ТРУДЫ

ИНСТИТУТА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ РАН

PROCEEDINGS OF THE INSTITUTE FOR SYSTEM PROGRAMMING OF THE RAS

ISSN Print 2079-8156
Том 33 Выпуск 4

ISSN Online 2220-6426
Volume 33 Issue 4

Институт системного
программирования
им. В.П. Иванникова РАН

Москва, 2021

ИСП РАН

ISSN 2079-8156 (Print), ISSN 2220-6426 (Online)

Труды Института системного программирования РАН
Proceedings of the Institute for System Programming of the RAS

Труды ИСП РАН – это издание с двойной анонимной системой рецензирования, публикующее научные статьи, относящиеся ко всем областях системного программирования, технологий программирования и вычислительной техники. Целью издания является формирование научно-информационной среды в этих областях путем публикации высококачественных статей в открытом доступе.

Издание предназначено для исследователей, студентов и аспирантов, а также практиков. Оно охватывает широкий спектр тем, включая, в частности, следующие:

- операционные системы;
- компиляторные технологии;
- базы данных и информационные системы;
- параллельные и распределенные системы;
- автоматизированная разработка программ;
- верификация, валидация и тестирование;
- статический и динамический анализ;
- защита и обеспечение безопасности ПО;
- компьютерные алгоритмы;
- искусственный интеллект.

Журнал издается по одному тому в год, шесть выпусков в каждом томе.

Поддерживается открытый доступ к содержанию издания, обеспечивая доступность результатов исследований для общественности и поддерживая глобальный обмен знаниями.

Труды ИСП РАН реферируются и/или индексируются в:

Proceedings of ISP RAS are a double-blind peer-reviewed journal publishing scientific articles in the areas of system programming, software engineering, and computer science. The journal's goal is to develop a respected network of knowledge in the mentioned above areas by publishing high quality articles on open access.

The journal is intended for researchers, students, and practitioners. It covers a wide variety of topics including (but not limited to):

- Operating Systems.
- Compiler Technology.
- Databases and Information Systems.
- Parallel and Distributed Systems.
- Software Engineering.
- Software Modeling and Design Tools.
- Verification, Validation, and Testing.
- Static and Dynamic Analysis.
- Software Safety and Security.
- Computer Algorithms.
- Artificial Intelligence.

The journal is published one volume per year, six issues in each volume.

Open access to the journal content allows to provide public access to the research results and to support global exchange of knowledge. **Proceedings of ISP RAS** is abstracted and/or indexed in:

УДК004.45

Адрес: 109004, г. Москва, ул. А. Солженицына, дом 25.
Телефон: +7(495) 912-44-25
E-mail: proceedings@ispras.ru
Сайт: https://ispranproceedings.elpub.ru/

Address: 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Tel: +7(495) 912-44-25
E-mail: proceedings@ispras.ru
Web: https://ispranproceedings.elpub.ru/

Содержание

Table of Contents

DOI: 10.15514/ISPRAS-2021-33(4)-1

Validation Automation of UML Diagrams Created by Students

T.S. Gasheva, ORCID: 0000-0002-8095-4538 <gasheva_99@mail.ru>
D.I. Vlasov, ORCID: 0000-0002-1968-5148 <dima.vlasov@icloud.com>
A.V. Otinov, ORCID: 0000-0002-4226-5694 <otinovandry@gmail.com>
N.N. Datsun, ORCID: 0000-0001-8560-7036 <nndatsun@inbox.ru>

Perm State University,
15, Bukireva st., Perm, 614990, Russia

Abstract. Unified Modeling Language (UML) is widely used standard for models visualization in software industry. Hence, a preparation of IT professionals involves the learning modeling process. Studies of student perception of UML modeling indicate that this process is perceived as quite complex. This paper presents software for validation activity, class and use-case diagrams by XMI representation. To achieve this goal, we researched existing methods and systems. Besides, we analyzed mistake catalogues and Perm State University's student models to propose a mistake classification and checklist that presents a list of validation to be done. This paper focuses on validation each type of diagram separately, without maintaining consistency between different UML models. However, all these validation modules are combined in one system, which allows to check any of the described types of diagrams.

Keywords: validation; UCD; AD; CD

For citation: Gasheva T.S., Vlasov D.I., Otinov A.V., Datsun N.N. Validation Automation of UML Diagrams Created by Students. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021, pp. 7-18. DOI: 10.15514/ISPRAS-2021-33(4)-1

Автоматизация проверки UML диаграмм, созданных студентами

Т.С. Гашева, ORCID: 0000-0002-8095-4538 <gasheva_99@mail.ru>
Д.И. Власов, ORCID: 0000-0002-1968-5148 <dima.vlasov@icloud.com>
А.В. Отинов, ORCID: 0000-0002-4226-5694 <otinovandry@gmail.com>
Н.Н. Дацун, ORCID: 0000-0001-8560-7036 <nndatsun@inbox.ru>

Пермский государственный национальный исследовательский университет,
614068, Россия, г. Пермь, ул. Букирева, д. 15

Аннотация. Унифицированный язык моделирования (UML) является широко используемым стандартом для визуализации моделей в отрасли программного обеспечения. Следовательно, подготовка ИТ-специалистов включает в себя обучение моделированию. Исследования восприятия студентами UML-моделирования показывают, что обучение проходит довольно непросто. В данной статье представлено программное обеспечение для проверки диаграмм активности, классов и прецедентов по их представлению в формате XMI. Для достижения этой цели мы исследовали существующие методы и системы. Кроме того, мы проанализировали каталоги ошибок и модели студентов Пермского государственного национального исследовательского университета, чтобы предложить классификацию ошибок и контрольный список, который представляет список необходимых проверок. В данной статье основное внимание уделяется проверке каждого типа диаграмм независимо друг от друга, без сохранения согласованности между различными моделями

UML. Однако все эти модули проверки объединены в одну систему, которая позволяет проверить любой из исследованных типов диаграмм.

Ключевые слова: проверка; UCD; AD; CD

Для цитирования: Гашева Т.С., Власов Д.И., Отинов А.В., Дацун Н.Н. Автоматизация проверки UML диаграмм, созданных студентами. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 7-18 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(4)–1

1. Introduction

Modern approaches to validation student models are UML is a standard that provides system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes [1]. This standard is widely used in the software industry.

On the one hand, it is used in Object-oriented analysis and design (OOAD) in the development of complex systems [2]. Formal methods or model-based specification [3] are used to verify such models [4], [5].

On the other hand, a preparation of IT professionals involves the learning modeling process [6] and Model Driven Architecture (MDA) [7]. Studies of student perception of UML modeling indicate that this process is perceived as quite complex. This opinion is shared by both computer scientists [8], [9] and computer science majors [8].

Besides, the process of manual validation of models is a time-consuming process, especially during the review of dozens of student models by the teacher. Therefore, the creation of such system is actual.

This paper presents a description of a system for automatically checking use case (UCD), class (CD) and activity diagrams (AD) based on an XMI [10] representation. This format can be exported from most Case-tools creating UML diagrams and contains description of the diagram elements.

Not all types of UML diagrams were chosen for research. To automate the validation, two analysis phase models were chosen - UCD and AD, and one design phase model - CD. This choice is based on the experience of checking these models and on others papers [8], [11].

We researched existing tools for validation UML diagrams. We put forward the criteria of applicability to the solution of our problem for the found tools but they do not fulfill our criteria.

Modern approaches to validation student models are based on the use of catalogues or lists of common mistakes [12], [13], [14], [15], [16]. Using the results of these catalogues and own review models of Perm State University (PSU) students we classify the mistakes in two dimensions. The first dimension represents the severity of the mistake. We distinguish three types of severity: Warning, Serious, Fatal. In the second dimension, we consider three main categories of mistakes: Model, Consistency, Layout. Model mistakes contains subcategory.

In this study, we consider in detail only the model mistakes. Using these mistakes, checklists for UCD, CD and AD are composed. These checklists are used to validate diagrams using Case-based reading method (CBR) [17]. Besides, for checking AD we use a graph with semantics similar to the Petri net, which was mentioned in the article [18], and colored tokens, similar to tokens in colored Petri nets [19].

To this end, we present the system implementation and demonstrate case studies. The quality of mistakes detections by the system is high, which confirms the achievement of the research goal.

2. Related Works

2.1 Analysis of Existing Mistakes Classifications

Chren et al. [12] evaluated over 2700 UML diagrams and examined students' mistakes with use case diagrams, activity diagrams, class diagrams, state machine diagrams, sequence diagrams,

communication diagrams, and entity-relationship diagrams. They produced their catalogue from papers and their own research and identified 146 mistakes. They split all mistakes in two dimensions – the first one represents the severity of mistakes, the second refers to the nature of the mistakes. The overview of the classification scheme is in fig. 1 [12].

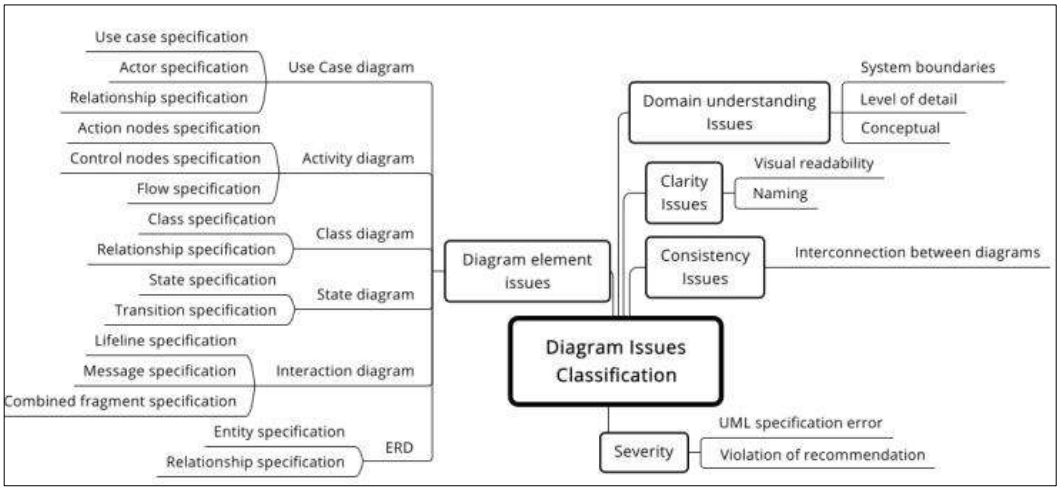


Fig. 1. Orthogonal classification scheme for diagram issues

They figured out that the most common mistakes for UCD, AD, and CD are Diagram element issues. The detection rate for top four mistakes in UCD is 31-42.9%, in AD - 26.2-42.9%, in CD - 42.9-59.5%.

Delgado et al. [16] considered use case, class, state-machine, sequence diagrams. They separated mistakes on six categories: Layout mistakes, Traceability, Notation, Semantic, Documentation, Naming, Conventions. They concluded that in UCD the most common mistakes are Notation mistakes and the least common are Traceability mistakes. In CD the most common defects are Documentation and the least common Traceability.

Reuter [8] classified mistakes in a category system that is commonly known from errors in programming languages and the Standard Classification for Software Anomalies [20]: Lexical, Syntactic, Semantic, Logic, Missing, and Unnecessary. Unfortunately, the numbers of mistakes were not presented, so that no conclusions could be drawn about the frequency of mistakes.

Bolloju [15] separated 380 mistakes in the 14 projects in three groups: Syntactic, Semantic, Pragmatic. They found out that the most frequent types of mistakes for UCD and CD are Syntactic (57% for UCD and 64% for CD) and Semantic (64% and 50%, respectively).

In our work, we use classification structure similar to Chren's. We also use two dimensions, one of which is severity and another links with the origin of the mistake. However, we use different severity subcategories. It relates with our future work – the development of system of qualifying the degree of deviation of the student model.

2.2 Analysis of Existing Software for Validation of UML Diagrams

Special criteria were identified for the existing UML diagram validation software:

- meet all the standards of the UML language;
- open source and free to use;
- support for graphical display of diagrams;
- software is easy to install and use.

The first two criteria are critical when analyzing existing software because of the target group of users – teachers and students.

For UC validation, two software products were discovered: FOAM [21] tool and Rational Rose [22]. FOAM it's open-source project, but disadvantage of this tool is that in order to analyze an existing UCD, you should independently translate all the contents of the diagram into a special text format, in which you must manually specify all the established elements and the relationships between them. Also, this tool doesn't have a graphical interface. Rational Rose it's a commercial product, that supports modeling UCDs and their continues validation. But the list of checking mistakes is quite small.

For AD validation, there are analyzed tools such as UML-VT [23] and Woflan [24]. These funds also do not match the main criteria identified earlier.

In existing publications [25], there are only brief descriptions of algorithms for validating CD, and it is impossible to study and analyze them in detail, since they are in the private domain. The set of libraries used in private validation systems: Eclipse (2000 LoC) and Java classes (11500 LoC) [26] [26], Dresden OCL toolkit [26], [27] extensible libraries (for processing and loading constraints) and MDR (for importing/exporting UML models from XMI [10]).

2.3 Analysis of UML Diagram Creation Software Providing Metadata

The choice of the software that will be used for the construction of diagrams is an important step in this work, since the chosen software tool will determine the possibility and success of further analysis, design and implementation of the prototype. That is why special requirements were defined for the selection process of competing modeling systems. The result of the research in this issue was the choice of the GenMyModel [28]. It combines a simple user interface, does not require installation and has the function to export the diagram in the required formats. The advantage of this tool is that when building diagrams, it does not allow you to perform some activities that can lead to mistakes. Due to this, the list of conditions to be checked can be reduced.

Based on this, it was decided that the input data (XMI and PNG files) will be generated using the GenMyModel tool.

3. Classification of Students Mistakes

Based on existing catalogues and own student's models analysis, we composed the classification.

1) Model: The Model category covers mistakes that violate the UML specification or recommendation.
   a) Lexical.
   b) Syntactic.
   c) Semantic.
2) Consistency: The Consistency category contains mistakes that are related to maintaining the dependencies between the diagrams.
3) Layout: The mistakes in the Layout category are caused by incorrect arrangement of elements on the diagram (overlapping elements, crossing of relationship lines); these mistakes are extremely difficult to detect using the XMI representation.
4) Severity.
   a) Warning.
   b) Serious.
   c) Fatal.

The Severity category depends on how accurately the mistake can be detected by system (warning – if the mistake cannot be accurately identified by the system), on the possibility of further validation

(fatal mistakes in AD lead to the termination of further validation) and on the recommendation (warnings for minor mistakes, fatal for major mistakes).

Model mistakes are also divided into lexical, syntactic, and semantic subcategories. On lexical step, the diagram elements are considered separately, individually (a mistake in the name of the element, the type of the element does not belong to the diagram). On syntactic step, we consider mistakes related to the constructing diagram rules, the rules for connecting diagram elements. Semantic captures the quality of a model. This category covers mistakes relating to invalid representation of domain, violation of the boundaries of the system, incorrect display of meaning.

In this work, we consider in detail only the model mistakes. GenMyModel does not allow to make some syntax mistakes in UCD, AD and CD, so we did not include them in the checklists. Table 1 presents the examples of Model mistakes.

Table 1. Examples of model mistakes

Mistake	Subcategory	Severity
Invalid actor name: should be represented by a noun, starting with a capital letter	Lexical	Serious
More than one initial node	Syntactic	Fatal
When specifying the roles multiplicity, some numbers are negative integers	Semantic	Serious

4. Validation Methods

4.1 Approach for Use-Case Diagram Validation

A research was carried out among the existing UCD validation methods. For the use case diagrams, the following were identified: Object-Oriented Reading Techniques (OORT) [29] and CBR [30] – reading based on a list of requirements.

Since we have a list of mistakes, the CBR methodology was chosen. CBR is a very common method. List of mistakes should be checked during the validation. CBR provides more aid and advice to the inspectors than ad-hoc reading and is therefore a very common technique.

Initially, the XML document is read into the program internal data representation. Each chart element has a unique identifier, coordinates on the image, and a name or description. On the reading stage, some types of mistakes can be verified (mostly lexical). Items not included in the UCD list are excluded.

The next stage involves conducting sequential checks of each type from the list of the most common mistakes of students.

4.2 Approach for Activity Diagram Validation

To solve the AD validation problem, the analysis of existing methods was carried out, such as the construction of a Petri net [31], the use of temporal logic [32], as well as graph with semantics similar to Petri Net [18].

For this work, we use a subset of UML elements. We consider the initial node, final node, decision node, merge node, activity node, fork node, join node, swimlane, comment node, flow.

Now we describe the basic idea of the validation process. For each element in AD, there is a class in our system. Each class has field for token and field for list of links on the next objects. Besides, it can contain additional information about AD element. For each AD element, the object is created

and placed in a graph. The graph's vertices represent AD's nodes and the graph's edges represent AD's flows.

The validation is divided into two steps. At the first step, lexical, semantic and part of syntactic mistakes are checked. Then we check unpaired using fork and join by modeling token flow through the graph.

After the model passes the first steps of validation, in order to continue validation, we impose some restrictions. The restrictions are as follows.

1) AD must have exactly one initial node, one or more final nodes and at least one activity node,
2) initial node has no incoming edge and the final node has no outgoing flow,
3) activity, merge, join and init nodes must have exactly one outgoing flow,
4) fork and decision nodes can have any number of outgoing flows,
5) activity, fork, decision, final nodes should have only one incoming flow,
6) each join and merge node can have any number of incoming flows,
7) flows cannot start and finish in the same node.

If the restriction was violated, we finish validation without graph checking.

The tokens flow through the graph along the edge directions from initial to final node. The validation is completed when all token flows are checked or a mistake is found. In this case mistakes are the situations when several tokens appear in one node at once or when a token remains in the graph upon reaching the final node or when deadlock occurs (the situation when there is no token can be moved).

According to [18], the graph uses token-flow semantics. Each element has own set of rules, which ensure the token flows through the graph.

The state of the graph at each step can be encoded with a sequence of zeros and ones, where zero means that the element is inactive (does not have a token), and one means that it is active. A stack of current masks and a set of checked masks are created. At each step, the top mask is taken from the current masks and processed. The processing's result is new masks, that are checked for use early using a set of used masks and, if they have not been previously used, are added to the list of current masks.

At each step, all existing tokens are moved to one of the next nodes. In the case of a decision node, a token can activate a random element. Therefore, it generates several possible next states that are pushed into the current mask stack.

However, the problem of unpaired use of a joint and a fork remains. When there are several fork nodes corresponding to one join nodes, the join can be activated wrongly. Fig. 2 presents this issue.

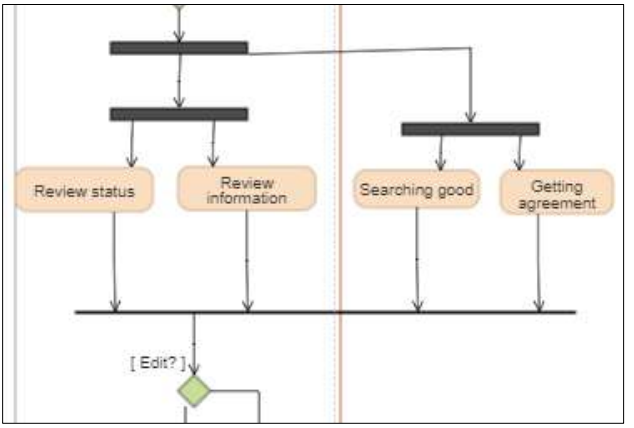


Fig. 2. Several fork nodes corresponding to one join nodes

To figure out this kind of mistake, it was proposed to use tokens of different colors. It is some additional data that is stored on the token's stack [19]. The fork node generates a unique color every time a token pass through it. The output tokens have the same color; it means that the fork's color is placed on the token's stack of colors. For join node activation, it is necessary that the colors at the top of the stacks have the same color. If the condition is right the join node becomes activated, and the output token remains all colors except the top color. In other case, a fatal mistake occurs and validation is completed.

4.3 Approach for Class Diagram Validation

There are very few fully automated methods for validating CD. Most of the existing solutions require a translation process into specific data formats that must be performed by a human. This approach is not suitable, since we need to quickly validate the diagrams, and the translation process takes a sufficient amount of time.

The validation process is based on and similar to the program compilation analysis stage, and it can be divided into three stages: the first stage is lexical analysis, the second is syntactic analysis and the third stage is semantic analysis. At each stage, the corresponding rules will be checked. During the first stage, metadata is converted into a set of tokens, the use of invalid tokens, incorrect names, designations and properties of tokens is detected. During the second stage, the correctness of creating constructions of the UML language from a valid set of tokens. And at the final stage of validation, the semantics of the constructed class diagram is considered, namely the correctness of the semantic meanings of words, phrases and elements.

Validation process begins with reading all data about the model from the XMI file. All properties of the CD tokens can be retrieved from these data. Already on the basis of these properties, it will be easy to detect some inconsistencies and mistakes.

The main point in this method is to designate a set of rules for constructing UML CD such as to identify all mistakes in the validated diagram. The set of rules was compiled using the UML specification [1]. The list of all rules that will be checked during validation was described in detail earlier. Also, special attention will be paid to common mistakes when constructing CD.

5. Implementation

For developing the system, we used C#. The process of validation system creating was divided into two stages:

1) implementation of the UCD, AD, and CD validation modules in prototype mode as a console application and presentation the result in the form of text message,
2) integration of UCD, AD, and CD validation modules into the system with a user graphical interface.

The system has the following features:

1) the ability to upload one or several files into the system;

2) the ability to validate one or several models;

3) the ability to automatically find the pair "metadata - image" while files are uploading into the system;

4) the ability to add and remove diagrams from the current list of models;

5) the ability to work only with metadata diagrams (without images);

6) dynamic changing the graphical presentation of diagrams while switching is occurring between them;

7) dynamic mistakes designation on the graphical presentation of the diagram while switching is occurring between mistakes;

8) highlighting each mistake in a different color depending on its severity;

9) the ability to check all diagrams with "not checked" status at once.

Fig. 3 shows the results of CD validation.

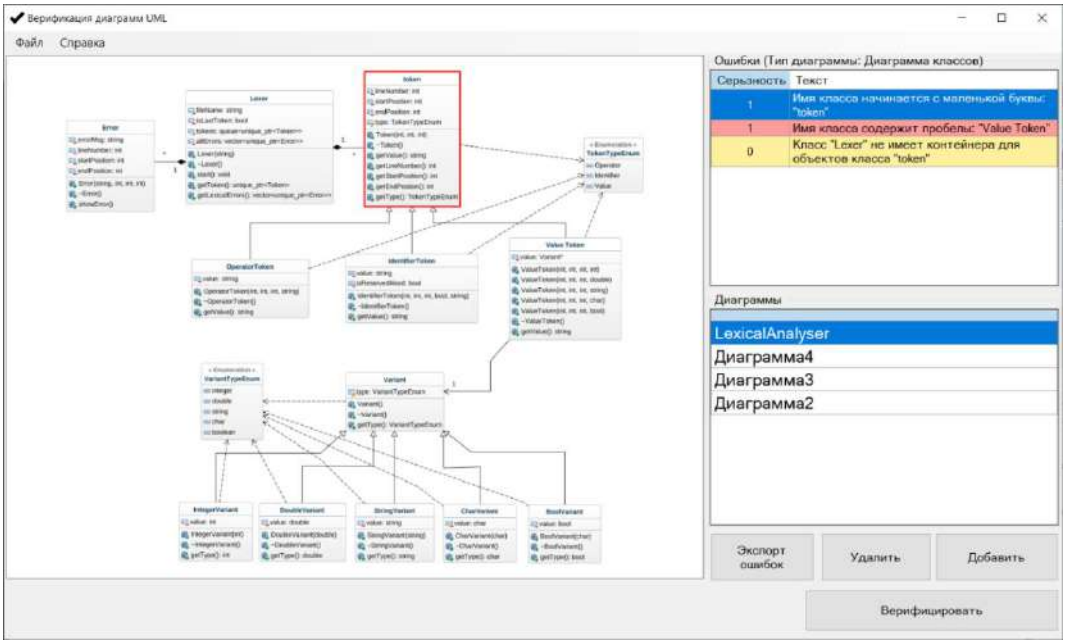


Fig. 3. Results of CD validation

6. Case Study

We analyzed the quality of projects developed in PSU by full-time undergraduate majors "Software" and "Computer security» (course «Modeling of Information Systems») and part-time specialty "Information technologies" (course "Design and implementation of information systems»). These are projects of information systems (IS) for business applications, computer security, and information technology. Each project includes models of three stages of IS life cycle: analysis (UCD, AD, sequence diagrams), design (collaboration diagrams and CD), implementation (component diagrams and deployment diagrams). The projects were carried out by teams of 2-3 students.

The checklists used in CBR to validation diagrams include 6, 10, and 5 lexical mistakes; 14, 15, and 8 syntactic, 7, 1, and 2 semantic mistake for CD, AD, and CD.

Fig. 4 presents examples of mistakes in Table 1 in: a) for UCD, b) for AD, c) for CD.

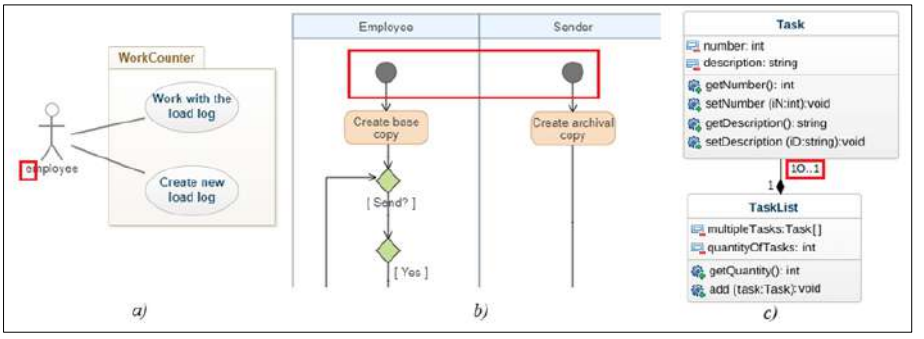


Fig. 4. Examples of mistakes found by system

7. Results

An analysis was carried out of 191 the work of IT students who create models of information systems based on an object-oriented approach to modeling. UCD module was tested on 70 student models, AD – 41, CD – 80. UCD module can detect 25 mistakes, AD – 26, CD – 26.

Fig. 5 presents percentage of mistakes found by system. System did not find semantic mistakes in students' CD models.

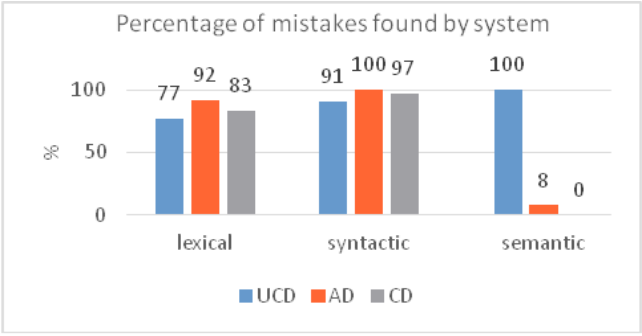


Fig. 5. Percentage of mistakes found by system

8. Discussion and Future Work

The current version of the UML diagram validation system solves the following tasks:

1) based on the exported XMI file, mistakes are searched in UCD, AD, and CD,
2) visualization of found mistakes and their display on exported diagram images.

It can be recommended for use by two categories of users:

1) students: to check models before submitting for teacher's grading.
2) teachers: to validate models of students.

For the future, we will work on the validation functions for teachers in order to qualify the degree of deviation of the student model from the task specification and to form the recommended score for the model.

9. Conclusion

We proposed a mistakes' classification with two dimensions and observed model mistakes in detail. We justified the choice of the tool for creating UML diagrams. Modules for validation of UCD, AD, CD were developed and realized. These modules were integrated into a system allowing package processing of model files. In the end, we tested the system on 191 student models.

References / Список литературы

[1]. Unified Modeling Language 2.5.1. (2017). Object Management Group, Available at: https://www.omg.org/spec/UML/About-UML/, accessed 03.09.2021.

[2]. Boberić Krstićev D., Tesendic D. Experience in Teaching OOAD to Various Students. Informatics in Education, vol. 12, 2013, pp. 43-58.

[3]. Koznov D.V. Methodology and tools for domain-specific modeling. Doctor Degree thesis. Saint-Petersburg, 2016, 430 p. (in Russian) / Кознов Д.В. Методология и инструментарий предметно-ориентированного моделирования. Диссертация доктора технических наук. СПб., 2016 г., 430 стр.

[4]. Baresi L., Morzenti A. et al. A logic-based approach for the verification of UML timed models. ACM Transactions on Software Engineering and Methodology, vol. 26, issue 2, 2017, article no. A7.

[5]. Daw Z., Mangino J., Cleaveland R. UML-VT: A Formal Verification Environment for UML Activity Diagrams. In Proc. of the MoDELS 2015 Demo and Poster Session co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015), 2015, pp. 48-51.

[6]. Bourque P., Dupuis R. et al. Guide to the software engineering body of knowledge. IEEE Software, vol. 16, no. 6, 1999, pp. 35-44.
[7]. Object Management Group. MDA Guide revision 2.0. 2014, Available at: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01, accessed 03.09.2021.
[8]. Reuter R., Stark T. et al. Insights in Students' Problems during UML Modeling. In Proc. of the IEEE Global Engineering Education Conference (EDUCON), 2020, pp. 592-600.
[9]. Matyokurehwa K., Makoni K.T. Students' Perceptions in Software Modelling Using UML in Undergraduate Software Engineering Projects. International Journal of Information and Communication Technology Education, vol. 15, no. 4, 2019, article no. 2.
[10]. Object Management Group. XML Metadata Interchange (XMI) Specification 2.5.1. 2015. Available at: https://www.omg.org/spec/XMI/2.5.1/PDF, accessed 03.09.2021.
[11]. Lima V., Talhi C. et al. Formal Verification and Validation of UML 2.0 Sequence Diagrams using Source and Destination of Messages. Electronic Notes in Theoretical Computer Science, vol. 254, 2009, pp. 143-160.
[12]. Chren S., Buhnova B. et al. Mistakes in UML Diagrams: Analysis of Student Projects in a Software Engineering Course. In Proc. of the IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), 2019, pp. 100-109.
[13]. Fernández-Sáez A.M., Caivano D. et al. On the use of UML documentation in software maintenance: Results from a survey in industry. In Proc. of the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2015, pp. 292-301.
[14]. Chytalová K. Catalog of errors in UML diagrams PB007 - software engineering I. Lasaris Lab, Faculty of Informatics, Masaryk University, 2018, Available at: https://drive.google.com/file/d/1J3_Ueb4E2YdAZjksryC4-F123Xqmyhkm/view, accessed 03.09.2021 (in Czech).
[15]. Bolloju N., Leung F.S.K. Assisting novice analysts in developing quality conceptual models with UML. Communications of the ACM, vol. 49, no. 7, 2006, pp. 108-112.
[16]. Delgado A., Dias A., Brito e Abreu F. Verification and Validation of UML Diagrams using Checklists. Available at: https://moodle.fct.unl.pt/pluginfile.php/22771/mod_folder/content/0/ArtigoGrupoB.pdf?forcedownload=1, accessed 03.09.2021.
[17]. Sabaliauskaite G., Matsukawa F. et al. An experimental comparison of checklist-based reading and perspective-based reading for UML design document inspection. In Proc. of the International Symposium on Empirical Software Engineering, 2002, pp. 148-157.
[18]. Rafe V., Rahmani A.T. Formal Analysis of Workflows Using UML 2.0 Activities and Graph Transformation Systems. Lecture Notes in Computer Science, vol. 5160, 2008, pp. 305-318.
[19]. Jensen K. A brief introduction to coloured Petri Nets. Lecture Notes in Computer Science, vol. 1217, Berlin, 1997, pp. 203-208.
[20]. IEEE 1044-2009 - IEEE Standard Classification for Software Anomalies. 2010, 23 p.
[21]. Vinarek J., Imko V., Hnetynka P. Verification of Use-Cases with FOAM Tool in Context of Cloud Providers. In Proc. of the 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2015), 2015, pp. 151-158.
[22]. Rational Rose. Available at: https://www.ibm.com/software/developer/rosexde/, accessed 03.09.2021.
[23]. UML-VT. Available at: http://www.cs.umd.edu/~rance/projects/uml-vt/, accessed 03.09.2021.
[24]. Woflan. Available at: https://www.win.tue.nl/woflan/doku.php/, accessed 03.09.2021.
[25]. Mokhati F., Gagnon P., Badri M. Verifying UML Diagrams with Model Checking: A Rewriting Logic Based Approach. In Proc. of the Seventh International Conference on Quality Software (QSIC 2007), 2007, pp. 356-362.
[26]. Cabot J., Claris'o R., Riera D. Verification of UML/OCL Class Diagrams using Constraint Programming. In Proc. of the IEEE International Conference on Software Testing Verification and Validation Workshop, 2008, pp. 73-80.
[27]. Dresden OCL. (2016), Available at: https://github.com/dresden-ocl/dresdenocl, accessed 03.09.2021.
[28]. GenMyModel. Available at: https://www.genmymodel.com/, accessed 03.09.2021.
[29]. Conradi R., Mohagheghi P. et al. Object-Oriented Reading Techniques for Inspection of UML Models – An Industrial Experiment. Lecture Notes in Computer Science, vol. 2743, 2003, pp. 483-500.
[30]. Naveed A., Ikram N. A novel checklist: Comparison of CBR and PBR to inspect use case specification. Communications in Computer and Information Science, vol. 558, 2015, pp. 109-125.

[31]. Baresi L., M. Pezzè. On Formalizing UML with High-Level Petri Nets. Lecture Notes in Computer Science, vol. 2001, 2001, pp. 276-304.

[32]. Araujo J., A. Moreira. Integrating UML Activity Diagrams with Temporal Logic Expressions. In Proc. of the 10th International Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'05), 2005, pp. 91-98.

Information about authors / Информация об авторах

Dmitry Igorevich VLASOV – student. His research interests include modeling languages, object-oriented programming, modeling tools, programming language toolkits.

Дмитрий Игоревич ВЛАСОВ – студент бакалавриата. Научные интересы включают языки моделирования, объектно-ориентированное программирование, средства моделирования, инструментарии языков программирования.

Tatiana Sergeevna GASHEVA – student. Her research interests include modeling, UML, object-oriented programming.

Татьяна Сергеевна ГАШЕВА – студент бакалавриата. Научные интересы включают моделирование, язык UML, объектно-ориентированное программирование.

Andrei Valerievich OTINOV – student. Research interests: object-oriented programming, modeling, information systems.

Андрей Валерьевич ОТИНОВ – студент. Научные интересы: объектно-ориентированное программирование, моделирование, информационные системы.

Nataliya Nikolaevna DATSUN – Candidate of Physical and Mathematical Sciences, associate professor of the Computer Science Department. Research interests: modeling languages, modeling tools, object oriented modeling.

Наталья Николаевна ДАЦУН – кандидат физико-математических наук, доцент, доцент кафедры математического обеспечения вычислительных систем. Сфера научных интересов: языки моделирования, средства моделирования, объектно-ориентированное моделирование.

DOI: 10.15514/ISPRAS-2021-33(4)-2

Integration of micro-services as components in modeling environments for low code development

[1,2]*H.A.A. Chaudhary, ORCID: 0000-0001-9272-2622 <ahmad.chaudhary@ul.ie>*
[1,3]*T. Margaria, ORCID: 0000-0002-5547-9739 <Tiziana.Margaria@ul.ieg>*

[1] *University of Limerick,*
Limerick, V94 T9PX, Ireland
[2] *Confirm Research Centre for Smart Manufacturing*
IBC Block2, University of Limerick, Limerick, Ireland
[3] *Lero: The Irish Software Research Centre*
Tierney Building, University of Limerick, Ireland

Abstract. Low code development environments are gaining attention due to their potential as a development paradigm for very large scale adoption in the future IT. In this paper, we propose a method to extend the (application) Domain Specific Languages supported by two low code development environments based on formal models, namely DIME (native Java) and Pyro (native Python), to include functionalities hosted on heterogeneous technologies and platforms. For this we follow the analogy of micro services. After this integration, both environments can leverage the communication with pre-existing remote RESTful and enterprise systems' services, in our case Amazon Web Services (AWS) (but this can be easily generalized to other cloud platforms). Developers can this way utilize within DIME and Pyro the potential of sophisticated services, potentially the entire Python and AWS ecosystems, as libraries of drag and drop components in their model driven, low-code style. The new DSLs are made available in DIME and Pyro as collections of implemented SIBs and blocks. Due to the specific capabilities and checks underlying the DIME and Pyro platforms, the individual DSL functionalities are automatically validated for semantic and syntactical errors in both environments.

Keywords: Domain Specific Language (DSL); Model Driven Development (MDD); eXtreme Model Driven Development (XMDD); Service Independent Building Blocks (SIBs); Low code development environments; DIME; Pyro

For citation: Chaudhary H.A.A., Margaria T. Integration of micro-services as components in modeling environments for low code development. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021, pp. 19-30. DOI: 10.15514/ISPRAS-2021-33(4)-2

Acknowledgements. This work was supported by the Science Foundation Ireland grants 13/RC/2094 (Lero, the Irish Software Research Centre) and 16/RC/3918 (Confirm, the Smart Manufacturing Research Centre).

Интеграция микросервисов как компонентов сред моделирования для малокодовой разработки

[1,2] *Х.А.А. Чаудхари, ORCID:* 0000-*0001-9272-2622 <ahmad.chaudhary@ul.ie>*
[1,3] *Т. Маргария, ORCID: 0000-0002-5547-9739 <Tiziana.Margaria@ul.ieg>*

[1] *Университет Лимерика,*
Ирландия, V94 T9PX, Лимерик
[2] *Confirm: Исследовательский центр интеллектуального производства*
Ирландия, Лимерикский университет, IBC Block2
[3] *Lero: Ирландский центр исследований программного обеспечения*
Ирландия, Лимерикский университет, Здание Тирни

Аннотация. Среды разработки с низким кодом привлекают внимание из-за их потенциала в качестве парадигмы разработки для очень крупномасштабного внедрения в ИТ будущего. В этой статье мы предлагаем метод расширения (приложений) предметно-ориентированных языков, поддерживаемых двумя средами разработки с низким уровнем кода, основанными на формальных моделях, а именно DIME (родная Java) и Pyro (родной Python), для включения функций, размещенных на гетерогенных технологиях и платформы. Для этого мы следуем аналогии с микросервисами. После этой интеграции обе среды могут использовать связь с уже существующими удаленными службами RESTful и корпоративных систем, в нашем случае Amazon Web Services (AWS) (но это можно легко распространить на другие облачные платформы). Таким образом, разработчики могут использовать в DIME и Pyro потенциал сложных сервисов, потенциально всей экосистемы Python и AWS, в виде библиотек перетаскиваемых компонентов в управляемом ими стиле с низким кодом. Новые DSL доступны в DIME и Pyro как коллекции реализованных SIB и блоков. Из-за особых возможностей и проверок, лежащих в основе платформ DIME и Pyro, отдельные функции DSL автоматически проверяются на семантические и синтаксические ошибки в обеих средах.

Ключевые слова: предметно-ориентированный язык (DSL); управляемая моделями разработка (MDD); экстремальная управляемая моделями разработка (XMDD); сервисно-независимые компоненты (SIB); среды малокодовой разработки; DIME; Piro

Для цитирования: Чаудхари Х.А.А., Маргария Т. Интеграция микросервисов как компонентов сред моделирования для малокодовой разработки. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 19-30 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(4)–2

Благодарности: Эта работа была поддержана грантами Ирландского научного фонда 13/RC/2094 (Lero, Ирландский исследовательский центр программного обеспечения) и 16/RC/3918 (Confirm, Исследовательский центр интеллектуального производства).

1. Introduction

Low code development platforms enable their users to design and develop applications with minimal coding knowledge [1], with the support of drag-and-drop visual interfaces that operate on representations of code as encapsulated code wrappers. The main aim [2] of these platforms is to produce flexible, cost effective and rapid applications in a model driven way. Ideally, they are adaptive to enhancements and less complex is terms of maintenance. Model-driven development (MDD) is an approach to develop such systems using models and model refinement from the conceptual modelling phase to the automated model-to-code transformation of these models to executable code [3]. The main challenges with traditional software development approaches are the complexity in development at large scale, the maintenance over time, and the adaptation to dynamic requirements and upgrades [1]. Doing this on source code is costly, and it systematically excludes the application domain experts. who are the main knowledge and responsibility carriers. At the same time, the cost of quality documentation and training of new human resources for code-based development are other concerns in companies and organizations that depend on code.

Domain Specific Languages (DSLs) conveniently encapsulate most complexities of the underlying application domain. Encapsulation of code and abstraction to semantically faithful representations in models empowers domain experts to take advantage of these platforms. They can develop products in an efficient manner and also meet the growing demands of application development without having deep expertise in software development. Based on a study [4] from 451 researches, the maintenance effort with low code platforms proved to be 50-90% more efficient as compared to changes with classical coding languages.

Software systems in general, and especially web apps in internet-centered ecosystems and digital threads in an Industry 4.0 context, are not isolated in nature: they demand interaction with various external systems, libraries and services. Frequent needs are (but not limited to)

- acquire sensors data from external systems,
- feed data to external dashboards for analytics and publishing,
- utilize the compute power of cloud systems,
- sophisticated enterprise services.

In this context, microservices [5] play an important role at the enterprise level. The microservices paradigm (SOA done right) defines certain methods to design software services as suite of independently deployable components with the purpose of modularity, reusability and autonomy [5]. Different versions of these services may coexist in a system as a set of loosely coupled collaborative components and must be independently replaceable without impacting the operations of heterogeneous systems.

This paper proposes the integration of microservices as components in two graphical modelling development environments based on formal models: the general purpose, desktop DIME [6] Integrated Modelling Environment and the special purpose, web based Pyrus (Pyro) [7]. Their extension and integration with external systems through services extends the capabilities of these platform to meet wider communication needs (e.g. in the cloud), and also to take advantage of existing sophisticated enterprise services (e.g. AWS).

Low-code programming both at the API and the platform level is considered to be a game changer for the economy of application development. Gartner Inc., for example, predicts [8] that the size of the low-code development tools market will increase by nearly 30% year on year from 2020 to 2021, reaching a $5.8 billion value in 2021. They state that so far, this is the fastest and probably the simplest and most economical method of developing applications.

In this paper, Section 2 discusses the state of art, Section [3] states the problem, Section [4] gives an overview of the platforms used to extend the low-code DSLs. Section [5] explains the integration, architecture and implementation of SIBs in DIME (the desktop IME) and blocks in Pyro/Pyrus (the web IME). Finally, in Section 6 we conclude and discuss.

2. State of the art

Most domain specific languages today are at the coding level and do not leverage a model driven approach at the platform level. The rise in re-usability and maintainability demands paved the path to low code development environments and gained the attention of the developer's community [9]. The construction of meta-models behind these DSLs is challenging, since they must capture all the domain knowledge, i.e. provide both semantic and syntactic rules.

Ktrain [10] is a popular coding level DSL: a python wrapper that encapsulates Tensor Flow functionalities and facilitates developers to augment machine learning tasks with fewer lines of python code. Xatkit [11], still in early stages of development, increases the reusability of chat bots by evolving NLP/NLU engine for text analytics. At the language level they support several versions of bots, but the generation of chatbots from existing data sources at the framework level is in future plans. jABC [12] is a general purpose XMDD framework for the development of desktop and enterprise applications in model driven fashion. It enables its users to compose models by drag and

drop of reusable blocks into hierarchical graph structures that are executable (interpreted) and compilable. Aurera [1] is a standalone desktop system for business modelling and addresses the challenges of frequent changes to IT solutions. The system is in early stages of development and does not support communication with external systems.

DIME [6] is a general purpose MDD platform-level tool, suitable for agile development due to its rapid prototyping for web application development. It follows the One Thing Approach based on XMDD [13], in a lineage of development environments that traces back to the METAFrame'95 [14]. DIME supports both control flow and data flow modelling in its process diagrams. Control flow models admit a single start node but may have multiple end nodes, and nodes (called SIBs) representing single functionalities or sub-models are graphs, i.e. formal models. The SIBs are connected via directed edges depending on the business logic, with distinct edge types for dataflow and control-flow.

Agent-based modelling paradigm [15] is another popular approach to increase the development productivity in simulation environments. CaaSSET [16] is a Context-as-a-Service based framework to ease the development of context services. The transformation into executable services is semi-automatic.

The market segment of web based development environments is still relatively young. Not having many established environments, there is a huge potential for research and collaboration in this area. Theia [17], is a textual DSL tool supporting both desktop and web based IDEs.

Pyro [7] is a web base graphical modelling environment for the collaborative development of web applications based on DSLs. Pyro, like DIME, is itself a product modelled with the Cinco [18] Meta Tooling Framework, which provides a suite of textual DSLs in which to specify the models for which to generate editors. The MGL («Meta Graph Language») defines the structural information on the tool's model; the «Meta Style Language» (MSL) file specifies the visual characteristics (e.g. shapes and colors) of this model. The «Cinco Product Definition» (CPD) file specifies the details of the tool generation.

Both DIME and Pyro are advanced graph model editors generated in this way from Cinco specifications. In this sense, they share a common philosophy, the semantic and syntactic characteristics of their respective models and edit/check/manipulate capabilities are described formally in their MGL, MSL and CPD files.

To interact with external entities, Micro service [19] is a popular way to develop modular, reusable and autonomous service components. We adopt this approach to extend the functionalities of two of the platforms in a model driven way. Following the same principles of graphical microservices architectures, AjiL [20] is a good effort in this direction, but due to performance delays in complex applications, they shifted their focus from graphical to textual notations.

3. Problem Statement

We consider the DIME [6] and Pyro [7] Cinco-products as our case study. Both are graphical Integrated Modelling Environments for low-code/no-code application development, used to develop research [21, 22] as well as industrial applications. We will use DSLs to virtualize the technological heterogeneity of the services, delivering a simple, coherent and efficient extension to both low-code modelling platforms.

Concretely, we show how to extend the capabilities of the DSLs through new, heterogeneous services. We

1) extend DIME, an offline eclipse-based general-purpose MDD environment for Web applications, by integrating a generic RESTful service as a new component, technically adding a new executable SIB that a) represents and b) executes this REST service;
2) extend Pyrus/Pyro, a collaborative, web based special-purpose MDD environment for data analytics and AI/ML, by integrating cloud-based enterprise services in a similar fashion. Here we chose Amazon Web Services.

The models in the 2 IMEs are different: DIME has rich models that cover processes, data, GUI, roles and security, and supports both dataflow and control flow models. Pyrus is simpler, and supports only dataflow modelling, which is popular and sufficient in the analytics pipelines it addresses. As the specific integration depends on the characteristic and expressive power of the models, there are differences.

The extension by integration adds to the tools the capability to communicate with sophisticated enterprise ecosystems, without sacrificing the flexible yet intuitive modelling style for the no-code users, who just use the DSLs that are available.

4. Overview of the IMEs

Domain-specific languages aim at minimizing the domain/IT knowledge gap between domain experts and software developers by lifting the vocabulary, granularity and structure of the application domain into the modelling language, so that the modelling entities stay familiar to the domain experts and their intuition is indeed correct. Domain experts prefer graphical languages because of the haptic functionality of drag-and-drop from a collection of functionalities is an apt metaphor for the construction of complex behaviors from an appropriate network of identifiable, well understood building blocks along intuitive control flow and data flow patterns.

The effort to develop these tools from scratch is enormous. Consequently, the specialization and evolution of such tools is hindered by the sheer cost and complexity of managing their code and its quality and support. Cinco [18] was a game changer: a meta-level platform that wipes out this cost and complexity by providing the above described domain specific graphical modelling and code generation capabilities. Most Cinco products are based on Eclipse, enhanced with graphical modelling tools and various plug-ins. Suddenly, one can create a new Integrated Modelling Environment by specifying properties in three files and availing of the Cinco code generation capability for the target execution environment (e.g., eclipse or web). Modifications are not anymore at the code level: to change DIME or Pyro, one edits the specifying files and re-generates the tool with the appropriate generator.

In this paper[1], we will discuss the extension and integration of external systems as micro services in two of the Cinco's products DIME and Pyro (particularly Pyrus).

4.1 DIME

DIME is an Integrated Modeling Environment based on J2EE eclipse, to design, develop and deploy web applications in an agile paradigm. Its model types help users to graphically model and develop different aspects of ordinary web application: (i) data model, (ii) GUI model, (iii) business logic in terms of processes and persistence, and (iv) roles and security model. The specific functional capabilities are provided to the users as a family of Graphical DSLs. The GUI DSL and a DSL providing a collection of generic blocks (called SIBs, for Service Independent Building Blocks) come with DIME, and other, domain specific DSLs can be added at need.

Modelling in DIME happens mainly by the mechanism of drag-and-drop of DSL components on a canvas, and components comprise a node and a predefined set of outgoing edges. DIME supports both data and control flow to implement different aspects of business logic. Consistency checks are built-in in the DIME MGL and MSL, so that errors are either prevented (e.g., an output cannot connect to another output) or detected (e.g. the model is incomplete because some edges are dangling, not connected). DIME follows the One Thing Approach philosophy [23] by enforcing the SIBs to be generic and encapsulate only the required functionality. This way, SIBs are easily understandable and reusable, and application experts that are not coders can develop complex applications by using the SIBs in the provided DSLs. GUI models represent single pages of the web application and links to the underlying functionalities. Process models can be hierarchical, i.e.

[1] The complete project code is available in Github: https://github.com/ahmadch1991/syrcose21.

contain other process models, this way easing the organization and comprehension of the structure and behaviour of complex applications. Once the models are ready, the product generation step feeds the models collection to successive model-to-model and model-to-code transformers, resulting in a complete code generation for a standard web application runtime.

The setup for the development environment for DIME requires Java version 1.8 and eclipse dependencies to be installed on the development machine.

4.2 Pyro

In contrast to DIME, Pyro is a web based Cinco-product that runs in a web browser and turns it into a collaborative domain-specific graphical modelling environment for data-flow applications.

%The high level metamodeling is braced by MGL and MSL: The MGL depicts the syntactical constraints including the available nodes and edges; and the visual appearance of these modelling artefacts is defined in MSL. Pyro stores objects and data types in a loosely coupled manner [24]. To incorporate the rich features of typical web application, like the built-in support of cross-platform and a reusable components focused architecture, its front-end is built upon the Angular Dart [25] framework. To meet the needs of uninterruptible user interaction with the modelling environment, data exchange is implemented via non-blocking REST-based asynchronous communication. As a more recent development, Pyro is being enhanced with performance optimization and integration of external systems.

Pyrus is a specific Pyro derivative specialized for dataflow models executing within the popular Jupyter notebook environment. It is therefore particularly attractive for data analytics and AI applications, that are frequently coded in Python.

Working with Pyro/Pyrus requires the platform deployment on a local or remote server accessible via browser.

5. Proposed NPU architecture modifications

5.1 Memory synchronization via the shared bus

In this approach, the following modifications are proposed: memory devices containing the algorithm state are added to each pipeline stage. Memory devices on pipeline stages with equal depths are connected using a shared bus. The bus is used to synchronize data in the memory devices. When a memory cell in some memory device is updated, the new value and memory cell address are sent to other memory devices via the bus.

There are two operations available for the processing unit: read a value from a memory cell and write a value to a memory cell. When the value is read it is taken from the memory device that is on the same pipeline stage. When the value is written to the memory it is written to the memory device on the same pipeline stages and then is sent to appropriate memory devices in other pipelines.

5.2 Combining mechanism

In this approach, memory devices and processing units are connected using a packet switched network-on-chip with special switches that allow to combine memory requests. Memory requests have a Read-Modify-Write form. Memory request has a form of a tuple *(id, addr, f)* where *id* is a unique request identifier, *addr* is a memory cell address and *f* is a memory operation identifier. The response consists of two values: request identifier and the value that was in the memory cell before the memory operation was done.

This approach requires memory cells and network switches to have specialized arithmetic units to perform memory operations and memory request combinations.

6. Extending the IMEs

We show now how to extend DIME and Pyrus with RESTful services and cloud-based AWS services, respectively. This happens by implementing a new DSL consisting of a collection of capabilities that run on an external platform in a different technology. Effectively, these are akin to microservices. We show here exemplarily how to implement one such microservice for each case. The extension to other RESTful services or other AWS or similar services is then easy to achieve following these blueprints.

6.1 RESTful extension of DIME

We show now how to develop a generic Service Independent Building Block (SIB) in DIME that communicates with any external RESTful system.

Extending the DIME functionality happens by using the support of native library it provides. In DIME's multi-model type architecture, the business/logic model type is where the new SIBs will be utilized. As shown in fig. 1, the existing model architecture is extended with the addition of a native library as a new block belonging to the process/business logic model type. The native block will be merged to the process/business logic models during the automated code generation phase for the web application. Concretely, the extended functionality will be integrated as Java code with the remainder of the application during the compilation, and this way it will not add any additional performance penalty.

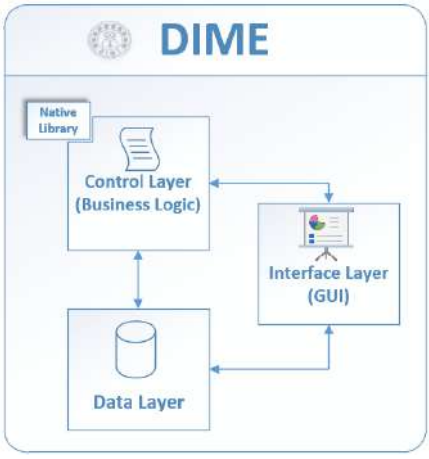


Fig. 1. DIME: Modelling Architecture and Native Library support

For the SIB implementation, we consider here a REST service that acts as a server and returns a list of country names on the basis of a country code input, e.g. United Kingdom for input ‘uk’, and the name of all countries from the database for input ‘all’. The service is implemented in PHP in a conventional fashion, and deployed on an external public server. It will respond to the requests generated by client SIBs

Now, we need to create a new client SIB with appropriate characteristics to communicate with RESTful service. This encompasses the SIB declaration and the SIB implementation.

The SIB declaration is shown in Listing 1.

```
package app.demo
sib rest_read_str_list : file_path#Java_fn
      url : text
      input_var : text
      input : text
      output : text
      -> success
```

```
        output: [text]
    -> noresult
    -> failure
```

Listing 1. SIB declaration for the «REST Read» SIB

- Firstly, in the project explorer we add a new, empty file with extension «.sib» and the name of the proposed SIB.
- This SIB declaration file contains the signature of the new SIB. It starts with the keyword «sib», followed by the new SIB name, that in our case is *REST_read_str_list*, followed by a colon and the path to the attached Java function. This is the function be invoked when the SIB is used in the process modelling.
- The next section contains the proper signature: the list of inputs and outputs, with name and data types. In our case, the SIB accepts the following I/O:
  - URL of an external server;
  - input variable name and data to create a valid URL at run time;
  - the output variable name is also added in the signature, to extract the requested data from server response for further JSON parsing.
- Finally, the list of different control branches based on outcomes. In our case the three branches are «success», which returns a text output provided by the external service, «noresult» of the external services returns no result, and «failure» in case of error in the communication with the external service.

For the SIB implementation, The RESTful «Rest Read» service is implemented in PHP in a conventional fashion, and deployed on an external public server. It will respond to the requests generated by this SIBs.

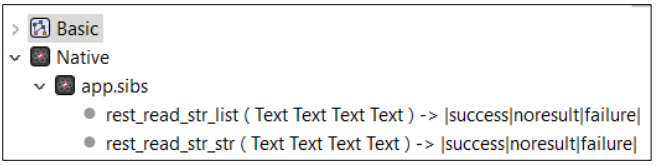


Fig. 2. SIBs explorer with the new Native SIBs

Once the declared SIB, its signatures and the attached Java function are validated by the platform, the SIB will be visible in the explorer as a Native SIB, with the other default SIBs as shown in fig. 2. At this point it is ready to be used, and available to the DIME users as a drag and drop item, ready to be inserted in any process model.

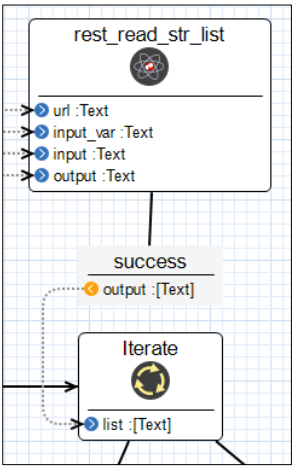


Fig. 3. The REST Read SIB in use: Visual representation in a model

Fig. 3 shows the visual representation of the newly developed SIB, as it appears when it is used in a process model. The required four inputs are being fed to this block using data flow (dotted) arrows. We see the three outgoing branches, labelled as defined. On success, the result will be conveyed as a string (or list of strings) to the successive SIB.

DIME automatically validates semantic and syntactic errors after the insertion and data connectivity of SIBs, ensuring this way the correctness of intended behaviour (automatic quality assurance [26] of models).

6.2 Cloud extension of Pyrus

We extend now the Pyrus is an online data analytics platform built using Pyro with Amazon Web Services (AWS), choosing the Amazon Translate service [27].

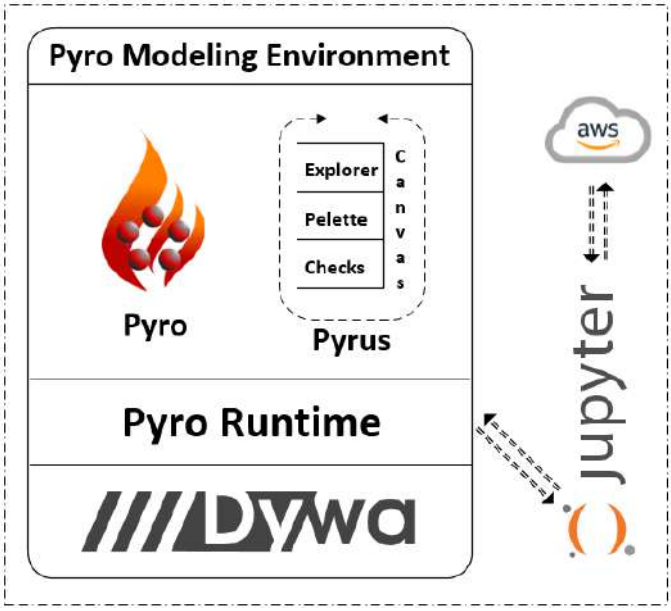


Fig. 4. The Pyrus/Pyro Architecture extended with AWS

Pyrus communicates with Jupyter hub at the backend. It uses the RESTful protocol to read function signatures and execute the attached python code. As shown in fig. 4, Jupyter and Pyrus communicates in asynchronously manner.

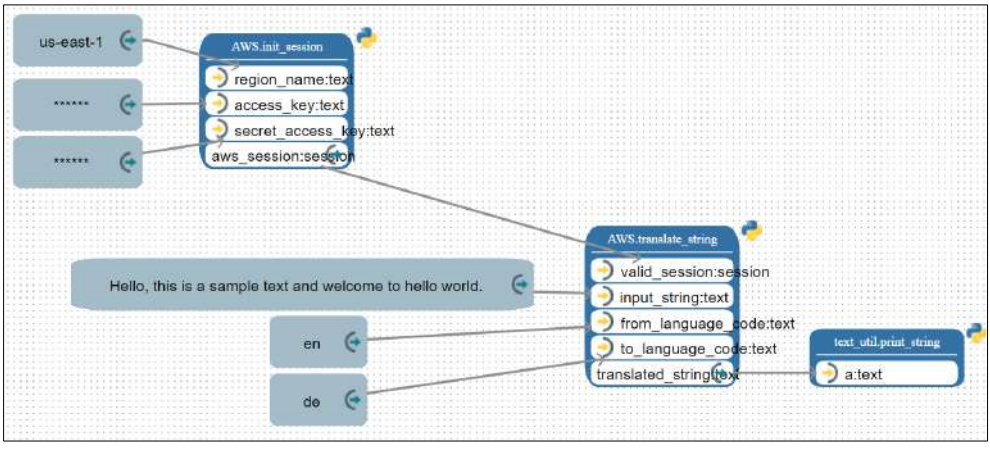


Fig. 5. Pyrus pipeline using AWS translate

The mechanism for the new AWS Translate block definition and implementation is similar to the DIME SIB declaration, but it only contains the signature, no outgoing branches. As Pyrus supports a dataflow modelling style, there are no control elements (the branches). The signature declaration starts with the # keyword, it is followed by meta data and the implementation of the functions in a python file. It has an extension «.py» and is located in the Jupyterhub space. Pyrus automatically reads these annotated signatures and shows them as drag-able blocks in its explorer.

Fig. 5 shows the working pipeline of *AWS_translate*: in reality we have defined 2 blocks, *AWS.init_session* and *AWS.translate_string*.

The initialization block must meet the preconditions of the external server in order to use its services. Communication with the AWS server/services requires a valid session, validated with credentials, i.e. access key, secret key, server information. The *AWS.init_session* initiates the communication transaction with the AWS server. It accepts the required inputs/tokens from connected grey blocks, which are constants.

Once successfully authenticated by AWS, a session token is provided for further communication with AWS. This token (output) is fed to the *AWS.translate_string* block along with the other required inputs: the text string to be translated and the code of the from and to languages.

Finally, the (translated text) result is passed to the next block, *text_util.print_string*, that prints it on screen.

The pipelines are automatically validated by the underlying modelling platform to check for connectivity errors of the blocks on the canvas.

6.2 Tool and Technologies

The tools and technologies used for these implementations and extensions are Eclipse, Java, JSON library, PHP, Python, DIME, Pyrus, Jupyter Hub and Amazon Web Services.

As the methodology is generic, it can be followed like a blueprint to implement communication and integrate a large variety of external services and platforms. The resulting drag and drop components enrich the DSL domain and expressive features of low code development in the mentioned platforms.

7. Conclusion and Discussion

We presented a generic extension mechanism to two low code development environments along a microservice philosophy. We showed it by integrating preexisting remote RESTful services and cloud-based enterprise system services as new drag and drop components in the respective DSLs. In DIME, an offline low-code IME, we used the native library mechanism, with signature declaration, linked Java backend code, and the code is merged with the logic layer at compile time. Pyrus, an online no code graphical data analytics tool, is linked with Jupiter Hub for functions discovery and code execution. To display new python functions as components in Pyrus, custom signatures are added to the python files defined in Jupyter hub, and the data flow pipeline of the service is modelled in the Pyrus frontend.

The simplicity and generality of the integration are an important feature of the chosen platforms. We envisage in fact a systematic integration of DSLs for various application domains stemming for our research collaborations. The simpler this is, the easier is the adoption of the approach across diverse application domains, research groups, and industrial partners. The (hand)code based extension approach of most popular low-code environments, that do not use formal models, nor generate «intelligent» modelling domains that have built-in checks for the model conformance are in fact inferior and sources again of complexity in the management of heterogeneity, code maintenance and evolution. The next application domains will be data visualization and data streaming platforms. We will support more AI/ML and data analytics functionality both in DIME and in Pyrus, adding

also cross-platform integration, in order to use the analytics capabilities of Pyrus pipelines in the DIME Digital Twin applications for Industry 4.0 as well as in the Digital Humanities [28].

References

[1] R. Waszkowski. Low-code platform for automating business processes in manufacturing. IFAC-PapersOnLine, vol. 52, issue 10, 2019, pp. 376-381.
[2] R. Sanchis, Ó. García-Perales et al. Low-code as enabler of digital transformation in manufacturing industry. Applied Sciences, vol. 10, no. 1, 2020, 17 p.
[3] S.J. Mellor, T. Clark, and T. Futagami. Model-driven development: guest editors' introduction. IEEE Software, vol. 20, no. 5, 2003, pp. 14-18.
[4] Intelligent process automation and the emergence of digital automation platforms. Available at: https://www.redhat.com/cms/managed-files/mi-451-research-intelligent-process-automation-analyst-paper-f11434-201802.pdf, accessed Feb, 2021
[5] S. Newman. Building microservices: designing fine-grained systems. O'Reilly Media, 2015, 280 p.
[6] S. Boßelmann, M. Frohme et al. Dime: A programming-less modeling environment for web applications. Lecture Notes in Computer Science, vol. 9953, 2016, pp. 809–832.
[7] P. Zweihoff, S. Naujokat, and B. Steffen. Pyro: Generating domain-specific collaborative online modeling environment. Lecture Notes in Computer Science, vol. 11424, 2019, pp. 101-115.
[8] Gartner forecasts. Available at: https://www.gartner.com/en/newsroom/press-releases/2021-02-15-gartner-forecasts-worldwide-low-code-development-technologies-market-to-grow-23-percent-in-2021, accessed Feb, 2021
[9] K. Ordoñez, J. Hilera, and S. Cueva. Model-driven development of accessible software: a systematic literature review. Universal Access in the Information Society, 2020, pp. 1-30.
[10] A.S. Maiya. ktrain: A low-code library for augmented machine learning. arXiv preprint arXiv:2004.10703, 2020, 9 p.
[11] G. Daniel, J. Cabot et al. Xatkit: A multimodal low-code chatbot development framework. IEEE Access, vol. 8, 2020, pp. 15332-15346.
[12] B. Steffen, T. Margaria et al. Model-driven development with the jabc. Lecture Notes in Computer Science, vol. 4383, 2007, pp. 92-108.
[13] T. Margaria and B. Steffen. eXtreme Model-Driven Development (XMDD) Technologies as a Hands-On Approach to Software Development Without Coding. In Tatnall A. (eds) Encyclopedia of Education and Information Technologies. Springer, 2020.
[14] B. Steffen, T. Margaria et al. The metaframe'95 environment. Lecture Notes in Computer Science, vol. 1102, 1996, pp. 450-453.
[15] F. Santos, I. Nunes, and A. L. Bazzan. Quantitatively assessing the benefits of model-driven development in agent-based modeling and simulation. Simulation Modelling Practice and Theory, vol. 104, 2020, article no. 102126.
[16] H. Moradi, B. Zamani, and K. Zamanifar. Caasset: A framework for model-driven development of context as a service. Future Generation Computer Systems, vol. 105, 2020, pp. 61-95.
[17] Cloud and desktop ide platform. Available at: https://theia-ide.org/, accessed Feb, 2021.
[18] S. Naujokat, M. Lybecait et al. Cinco: a simplicity-driven approach to full generation of domain-specific graphical modeling tools. International Journal on Software Tools for Technology Transfer, vol. 20, issue 3, pp. 1-28, 2018.
[19] L. Baresi and M. Garriga. Microservices: The evolution and extinction of web services? In Microservices: Science and Engineering, Springer, 2020, pp. 3-28.
[20] F. Rademacher, J. Sorgalla et al. Graphical and textual model-driven microservice development. In Microservices: Science and Engineering, Springer, 2020, pp. 147-179.
[21] T. Margaria and A. Schieweck. The digital thread in industry 4.0. Lecture Notes in Computer Science, vol. 11918, 2019, pp. 3–24.
[22] S. Jorges, C. Kubczak et al. Model driven design of reliable robot control programs using the jabc. In Proc. of the Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe'07), 2007, pp. 137–148.
[23] T. Margaria and B. Steffen. Business process modeling in the jabc: the one-thing approach. In Handbook of research on business process modeling. IGI Global, 2009, pp. 1-26.
[24] J. Neubauer, M. Frohme et al. Prototype driven development of web applications with dywa. Lecture Notes in Computer Science, vol. 8802, 2014, pp. 56-72.

[25] Angular dart open source packages. Available at: https://github.com/angulardart, accessed Feb, 2021.
[26] S. Windmüller, J. Neubauer et al. Active continuous quality control. In Proc. of the 16th International ACM SIGSOFT Symposium on Component-based Software Engineering, 2013, pp. 111-120.
[27] Amazon translate; fluent and accurate machine translation. Available at: https://aws.amazon.com/translate/, accessed Feb, 2021.
[28] Khan R., Schieweck A. et al. Historical Civil Registration Record Transcription Using an eXtreme Model Driven Approach. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 123-142.

Information about authors / Информация об авторах

Hafiz Ahmad Awais CHAUDHARY is a PhD student in University of Limerick and associated with Confirm – a SFI research centre in Smart Manufacturing, Ireland. His area of research includes «Interoperability and Data Integrations in the Digital Smart Manufacturing». Before moving to Ireland, he has been working as a Lecturer in University of Engineering and Technology, Lahore, Pakistan.

Хафиз Ахмад Аваис ЧАУДХАРИ – аспирант Лимерикского университета, сотрудничающий с Confirm – исследовательским центром Ирландского научного фонда в области интеллектуального производства. Его область исследований включает «Функциональная совместимость и интеграция данных в интеллектуальном цифровом производстве». До переезда в Ирландию он работал преподавателем в инженерно-технологическом университете в Лахоре, Пакистан.

Tiziana MARGARIA, PhD in Computer and Systems Engineering, Politecnico di Torino, Italy, Professor at Computer Science and Information System Department at the University of Limerick, Ireland. Research Interest: eXtreme Model Driven Design, lightweight formal methods, automatic program synthesis, system correctness, in particular compliance and security, future education in SE and IT.

Тициана МАРГАРИА, кандидат компьютерных наук и системной инженерии, Туринский политехнический университет, Италия, профессор кафедры компьютерных наук и информационных систем. Область научных интересов: проектирование на основе экстремальных моделей, упрощенные формальные методы, автоматический синтез программ, правильность системы, в частности соответствие и безопасность, будущее образование в области программной инженерии и информационной технологии.

DOI: 10.15514/ISPRAS-2021-33(4)-3

Идентификация прозрачных, сжатых и шифрованных данных в сетевом трафике

[1,2]*А.И. Гетьман, ORCID: 0000-0002-6562-9008 <thorin@ispras.ru>*
[1]*М.К. Иконникова, ORCID: 0000-0003-1530-5133 <mikonnikova@ispras.ru>*
[1]*Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2]*Национальный исследовательский университет «Высшая школа экономики», 101978, Россия, г. Москва, ул. Мясницкая, д. 20*

Аннотация. В статье рассматривается задача классификации сетевого трафика на три типа, в зависимости от представления данных в нём: прозрачный, сжатый и шифрованный. Описываются существующие методы классификации, служащие для разделения трафика на прозрачный и непрозрачный, сжатый и шифрованный применительно к сетевым данным и документам. На основе них выбираются методы, показавшие лучшие результаты, и производится отбор лучшей их комбинации и вывод единого результата с применением методов машинного обучения (случайный лес). Также исследуется вопрос классификации потоков как единого целого и предлагается новый, отличный от существующих способ. Завершается статья анализом направлений для дальнейших исследований.

Ключевые слова: анализ сетевого трафика; классификация сетевого трафика; машинное обучение**;** шифрованный трафик

Для цитирования: Гетьман А.И., Иконникова М.К. Идентификация прозрачных, сжатых и шифрованных данных в сетевом трафике. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 31-48. DOI: 10.15514/ISPRAS–2021–33(4)–3

Identification of transparent, compressed and encrypted data in network traffic

[1,2]*A.I. Getman, ORCID: 0000-0002-6562-9008 <thorin@ispras.ru>*
[1]*M.K. Ikonnikova, ORCID: 0000-0003-1530-5133 <mikonnikova@ispras.ru>*
[1]*Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2]*National Research University, Higher School of Economics, 20, Myasnitskaya Ulitsa, Moscow, 101978, Russia*

Abstract. The article is dedicated to the problem of classifying network traffic into three categories: transparent, compressed and opaque, preferably in real-time. It begins with the description of the areas where this problem needs to be solved, then proceeds to the existing solutions with their methods, advantages and limitations. As most of the current research is done either in the area of separating traffic into transparent and opaque or into compressed and encrypted, the need arises to combine a subset of existing methods to unite these two problems into one. As later the main mathematical ideas and suggestions that lie behind the ideas used in the research done by other scientists are described, the list of the best performing of them is composed to be combined together and used as the features for the random forest classificator, which will divide the provided traffic into three classes. The best performing of these features are used, the optimal tree parameters are chosen and, what's more, the initial three class classifier is divided into two sequential ones to save time needed for classifying in case of transparent packets. Then comes the proposition of the new method to classify

the whole network flow as one into one of those three classes, the validity of which is confirmed on several examples of the protocols most specific in this area (SSH, SSL). The article concludes with the directions in which this research is to be continued, mostly optimizing it for real-time classification and obtaining more samples of traffic suitable for experiments and demonstrations.

Keywords: network traffic analysis; network traffic classification; machine learning; encrypted traffic

For citation: Getman A.I., Ikonnikova M.K. Identification of transparent, compressed and encrypted data in network traffic. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021. pp. 31-48 (in Russian). DOI: 10.15514/ISPRAS–2021–33(4)–3

1. Введение

Данные, передаваемые в сетевых пакетах, неоднородны по своему представлению. Они могут содержать в себе тексты, написанные на естественном языке, передавать данные через строго структурированные, но всё же человекочитаемые протоколы, содержать документы различных форматов. Также, содержимое пакетов может быть сжато для уменьшения размера передаваемых данных и оптимизации использования каналов связи или зашифровано для обеспечения конфиденциальности.

Разные представления передаваемых данных требуют разных видов их обработки и анализа. Например, сигнатурный поиск может помочь определить протокол передачи сообщения и тип приложенных к сообщению документов, но сжатый трафик нужно предварительно разархивировать. Зашифрованный трафик не поддаётся такому способу анализа, поэтому его классификация и обработка требуют особых подходов.

Задача определения представления передаваемых данных имеет разнообразные практические приложения. Такие сведения могут использоваться в системах обнаружения атак. Наличие сжатого или зашифрованного трафика в ситуации, для которой обычно используется только прозрачный трафик, может свидетельствовать, например, о работе сервера команд ботнета [1].

Сведения о шифровании данных внутри пакетов могут применяться для более точного определения протоколов и типов содержимого, что, в свою очередь, служит для повышения качества обслуживания, выделения приоритетного трафика [2] и анализа устройства и эффективности сети в целом. Эти данные могут использоваться как сами по себе, так и в качестве только одного из признаков в глобальной системе принятий решений.

Также, в зависимости от состава передаваемых данных, могут применяться разные требования к их представлению для обеспечения конфиденциальности личных данных пользователей [3]. Так, медицинские данные не могут передаваться в открытом или сжатом виде - тогда их смогут прочитать злоумышленники.

Архиваторы обычно добавляют свои признаки и метки в файл, чтобы можно было легко понять его формат и алгоритм сжатия, но по сети документы передаются по частям, разделённым между множеством пакетов, и нельзя гарантировать, что система анализа сможет получить и извлечь нужные данные. На практике отличить фрагмент сжатых данных от шифрованных – нетривиальная задача, которой посвящено не одно исследование.

В целом, можно выделить три основных класса представления данных:

- нешифрованный (transparent),
- сжатый (compressed),
- шифрованный (encrypted).

В некоторых исследованиях не делается различий между вторым и третьим классом из-за схожести некоторых их свойств. Тогда они объединяются в единый класс непрозрачного (opaque) трафика.

Таким образом, если производить предварительную обработку трафика его разделением на указанные выше классы, это даст нам данные для решения некоторых из поставленных задач

или позволит к каждому из них применять уже целенаправленно свои методы анализа, что сократит время и трудоёмкость обработки получаемых данных (рис. 1).

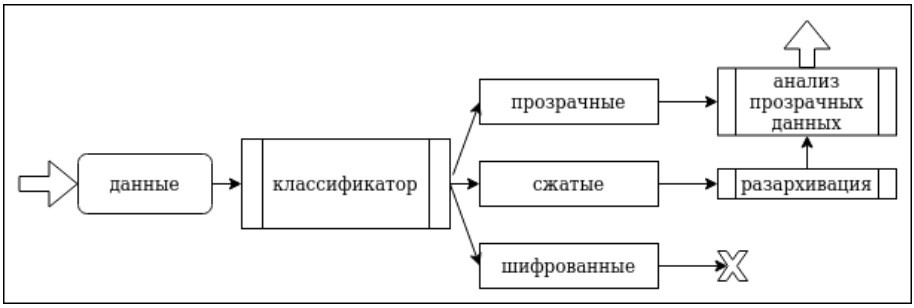


Рис.1 Схема классификации и обработки данных
Fig.1 Data classification and processing scheme

Такая классификация может проводиться как для отдельных пакетов, так и для потока пакетов в целом, где поток – это подмножество пакетов, определяемое пятёркой значений <IP адрес отправителя, IP адрес получателя, номер порта отправителя, номер порта получателя, транспортный протокол>. В зависимости от задачи может быть предпочтителен тот или иной способ.

Отдельные пакеты потока могут иметь разный тип. Например, для протокола HTTPS можно выделить следующие части потока.

- TCP рукопожатие: пакеты не содержат полезной нагрузки, поэтому формально не относятся ни к одному из классов.
- Обмен ключами: содержимое пакетов нешифрованное.
- Передача сообщений: зашифрованный трафик.

Таким образом, для классификации потока как единого целого нужно установить специальные правила соотношения между встречающимися в нём типами пакетов.

В зависимости от стоящей перед нами практической задачи анализ трафика может производиться как в онлайн-режиме, на лету, так и офлайн, постфактум. Это определяет, какое время мы можем тратить на классификацию отдельного пакета или потока, чтобы достичь компромисса между качеством полученного решения и скоростью его принятия (чтобы принимать достаточно хорошее решение с минимальной задержкой). Также, некоторые потоки могут продолжаться достаточно длительное время или вообще не заканчиваться в обозримом будущем (постоянная поддержка HTTP соединения через Keep-Alive), но их всё равно нужно уметь как-то классифицировать.

На основе всех описанных выше идей рассмотрим задачу классификации трафика по признаку его представления на три класса по потокам и в онлайн-режиме. В разд. 2 будут описаны существующие методы решения близких задач. В разд. 3 будут рассмотрены их особенности, преимущества и недостатки и выделены лучшие из них. Там же будет предложен основанный на изученном материале наш подход к решению данной задачи. В разд. 4 будут описаны эксперименты, проведённые для определения его характеристик. И, наконец, в разд. 5, будут сделаны выводы из полученных результатов.

2. Обзор существующих методов решения

К области классификации зашифрованных данных можно отнести работы, описывающие классификацию трафика, классификацию документов (файлов), а также работы о шифровании и генерации случайных значений.

В [4] предметом классификации являются потоки трафика, которые разделяются на шифрованные и нешифрованные на основе анализа первого значимого пакета (пакета,

содержащего полезную нагрузку). Для этого сначала определяется энтропия (более подробные объяснения применяемых методов будут даны в разд. 3) полезной нагрузки пакета (не менее 16 байтов), а затем проверяется доля печатных символов в первых 96 байтах. Такие ограничения были выбраны для ускорения процесса классификации. Для проверки полученных результатов использовался инструмент SPID [5], который сначала задавал истинные значения протоколов для каждого потока, а затем применялся для определения трафика, классифицированного системой как зашифрованный. Авторы указывают высокие результаты работы своей системы, но сами говорят о её неприменимости к некоторым популярным протоколам (SSH, SSL) из-за особенностей их устройства. Также, система не выделяет сжатые данные в отдельный класс.

В [6] исследователи используют свою систему чтобы отфильтровать трафик, который не может быть анализирован методами DPI. Тем самым они улучшают пропускную способность и качество работы фильтров Snort [7] за счёт предотвращения потерь значимых пакетов и отсутствия временных потерь на анализ зашифрованных пакетов, которые не могут быть анализированы таким способом. В качестве единицы классификации они выбирают пакет сетевого трафика, к которому применяются вероятностные тесты. Для тестирования использовались наборы разных типов файлов, переданные по протоколам поверх TCP на стенде, и выделенные из реальных сетевых трасс, собранных на кампусе, потоки протоколов SSL, SSH, SMTP и HTTP, причём внутри протокола HTTP было также проведено внутреннее деление пакетов на типы в соответствии с полями Content-Type и Content-Encoding. Из представленных результатов можно видеть, что предложенный авторами подход является довольно эффективным, однако, поскольку в нашем случае предпочтительной является классификация не пакетов, а потоков, он нуждается в улучшениях. Также желательно добавить отличия между шифрованным и сжатым трафиком и расширить множество изучаемых протоколов.

Авторы исследования [8] используют комбинацию трёх тестов (энтропия, критерий хи-квадрат, арифметическое среднее значений байтов), применяя их к полезной нагрузке k байтов пакетов (где k меньше размера пакета) и используя методы машинного обучения для определения пороговых значений. Эксперименты на трафике протоколов HTTP, FTP, Telnet, SSH показали лучшие результаты при использовании классификаторов CART [9] и NB [10], также CART показал наилучшее соотношение качества и скорости классификации при использовании только 32 байтов содержимого пакета. Использование сочетания разных тестов позволило разделять трафик на три типа, отделяя сжатый трафик от шифрованного. Также было предложено выбирать байты для анализа не из начала пакета, а случайным образом, что поможет лучше классифицировать разнородные пакеты.

Главной целью исследования [1] является определение блоков данных с высокой энтропией для поиска и профилировки протоколов обмена ключами, что позволит эффективнее искать ботов и потоки их команд в сети. Для этого используется сокращённая энтропия в скользящем окне, пороговые значения выбираются экспериментально. Эксперименты были проведены на выделенные tshark трассах протоколов TLS и некоторых других, отнесённых к прозрачным, а также на наборе данных трафика ботнета. Полученные результаты позволили авторам выдвинуть гипотезу, что созданный метод поможет выделять для дальнейшего исследования нестандартные случаи использования шифрования потоков, которые могут использоваться для передачи командной информации ботам в сети. Отделение шифрованного трафика от сжатого в работе не рассматривалось.

Хотя данное исследование [11] проводилось не на сетевом трафике, а на различных видах файлов (текст, аудио, видео и т.д.), оно интересно тем, что авторы ставят задачу научиться различать случаи сжатия и шифрования данных. Для этого они предлагают применять некоторые методы, которые обычно используются для тестирования генераторов случайных чисел. Из полученных результатов можно видеть, что этот метод успешно применим и что точность классификации растёт с увеличение размера анализируемых файлов (лучшая

получается при анализе 32 или 64 КБ), то есть по мере увеличения количества доступной для тестирования информации.

В [3] авторы исследуют подключённые к интернету вещей медицинские устройства и ищут исходящий от них прозрачный трафик, с целью проверки защиты конфиденциальности данных пользователей (и приходят к неутешительным выводам). Для этой цели они сравнивают использование энтропии, подхода, основанного на символах ASCII, и теста хи-квадрат, который и показывает в их работе лучшие результаты.

[12] является в некотором роде продолжением [3], здесь авторы прибегают к методам машинного обучения для классификации пакетов с высокой энтропией на шифрованные и сжатые (так как сжатие тоже не обеспечивает защиту данных пользователей). В качестве моделей машинного обучения они рассматривают различные виды искусственных нейронных сетей, из которых останавливаются на использовании для своих целей свёрточной нейронной сети, а в качестве признаков выбрана мера хи-квадрат теста, вычисленная на четверти рассматриваемого пакета. Их лучший результат составляет около 70%, что хуже, чем результаты, получаемые в аналогичных тестах другими методами.

В [13] рассматривается классификация потоков на прозрачные и непрозрачные с использованием как можно меньшего числа пакетов. Согласно исследованиям авторов, для этого достаточно найти N последовательных пакетов с высокой энтропией, а затем измерить совместную энтропию следующих M пакетов (то есть рассматривать пакеты как единое целое).

Здесь [14] была предпринята попытка определить тип содержимого (например, текст, изображение, сжатое изображение, зашифрованный текст и т.д.) применением классификатора SVM [15] к вектору энтропии и дополнительному признаку в виде частот последовательностей из 4 битов (для разделения шифрованного и сжатого трафика). Однако, основные эксперименты были проведены на разных видах файлах, а не на сетевом трафике, что оставляет открытым вопрос о полной применимости этого метода в сети.

В [16] описывается использование энтропии для отделения зашифрованных или упакованных PE файлов от обычных.

В [2] потоки VoIP классифицируются на шифрованные и сжатые с использованием набора тестов NIST (англ. The National Institute of Standards and Technology, Национальный институт стандартов и технологий США), разработанных для проверки генераторов псевдослучайных чисел [17]. В некоторых случаях (для некоторых кодеков) из-за особенностей их устройства нельзя выделить явные различия, поэтому предложено дополнительно удалять часть байтов из середины пакетов для искусственного добавления такого различия. Это предложение основано на основных свойствах шифрованного и сжатого трафика: шифрованный трафик должен быть равномерно случайным, в то время как сжатый трафик имеет определённую структуру, и удаление части байтов должно влиять на видимость этой структуры.

В [18] используется NIST для определения сжатых и зашифрованных документов.

В [19] (и расширенной версии статьи [20]) для классификации шифрованного и популярных форматов сжатого трафика используется нейронная сеть со значениями функции плотности вероятности поинтервального распределения значений байтов в фрагментах данных в качестве признаков. По результатам экспериментов на наборе файлов этот метод показывает высокие качество и скорость работы, но для хороших результатов минимальный размер анализируемого фрагмента ограничен снизу примерно 2 КБ.

Таким образом, из описания существующих исследований данной задачи можно видеть, что, хотя многие из подзадач отдельно уже были решены, нет единого решения задачи классификации потоков сетевого трафика на три типа (нешифрованный, сжатый, шифрованный) в онлайн режиме с достаточной скоростью и точностью и на основе небольшого количества данных, анализируемых в пакетах потока.

3. Подходы к решению задачи определения представления трафика

Условно, в решаемой задаче можно выделить следующие подзадачи.

1) Классификация пакетов на прозрачные (нешифрованные) или непрозрачные (сжатые или шифрованные).
2) Классификация непрозрачных пакетов на сжатые и шифрованные.
3) Классификация целого потока на основе входящих в него пакетов или их части.

Первые две подзадачи могут решаться одновременно. Опишем для каждой из подзадач существующие методы её решения и выберем методы, показавшие свою эффективность в существующих исследованиях и соответствующие нашей постановке задачи.

3.1 Классификация пакетов на прозрачные и непрозрачные

Среди методов и метрик, использующихся для решения этой задачи, можно выделить следующие.

3.1.1 Использование особенностей кодировки

В кодировке ASCII печатные символы – это значения 1 байта от 32 до 127. При равномерном распределении вероятностей появления каждого из символов в сообщении, свойственном шифрованному и, чуть в меньшей степени, сжатому трафику, их доля составила бы примерно 37.5%. Однако, для нешифрованного трафика их процент сильно увеличивается, поэтому в качестве порогового значения можно выбрать значительно более высокое значение (например, в [4] авторами было выбрано 75%).

Иногда, для облегчения процесса подсчёта первые 32 байта не выделяются в отдельную категорию и все байты меньше 128 считаются принадлежащими прозрачному (читаемому) трафику. Это может ускорить процесс вычислений, но оказать влияние на точность, например, при возникновении длинных цепочек нулевых символов, не свойственных прозрачному трафику.

3.1.2 Арифметическое среднее

Аналогично предыдущему варианту, можно считать не долю байтов с определёнными значениями, а среднее арифметическое значение всех байтов в последовательности [8]. Для равномерного распределения, это значение будет близко к 127.5, для прозрачных данных, соответственно, меньше.

3.1.3 Энтропия

Эта мера, введённая Шенноном [21] используется для численного выражения значения неопределённости, в частности характеризует непредсказуемость появления какого-либо символа в последовательности. Для структурированных данных и данных на естественном языке такая неопределённость ниже, в то время как сжатые и шифрованные данные по своей природе характеризуются высокой энтропией. Для шифрованного трафика это связано с необходимостью представить результат случайным на вид, чтобы исключить возможность простой расшифровки на основе статистического анализа текста. Для сжатого трафика это связано с максимальным удалением повторяющихся конструкций, чтобы избежать избыточности в данных [2]. Из этого видно, что энтропия может быть эффективна для классификации трафика на прозрачный и непрозрачный, но не для классификации непрозрачного трафика.

Для алфавита $\Sigma = \{0, 1, \dots, m-1\}$ и распределения $p = (p_i)_{i \in \Sigma}$:
$H(p) = -\sum_{i=0}^{m-1} p_i \log p_i$ (формула Шеннона для энтропии, где p_i – вероятности событий из пространства Σ).

Если ω - слово длины N в алфавите Σ, а n_i - количество букв i в слове, то можно вычислить частоту буквы i $f_i = \frac{n_i}{N}$ и энтропию слова как $\widetilde{H}_N^{\mathrm{MLE}}(w) = -\sum_{i=0}^{m-1} f_i \log f_i$, где MLE (Maximum Likelihood Estimator) – метод максимального правдоподобия.

Такое приближение будет сходиться к значению энтропии при $N \to \infty$. Соответственно, это приближённое вычисление энтропии даёт близкое к настоящему значение только при достаточной относительно размера алфавита длине последовательности. В частности, в [1] приводятся расчёты, что для 256 значений байтов потребовались бы примеры длиной около 2000 байтов, что превышает обычное значение MTU.

3.1.4 Сокращённая энтропия

В статьях [22] и [23] рассматриваются теоретические основы энтропии, строятся зависимости вычисленных методом максимального правдоподобия приближенных значений энтропии и реальных значений, описываются корректоры, применяемые для устранения подобных расхождений. Для большего удобства работы с ограниченными размерами сетевых пакетов вводится сокращённая (N-truncated) энтропия, которая определяется как среднее энтропии, посчитанной методом максимального правдоподобия, примеров среди всех слов длины N, выбранных случайным образом в соответствии с вероятностями символов заданного алфавита.

Например, для N=32 (что значительно меньше 256), это значение будет равно 4.87816 с доверительными интервалами $\pm$4×0.081156. Следовательно, для каждых фиксированных N и m (размер алфавита, в данной задаче m=256), можно вычислить такое значение по специальной формуле и сравнивать вычисляемое значение H^{MLE} уже с ним. Если получено значение ниже этой заданной границы, то фрагмент считается прозрачным. Чем больше N, тем меньше доверительный интервал.

3.1.5 Проверка отношений вероятностей

В статье [6] исследуются примеры статистических тестов для проверки гипотез, нулевой H_0 (о том, что распределение равномерно) и альтернативной H_1 (о том, что большинство байтов имеет значения меньше 128, процент байтов, значение которых должно быть меньше 128, параметризуется значением δ). В ходе экспериментов лучшие результаты дало использование последовательной проверки отношения вероятностей (sequential probability ratio test). Этот тест устроен следующим образом:

$\Lambda(\bar{X})$ - это отношение вероятности X при альтернативной гипотезе к вероятности X при нулевой гипотезе.

$\alpha = P(\text{accept } H_1 | H_0)$ – желаемая вероятность false negative результата.

$\beta = P(\text{accept } H_0 | H_1)$ – желаемая вероятность false positive результата.

$g_0(m) = \frac{\beta}{1-\alpha}$ и $g_1(m) = \frac{1-\beta}{\alpha}$ – пороговые значения.

Тогда процедура последовательной проверки отношения вероятностей выглядит следующим образом: на каждой итерации m принимается решение

$$\begin{cases} \text{accept } H_0 \text{ if } \Lambda_m(X_1, X_2, \dots, X_{\mathrm{m}}) \le g_0(m) \\ \text{accept } H_1 \text{ if } \Lambda_m(X_1, X_2, \dots, X_{\mathrm{m}}) \ge g_1(m) \\ \text{continue otherwise} \end{cases}$$

Для того, чтобы гарантировать сходимость метода, рассматривались два способа: ограничить длину анализируемой последовательности байтов или уменьшать пороговые значения в ходе анализа последовательности. Первый способ оказался предпочтительным. Если заданное число байтов проанализировано, а пороговое значение ни в одну из сторон не преодолено, выдаётся тот результат, к которому текущее значение ближе.

В реализации данного алгоритма SPRT использует правдоподобие количества байтов, меньших 128, по распределению Бернулли. Получая пакет на вход, алгоритм сначала пропускает заданное как параметр программы количество байтов $offset$, а затем в каждом шаге анализирует количество байтов $stepsize$, рассчитывая правдоподобность двух описанных выше гипотез по следующей формуле:
$L(\Theta_i|x_j) = C_n^k * \Theta_i^k * (1-\Theta_i)^{n-k}$, где:

- $n = stepsize$ – количество байтов, проверяемых за один шаг;
- k – количество байтов, меньших 128, в одном шаге;
- Θ_i – доля байтов, меньших 128, при биноминальном распределении, относящемся к гипотезе i;
- x – набор байтов, анализируемых на шаге j.

Затем рассчитывается относительное правдоподобие для двух гипотез

$$\Lambda_j = \frac{L(\Theta_1|x_j)}{L(\Theta_0|x_j)}$$

и сумма логарифмов относительных правдоподобий обновляется следующим образом:
$S_j = S_{j-1} + \ln \Lambda_j$ (при $S_0 = 0$)
Для g_0 и g_1 также берутся логарифмы. Если после анализа заданного как параметр $maxBytes$ количества байтов, решение не принято, алгоритм останавливается и принимает одну из гипотез на основании текущего значения S_j.

3.1.6 Критерий хи-квадрат

Критерий хи-квадрат сравнивает наблюдаемую частоту каждого символа f_i с ожидаемой в случае равномерного распределения e_i. Результат вычисляется по формуле

$$\chi^2 = \sum_{i=1}^{n} \frac{(f_i - e_i)^2}{e_i}$$

Чем сильнее реальные частоты символов отличаются от ожидаемых, тем выше значение χ^2. Таким образом, можно установить порог для разделения данных на прозрачные и непрозрачные. Это метод показал лучшие результаты из нескольких методов в [3].

3.2. Классификация непрозрачных пакетов

И сжатые, и шифрованные пакеты показывают более высокие значения энтропии, чем прозрачный трафик, благодаря чему их можно выделить в отдельный класс. Но при этом на первый взгляд оба эти типа представления выглядят как случайный набор значений, и их разделение на классы выглядит нетривиальной задачей. В качестве методов классификации пакетов на сжатые и шифрованные можно выделить следующее.

3.2.1 Использование тестов для генераторов случайных чисел

По определению, цели сжатия и шифрования данных различны. Шифрование нужно для обеспечения конфиденциальности, и хорошая криптосистема должна распределять данные по сообщению равномерно и не давать никаких закономерностей для статистического анализа, который мог бы позволить расшифровать данные третей стороне. Сжатие требуется для уменьшения количества битов, необходимых для передачи информации, путём устранения избыточности, и обычно сжатые данные наоборот имеют определённую структуру. В связи с этим можно считать распределение символов в шифрованных данных более близким к истинно случайным, в отличии от сжатых данных, и использовать для их

разделения тесты, предназначенные для определения качества работы программных генераторов случайных чисел. Одним из таких наборов тестов является NIST [17].

NIST – это пакет статистических тестов, разработанный Лабораторией информационных технологий Национального института стандартов и технологий. В его состав входят 15 тестов для определения меры случайности двоичных последовательностей, которые часто используются для проверки работы генераторов случайных чисел.

Разные тесты NIST имеют разную рекомендуемую длину входной последовательности, в связи с чем не все из них подходят для использования на сетевых пакетах с их ограниченными размерами. Кроме того, некоторые методы не так хороши для различения шифрованного и сжатого трафика согласно результатам экспериментов [11] или слишком вычислительно сложны для применения при классификации пакетов в онлайн режиме.

Из подходящих можно описать следующие тесты.

1) Частотный побитовый тест. Тест определяет, является ли количество нулей и единиц в двоичной последовательности приблизительно одинаковым. Для этого вычисляются:

- S_n – сумма цифр в примере (где 0 заменяется значением -1);
- статистика $s_{\text{obs}} = \frac{|S_n|}{\sqrt{n}}$, где n – длина примера;
- $p - \text{value} = \text{erfc}(\frac{s_{\text{obs}}}{\sqrt{2}})$.

  При $p - \text{value} < 0.01$ последовательность считается неслучайной. Рекомендуется брать минимум 100 битов для анализа.

2) Частотный блочный тест (частота единиц в блоках). Аналогичен предыдущему тесту, но для N блоков фиксированной длины M внутри последовательности. Вычисляются:

- $\pi_i = \frac{\sum_{j=1}^{M} \varepsilon_{(i-1)M+j}}{\text{M}}$ - пропорция единиц в каждом блоке длины M (ε - очередной бит);
- $\chi^2(\text{obs}) = 4\text{M}\sum_{\text{i}=1}^{N}(\pi_{\text{i}} - 1/2)^2$ - статистика хи-квадрат для наблюдений;
- $\text{p} - \text{value} = \text{igamc}(\frac{\text{N}}{2}, \frac{\chi^2(\text{obs})}{2})$.

  При $p - \text{value} < 0.01$ последовательность считается неслучайной. Рекомендуется брать минимум 100 битов для анализа, $M \geq 20$ и $N < 100$.

3) Тест на последовательность одинаковых битов. Исходно тест заключается в подсчёте числа рядов в исходной последовательности, где ряд представляет собой непрерывную подпоследовательность одинаковых битов. Целью данного теста является вывод о том, действительно ли количество рядов разных длин из 0 и 1 соответствует их количеству в случайной последовательности. Также, можно использовать упрощённый вариант теста, заключающийся просто в вычислении максимального непрерывного количества одинаковых символов подряд.

4) Тест кумулятивных сумм. Для произвольного обхода вычисляется кумулятивная сумма значений битов в подпоследовательности (где 0 заменяется значением -1). Целью является сравнение определяемых сумм с ожидаемым их поведением в абсолютно случайной последовательности.

Пакет считается шифрованным, если он успешно проходит все тесты, пороговые значения в которых были выбраны в [11] на экспериментальной основе.

3.2.2 Использование машинного обучения

Для классификации сжатых и шифрованных пакетов в некоторых работах используются различные методы машинного обучения. Они могут работать как с признаками самого пакета, так и помогать определять пороговые значения для комбинаций других описанных тестов. Для них требуется некоторое количество достоверно размеченных иными способами тренировочных данных.

- **SVM** (англ. Support Vector Machine, метод опорных векторов) [15] – метод машинного обучения, основанный на построении разделяющей гиперплоскости в пространстве признаков объектов. В работе [14] в качестве признаков для этого классификатора используется вектор энтропии всех возможных подпоследовательностей байтов и частоты различных 4-битовых символов. Видно, что даже для частей пакетов в 1024 байта этот способ является вычислительно затратным, а его показанная эффективность не превышает эффективность других методов, поэтому такой вариант классификации далее рассматриваться не будет.
- **CART** (англ. Classification And Regression Tree) – один из алгоритмов обучения дерева решений [9]. В [8] этот метод используется для автоматического определения пороговых значений для трёх статистических методов (энтропия, арифметическое среднее, хи-квадрат) и их объединения, показав лучшие результаты, чем другие из опробованных алгоритмов машинного обучения. Этот метод в исследовании показал хорошие качество и скорость работы. Также используется в [24].
- **Метод k-ближайших соседей** (англ. k-nearest neighbors algorithm, kNN) [25] присваивает объекту тот класс, который наиболее распространён среди k его ближайших по используемой метрике соседей, класс которых уже известен. Для этого метода нужен предварительный этап выделения признаков (в [12] это локальные значения по методу хи-квадрат), а в работе он показал не самые высокие результаты.
- **Нейронные сети.** Также есть исследования по применению искусственных сетей к решаемой задаче. Среди архитектур есть как сети прямого распространения ([12], [19]), так и свёрточные ([12]). В [12] авторам не удалось добиться высоких результатов классификации, хотя лучший из полученных (около 70%) всё-таки статистически превосходит классификацию случайным выбором). В [19] классификаторы, объединяющие признаки, получаемые тестами из NIST, тест хи-квадрат и сам фрагмент, показали хорошие результаты, превосходящие другие методы, при достаточном размере анализируемого фрагмента данных.

3.3 Классификация потоков

Для того, чтобы классифицировать не отдельные пакеты, а весь поток целиком, было предложено следующее.

1) Классифицировать весь поток по первому значимому пакету (не учитывая TCP рукопожатие) [4]. Такой подход работает далеко не во всех случаях. Например, шифрованные протоколы SSH и SSL начинаются с открытого обмена ключами, а протокол HTTP, даже если передаёт сжатые данные, начинается с прозрачных заголовков, поэтому анализа только первого пакета, а тем более его части может не хватить.
2) Склеивать данные полезной нагрузки пакетов и анализировать уже их [2]. Такой подход более перспективен с точки зрения результатов классификации, чем предыдущий, но требует хранения большого количества данных, особенно при одновременном анализе множества потоков. Также, необходимо ограничить количество анализируемых пакетов в потоке, чтобы обеспечить работу в режиме реального времени, и реализовать определение конца потока, если он наступит раньше этой границы.
3) Искать в потоке непрерывные последовательности непрозрачных пакетов [13]. Для разделения потоков на прозрачные и непрозрачные в этом исследовании предлагается найти N (выбрано N=3) последовательных пакетов с высокой энтропией среди 640 первых байтов, а затем измерить совместную энтропию следующих M пакетов. Такое решение позволяет хорошо выделять непрозрачные потоки, однако в случае прозрачных

потоков требуют излишних вычислений (нужно узнать энтропию всех пакетов до конца потока).

3.4. Наш подход

Ни один из описанных выше способов классификации потоков не подходит для нашей постановки задачи или не даёт достаточно хорошие результаты, поэтому мы предложим свой метод, исходя из следующих соображений:

- количество анализируемых пакетов в потоке должно быть ограничено для возможности классификации в онлайн режиме, но должно быть достаточным, чтобы дать представление о потоке и дойти до фазы собственно обмена информацией (после рукопожатия, приветствия, обмена ключами и т.п.);
- поток классифицируется на основе всех анализируемых пакетов, но хранимая до момента принятия решения информация об исследованных пакетах должна быть минимальна для оптимизации работы по памяти;
- все пакеты прозрачного потока прозрачны;
- в случае сжатого или шифрованного потока в начале его могут идти несколько прозрачных пакетов;
- некоторые потоки не содержат полезной нагрузки вообще - их имеет смысл считать прозрачными;
- точность классификации непрозрачных потоков неидеальна, поэтому принимать решение лучше на основе не одного непрозрачного пакета, а нескольких.

На основе приведённых выше решений подзадач, их преимуществ и недостатков относительно поставленных целей, нами было предложены и опробованы следующие тесты для классификации пакетов в различных их сочетаниях.

1) Использование особенностей кодировки (*freq*): вычисление процента байтов со значениями меньше 128 в полезной нагрузке пакета.
2) Энтропия (*entropy*): вычисление энтропии полезной нагрузки пакета.
3) Проверка отношений вероятностей (sprt): в качестве начальных значений выбираются $\alpha = 5, \beta = 5, \text{offset} = 32, \text{stepsize} = 32, \text{theta} = 85.$
4) Критерий хи-квадрат (*chi*): вычисление значения данного теста при условии равномерного распределения значений байтов в качестве нулевой гипотезы.
5) Кумулятивная сумма (адаптация теста из NIST) (*cusum*): вычислить максимальное отклонение байтов (количество таких байтов подряд) от 128 в одну или другую сторону.
6) Наибольшее количество одинаковых символов подряд (адаптация теста из NIST) (runs): в качестве результата возвращается такое значение; в отличие от NIST используются значения байтов, а не битов.
7) Побитовая частота (тест NIST) (*bit freq*).
8) Максимальное количество битовых единиц подряд (адаптация теста NIST) (*bit ones*): как результат теста возвращается такое значение.
9) Частота единиц в блоках (тест NIST) (*bit freq block*).

4. Эксперименты

Для проведения экспериментального исследования предложенного решения данной задачи был подготовлен набор данных, состоящих из потоков, выделенных из реальных и искусственных сетевых трасс.

4.1 Классификация пакетов

Для экспериментов по сравнению эффективности методов классификации пакетов на три класса использовались выделенные из сетевых трасс пакеты, по 30000 пакетов на класс (20000 для тренировочной выборки, по 5000 для валидационной и тестовой). Эти классы состояли из пакетов протоколов:

- прозрачный трафик: *pop, ftp, smtp, imap, http* (текстовые незашифрованные данные согласно значениям полей *content-type* и *content-encoding*).
- сжатый трафик: *http* (*content-encoding=gzip* или *content-type=png, gif, jpeg*).
- шифрованный трафик: *gquic, ssh, tls*.

Эти пакеты были распределены по указанным классам согласно свойствам протоколов с минимальной ручной валидацией.

Далее были построены три типа классификаторов:

- классификатор на все три класса;
- классификатор для разделения трафика на прозрачный и непрозрачный;
- классификатор для разделения непрозрачного трафика на шифрованный и сжатый.

Для построения классификаторов использовалась модель случайного леса (*RandomForestClassifier*) из библиотеки *scikit-learn*, для которой на валидационной выборке был проведён подбор оптимальный параметров максимальной глубины дерева и количества деревьев (табл. 1). Для каждой комбинации параметров в связи с элементами случайности в процессе обучения производилось несколько обучений модели для выбора лучшей.

Табл. 1. Выбранные для классификаторов на основе модели случайного леса лучшие комбинации параметров
Table 1. Best combinations of parameters chosen for Random Forest classifiers

Тип классификатора	макс. глубина	количество деревьев
1 классификатор	30	25
2 классификатор	25	25
3 классификатор	25	40

Табл. 2. Полученные для указанных параметров результаты (с использованием всех признаков)
Table 2. Results obtained for chosen parameters (using all features)

Тип классификатора	точность	полнота	F1 мера (макро)
1 классификатор	0.9594	0.9604	0.9593
2 классификатор	0.9996	0.9994	0.9995
3 классификатор	0.9395	0.9408	0.9395

Можно видеть (табл. 2), что на задаче разделения трафика на прозрачный и непрозрачный достигается очень хорошее качество, в то время как задача классификации непрозрачного трафика оказывается несколько сложнее (анализ ошибок показывает, что и для классификатора на три класса эта ситуация является основным источником ошибок).

Также, с помощью параметра, характеризующего важность каждого из признаков для процесса классификации, были получены относительные значения важности признаков при классификации согласно критерию Джини. Эти значения представлены в табл. 3, чем больше число, тем важнее признак.

Табл. 3. Важность признаков для классификации согласно критерию Джини
Table 3. Feature importance according to Gini criterion

	freq	***entropy***	***sprt***	***chi***	***cusum***	***runs***	***bit freq***	***bit ones***	***bit freq block***
1	0.1784	0.0711	0.0563	0.1571	0.2764	0.0026	0.0993	0.0640	0.0645
2	0.3244	0.0419	0.1208	0.0016	0.3828	0.0676	0.0145	0.1102	0.0012
3	0.0395	0.1747	0.0064	0.3093	0.0251	0.0329	0.2147	0.0640	0.0766

Для получения общего представления было измерено время работы каждой из функций получения одного из признаков на всех пакетах (Python). Результаты приведены в табл. 4.

Табл. 4. Время получения признака для всех примеров
Table 4. Time to compute the feature for all data samples

	freq	***entropy***	***sprt***	***chi***	***cusum***	***runs***	***bit freq***	***bit ones***	***bit freq block***
время (с)	8.84	10.19	1.15	21.22	9.67	15.86	31.73	128.18	49.95

Можно видеть, что далеко не все признаки одинаковы важны для классификации, кроме того их вычисление занимает разное время, для некоторых большое, относительно других, поэтому имеет смысл использовать только часть из этих признаков для классификации, определив самые важные в каждом из случаев.

Для отбора признаков были выбраны два метода из специализированного раздела библиотеки *scikit-learn*.

- *SequentialFeatureSelector* (последовательный выбор признаков) – формирует подмножество признаков заданного размера жадным способом, на каждом шаге добавляя лучший из оставшихся признаков на основе оценки, полученной кросс-валидацией.
- *RFE* (*Recursive Feature Eliminator* – рекурсивный элиминатор признаков) – используя получаемые обучаемой моделью веса функций, рекурсивно выбирает всё меньшие и меньшие подмножества признаков, пока не достигнет нужного размера подмножества.

Табл. 5. Лучшие признаки для каждого из классификаторов, полученные методами SFS и RFE
Table 5. Best features for each of the classifiers according to the SFS and RFE methods

метод	**классификатор**	**1 признак**	**2 признак**	**3 признак**	**4 признак**	**5 признак**
SFS	1	*freq*	*chi*	*entropy*	*bit_freq*	*bit_ones*
	2	*cusum*	*freq*	*entropy*	*runs*	*bit_ones*
	3	*chi*	*runs*	*sprt*	*entropy*	*bit_freq*
RFE	1	*chi*	*freq*	*cusum*	*entropy*	*bit_ones*
	2	*bit_ones*	*cusum*	*freq*	*sprt*	*entropy*
	3	*entropy*	*chi*	*bit_freq*	*bit_freq_block*	*bit_ones*

Табл. 6. Выбранные для классификаторов на основе модели случайного леса лучшие комбинации параметров (с использованием сокращённого числа параметров)
Table 6. Best combinations of parameters chosen for Random Forest classifiers (using only chosen features)

Тип классификатора	**макс. глубина**	**количество деревьев**
1 классификатор	30	40
2 классификатор	15	20
3 классификатор	30	40

На основе сопоставления результатов, полученных этими двумя методами, и изначальной оценки важности признаков, а также с учётом времени вычисления отдельных признаков,

можно определить самые важные признаки для каждого из классификаторов (табл. 5 и 6), сократив таким образом множество признаков:

1: *freq, chi, entropy, cusum, bit_ones*;
2: *freq, cusum, sprt, entropy, bit_ones*;
3: *bit_freq, chi, entropy.*

Табл. 7. Полученные для указанных параметров результаты (с использованием только выбранных признаков)
Table 7. Results obtained for chosen parameters (using only chosen features)

Тип классификатора	точность	полнота	F1 мера (макро)
1 классификатор	0.9324	0.9341	0.9322
2 классификатор	0.9993	0.9989	0.9991
3 классификатор	0.9147	0.9170	0.9146

Так как классификатор пакетов на прозрачные и непрозрачные показывает хорошее качество и требует меньше признаков, чем классификатор для всех трёх классов, имеет смысл рассмотреть схему классификации в два этапа (выделение непрозрачных пакетов и отдельная их классификация) и сравнить её по качеству и времени с одноэтапным классификатором (табл. 7).

Используя для одноэтапного классификатора четыре выбранных выше признаков, мы получаем следующую оценку по времени вычисления признаков: 49,91 с

Для двухэтапного классификатора:

- 4 признака: 29,85 с;
- + 2 признака для части пакетов: 52,95 * <доля пакетов> с, так как эти признаки нужно вычислять уже не для всех пакетов. В нашем случае, при условии, что треть пакетов прозрачная, получается значение 35,3 с => общее время 65,14 с.

Табл. 8. Результаты, полученные для двух типов классификаторов
Table 8. Results obtained for two types of classifiers

классификатор	точность	полнота	F1 мера (макро)
одноэтапный	0.9324	0.9341	0.9322
двухэтапный	0.9423	0.9438	0.9422

Табл. 9. Матрица ошибок для одноэтапного классификатора (сверху истинные значения, слева - полученные)
Table 9. Confusion matrix for one-step classifier

трафик	прозрачный	сжатый	шифрованный
прозрачный	14999	10	0
сжатый	1	14286	299
шифрованный	0	704	14701

Табл. 10. Матрица ошибок для двухэтапного классификатора
Table 10. Confusion matrix for two-step classifier

трафик	прозрачный	сжатый	шифрованный
прозрачный	14998	10	0
сжатый	2	14379	242
шифрованный	0	611	14758

Можно видеть, что оба классификатора показывают примерно одинаковые по качеству результаты, при этом, в зависимости от доли шифрованных пакетов, двухэтапный классификатор может работать быстрее, не производя вычислений некоторых признаков (табл. 8-10).

Также был проведён выбор оптимальных параметров для SPRT (единственный из используемых тестов с переменными параметрами) (табл. 11-13).

Табл. 11. Экспериментально полученные оптимальные значения для теста SPRT
Table 11. Experimentally obtained optimal SPRT test parameters

параметр	значение
отступ от начала (offset)	0
шаг анализа (step)	32
(альтернативная гипотеза) theta	85
(желаемый false negative) alpha	5
(желаемый false positive) beta	5

Табл. 12. Результаты при выборе оптимальных параметров теста SPRT
Table 12. Results for optimal SPRT test parameters

мера	точность	полнота	F1 мера (макро)
результаты	0.9	0.9011	0.8998

Табл. 13. Матрица ошибок при выборе оптимальных параметров теста SPRT
Table 13. Confusion matrix for optimal SPRT test parameters

трафик	прозрачный	сжатый	шифрованный
прозрачный	14997	3	0
сжатый	3	14907	54
шифрованный	0	90	14966

Хотелось бы произвести дальнейшее ускорение процесса принятия решения о пакете для возможности классификации в режиме онлайн. Для этого было предложено не рассматривать пакеты меньше минимального размера и рассматривать не весь пакет целиком, а только его N байтов. Как можно видеть из графика зависимости качества классификации от размера пакета (рис. 2), такая оптимизация возможна.

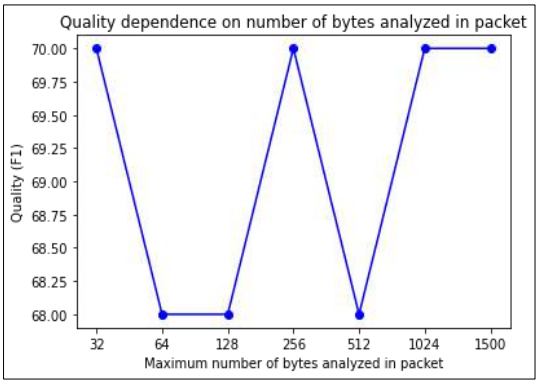


Рис.2. График зависимости качества классификации (в F1 мере) от максимального количества анализируемых байтов в пакете
Fig.2. Dependency of quality of classification (in F1) on maximum number of bytes analized in one packet

На основе экспериментов лучшим значением для ограничения сверху размера анализируемой части пакета стало 1024 байта. Это значение несколько варьировалось для разных типов трафика (прозрачного и непрозрачного) и было выбрано как компромиссное для этих ситуаций. Также, стало возможно ограничить размер анализируемых данных в тесте побитовой частоты (*bit_freq*), разделив его на 8, и сократить длину анализируемой части пакета в тестах проверки отношения вероятностей (*sprt*) и кумулятивных сумм (*cusum*) до 64 байтов без существенных потерь в качестве и с выигрышем по времени.

Для оценки производительности предлагаемые методы были реализованы в виде модулей расширения на языке C++ для разрабатываемой в ИСП РАН системы анализа сетевого трафика Конвеер. Таким образом, обученные в библиотеке *scikit-learn* модели были также переведены на C++ и загружены в систему. При описанных выше ограничениях получились следующие результаты (табл. 14) для файлов трасс, содержащих по 30000 пакетов (тех, на которых обучалась и тестировалась система на Python).

Табл. 14. Результаты и производительность модуля на C++ в составе системы Конвеер
Table 14. Results and performance of C++ module in Konveyer system

трасса	время (нс)*	скорость (Гб/с)	прозрачные	сжатые	шифрованные
transparent	1907	0.7	29952	48	0
compressed	5378	0.24	192	21962	7846
encrypted	5803	0.22	0	918	28868
*время - среднее время классификации одного пакета в нс					

4.2 Классификация потоков

Для классификации потоков предлагается следующая схема: классифицируются первые N пакетов потока, для них возможны варианты: прозрачный (сюда же относятся пакеты, не содержащие полезной нагрузки), сжатый, шифрованный или неопределённый (если задана нижняя граница размера пакета и он меньше неё). Далее для каждого потока рассматриваются N пакетов и применяются следующие правила в указанном порядке:

- если среди них есть шифрованный пакет, то весь поток классифицируется как шифрованный;
- если среди них есть сжатый пакет, то весь поток классифицируется как сжатый;
- если есть неопределённые пакеты, то весь поток классифицируется неопределённым, так как можно было пропустить какую-то важную информацию;
- иначе поток считается прозрачным.

Для некоторых протоколов были проведены эксперименты на выделенных из реального трафика примерах для определения минимального числа пакетов потока для анализа.

Табл. 15. Результаты для разного количества анализируемых пакетов в потоке
Table 15. Results for different number of packets analyzed in network flow

протокол	количество пакетов	всего потоков	прозрачные	сжатые	шифрованные
SSL	10	200	0	75	125
	15	199	0	15	184
	20	194	0	3	191
	25	194	0	2	192
SSH	10	200	0	295	5
	15	200	0	98	102
	20	200	0	1	199
	25	200	0	0	200
GQUIC	10	200	0	0	200
	25	200	0	0	200

Из табл. 15 можно видеть, что в соответствии с устройством протоколов SSL и SSH, где вначале идёт нешифрованная часть, наилучшим количеством пакетов для анализа в потоке можно считать 20: при нём протоколы уже доходят до шифрованной части, что позволяет их

правильно классифицировать, но количество минимально из возможных для более быстрой классификации.

Эксперименты на других протоколах менее показательны, так как чисто прозрачный трафик, например, гораздо сложнее выделить из реально передаваемых в сети данных. Однако, из имеющихся данных о протоколах, данного количества пакетов будет достаточно для классификации.

5. Выводы

Были изучены существующие подходы к классификации трафика на прозрачный, сжатый и шифрованный и выявлены их недостатки с точки зрения нашей постановки задачи. На основе существующих идей и методов, была предложена реализация, объединившая те из них, которые показали хорошие результаты в проведённых экспериментах.

Наша реализация способна работать с сетевым трафиком, классифицировать как пакеты, так и целые потоки, показывая при этом достаточно хорошие результаты по качеству и приемлемые - по времени. Эта реализация встроена как модуль в более широкую систему анализа трафика и может служить одним из этапов его обработки в ходе решения более сложных задач.

В дальнейшем планируется продолжить сбор и разметку примеров трафика, чтобы обучать и тестировать систему на большем количестве реальных примеров, с учётом их разнообразия и изменчивости. Желательно выделить больше примеров разных типов трафика для разных протоколов. Также, предполагается продолжить поиск оптимальных комбинаций методов и параметров для ускорения получения результатов в условиях работы в реальном времени.

Список литературы / References

[1]. Luo S., Seideman J.D., Dietrich S. Fingerprinting Cryptographic Protocols with Key Exchange using an Entropy Measure. In Proc. of the IEEE Security and Privacy Workshops (SPW), 2018, pp. 170-179.

[2]. Choudhury P., Kumar K.R.P. et al. An empirical approach towards characterization of encrypted and unencrypted VoIP traffic. Multimedia Tools and Applications, vol. 79, issue 1, 2020, pp. 603-631.

[3]. Wood D., Apthorpe N., Feamster N. Cleartext data transmissions in consumer iot medical devices. In Proc. of the 2017 Workshop on Internet of Things Security and Privacy, 2017, pp. 7-12.

[4]. Dorfinger P., Panholzer G., John W. Entropy estimation for real-time encrypted traffic identification. Lecture Notes in Computer Science, vol. 6613, 2011, pp. 164-171.

[5]. Hjelmvik E., John W. Breaking and improving protocol obfuscation. Chalmers University of Technology, Technical Report No. 2010-05, 2010, 34 p.

[6]. White A. M., Krishnan S. et al. Clear and Present Data: Opaque Traffic and its Security Implications for the Future. In Proc. of the 20th Annual Network & Distributed System Security Symposium, 2013, 16 p.

[7]. Roesch M. Snort: Lightweight intrusion detection for networks. In Proc. of the 13th USENIX Conference on System Administration (LISA '99), 1999, pp. 229–238.

[8]. Cha S., Kim H. Detecting encrypted traffic: a machine learning approach. Lecture Notes in Computer Science, vol. 10144, 2016, pp. 54-65.

[9]. Lewis R.J. An introduction to classification and regression tree (CART) analysis. In Proc. of the Annual Meeting of the Society for Academic Emergency Medicine in San Francisco, 2000, 15 p.

[10]. Rish I. An empirical study of the naive Bayes classifier. In Proc. of the Workshop on Empirical Methods in Artificial Intelligence, 2001, pp. 41-46.

[11]. Casino F., Choo K. K. R., Patsakis C. HEDGE: efficient traffic classification of encrypted and compressed packets. IEEE Transactions on Information Forensics and Security, vol. 14, issue 11, 2019, pp. 2916-2926.

[12]. Hahn D., Apthorpe N., Feamster N. Detecting compressed cleartext traffic from consumer internet of things devices. arXiv preprint arXiv:1805.02722, 2018.

[13]. Zhang H., Papadopoulos C. Early detection of high entropy traffic. In Proc. of the IEEE Conference on Communications and Network Security (CNS, 2015, pp. 104-112.

[14]. Wang, Y., Zhang, Z. et al. Using entropy to classify traffic more deeply. In Proc. of the IEEE Sixth International Conference on Networking, Architecture, and Storage, 2011, pp. 45-52.

[15]. Wang L. (ed.). Support vector machines: theory and applications. Springer Science & Business Media, 2005, 412 p.
[16]. Lyda R., Hamrock J. Using entropy analysis to find encrypted and packed malware. IEEE Security & Privacy, vol. 5, issue 2, 2007, pp. 40-45.
[17]. Rukhin A., Soto J. et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST Special Publication 800-22, 2001, 131 p.
[18]. Sturgill, M., & Simske, S. (2016). Mass Serialization Method for Document Encryption Policy Enforcement. In Proc. of the ACM Symposium on Document Engineering, 2016, pp. 193-196.
[19]. De Gaspari F., Hitaj D. et al. Encod: Distinguishing compressed and encrypted file fragments. Lecture Notes in Computer Science, vol. 12570, 2020, pp. 42-62.
[20]. De Gaspari F., Hitaj D. et al. Reliable Detection of Compressed and Encrypted Data. arXiv preprint arXiv:2103.17059, 2021.
[21]. Shannon C.E. A mathematical theory of communication. The Bell System Technical Journal, vol. 27, no. 3, 1948, pp. 379-423.
[22]. Goubault-Larrecq J., Olivain J. Detecting subverted cryptographic protocols by entropy checking. Research Report LSV-06-13, Laboratoire Spécification et Vérification, ENS Cachan, 2006.
[23]. Goubault-Larrecq J., Olivain J. On the efficiency of mathematics in intrusion detection: the NetEntropy case. In Proc. of the International Symposium on Foundations and Practice of Security, 2013, pp. 3-16.
[24]. Kozachok A. V. et al. Classification of pseudo-random sequences based on the random forest algorithm. In Proc. of the 2020 Ivannikov Memorial Workshop (IVMEM), 2020, pp. 55-58.
[25]. Zahid N., Abouelala O. et al. Fuzzy clustering based on K-nearest-neighbours rule. Fuzzy Sets and Systems, vol 120, issue 2, 2001, pp. 239-247.

Информация об авторах / Information about authors

Александр Игоревич ГЕТЬМАН – старший научный сотрудник, кандидат физико-математических наук. Сфера научных интересов: анализ бинарного кода, восстановление форматов данных, анализ и классификация сетевого трафика.

Aleksandr Igorevich GETMAN – senior researcher, PhD in physical and mathematical sciences. Research interests: binary code analysis, data format recovery, network traffic analysis and classification.

Мария Кирилловна ИКОННИКОВА – аспирант. Научные интересы: анализ сетевого трафика, машинное обучение.

Maria Kirillovna IKONNIKOVA – postgraduate student. Research interests: network traffic analysis, machine learning.

DOI: 10.15514/ISPRAS-2021-33(4)-4

Средства захвата и обработки высокоскоростного сетевого трафика

[1]*Д.В. Ларин, ORCID: 0000-0001-8686-8916 <larin.dv@ispras.ru>*
[2,3]*А.И. Гетьман, ORCID: 0000-0002-6562-9008 <thorin@ispras.ru>*

[1]*Московский физико-технический институт,*
141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9
[2]*Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[3]*Национальный исследовательский университет «Высшая школа экономики»,*
101978, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. В данной работе дается обзор научных исследований в области захвата и обработки высокоскоростного трафика, а также рассматриваются конкретные программно-аппаратные решения. В работе выделены основные направления развития технологий перехвата трафика и описано их взаимодействие между собой. На основе обзора выделены основные проблемы сетевого стека операционных систем и способы их решения. Рассматриваются реализованные в программно-аппаратных средствах алгоритмы и структуры, а также их архитектура. Для каждого из решений описывается процесс получения пакетов с сетевого интерфейса. Кроме того, проведено сравнение их производительности, общий анализ реализаций, а также приведены рекомендации по области применимости.

Ключевые слова: перехват высокоскоростного трафика; программная обработка трафика; Анализ производительности; libpcap; PF_RING; DPDK; Netmap; eBPF; XDP; AF_XDP

Для цитирования: Ларин Д.В., Гетьман А.И. Средства захвата и обработки высокоскоростного сетевого трафика. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 49-68. DOI: 10.15514/ISPRAS–2021–33(4)–4

High-speed network traffic capturing and processing tools

[1]*D.V. Larin, ORCID: 0000-0001-8686-8916 <larin.dv@ispras.ru>*
[2,3]*A.I. Getman, ORCID: 0000-0002-6562-9008 <thorin@ispras.ru>*

[1]*Moscow Institute of Physics and Technology,*
9, Institutskiy per., Dolgoprudniy, 141701, Russia
[2]*Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[3]*National Research University Higher School of Economics,*
20, Myasnitskaya Ulitsa, Moscow, 101978, Russia

Abstract. Network stacks currently implemented in operating systems can no longer cope with the packet rates offered by 10 Gbit Ethernet. Thus, frameworks were developed claiming to offer a faster alternative for this demand. These frameworks enable arbitrary packet processing systems to be built from commodity hardware handling a traffic rate of several 10 Gbit interfaces, entering a domain previously only available to custom-built hardware. In this paper, we survey various frameworks for high-performance packet IO and their interaction with a modular frameworks and specialized virtual network functions software for high-speed packet processing. We introduce a model to estimate and assess the performance of these packet processing

frameworks. Moreover, we analyze the performance of the most prominent frameworks based on representative measurements in packet capturing scenarios. Therefore, we provide a comparison between them and select the area of applicability.

Keywords: high-speed traffic capturing; software packet processing; performance measurements; libpcap; PF_RING; DPDK; Netmap; eBPF; XDP; AF_XDP

For citation: Larin D.V., Getman A.I., Hight-speed network traffic capturing and processing tools. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021. pp. 49-68 (in Russian). DOI: 10.15514/ISPRAS–2021–33(4)–4

1. Введение

С развитием сети Интернет и увеличением доступности операторы связи и интернет-провайдеры вынуждены расширять каналы передачи данных и применять более современные с технологической точки зрения решения для обработки и анализа сетевого трафика на скоростях 100 Гбит/с и более. Такие задачи, как балансировка и фильтрация трафика, обнаружение и предотвращение различных сетевых угроз, высокопроизводительные коммутация и маршрутизация потребовали разработки новых подходов, способных обеспечивать хорошую производительность на высоконагруженных сетях.

Для решения возникающих проблем сетевые операторы начали использовать специализированные аппаратные решения (например, FPGA, ASIC, SmartNIC, архитектура P4). Такие устройства настраиваются под нужды какой-либо конкретной задачи, требующей очень высокой производительности – например, захват сетевого трафика без потерь, разбор пакетов, коммутация трафика. Однако, эти решения зачастую страдают от недостаточной гибкости и масштабируемости, что становится проблемой в условиях современных высокомасштабируемых систем.

Альтернативой стало использование возможностей стандартного сетевого оборудования и программной обработки трафика. Перед аппаратными реализациями такой подход обладает рядом преимуществ, среди которых возможность адаптации аппаратуры для решения различных задач, снижение операционных и капитальных расходов при построении больших систем для мониторинга и анализа сети, допустимость использования широкого класса программных решений с открытым исходным кодом. В то же время, данные решения требуют разработки эффективных методов взаимодействия с сетью и ставят перед исследователями новые задачи по созданию гибких и масштабируемых систем для перехвата, анализа и обработки высокоскоростного трафика.

Дальнейшая статья устроена следующим образом. В разд. 2 описаны особенности программной обработки трафика. В разд. 3 приведен подробный обзор существующих программных решений. Разд. 4 посвящен анализу и сравнению производительности описанных решений. В разд. 5 подведены итоги работы.

2. Программная обработка трафика

Программные решения по обработке сетевого трафика можно разделить на три основных категории [1-3]:

1) низкоуровневые решения, обеспечивающие ввод/вывод трафика: DPDK [4], PF_RING [5], XDP [6], netmap [7], libpcap [8];
2) специализированные программные реализации виртуальных сетевых функций: Open virtualSwitch (Open vSwitch) [9], ESwitch [10], PacketShader [11], DPDKStat [12];
3) модульные фреймворки для создания сетевых функций: Click [13], FastClick [14], VPP (Vector Packet Processing) [15], BESS [3], Snabb NFV [16], PacketMill [17].

Данная статья фокусируется на исследовании решений первой категории, т.е. низкоуровневых решений по перехвату и передаче трафика. Они могут быть использованы

для всех типов приложений, которым необходим доступ к сетевому трафику. Их можно разделить на два типа в зависимости от используемой архитектуры:

Kernel-Bypass. Основная особенность данной архитектуры заключается в передаче трафика от сетевой карты в пространство пользователя “в обход” ядра ОС, т.е. вся обработка пакетов производится не в сетевом стеке операционной системы, а в пользовательском пространстве. По этому принципу функционирует, например, прием и обработка пакетов с помощью libpcap. К частному случаю архитектуры Kernel-Bypass можно отнести схемы по перехвату трафика “без копирований” или “0-copy”. Последнее стало возможным, благодаря “эффективному” использованию механизма DMA (Direct Memory Access) – контроллер DMA записывает пришедшие на сетевую карту пакеты в предварительно выделенную область памяти ядра, которая, в свою очередь, отображена на область памяти в пространстве пользователя. Таким образом, пользовательские приложения могут напрямую обращаться к содержимому пакетов без необходимости его копирования. Поддержкой DMA снабжаются все современные сетевые карты, а на его основе сделаны такие решения, как PF_RING Zero-Copy (ZC) и DPDK (Data Plane Development Kit)

In-Kernel FastPath. Особенность данной архитектуры заключается в запуске первичной обработки трафика в пространстве ядра ОС до того, как пакет передается в сетевой стек ОС. Исторически программирование ядра было сложной задачей из-за необходимости написания кода на низкоуровневых языках программирования и строгих требований к безопасности кода – ошибка в программе могла привести к отказу ОС. С приходом технологий eBPF (extended Berkeley Packet Filter) и XDP (eXpress Data Path) исследователям удалось добиться загрузки и выполнения пользовательских программ, написанных на высокоуровневых языках программирования, в ядре ОС с сохранением всех требований по безопасности кода. Архитектура подразумевает использование возможностей ядра ОС для работы с входящим трафиком, без необходимости передавать трафик в сетевой стек или в пространство пользователя. Если последнее все же необходимо, то для этих целей используется сокет AF_XDP.

2.1 Receive Side Scaling (RSS)

С увеличением количества входящих сетевых пакетов в случае обработки в однопоточном режиме процессор может не справляться с обработкой всего поступающего трафика. Для решения этой проблемы можно воспользоваться преимуществами многопоточной обработки, одной из составляющих которой является механизм Receive Side Scaling (RSS).

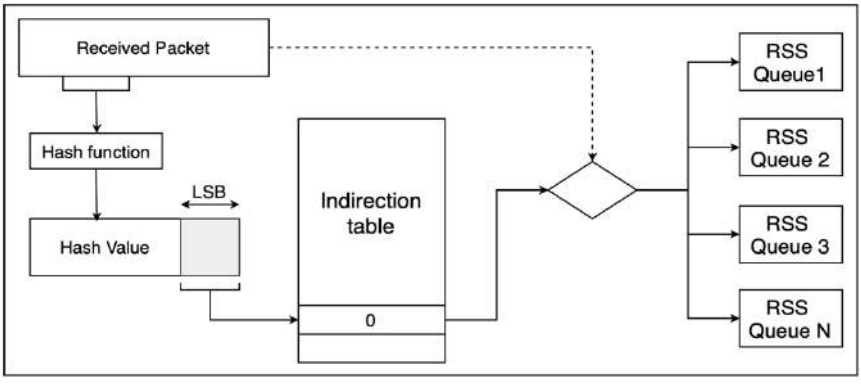


Рис. 1. Механизм работы RSS [18]
Fig. 1. RSS architecture [18]

RSS позволяет распределять полученные сетевой картой пакеты по нескольким входным очередям, каждой из которых операционная система назначает свое ядро процессора. Таким образом, нагрузка по обработке входящего трафика распределяется на несколько ядер

многоядерной системы, что позволяет избежать проблем, связанных с обработкой на одном процессорном ядре, и оптимизировать использование кэша. Говоря более детально, RSS распределяет трафик по входным очередям при помощи значений, полученных после применения хэш-функции к нескольким полям входящих пакетов, и таблицы косвенной адресации (indirection table).

Как изображено на рис. 1, наименее значащие биты (Least Significant Bits, LSB) хэша используются в качестве ключей для доступа к соответствующим позициям в таблице косвенной адресации. Такая таблица содержит в себе значения, служащие для распределения полученных данных на обработку конкретным ядром процессора. Для вычисления хэшей используется хэш-функция Тёплица (Toeplitz), на вход которой подается массив данных (т.н. 5-tuple или 5-ка полей пакета [19]), состоящий из IPv4/IPv6 адресов отправителя/получателя, TCP/UDP портов отправителя/получателя и опциональных расширенных заголовков IPv6, и секретный 40-байтовый ключ - как правило, битовая маска. Стандартный ключ распределяет трафик по очередям, поддерживая однонаправленную когерентность на уровне потока - пакеты, которые содержат одинаковые адреса и порты отправителя/получателя, будут направлены на обработку в одну и ту же очередь [18, 20].

2.2 Сетевой стек операционной системы

Современная сетевая аппаратура стремительно развивается для работы с высокоскоростным трафиком, однако программные решения зачастую не следуют этому тренду. Большинство существующих операционных систем предоставляют универсальный сетевой стек с простым пользовательским интерфейсом на основе сокетов для получения/отправки данных и поддержки большого числа сетевых протоколов и аппаратуры, в приоритете которого стоит совместимость, а не производительность.

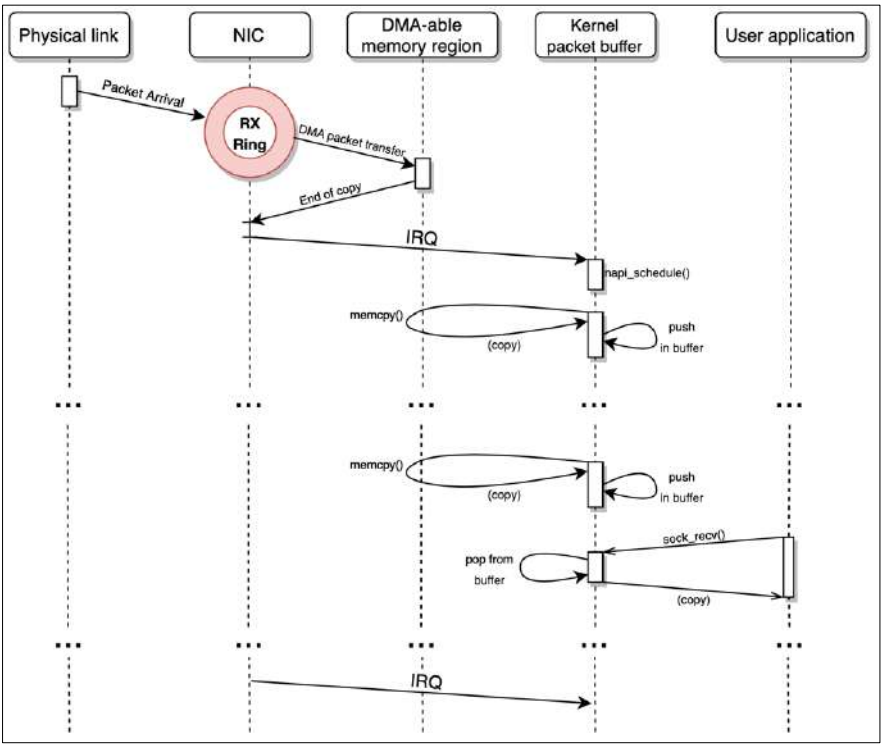


Рис. 2. Сетевой стек GNU/Linux, механизм NAPI [20]
Fig. 2. Linux NAPI RX scheme [20]

В данной статье рассматриваются существующие решения для GNU/Linux как для самого широко используемого семейства операционных систем с открытым исходным кодом, предоставляющего множество средств для анализа производительности системы. Для каждого решения будет отдельно указано, поддерживается ли оно на других ОС.

Большинство современных высокоскоростных драйверов, начиная с версии 2.6.0 ядра Linux (2003 г.), используют механизм NAPI для захвата пакетов с сетевой карты. NAPI позволяет существенно снизить количество прерываний по сравнению с предыдущими схемами. Более детально происходит это следующим образом: при получении первого прерывания драйвером запускается ответственная за прерывания NAPI функция, но, в отличие от традиционного подхода, она не приступает к копированию и помещению пакета в очередь, а планирует (shedule) выполнение функции poll() и отключает аналогичные прерывания в дальнейшем. Данная функция проверяет наличие новых пакетов, и, в случае их готовности, копирует и помещает в очередь сетевого стека GNU/Linux, без ожидания прерывания. В дальнейшем функция poll() планирует свой следующий запуск, продолжая до тех пор, пока на сетевую карту приходят пакеты. Алгоритм работы изображен на рис. 2.

Кроме того, если система не справляется с обработкой высокоскоростного трафика, механизм NAPI позволяет отбрасывать пакеты на уровне сетевого адаптера – т.е. до их прихода на уровень ядра. Стоит отметить, что механизм NAPI сильнее нагружает CPU при обработке медленного трафика по сравнению с традиционными механизмами. С увеличением скорости входящего трафика нагрузка снижается [20].

2.3 Проблемы сетевого стека

Среди проблем сетевого стека GNU/Linux и, в частности, механизмов NAPI и RSS, в соответствии с [20], можно выделить следующие.

2.3.1 Попакетное выделение и освобожение памяти

Каждый раз, когда на сетевую карту приходит новый пакет, для хранения информации о теле и заголовке пакета в памяти выделяется пакетный дескриптор. После завершения работы с пакетом этот дескриптор освобождается. Такой процесс работы с памятью приводит к значительным временным издержкам, что серьезно влияет на производительность при работе с высокоскоростным трафиком – 14.88 миллионов пакетов в секунду (Million packets per second, Mpps) для 10GbE. Кроме того, структура sk_buff, используемая для хранения всей информации о пакетах, обладает значительным размером, так как содержит множество информации о различных протоколах на нескольких уровнях. Как описано в [21], обращение и выделение sk_buff использует около 1200 тактов CPU на каждый пакет, а освобождение порядка 1100 тактов. А на операции по взаимодействию с sk_buff приходится порядка 63% нагрузки CPU при обработке пакетов Ethernet минимального размера, составляющего 64 байта [22].

Для решения этой проблемы предлагается использовать заранее выделенные участки памяти и переиспользовать их в дальнейшем. По этому принципу работают кольцевые буферы. Перед началом обработки, для нужного количества кольцевых буферов выделяется память, в которую записывается информация о пакетах. После обработки пакета, память не освобождается, а переиспользуется снова для новых пакетов. Таким образом, удается избежать множественных временных задержек. К минусам подхода можно отнести временные затраты на выделение памяти перед началом обработки и ее постоянную занятость во время обработки.

2.3.2 Сериализуемость доступа к трафику

Современные сетевые карты поддерживают получение пакетов с использованием аппаратных Receive Side Scaling (RSS) очередей. Они позволяют разделять входящий трафик

на несколько потоков, за счет применения хэш-функции, например, к 5-ке полей пакета, и параллелизовать процесс захвата пакетов, путем отображения каждой входной очереди на собственное ядро процессора (см. подраздел 2.1). Проблемы начинают появляться несколькими уровнями выше, когда сетевой стек GNU/Linux начинает совмещение пакетов из всех очередей в одну на сетевом и транспортных уровнях для их последующего анализа. В связи с этим возникает две проблемы. Во-первых, весь трафик собирается в одной точке, что ведет к снижению производительности. Во-вторых, пользовательский процесс не может получать данные из какой-то конкретной RSS очереди.

Чтобы решить эти проблемы необходимо организовать прямой доступ к RSS очередям для пользовательских процессов. Максимальная производительность данной архитектуры достигается, когда один процесс выполняет сразу две задачи - прием пакетов из RSS очереди и их дальнейшую передачу в пространство пользователя. Кроме того, такой подход увеличивает масштабируемость системы - можно добавлять новые потоки обработки по мере того, как нарастает число задействованных ядер и RSS очередей. К минусам данного подхода можно отнести следующее.

1) Необходимость использовать сразу несколько ядер, которые можно использовать под другие задачи.
2) Использование хэш-функции для распределения пакетов по очередям. В том случае, если задача подразумевает работу со связанными пакетами, потоками или сессиями, использование хэш-функции для распределения трафика по входным очередям может привести к тому, что связанный трафик будет обрабатываться на разных очередях [23]. Эта проблема актуальна, например, для систем по мониторингу Voice over IP (VoIP), а именно для потоков SIP и RTP трафика, которые могут не разделять заголовки сетевого и транспортного уровней.

2.3.3 Множественные копирования пакетов на пути от сетевой карты до пространства пользователя

Пакеты, проходящие через сетевую подсистему Linux, несколько раз копируются до их получения пользовательским приложением. Как минимум дважды – один раз копирование происходит во время переноса пакета из DMA-области памяти в буфер на уровне ядра, второй раз из буфера в пространстве ядра в буфер пользователя. К примеру, на одно копирование требуется от 500 до 2000 тактов процессора в зависимости от размера пакета [21]. Вдобавок к этому, множественные копирования небольших по размеру пакетов приводят к значительной потере производительности.

Для решения этой проблемы предлагается использовать отображение DMA-участков памяти ОЗУ напрямую в пространство пользователя. Такой подход позволяет свести количество копирований данных пакетов к нулю (zero-copy), однако возможен только при поддержке со стороны сетевой карты. В качестве альтернативы, которая поддерживается на большем числе сетевых карт, возможно использование отображения буфера на уровне ядра в пространство пользователя, что является реализацией схемы с одним копированием (1-copy).

3.3.4 Переключения контекста между ядром и пространством пользователя

Пользовательское приложение должно осуществлять системный вызов для получения каждого пакета. Каждый такой вызов влечет за собой переключение контекста, между пространством пользователя и пространством ядра и наоборот. Это ведет к возрастанию использования ресурсов CPU. Каждый системный вызов и переключение контекста могут потребовать до 1000 тактов процессора на обработку одного пакета [21].

Решается данная проблема путем получения пакетов группами (batch-processing). Решение подразумевает сбор нескольких пакетов в буфер и копирование полученной группы в память пространства ядра или пользователя. С помощью данной методики удается уменьшить

количество системных вызовов и соответствующих им прерываний, а также снизить количество копирований. В соответствии с архитектурой NAPI, если трафик перехватывается при помощи механизма опроса, то можно запросить получение не одного пакета, а сразу нескольких. Если трафик принимается с помощью механизма прерываний, то можно использовать дополнительный буфер-посредник для хранения полученных пакетов, пока они не будут запрошены приложением. Основная проблема получения пакетов группами заключается в увеличении задержек и джиттере, а также в неточности выставления временных меток на полученные пакеты, так как пакетам необходимо ждать заполнения группы или истечения таймера [24].

2.3.5 Неэффективное использование локализации памяти

Первичное обращение к DMA-области памяти вызывает принудительные промахи кэша (compulsory cache-misses), происходящие из-за инвалидации линий кэша процессора для консистентности памяти. Такие промахи кэша составляют 13.8% от общего числа тактов процессора, затраченных на получение одного 64-байтного пакета [22]. Кроме того, в системах, основанных на использовании NUMA (Non-Uniform Memory Access), задержки при обращении к памяти зависят от того, к какому узлу NUMA происходит обращение. Таким образом, неэффективное использование локализации памяти приводит к снижению производительности.

Архитектура NUMA подразумевает разделение доступной в системе памяти между различными процессорами, назначая каждому из них по своей области. Комбинация из процессора и области памяти называется NUMA-узлом. Для увеличения производительности и использования локализации памяти, процесс должен работать в области памяти, назначенной тому же процессору, на котором он исполняется. Такая техника называется закрепление памяти (memory affinity). Кроме того, существуют привязка к процессору (CPU affinity) и привязка прерываний (interrupt affinity). Первое подразумевает закрепление процесса или потока за каким-то конкретным процессором. Второе отвечает за привязку обработки прерываний к различным процессорам или ядрам (smp_affinity). Важность обработки прерываний и потоков входных данных на одном ядре заключается в повышении эффективности обращений к данным кэша и распределении нагрузки по ядрам системы. Как только потоку требуется получить доступ к полученным данным, он быстрее сможет найти их в локальном кэше процессора, если до этого эти данные были получены обработчиком прерываний, назначенным на то же ядро. Все необходимые настройки можно произвести при помощи утилиты *numactl*.

Помимо описанного, необходимо также обратить внимание на привязку потоков входных данных к узлам NUMA, назначенным на ту же шину PCIe, к которой подключена сетевая карта. Стоит упомянуть и средства аппаратной предвыборки (prefetching) данных следующего пакета и его дескриптора во время обработки текущего пакета. Задача предвыборки заключается в загрузке данных в кэш-память из оперативной памяти до того, как они потребуются.

3. Обзор программных решений

3.1 libpcap

libpcap – библиотека, предоставляющая высокоуровневый интерфейс для систем по захвату трафика, а также поддерживающая возможности чтения пакетов в неразборчивом режиме (promiscuous mode) и их записи в файл [25]. С использованием libpcap разработаны такие приложения, как Wireshark, tcpdump, snort [26-28] и многие другие. Данная библиотека не предназначена для перехвата высокоскоростного трафика, поскольку в своей основе она использует драйвер уровня ядра, а именно Berkeley Packet Filter [8, 19]. Таким образом, она

переносит пакеты через сетевой стек Linux, производя их копирование на уровне ядра в структуру sk_buff, которая используется для хранения всей информации о пакетах. Такая схема сильно влияет на итоговую производительность по сравнению с реализованными без копирований архитектурами. Процесс захвата пакетов с использованием библиотеки libpcap принципиально не отличается от описанной ранее (см. рис. 2) схемы по устройству сетевого стека GNU Linux.

К преимуществам libpcap можно отнести поддержку всеми популярными операционными системами и сетевыми картами. Использование libpcap оправданно, если для решения необходима не производительность, а совместимость с оборудованием.

3.2 PF_RING

PF_RING — это фреймворк для захвата сетевого трафика на различных сетевых картах [5] (см. рис. 3). Существует несколько реализаций, которые применяются в зависимости от аппаратных возможностей и драйверов сетевых карт. А именно – не требующий специальной аппаратной поддержки Vanilla PF_RING и, наоборот, требующий такой поддержки PF_RING Zero-Copy (ZC). Помимо этого, вне зависимости от режима, с PF_RING можно взаимодействовать при помощи стандартного libpcap API, а также использовать собственный API, предоставляющий возможности написания BPF фильтров, приема пакетов из нескольких входных очередей, закрепления программ за выбранными ядрами и другой функционал, позволяющий применять PF_RING для решения различных задач.

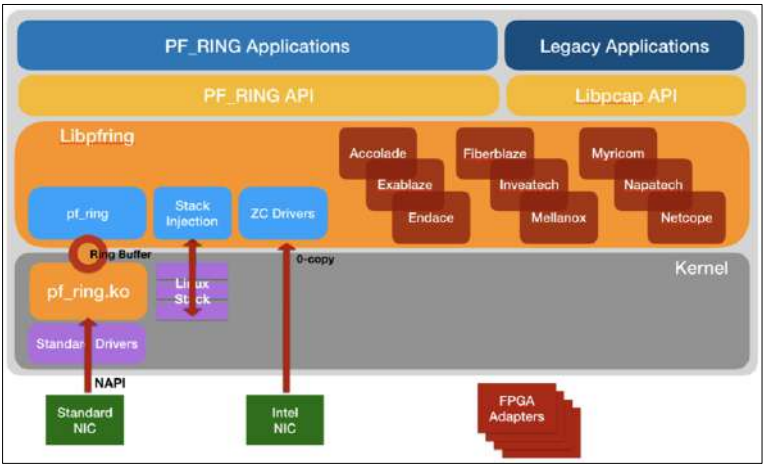


Рис. 3. Общая схема PF_RING [5]
Fig. 3. PF_RING architecture [5]

3.2.1 Vanilla PF_RING

Vanilla PF_RING получает трафик с сетевой карты при помощи NAPI, копируя пришедшие на сетевую карту пакеты в кольцевой буфер, из которого пользовательское приложение может считать пакеты. При необходимости, поддерживается использование сразу нескольких кольцевых буферов. Такая схема позволяет избежать издержек, возникающих в классических механизмах 2-copy, тем не менее в ней все еще присутствует одно копирование. Проблемы данного подхода начинают проявляться, когда скорость входящего трафика возрастает и кольцевые буферы начинают быстро переполняться.

Vanilla PF_RING основан на использовании кольцевого буфера в пространстве ядра, в который копируются все входящие пакеты (см. рис. 4) [29]. Данный буфер выделяется при создании сокета PF_RING и освобождается при его удалении и может быть прикреплен к сетевому адаптеру при помощи системного вызова *bind()*. Как только сетевая карта получает

пакет, драйвер переносит его в пространство ядра при помощи механизма DMA. В случае использования сокета PF_RING каждый входящий пакет копируется и помещается в кольцевой буфер. В случае переполнения буфера пакет отбрасывается. Кольцевой буфер экспортируется в пространство пользователя при помощи *mmap()*.

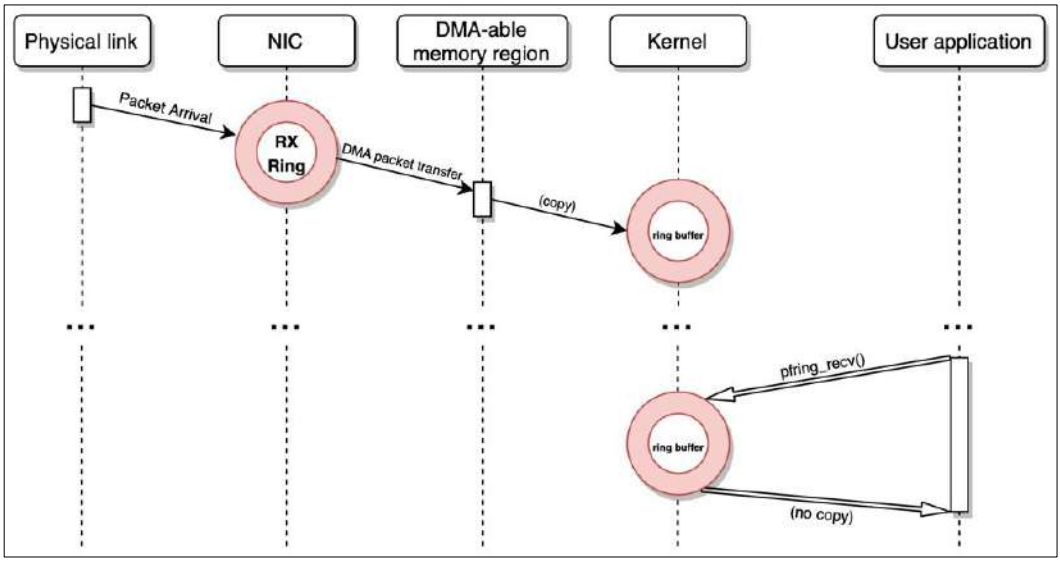


Рис. 4. Процесс получения пакета Vanilla PF_RING [20]
Fig. 4. Vanilla PF_RING Rx process [20]

Когда пользовательскому приложению необходимо получить доступ к сетевому трафику, оно вызывает *mmap()* для получения указателя на кольцевой буфер. Ядро копирует пакеты в кольцевой буфер и перемещает указатель на запись, пользовательское приложение производит аналогичную операцию для указателя на чтение. При этом новые пакеты перезаписывают уже прочитанные пользовательским приложением, то есть не происходит затратных с точки зрения потребления тактов процессора операций по выделению и освобождению памяти. К преимуществам данного подхода можно отнести следующее:

1) входящий трафик не отправляется в структуры сетевого стека ядра;
2) *mmap()* позволяет пользовательскому приложению получать доступ к трафику без необходимости осуществлять его копирование;
3) несколько приложений могут создавать собственные кольцевые буферы для чтения из нескольких входных RSS очередей.

Данный механизм осуществляет одно копирование на уровне ядра, в связи с чем пропускная способность модуля чтения с использованием Vanilla PF_RING заведомо ниже, чем с использованием механизма без копирований.

3.2.2 PF_RING Zero-Copy

PF_RING Zero-Copy основан на использовании модифицированных драйверов сетевых карт, которые позволяют осуществлять передачу пакетов пользовательским приложениям без копирований (рис. 5). PF_RING ZC отображает память пользовательского пространства в область памяти DMA, в связи с чем отпадает необходимость в использовании промежуточных буферов в пространстве ядра. Механизм предоставляет возможность чтения пакетов из нескольких входных очередей, что вкупе с отсутствием копирований дает значительный прирост производительности.

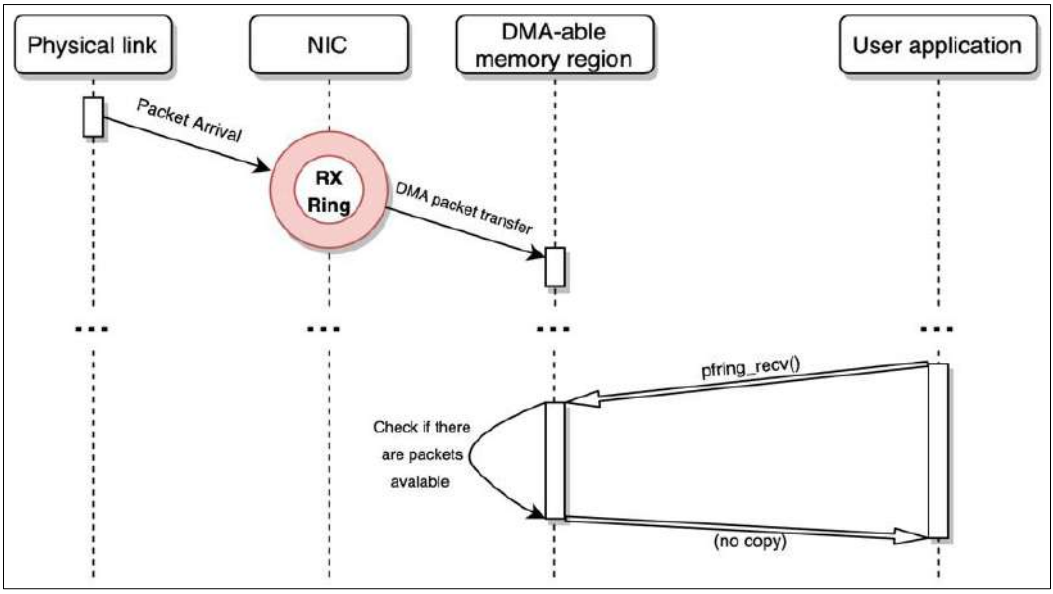


Рис. 5. Процесс получения пакета PF_RING ZC [20]
Fig. 5. PF_RING ZC Rx process [20]

Однако, данный подход обладает и очевидным недостатком – из-за необходимости использовать модифицированные драйверы для сетевых карт, количество оборудования на котором поддерживается PF_RING ZC ограничено (на момент написания работы поддерживаются только сетевые карты Intel).

Для всех перечисленных решений PF_RING предоставляет libpcap-like API, что сильно упрощает разработку приложений с использованием данного фреймворка. Кроме того, в PF_RING реализована возможность использовать XDP и сокет AF_XDP для захвата трафика. Интерфейс PF_RING можно использовать совместно с такими приложениями, как Snort, Suricata и Zeek (более известный как Bro).

3.3 DPDK

Data Plane Development Kit (DPDK) – это фреймворк, предоставляющий набор библиотек и драйверов для быстрой обработки трафика в пространстве пользователя, используя архитектуру Kernel Bypass (см. рис. 6). DPDK полностью замещает сетевой стек Linux - при подключении к сетевой карте все взаимодействие с ней будет происходить через компоненты DPDK и никакие другие приложения не смогут получить к ней доступ.

Стоит отметить, что сам по себе DPDK сетевым стеком не является – разбор пакетов необходимо реализовывать самостоятельно с использованием предоставленных библиотек. Совокупность библиотек DPDK образует EAL (Environment Abstraction Layer), который скрывает различия в аппаратной и программной частях систем и предоставляет API для взаимодействия с системой из пространства пользователя. Для перемещения сетевых пакетов DPDK использует буферы памяти (mbuf). Каждый mbuf состоит из трех секций: (i) структуры данных rte_mbuf, содержащей метаинформацию о пакете (например, номер VLAN или RSS очереди, ссылку на следующий пакет и тд), (ii) фиксированных по размеру областей памяти (headroom/tailroom) для добавления дополнительной информации и (iii) сегмента памяти для хранения всего пакета [4]. Каждая структура rte_mbuf использует только 2 блока кэша (cache line) для минимизации занимаемой памяти. Помимо этого, DPDK предоставляет драйверы для сетевых карт - т.н. Poll Mode Driver (PMD), которые позволяют приложениям напрямую взаимодействовать с сетевыми картами.

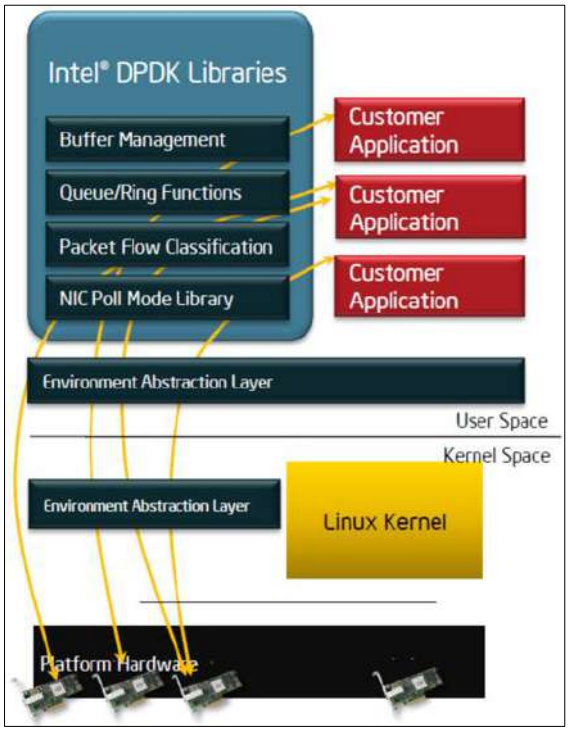


Рис. 6. Общая схема DPDK [30]
Fig. 6. DPDK architecture [30]

PMD использует mbuf для получения и передачи пакетов во время работы, а выделение памяти для них происходит на стадии инициализации приложения. Для захвата пакетов, PMD передает указатель на mbuf и его дескрипторы, зависящие от драйвера, сетевой карте, что позволяет ей при помощи DMA записывать полученные пакеты и их метаинформацию в необходимые разделы памяти. Затем, PMD при помощи опроса обнаруживает завершение операции DMA и копирует необходимую информацию из дескрипторов драйвера в структуру rte_mbuf. Все буферы памяти хранятся в объекте rte_mempool из которого при помощи кольцевых очередей (rte_ring) пользовательские приложения могут получать доступ к данным принятых пакетов [17, 30]. Схема взаимодействия DPDK с приложениями пользователя принципиально не отличается от аналогичной для PF_RING ZC, изображенной на рис. 6.

Помимо описанного, DPDK предоставляет и другие возможности для получения и обработки трафика. К ним можно отнести захват трафика при помощи сокета AF_XDP, библиотеку с реализацией алгоритма сопоставления максимального префикса (Longest Prefix Match, LPM) для IPv4 и IPv6 адресов, библиотеку таймеров для асинхронного выполнения функций, библиотеку хэширования и многие другие. Однако он обладает значительного размера документацией, которая сильно усложняет работу с данным фреймворком.

Ресурсы DPDK для захвата и передачи пакетов используют такие фреймворки, как VPP, PacketMill, FastClick и BESS, а также возможно его использование с Open vSwitch и для работы с виртуальными машинами (SR-IOV и VMDq режимы драйвера).

3.4 XDP/eBPF и сокет AF_XDP

eBPF (extended Berkeley Packet Fiter) и XDP (eXpress Data Path) необходимо рассматривать в совокупности из-за их тесной связи между собой (см. рис. 7). XDP – это фреймворк, который

выделяет ограниченную среду выполнения внутри виртуальной машины eBPF. Эта среда позволяет запускать различные программы непосредственно в контексте ядра, еще до того, как само ядро получит доступ к входным данным. Это дает возможность обработки трафика (в том числе и его перенаправление) на самой ранней из возможных стадий – сразу после получения пакетов от сетевой карты. Программа XDP запускается драйвером сетевой карты каждый раз, когда в систему приходит новый пакет. Запуск происходит внутри виртуальной машины eBPF, что позволяет добавить необходимую предобработку. Кроме того, eBPF предоставляет возможность использовать функции помощники ядра (kernel helpers) для доступа к структурам ядра и различным системным вызовам, а также разделяемую память - eBPF maps, которую можно применять как для хранения состояния между вызовами, так и для связи с другими eBPF программами и пространством пользователя (рис. 7). Помимо перечисленного, eBPF использует Verifier - верификатор пользовательских программ, который проверяет код на безопасность (например, на отсутствие бесконечных циклов, что гарантирует завершение программы) перед его загрузкой в пространство ядра [6].

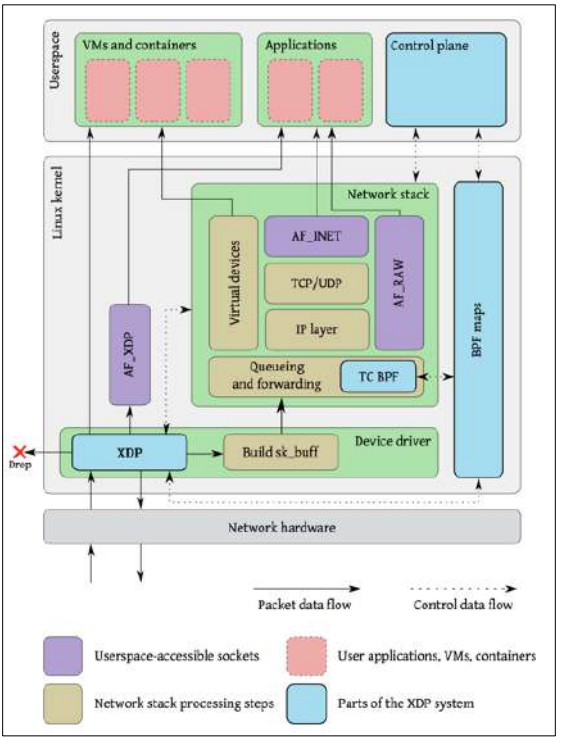


Рис. 7. Общая схема XDP/eBPF [6]
Fig. 7. XDP/eBPF architecture [6]

Действия, которые производит программа XDP можно разделить на отбрасывание пакета (XDP_DROP), его пропускание в сетевой стек Linux (XDP_PASS) или передачу пакета в тот же (XDP_TX) или в другой сетевой интерфейс (XDP_REDIRECT), как указано на рис. 8. Так как программа XDP исполняется на уровне драйвера, то выполнять задачи она может достаточно быстро. Но она ограничена в размере и может выполнять только ограниченное число инструкций, поэтому, в случае необходимости проведения более сложных операций, пакет может быть передан в сетевой стек Linux или в пространство пользователя, используя множество существующих сокетов, таких как AF_PACKET, стандартный для TCP/IP сокет AF_INET или AF_XDP. Рассмотрим работу последнего более подробно. AF_XDP – это тип

сокета, который позволяет передавать данные пакетов напрямую с сетевой карты в пространство пользователя используя XDP и избегая копирований. AF_XDP работает в трех режимах, начиная от самого медленного до самого быстрого:

1) skb режим, работающий на любой сетевой карте;
2) XDP-copy режим, который требует поддержки драйвером сетевой карты;
3) Режим без копирований, который работает только на драйверах с поддержкой XDP и расширением для zero-copy.

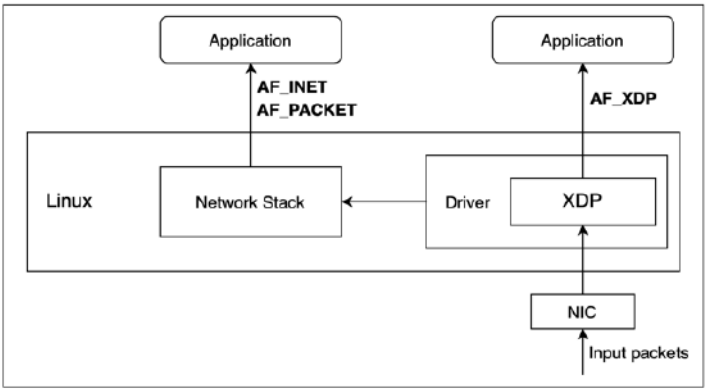


Рис. 8. Поток трафика с использованием XDP [31]
Fig. 8. XDP traffic flow [31]

Далее мы будем рассматривать только архитектуру zero-copy как самую производительную. С точки зрения приложения все пакеты находятся в выделенной области памяти, называющейся UMEM, как изображено на рис. 9. Эта область состоит из сегментов одного размера - пакетных буферов в которых находится информация о пакетах. Вместе с UMEM создаются два кольца: кольцо заполнения (fill ring) и кольцо завершения (completion ring). Кольцо заполнения используется для передачи управления пакетным буфером от пользовательского пространства в ядро. Кольцо завершения, наоборот, сигнализирует о передаче управления пакетным буфером от ядра в пространство пользователя. Приложение уведомляет о передаче управления записывая относительный адрес пакета в кольцо заполнения. Аналогичным образом поступает ядро, используя кольцо завершения. При помощи данных очередей мы можем передавать управление пакетными буферами без необходимости получать и отправлять данные. Для получения и отправки пакетов используются еще два кольца: кольцо получения (Rx ring) и кольцо отправки (Tx ring).

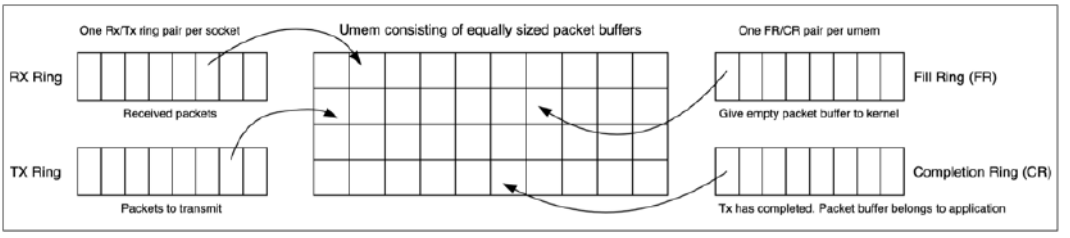


Рис. 9. 4 кольца AF_XDP и структура UMEM для хранения данных о пакетах [31]
Fig. 9. Four AF_XDP rings and UMEM structure [31]

При получении пакета ядро записывает дескриптор пакета в кольцо получения, указывая, что пакетный буфер содержит данные пакета и задает его относительный адрес и длину. Проверяя кольцо получения, приложение определяет, что пакет получен и считывает его.

Аналогичным образом приложение поступает при отправке пакета, используя очередь отправки. Стоит отметить, что очереди отправки и получения создаются для одного сокета,

и каждый сокет привязывается к одной структуре UMEM, которая имеет лишь одну пару колец заполнения и завершения. Однако, к структуре UMEM можно привязать несколько сокетов, в этом случае буферов получения и отправки будет несколько.

Процесс получения пакета можно описать следующим образом (см. рис. 10). Пакет приходит на сетевую карту и забирается драйвером, выполняющим программу XDP, которая принимает решение об отправке пакета в какой-то конкретный сокет AF_XDP. Так как сокет AF_XDP работает в режиме zero-copy, то сетевая карта уже записала данные пакета в пакетный буфер структуры UMEM, поэтому все, что остается сделать ядру — это записать в кольцо получения дескриптор пакета, чтобы уведомить приложение о нахождении пакета в памяти. Затем приложение проверяет кольцо получения на наличие пакетов. Как только оно заканчивает обработку пакета, дескриптор пакета возвращается в ядро при помощи кольца заполнения, таким образом новый пакет может быть записан в тот же пакетный буфер.

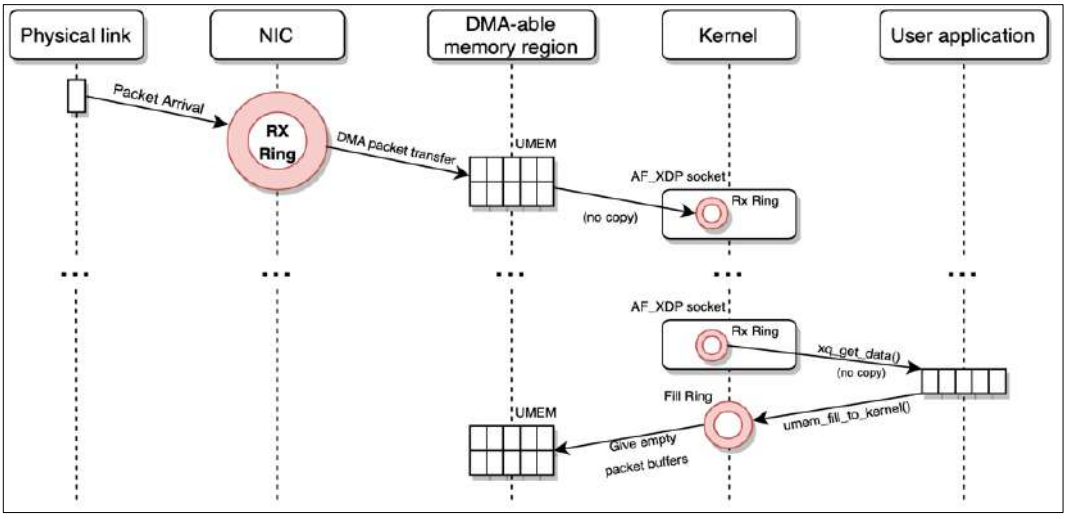


Рис. 10. Процесс получения пакета AF_XDP
Fig. 10. AF_XDP Rx process

Интерфейс AF_XDP для захвата и передачи пакетов может быть использован с такими фреймворками, как VPP, FastClick и BESS, а также с Open vSwitch.

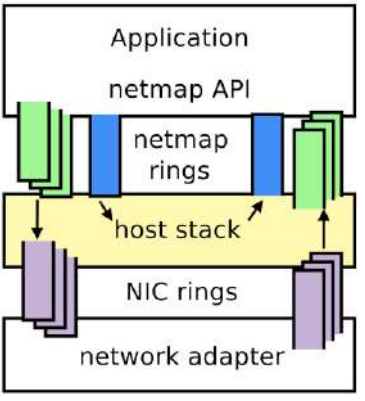


Рис. 11. Общая схема Netmap [7]
Fig. 11. Netmap architecture [7]

3.5 Netmap

Netmap – это фреймворк для работы с высокоскоростным сетевым трафиком на стандартном сетевом оборудовании (рис. 11). В соответствии с [7] решение использует предвыделение памяти на стадии инициализации, буферы фиксированного размера (2048 байт), обработку пакетов группами, а также предоставляет возможность параллельной обработки трафика. Кроме того, netmap использует отображение памяти для обеспечения пользовательских приложений прямым доступом к пакетным буферам с простым и оптимизированным представлением метаданных - кольцом памяти netmap (netmap memory ring). Оно содержит информацию о размере кольца памяти, указатель на текущую позицию буфера, количество полученных буфером пакетов или число свободных позиций в нем в зависимости от работы в режиме приема или передачи трафика соответственно, флаги о текущем статусе, отступ памяти пакетного буфера и массив с метаинформацией. Также на каждый пакет выделена память для хранения размера пакета, его индекса в пакетном буфере и некоторых флагов.

Кольца памяти netmap создаются по одному на каждую входную RSS очередь, что позволяет обрабатывать трафик в многопоточном режиме. Процесс захвата пакета изображен на рис. 12.

Аналогично XDP, интерфейс Netmap для захвата и передачи пакетов может быть использован с VPP, FastClick, BESS и Open vSwitch.

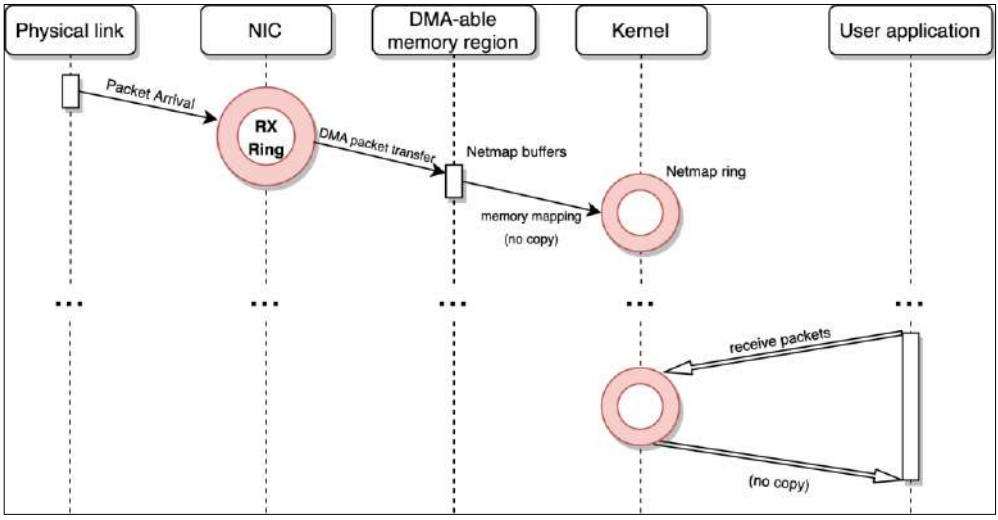


Рис. 12. Процесс получения пакета Netmap
Fig. 12. Netmap Rx process

4. Анализ программных решений

Далее приводится сравнительный анализ описанных решений: Vanilla PF_RING, PF_RING Zero-Copy, DPDK, Netmap и XDP/eBPF с AF_XDP.

Табл. 1. Сравнение решений (D - драйвер, K - ядро, K-U - интерфейс ядро/пользователь)
Table 1. Comparison of solutions (D - driver, K - kernel, K-U - kernel / user interface)

Характеристики / решения	Vanilla PF_RING	PF_RING ZC	DPDK	XDP/eBPF AF_XDP	Netmap
Бесплатность	✓	X	✓	✓	✓

Open-source	✓	✓	✓	✓	✓
Переиспользование и предвыделение памяти	✓	✓	✓	✓	✓
Поддержка RSS и исполнения в несколько потоков	✓	✓	✓	✓	✓
Отображение памяти	✓	✓	✓	✓	✓
Zero-Copy	X	✓	✓	✓	✓
Обработка пакетов группами	✓	✓	✓	✓	✓
Закрепление CPU и прерываний	✓	✓	✓	✓	✓
Закрепление памяти	✓	✓	✓	✓	X
Обработка на уровне ядра ОС	X	X	X	✓	X
Обработка в пространстве пользователя	✓	✓	✓	✓	✓
Уровни модификаций	D, K, K-U	D, K, K-U	D, K-U	D, K, K-U	D, K, K-U
API	libpcap-like	libpcap-like	Custom	Custom	Standard libc
Архитектура	Kernel-Bypass	Kernel-Bypass	Kernel-Bypass	In-Kernel Fastpath	Kernel-Bypass
Поддерживаемые ОС	GNU/Linux	GNU/Linux	GNU/Linux, Windows	GNU/Linux, Windows	GNU/Linux, FreeBSD
Поддерживаемые сетевые карты	Любые*	Intel	Любые*	Любые*	Intel, Mellanox, Nvidia, Realtek
• необходима поддержка драйверами сетевых карт					

4.1 Анализ производительности

Данный подраздел содержит результаты тестирования Vanilla PF_RING, PF_RING Zero-Copy, XDP/eBPF с AF_XDP и библиотеки libpcap на тестовом стенде, построенном на базе двух серверов с ОС Ubuntu 18.04 (128 Gb RAM, процессор: AMD Ryzen Threadripper 3960X с тактовой частотой 3.8 GHz и 24 физическими ядрами) на каждом из которых установлены

сетевые карты Intel i40e XL710 с двумя портами 40G QSFP+. Сервера соединены напрямую, один из них выступает в роли генератора трафика, на втором запущен анализатор трафика и происходит перехват.

В работе использовался программный генератор трафика PF_RING (pfsend), позволяющий в режиме zero-copy отправлять трафик на 40 Gb сетевых картах на скорости от 30 Гбит/сек для пакетов минимального размера (64 байта), достигающей 40 Гбит/сек для пакетов, размер которых превышает 100 байт. Отправка происходит в несколько потоков с использованием RSS очередей. Для достижения максимальной производительности в большинстве случаев достаточно одновременно запустить 3 генератора. Дальнейшее увеличение числа генераторов к улучшению производительности не приводит. Кроме того, генератор PF_RING позволяет балансировать трафик для передачи его в разные RSS очереди на приемной стороне. Реализовано это за счет выставления и изменения адресов и портов отправителя в сгенерированных пакетах.

На рис. 13 и 14 приведены результаты тестирования PF_RING (Vanilla и Zero-Copy), XDP/eBPF с AF_XDP и библиотеки libpcap на стенде с 40 Gb сетевыми картами. На графиках изображена скорость вхождения пакетов в систему в зависимости от их размера. Измерение скорости осуществляется при помощи специально разработанной программы, которая получает пакеты с сетевого интерфейса при помощи одного из фреймворков и передает трафик в выделенный кольцевой буфер. Таким образом, программа производит одно копирование и предоставляет возможность дальнейшей обработки пакетов в пространстве пользователя.

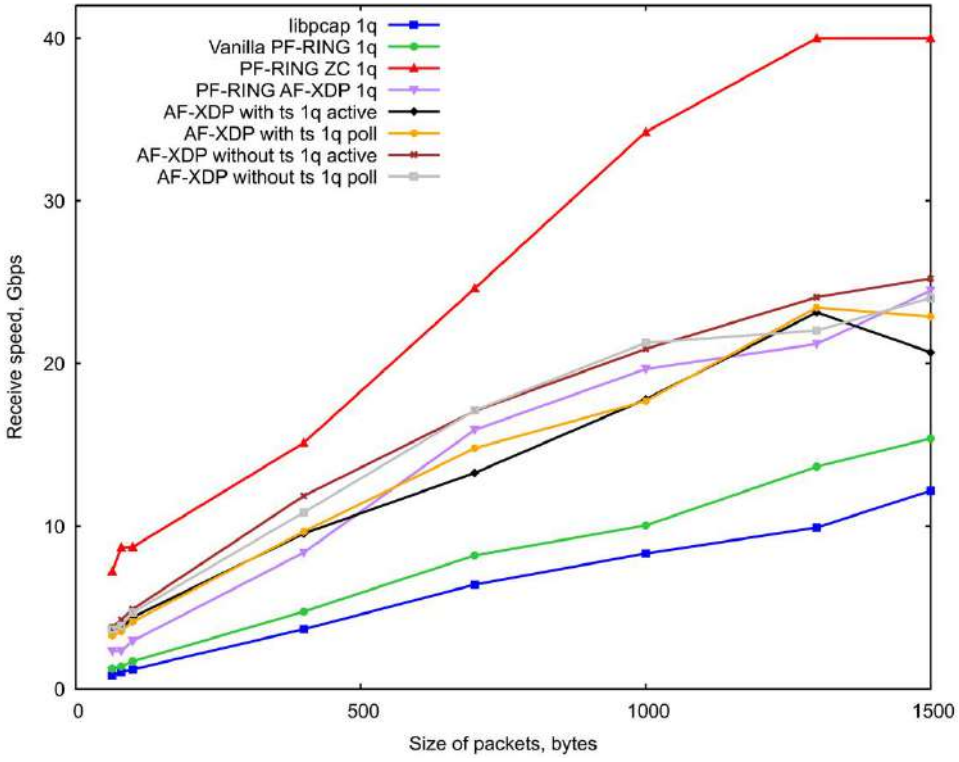


Рис. 13. Сравнение производительности решений на одной входной очереди
Fig. 13. Performance comparison on one input queue

На рис. 13 сравниваются производительности программных решений, работающих в режиме захвата пакетов из одной входной очереди. По всем приведенным графикам можно заметить, что использование libpcap и Vanilla PF_RING закономерно приводит к наименьшим

пропускным способностям. Кроме того, из всех решений необходимо выделить PF_RING ZC, который способен принимать пакеты на скорости 40 Гбит/сек, используя только одну входную очередь. Наиболее близок к нему AF_XDP, работающий в режиме активного ожидания без выставления временных меток. Кроме того, каждое из программных решений тестировалось на приеме пакетов из одной, шести и двенадцати входных RSS очередей в режиме активного ожидания (см. рис. 14).

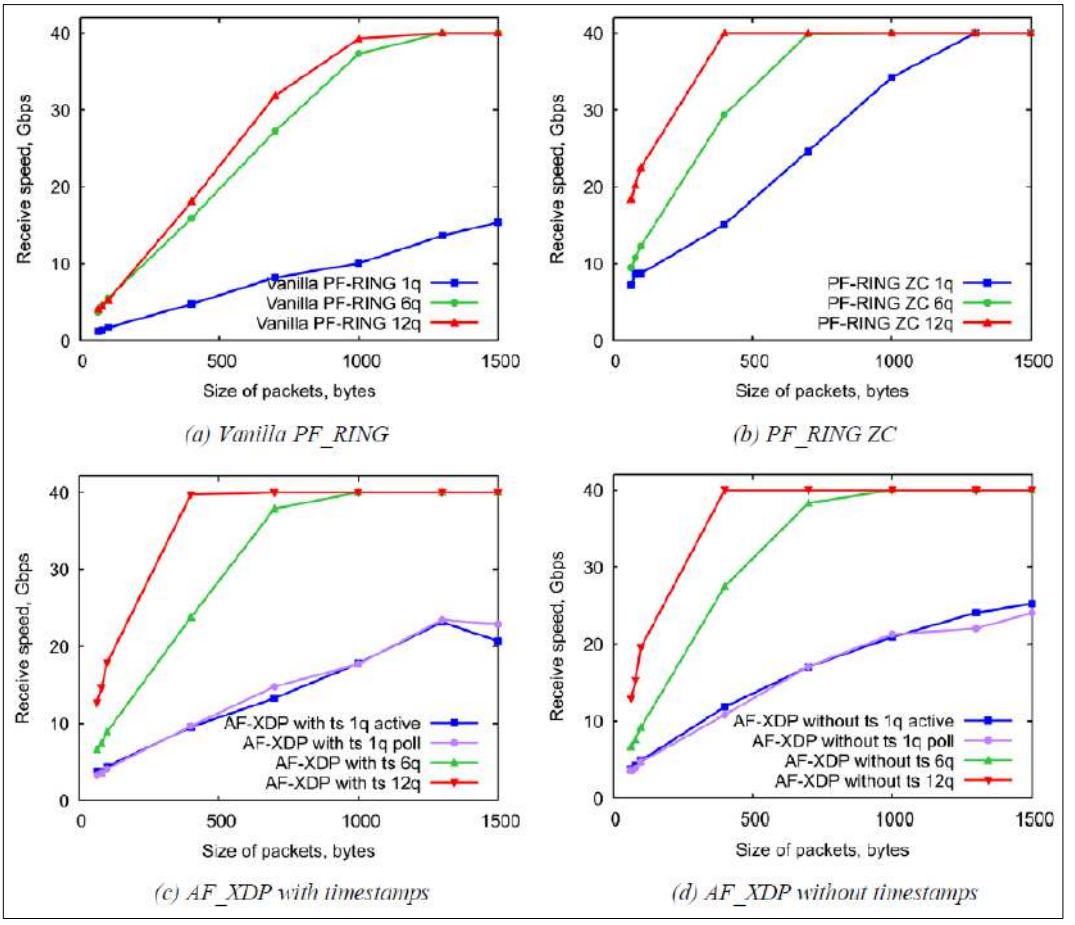


Рис. 14. Измерения производительности
Fig. 14. Performance measurements

Для XDP/eBPF с AF_XDP снята производительность захвата пакетов в режиме опроса (polling), что отражено на рис. 14 (c, d). Помимо этого, поскольку в стандартном режиме AF_XDP не поддерживает выставления временных меток, была реализована eBPF программа, которая записывает временные метки в метаданные пакета и позволяет пользовательским приложениям их считывать. Результаты тестирования данного режима изображены на рис. 14 (c), однако использование дополнительных системных вызовов приводит к ухудшению производительности. С увеличением количества входных очередей все решения показывают прирост производительности. Минусом данного подхода является сильная нагрузка на CPU из-*за* параллельной работы нескольких процессов.

Полученные результаты необходимо интерпретировать как оценочные, поскольку в зависимости от архитектуры пользовательского приложения, решаемой задачи и конфигурации аппаратуры производительность программных решений может сильно отличаться.

5. Заключение

В рамках данной работы был проведен анализ решений по перехвату и обработке высокоскоростного трафика на стандартных сетевых картах. Для каждого из решений выделена область применимости. Libpcap позволяет осуществлять прием пакетов на любых сетевых картах и операционных системах. PF_RING реализует перехват трафика на операционной системе Linux, а при поддержке драйвером сетевой карты может работать в режиме 0-copy. DPDK предоставляет наибольшую совместимость с решениями по программной обработке трафика. AF_XDP позволяет осуществлять прием пакетов без копирований на поддерживающих XDP сетевых картах, кроме того, все больше решений по обработке трафика реализуют его поддержку. Netmap – достаточно легковесный инструмент для перехвата трафика с возможностями расширения функционала.

Полученные при тестировании программных решений результаты отражают возможности современных программно-аппаратных средств в задачах перехвата сетевого трафика, поступающего на скорости порядка 40 Гбит/сек. В качестве дальнейшего развития работы предлагается разработка и тестирование интерфейсов захвата высокоскоростного сетевого трафика с применением фреймворков DPDK и Netmap.

Список литературы / References

[1] D. Cerović, V. Del Piccolo et al. Fast Packet Processing: A Survey. IEEE Communications Surveys & Tutorials, vol. 20, no. 4, 2018, pp. 3645-3676.

[2] L. Linguaglossa, S. Lange et al. Survey of Performance Acceleration Techniques for Network Function Virtualization. Proceedings of the IEEE, vol. 107, no. 4, 2019, pp. 746-764.

[3] Sangjin Han, Keon Jang et al. Berkeley Extensible Software Switch (BESS). Technical Report No. UCB/EECS-2015-155, University of California at Berkeley, 2015, 17 p.

[4] DPDK (Data Plane Development Kit). URL: https://core.dpdk.org.

[5] ntop. URL: https://www.ntop.org/.

[6] T. Høiland-Jørgensen, J. D. Brouer et al. The eXpress data path: fast programmable packet processing in the operating system kernel. In Proc. of the 14th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '18), 2018, pp. 54-66.

[7] L. Rizzo. netmap: A Novel Framework for Fast Packet I/O. In Proc. of the USENIX Annual Technical Conference (USENIX ATC 12), 2012, pp. 101-112.

[8] S. McCanne and V. Jacobson. The BSD packet filter: a new architecture for user-level packet capture. In Proc. of the Winter USENIX Conference (USENIX'93), 1993, 11 p.

[9] Ben Pfaff, Justin Pettit et al. The Design and Implementation of Open vSwitch. In Proc. of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), 2015, pp. 117-130.

[10] László Molnár, Gergely Pongrácz et al. Dataplane Specialization for High-Performance OpenFlow Software Switching. In Proc. of the ACM SIGCOMM Conference (SIGCOMM '16), 2016, pp. 539-552.

[11] Sangjin Han, Keon Jang et al. PacketShader: A GPU-Accelerated Software Router. ACM SIGCOMM Computer Communication Review, vol. 40, issue 4, 2010, pp. 195–206.

[12] M. Trevisan, A. Finamore et al. Traffic Analysis with Off-the-Shelf Hardware: Challenges and Lessons Learned. IEEE Communications Magazine, vol. 55, no. 3, 2017, pp. 163-169.

[13] Robert Morris, Eddie Kohler et al. The Click Modular Router. In Proc. of the Seventeenth ACM Symposium on Operating Systems Principles, 1999, pp. 217-231.

[14] Tom Barbette, Cyril Soldani, and Laurent Mathy. Fast Userspace Packet Processing. In Proc. of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2015, pp. 5-16.

[15] FD.io. Vector Packet Processing - One Terabit Software Router on Intel Xeon Scalable Processor Family Server. White Paper. Cisco, Intel Corporation, FD.io, 2017.

[16] M. Paolino, N. Nikolaev et al. SnabbSwitch user space virtual switch benchmark and performance optimization for NFV. In Proc. of the IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), 2015, pp. 86-92.

[17] A. Farshin, T. Barbette et al. PacketMill: Toward per-core 100-GBPS Networking. In Proc. of the Twenty-Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2021), 2021, pp. 1-17.

[18] Intel: 82599 10 GbE controller datasheet, 2019. http://www.intel.com/content/www/us/en/ethernet-controllers/82599-10-gbe-controller-datasheet.html.
[19] А.И. Гетьман, Е.Ф. Евстропов, Ю.В. Маркин. Анализ сетевого трафика в режиме реального времени: обзор прикладных задач, подходов и решений. Препринт ИСП РАН, вып. 28, 2015, стр. 1-52 / A.I. Getman, E.F. Evstropov, Yu.V. Markin. Analysis of network traffic in real time: an overview of applications, approaches and solutions. ISP RAS Preprint, no. 28, 2015, pp. 1-52 (in Russian).
[20] V. Moreno, J. Ramos et al. Commodity packet capture engines: tutorial cookbook and applicability. IEEE Communications Surveys & Tutorials, vol. 17, no. 3, 2015, pp. 1364-1390.
[21] G. Liao, X. Znu, L. Bnuyan. A new server I/O architecture for high speed networks. In Proc. of the Symposium on High-Performance Computer Architecture, 2011, pp. 255-265.
[22] S. Han, K. Jang, K.S. Park, S. Moon. PacketShader: a GPU-accelerated software router. ACM SIGCOMM Computer Communication Review, vol. 40, issue 4, 2010, pp. 195–206.
[23] W. Wu, P. DeMar, M. Crawford. Why can some advanced Ethernet NICs cause packet reordering? IEEE Communications Letters, vol. 15, issue 2, 2011, pp. 253-255.
[24] V. Moreno, P.M. Santiago del Río, J. Ramos, J.J. Garnica, J.L. García-Dorado. Batch to the future: Analyzing timestamp accuracy of high-performance packet I/O engines. IEEE Communications Letters vol. 16, issue 11, 2012, pp. 1888–1891.
[25] pcap(3pcap). URL: https://www.tcpdump.org/manpages/pcap.3pcap.html.
[26] Wireshark. URL: https://www.wireshark.org/.
[27] Tcpdump. URL: https://www.tcpdump.org.
[28] Snort. URL: https://www.snort.org.
[29] L. Deri. Improving passive packet capture: Beyond device polling. In Proc. of the 4th International System Administration and Network Engineering Conference, 2004, 9 p.
[30] D.Scholz. A look at Intel's Dataplane Development Kit. In Proc. of the Seminars on Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM), 2014, pp. 115-122.
[31] M. Karlsson and B. Töpel. The Path to DPDK Speeds for AF_XDP. In Proc. of the Linux Plumbers Conference, 2018, 9 p.

Информация об авторах / Information about authors

Дмитрий Викторович ЛАРИН является специалистом кафедры системного программирования МФТИ. Его научные интересы включают телекоммуникационные сети, распределенные системы и программную обработку сетевого трафика.

Dmitry Victorovich LARIN is a specialist of the Department of system programming of Moscow Institute of Physics and Technology. His research interests include telecommunication networks, distributed systems and software processing of network traffic.

Александр Игоревич ГЕТЬМАН – старший научный сотрудник, кандидат физико-математических наук. Сфера научных интересов: анализ бинарного кода, восстановление форматов данных, анализ и классификация сетевого трафика.

Aleksandr Igorevich GETMAN – senior researcher, PhD in physical and mathematical sciences. Research interests: binary code analysis, data format recovery, network traffic analysis and classification.

DOI: 10.15514/ISPRAS-2021-33(4)-5

A Method for the Stateful Data-Plane Algorithm State Synchronization in the Network Processing Unit

Y.K. Kuzmin, ORCID: 0000-0001-9204-2141 <yaroslav_konst@lvk.cs.msu.ru>
D.Y. Volkanov, ORCID: 0000-0001-9940-5822 <volkanov@asvk.cs.msu.ru>
J.A. Skobtsova, ORCID: 0000-0001-8351-3191 <xenerizes@lvk.cs.msu.ru>

Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. This work presents a network processing unit based on specialized computational cores that is used for packet processing in network devices (e.g. in network switches). Nowadays stateful data-plane algorithms are developing in software-defined networks. The idea of stateful data-plane algorithms is to move a part of control information from control plane to data plane. But these algorithms require hardware support because they need resources for state handling. This work presents the network processing unit architecture modifications that allow to use stateful data-plane algorithms that require state synchronization between the NPU processing pipelines.

Keywords: software-defined networks; network processing unit; stateful packet processing; network protocols

For citation: Kuzmin Y.K., Volkanov D.Y., Skobtsova J.A. A Method for the Stateful Data-Plane Algorithm State Synchronization in the Network Processing Unit. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021, pp. 69-76. DOI: 10.15514/ISPRAS-2021-33(4)-5

Acknowledgements. This work is partially supported by the Russian Foundation for Basic Research under grant 19-07-01076.

Об одном методе синхронизации состояния алгоритма обработки пакетов в сетевом процессорном устройстве

Я.К. Кузьмин, ORCID: 0000-0001-9204-2141 <yaroslav_konst@lvk.cs.msu.ru>
Д.Ю. Волканов, ORCID: 0000-0001-9940-5822 <volkanov@asvk.cs.msu.ru>
Ю.А. Скобцова, ORCID: 0000-0001-8351-3191 <xenerizes@lvk.cs.msu.ru>

Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. В данной работе рассматривается архитектура программируемого сетевого процессорного устройства (СПУ), основанного на специализированных вычислительных ядрах. В настоящее время в программно-конфигурируемых сетях развиваются алгоритмы обработки пакетов с хранением состояния. Особенностью алгоритмов данного типа является перенос части управляющих функций из плоскости управления в плоскость передачи данных. Но для работы алгоритмов обработки пакетов с хранением состояния требуется поддержка со стороны СПУ. В работе предложены модификации архитектуры СПУ, позволяющие использовать алгоритмы обработки пакетов с хранением состояния и синхронизировать состояние алгоритма обработки пакетов между портами СПУ. Проведено экспериментальное исследование модифицированной архитектуры СПУ.

Ключевые слова: программно-конфигурируемые сети; сетевое процессорное устройство; алгоритмы обработки пакетов с хранением состояния; сетевые протоколы

Для цитирования: Кузьмин Я.К., Волканов Д.Ю., Скобцова Ю.А. Об одном методе синхронизации состояния алгоритма обработки пакетов в сетевом процессорном устройстве. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 69-76 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(4)–5

Благодарности: Работа выполнена при частичной поддержке РФФИ, грант № 19-07-01076.

1. Introduction

Nowadays software-defined networks (SDN) are being developed [1]. The main principle of SDN technology is placing conrol functions in separate server called controller. Control functions are moved from network devices to the controller.

The main functional element of a network device is the network processing unit (NPU). NPU is a specialized integrated curcuit that is used for packet processing in network devices.

Nowadays programmable NPU are being developed. NPU of this type allow to load new packet processing programs and define new data transfer protocols [2].

Packet processing in programmable NPU is done according to the packet processing program that implements data-plane algorithm. One class of such algorithms is a class of stateful data-plane algorithms. Stateful data-plane algorithms are used in data processing centers' networks and in telecommunication providers' networks [3]. The state of data-plane algorithm is a set of changeable variables, keeping their values on moving to next packet processing. The examples of such algorithms are load balancing with consistency [4], port knocking algorithm [5], failure recovery algorithm [4]. The main feature of stateful data-plane algorithm is the ability to introduce dependency of the process of packet processing on the properties of packets, processed by this NPU before. With the development of SDN and programmable NPU the task of implementing stateful data-plane algorithms on programmable NPU appears.

If NPU does not have state handling support, the state will be stored on the controller. But the state can change depending on the properties of packets, processed by the NPU. If the controller stores the state, the network device will access the controller for every packet that will lead to network device perfomance reduction, packet loss and network services work failures. So, implementing the hardware support mechanism for stateful data-plane algorithm state storage in NPU becomes actual.

The architecture of the considered NPU RuNPU does not support stateful data-plane algorithms. Thus, this work proposes the RuNPU architecture modifications that allow to use stateful data-plane algorithms with synchronization between the ports of the network device.

2. Formulation of the problem

It is necessary to propose the RuNPU architecture modifications that will allow to support stateful data-plane algorithms with state synchronization between NPU ports.

RuNPU architecture has the following features:

- Each NPU port has its processing pipeline.
- Processing pipelines are not connected to each other and operate in parallel.
- Packet processing time on one pipeline stage is limited to 250 ticks.

Data-plane algorithm state is represented as a set of variables. The following symbols are introduced:

- n is a number of pipelines in the NPU;
- l is a number of stages in each pipeline;
- q is a number of state variables available for each pipeline stage;
- $P = \{P_1, \ldots, P_n\}$ is a set of the NPU's pipelines;
- D_{ij} is a j-th stage of i-th pipeline;

- $S_{ijk}(t)$ is a value of k-th state variable of the stage D_{ij} in the time moment t;
- $S_{ij}(t) = \{S_{ij1}(t), \dots, S_{ijq}(t)\}$ is a state of the stage D_{ij} in the time moment t;
- $S_i(t) = \{S_{i1}(t), \dots, S_{il}(t)\}$ is a state of the i-th pipeline in the time moment t.

Time moments $t_m, m = 0,1,2, \dots$ are considered. They correspond to the moment of the next instruction execution beginning. The following situation is considered: during the instruction $i-1$ execution the state of the x-th pipeline has changed. It means that $\exists y, \exists k: S_{xyk}(t_{i-1}) \neq S_{xyk}(t_i)$. It is necessary to propose state synchronization system that will allow other pipelines to have an access to an updated state variable $\exists a \in N \cup \{0\}: \forall p \in \{1,2, \dots, n\} \Rightarrow S_{pyk}(t_{i+a}) = S_{xyk}(t_i)$ with minimum a. And the proposed modifications must take into consideraton the RuNPU architecture features and limits.

3. RuNPU architecture description

During the processing the packet header and metadata are moving through every stage of the pipeline (fig. 1). Every stage processes the packet header for fixed number of ticks. Now this value is equal to 250 ticks. The stage consists of RISC processing core and memory device containing packet processing program. Packet header processing on every stage runs according to the following scheme. First of all, the packet header and metadata are loaded into the pipeline stage memory. After it, the program loaded into the stage memory device is executed. When the program finishes its work, the packet header and metadata are transferred to the next pipeline stage and new packet header and metadata are loaded into the current pipeline stage memory.

This architecture has two aspects that do not allow to use stateful data-plane algorithms. Firstly, pipelines do not have memory devices for the algorithm state. Pipeline stage memory device contains only program microcode, packet header and metadata. Secondly, NPU architecture contains a set of pipelines that work in parallel and do not have any connections to each other. It means that stages of different pipelines can not exchange data to update state in all pipelines. So, this NPU architecture does not allow to use stateful data-plane algorithms.

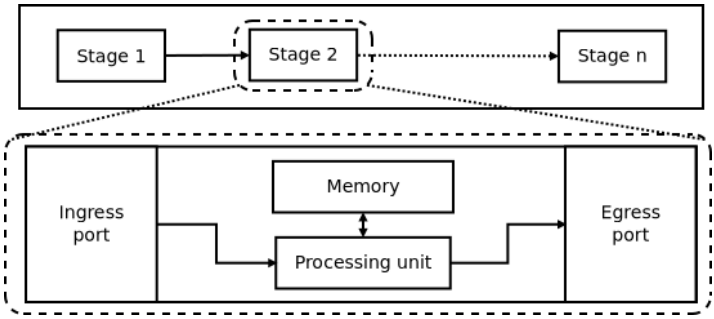


Fig. 1. NPU processing pipeline scheme

4. Related work

The analysis of existing methods of memory synchronization in multicore systems was done. The overview is carried out according to the following criteria:

- Synchronization delay, ticks (the paramerer a value).
- Memory access time, ticks.
- A possibility of long-time blocking.

4.1 Flexible multiprocessor locking protocol

Flexible multiprocessor locking protocol (FMLP) is a mutex based synchronization algorithm for real time systems [6]. In this approach shared memory resources are protected using mutexes.

System operation time is divided into two phases: reading phase and writing phase. To get an access to a shared memory the process must acquire the appropriate mutex for reading or for writing depending on the access type. All memory reading operations start in the beginning of the phase and the phase is not changed until all requests in the current phase are done. Writing operations are done in the writing phase in FIFO order.

4.2 Combining mechanism

The work [7] proposes the following approach: the memory is a set of memory devices. Devices and processing units are connected using a layered packet switched network.

The network has the following properties:

- The network is no overtaking. It means that if two messages are sent from one node in some order and arrive later at some other node then they arrive in the same order as they were sent.
- A reply message is sent back using the same path as the request message.

The memory requests are Read-Modify-Write (RMW) operations. An RMW operation is equivalent to the execution of the following function:

```
function RMW(X, f)
begin
   temp := X;
   X := f(X);
   return temp;
end
```

This operation applies the transformation *f* to the value *X* stored in memory and returns the old *X* value to the processing unit.

The combining mechanism is an approach to handle parallel requests to the same memory location. A memory request has the form *(id, addr, f)*, where *id* as a unique request identifier, *addr* is an address of the memory location and *f* is an identifier of the transformation function. When two requests to the same address are received on the same network switch they are combined into single request. It is done in the following way: the messages are *(id1 , addr, f)* and *(id2 , addr, g)*. They have the same memory address and conflict. Combining them into single request is done according to the following steps:

- The switch saves *id1*, *id2* and *f* and forwards the message *(id1, addr, f ∘ g)*.
- When a reply message *(id1, val)* reaches the switch, the saved information is retrieved by matching the ids. The message *(id1, val)* is forwarded as a reply to the first request and a message *(id2, f (val)* is forwarded as a reply to the second request.

4.3 Write-update algorithm

In the work [8] various cache coherent algorithms are overviewed. One of them is write-update cache coherence policy. According to this policy, if some processing unit writes data to the memory it is updated in the same cache blocks in other processing units' cache.

It can be applied to the NPU in the following way: each processing unit has its memory device for the packet processing algorithm state. Memory devices located at pipeline stages with equal depths are connected with a shared bus. When a processing unit reads data from the state memory the value is taken from the local memory device. When some processing unit writes data to the state memory it is written to the local memory device and updated in all other memory devices using the bus.

4.4 Result

The problem of the FMLP algorithm is that phases do not have fixed length, so this approach can lead to request locking for an unestimated period of time what can lead to pipeline errors because it has only 250 ticks to process a packet header. Two other approaches do not allow such behavior, so their applicability and exact properties will be evaluated during the experimental research. The overview results are shown in the table 1.

Table 1 Algorithm overview result

Algorithm	**Value**	**Memory access time (read / write), ticks**	**Possibility of long-time blocking**
FMLP	Depends on the blocking duration	1 / 1	+
Cobining mechnism	1	Number of switches on the route to memory device	-
Write-update algorithm	1	1 / 2	-

5. Proposed NPU architecture modifications

5.1 Memory synchronization via the shared bus

In this approach, the following modifications are proposed: memory devices containing the algorithm state are added to each pipeline stage. Memory devices on pipeline stages with equal depths are connected using a shared bus. The bus is used to synchronize data in the memory devices. When a memory cell in some memory device is updated, the new value and memory cell address are sent to other memory devices via the bus.

There are two operations available for the processing unit: read a value from a memory cell and write a value to a memory cell. When the value is read it is taken from the memory device that is on the same pipeline stage. When the value is written to the memory it is written to the memory device on the same pipeline stages and then is sent to appropriate memory devices in other pipelines.

5.2 Combining mechanism

In this approach, memory devices and processing units are connected using a packet switched network-on-chip with special switches that allow to combine memory requests. Memory requests have a Read-Modify-Write form. Memory request has a form of a tuple *(id, addr, f)* where *id* is a unique request identifier, *addr* is a memory cell address and *f* is a memory operation identifier. The response consists of two values: request identifier and the value that was in the memory cell before the memory operation was done.

This approach requires memory cells and network switches to have specialized arithmetic units to perform memory operations and memory request combinations.

6. RuNPU simulation model

The RuNPU simulation model is used for the experimental research. It is written in Python programming language and allows to evaluate various NPU parameters such as pipeline throughput and power consumption. The simulation model input consists of two parts: pcap files with test packets and the packet processing program written in the assembly language. Each ingress port has separate pcap file with a sequence of packets. The output consists of the statistics and pcap files with packets that were sent via the NPU egress ports.

The simulation model consists of a number of the modules that are responsible for NPU modules work simulation. These modules are:

- Main application module. This module is responsible for other modules initialization and configuration.
- Pipeline module performs pipeline initialization and controls all pipeline components. There are 24 pipelines in the RuNPU simulation model.
- InFIFO module reads the network packets from the pcap file that corresponds to the NPU ingress port.
- OutFIFO module writes processed packets to the pcap file that correponds to the NPU egress port.
- DE (Decision Engine) module represents the NPU pipeline stage and is responsible for packet header processing according to the packet processing program.
- PacketMem module stores the packet bodies while packet headers are being processed in the pipelines.

For each proposed approach the simulation model was modified. For the synchronization method based on shared bus the following modules were added: memory devices for the data-plane algorithm state and the shared bus modules. For the combining mechanism approach the network module, network switch modules and memory devices for the algorithm state were added to the simulation model.

7. Experimental research

For the experimental research a program that implements flowlet switching algorithm was written. This algorithm is used to balance packet flows on transport layer and requires state synchronization between the NPU pipelines. Pcap files with test packets were generated. Test packet sequences contain a number of packet groups. Each group consists of packets that belong to a single transport flow.

During the experimental research the test packets were processed by the simulation model and the required statistics was collected. The experimental research consists of a series of the NPU simulation model runs with a different number of pipelines working with the state. It allows to evaluate how the number of pipelines actively working with state affects the time required to process a packet header.

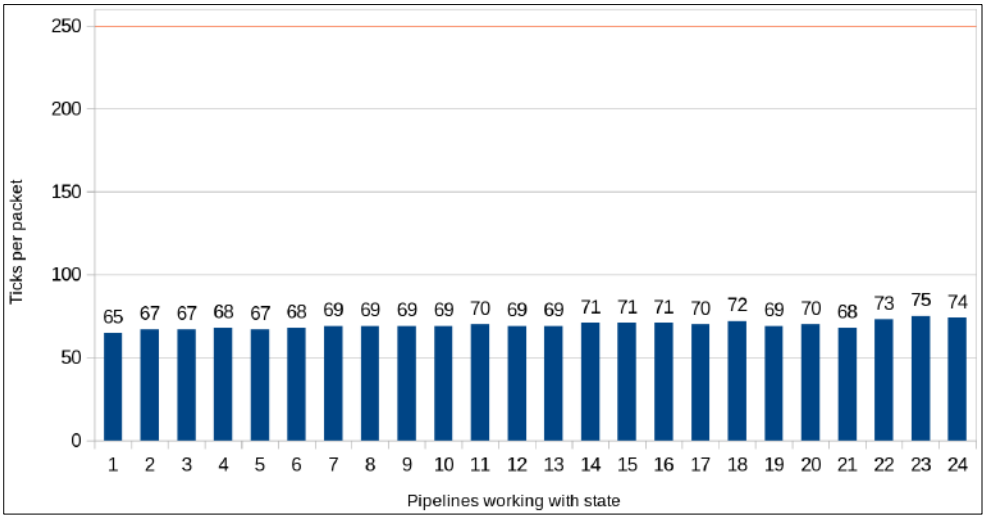


Fig. 2 Shared bus approach evaluation results

The experimental research results for the shared bus approach was finished (Fig. 2). It shows that the number of ticks required to process a packet header does not exceed 250 ticks.

9. Conclusion and future work

This work proposes the RuNPU architecture modifications that allow to synchronize stateful data-plane algorithm state between the NPU processing pipelines. An overview of the existing methods was done and two approaches were selected for further implementation and experimental research. A simulation model and test data for the experimental research were prepared.

In the future it is planned to finish an experimental research for the combining mechanism approach and determine what approaches are applicable.

Список литературы / References

[1]. S Smeliansky R.L. System Defined networks. Open Systems. DBMS, issue 9, 2012, pp. 15-26 (in Russian) / Смелянский Р.Л. Программно-конфигурируемые сети. Открытые системы. СУБД, вып. 9, 2012 г., стр. 15-26.

[2]. Bezzubtsev S.O., Vasin V.V. et al. An Approach to the Construction of a Network Processing Unit. Modeling and Analysis of Information Systems, vol. 26, no. 1, 2019, pp. 39-62. (in Russian) / Беззубцев С.О., Васин В.В. и др. Об одном подходе к построению сетевого процессорного устройства. Моделирование и анализ информационных сист1 ем, том 26, no. 1, 2019 г., стр. 39-62.

[3]. Bifulco Roberto, and Gábor Rétvári. A Survey on the Programmable Data Plane: Abstractions, Architectures, and Open Problems. In Proc. of the IEEE 19th International Conference on High Performance Switching and Routing (HPSR), 2018, pp. 1-7.

[4]. Carmelo C., Pollini L. et al. Traffic Management Applications for Stateful SDN Data Plane. In Proc. of the Fourth European Workshop on Software Defined Networks, 2015, pp. 85-90.

[5]. Bianchi Giuseppe, Bonola Marco et al. OpenState: programming platform-independent stateful openflow applications inside the switch. ACM SIGCOMM Computer Communication Review, vol. 44, issue 2, 2014, pp. 44-51.

[6]. Brandenburg Björn B. and James H. Anderson. Reader-Writer Synchronization for Shared-Memory Multiprocessor Real-Time Systems, In Proc. of the 21st Euromicro Conference on Real-Time Systems. 2009, pp. 184-193.

[7]. Kruskal Clyde P., Larry Rudolph, and Marc Snir. Efficient synchronization of multiprocessors with shared memory. ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 10, issue 4, 1988, pp. 579-601.

[8]. Stenstrom Per. A survey of cache coherence schemes for multiprocessors. Computer, vol. 23, no. 6, 1990, pp. 12-24.

Information about authors / Информация об авторах

Yaroslav Konstantinovich KUZMIN is a student at the department of Computer Systems and Automation of the Faculty of Computational Mathematics and Cybernetics. Research interests include software-defined networks; network processing units.

Ярослав Константинович КУЗЬМИН – студент магистратуры кафедры АСВК ф-та ВМК. Научные интересы включают программно-определяемые сети, сетевые процессоры.

Dmitry Yuryevitch VOLKANOV – Candidate of Physical and Mathematical sciences, Associate Professor at the department of Computer Systems and Automation of the Faculty of Computational Mathematics and Cybernetics. Areas of research: analysis and design of network processing unit architecture.

Дмитрий Юрьевич ВОЛКАНОВ – кандидат физико-математических наук, доцент кафедры АСВК ф-та ВМК. Направления исследований: анализ и разработка архитектуры сетевого процессора.

Julia Alexandrovna SKOBTSOVA – Programmer, Faculty of Computational Mathematics and Cybernetics, department of Computer Systems and Automation, laboratory of Computer Systems. Research interests: software-configurable networks, network processor units, hardware description languages.

Юлия Александровна СКОБЦОВА – специалист, факультет ВМК, кафедра автоматизации систем вычислительных комплексов, лаборатория вычислительных комплексов. Научные интересы: программно-конфигурируемые сети, сетевые процессорные устройства, языки описания аппаратуры.

DOI: 10.15514/ISPRAS-2021-33(4)-6

Data compression algorithms for flow tables in Network Processor RuNPU

N.I. Nikiforov, ORCID: 0000-0003-3394-1766 <nickiforov.nik@gmail.com>
D.Yu. Volkanov, ORCID: 0000-0001-9940-5822 <volkanov@asvk.cs.msu.ru>

Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. This paper addresses the problem of packet classification within a network processor (NP) architecture without the separate associative device. By the classification, we mean the process of identifying a packet by its header. The classification stage requires the implementation of data structures to store the flow tables. In our work, we consider the NP without the associative memory. Flow tables are represented by an assembly language program in the NP. For translating flow tables into assembly language programs, a tables translator was used. The main reason for implementing data compression algorithms in a flow tables translator is that modern flow tables can take up to tens of megabytes. In this paper, we describe the following data compression algorithms: Optimal rule caching, recursive end-point cutting and common data compression algorithms. An evaluation of the implemented data compression algorithms was performed on a simulation model of the NP.

Keywords: network processor; software-defined networks; packet classification; data compression

For citation: Nikiforov N.I., Volkanov D.Yu. Data compression algorithms for flow tables in Network Processor RuNPU. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021, pp. 77-86. DOI: 10.15514/ISPRAS-2021-33(4)-6

Acknowledgments. This work is partially supported by the Russian Foundation for Basic Research under grant 19-07-01076.

Исследование применимости алгоритмов сжатия данных для таблиц потоков в сетевом процессоре RuNPU

Н.И. Никифоров, ORCID: 0000-0003-3394-1766 <nickiforov.nik@gmail.com>
Д.Ю. Волканов, ORCID: 0000-0001-9940-5822 <volkanov@asvk.cs.msu.ru>

Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. Данная статья посвящена проблемам классификации пакетов в архитектуре сетевого процессорного устройства (СПУ) без выделенного ассоциативного устройства. Под классификацией мы понимаем процесс идентификации пакета по его заголовку. На этапе классификации требуется реализация структур данных для хранения таблиц потоков. В данной работе рассматривается СПУ без адресуемой памяти, а таблицы потоков представляются в виде программы на языке ассемблера СПУ. Для перевода таблиц потоков в программу на языке ассемблера используется транслятор таблиц потоков. Необходимость реализации алгоритмов сжатия данных в трансляторе таблиц потоков обуславливается тем, что современные таблицы потоков могут занимать память объёмом до десятков мегабайт. В настоящей статье рассматриваются следующие алгоритмы сжатия данных: алгоритм оптимального кэширования, рекурсивного отсечения и общие алгоритмы сжатия данных. Оценка реализованных алгоритмов сжатия данных проводилась на имитационной модели СПУ.

Ключевые слова: алгоритмы сжатия данных; сетевое процессорное устройство; ПКС; классификация пакетов

Для цитирования: Никифоров Н.И., Волканов Д.Ю. Исследование применимости алгоритмов сжатия данных для таблиц потоков в сетевом процессоре RuNPU. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 77-86 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(4)–6

Благодарности. Работа частично поддержана Российским фондом фундаментальных исследований (грант 19-07-01076).

1. Introduction

At present, software-defined networks (SDN) are in active development and require high-performance switches [1]. The main functional element of the high performance SDN switch is a programmable network processor (NP). The network processor is a system-on chip specialized for network packet processing. In this work, we consider a programmable NP. A programmable NP is one that supports on-the-fly modification of the packet processing program and the set of header fields to be processed.

In this article, we discuss data compression algorithms used for flow tables. Flow tables are needed for packet classification process. A flow table is the set of rules defined by OpenFlow protocol. OpenFlow is one of the most common protocols for controlling a network SDN switch. This paper considers OpenFlow version 1.3 [2]. Each rule contains a match field, a bit string by witch a packet can be identified and a set of actions, that the NP performs on this packet. Classification is the process of the identification of a network packet by its header.

This article has the following structure: in second section we introduce problem, in third section we introduce the NP architecture and flow tables translator, in fourth section we describe related work, in fifth section we describe data compression algorithm implementation and in sixth section we introduce our evaluation methodology.

2. The problem

Let us consider OpenFlow tables formalisation. An ordered set of all considered attributes is denoted as $I = \{m_1, m_2, \ldots, m_k\}$. Every attribute m_i from the set I is described by a bit string $m_i \in \{0, 1, *\}_i^W$. In this article symbol $*$ denotes any bit. But, if $\exists m_i^j \in m_i$ and $m_i^j = *$, then for $\forall m_i^k$, where $k > j, m_i^k = *$. The length of the attribute is denoted $len(m_i) = W_i$.

Flow tables are represented by a set of rules $R = \{r_1, r_2, \ldots, r_n\}$. With every rule r_i binding the features:

- An index i;
- A priority $p_i \in Z_+$;
- A vector of values of attributes $f_i = \{f_i^1, f_i^2, \ldots, f_i^k\}$, where f_i^j is an attribute value $m_i \in I$;
- A set of actions $Ai = \{a_1, a_2, \ldots, a_z\}$.

A network packet header x and its metadata with vector values of attributes $g = \{g^1, g^2, \ldots, g^k\}$ $(x \rightarrow g)$, a match rule $r_i \in R$ with a vector of values of the attributes $f_i = \{f_i^1, f_i^2, \ldots, f_i^k\}$ and a priority p_i (a rule $r_i \in R$ identifies a network packer with a vector values of attributes g), if:

1) a vector values of attributes g match a vector of values of the attributes $f, \forall g_i \in g,\ len(g_i) = len(f_i),\ \forall f_i^{lj} \in f_i^l,\ f_i^{lj} \in \{*, g^{lj}\},\ l = \overline{1,k}$;
2) a priority p_i is the highest among all rules $r_i \in R$, if a vector g match a vector f_j.

The set of rules *R* must satisfy the following constraint. For any two rules $r_i, r_j \in R, r_i \neq r_j$, if their vectors of values intersect, there is a set of attribute values. This set corresponds to vectors of values of attributes of both rules $p_i \neq p_j$.

Let us introduce the function for network packet identification $x \to gin$ flow table R, (denotes as $R(x)$). It returns a set of actions, that corresponded to the rule $x \to gin$.

$R(x) = A_{r_i}$ where A_{r_i} is the set of actions $r_i \in R$.

We need to introduce a **similar concept** of the sets of rules R_1 and R_2. The set R_1 is similar to the set R_2 when for any network packet header, that can be identified by some rule from the set $r_i \in R_1$, and there exists another rule that identifies it as $r_j \in R_2$, and $A_i = A_j$.

We need to develop an algorithm for compressing flow tables. This algorithm must translate an input flow table (a set of rules R_1) into a new compressed set of rules R_2.

1) The set of rules R_1 is similar to the set of rules R_2.
2) The cardinality of the set R_2 must be lower than the cardinality of the set R_1.

3. Network processor architecture

In the considered NP the pipeline architecture is used, with each pipeline consisting of 10 computing blocks. To avoid complex memory organization, there is no associative memory in the considered NP. The NP uses the same memory both for commands and data.

Let us consider the pipeline NP architecture. Each computing block has an access to the memory area where the program with data is located. There is a limit of 25 clock cycles per packet on each processing block. There is up to 512 kilobytes to store assembly language program representing flow tables. Due to the instruction set architecture, there is no separate memory area where data is stored. Therefore, the microcode contains all the data, required to classify packets.

3.1 Flow tables translator

Flow tables translator is a tool that is executed on CPU. It is used for flow tables translating into assembly language programs, that can be interpreted by NP. Flow tables translator uses tree structures for flow table representation. Every node of the tree structure can be associated with a table rule. After building a tree every node is translated into a part of an assembly language program. Here is a flow tables translator workflow:

1) Load a flow table from file.
2) Check every rule in the table.
3) Build a tree structure from a set of rules.
4) Translate every node into a part of an assembly language program.
5) Combine all translated parts into the one assembly language program.
6) Add a header that corresponds to used protocol.
7) Write the assembly language program into file.

This tool was implemented in work [3]

4 Related work

In this section, we introduce a review on data compression algorithms, that already used for other network processors [4]. To choose algorithms for implementation in NP we used the following criteria:

1) Compression rate, is needed for algorithm performance evaluation.
2) Evaluation of compression algorithm complexity.
3) Usability of compressed flow tables without decompression.
4) The necessity to use external memory by the algorithm.

4.1 Most common data compression algorithms

Data compression algorithms have evolved over the years. Nowadays compression algorithms can be used in many different ways. In this section we describe the algorithms that compress data in binary format. There are most known of them: Huffman codding, JPEG, LWZ, zip. These algorithms require decompression for data usage. And this is why we will not use them in our flow table translator.

4.2 Optimal rule caching

Optimal rule caching algorithm is more specific data compression algorithm. It is used for table compressing in SND switches [5]. It is based on search tree structure, that is built based on rules usage frequencies. There are two trees: the first tree consists of the most used rules. This tree is translated into assembly language program. The second tree consist from other rules; it is stored in CPU memory.

4.3 Recursive end point cutting

Recursive end-point cutting algorithm is based on HyperSplit tree usage. Compressing is performed by destroying duplication rules [6]. This algorithm permits operations with flow tables without full rebuilding tree. By rules duplication we understand the following rules:

- A rule storing in a node duplicates the rules in leaf nodes (particle duplication).
- A rule storing in a node duplicates the rules in all leafs nodes (full duplicating rule).

This algorithm recursively uses NewHypersplit tree to remove duplicate rules from the currently being built tree. The deleted duplicate rules are then collected into a second rule table, called a recursive table, to build a second tree. It is possible that duplicate rules still exist in the second tree, and some of them are also removed and used to build the third tree.

This tree building process is performed recursively while there are duplicate rules in the last tree.

4.4 Algorithms comparison

Let us describe data compression algorithms comparison in Table 1. Each algorithm has its own pros and cons.

1) Optimal rule caching – has the highest compression ratio, and is quickly implemented in the considered NP. The need to use external memory imposes additional overhead on some packet processing.

2) Recursive end-point cutting – has the lowest compression ratio, it is more difficult to implement than the optimal rule caching algorithm. Moreover, this algorithm does not require the use of external memory.

3) Common data compression algorithms – have good compression ratios on average, but require data decompression.

Table1. Data compression algorithms comparison

Name	**Complexity of construction**	**Compression rate**	**CPU memory usage**	**Decompression needed**
Optimal rule caching	$O(N^2)$	0.1 … 0.9	yes	no
Recursive end-point cutting	$O(N * log(N))$	0.1	no	no
Common data compression algorithms	$O(K * log_2 N)$	0.1 … 0.8	no	yes

5. Our solution

In this section, we introduce our solution of flow tables compressing.

5.1 Flow table optimization

First of all, we need to introduce operation **getting last important bit** $last(m_i) = j, m_i^j \in \{0,1\}$ and $m_i^{(j+1)} = *$. We claim that the rules $r_i \in R$ and $r_j \in R$ are the **same** if $\forall u \in len(f_i)\ last(f_i^u) = last(f_j^u) = l$ but $f_i^{ul} \neq f_j^{ul}$ and $A_i = A_j$. For flow table optimization we need to remove all **same** rules.

5.2 Main flow table compression algorithm

Let us introduce a packet header distribution P where p_x mean network packet income probability $x \to g = \{g^1, g^2, \ldots, g^k\}$. We need a correction ratio $T_P(R_1, R_2)$ where R_1 and R_2 are two different flow tables. Thus correction ratio means probability of incoming network packet header by distribution P. As well as probability of identifying this network packet by rules $r_1 \in R_1$ and $r_2 \in 2$. Moreover, the sets of actions of this rules are similar $A_1 = A_2,\ A_1 \in r_1,\ A_2 \in r_2$.

$$T_P(R_1, R_2) = \sum_{x \to g, R_1(x) = R_2(x)} p_x$$

The optimal correction ratio for flow table R and a number of rules n and a network packet header distribution P is:

$$\zeta(n, R, P) = \max_{R_i, |R_i| <= n} T_P(R, R_i).$$

Let p^i be probability of choosing rule $r_i \in R$, in distribution P. Let rules in flow table R be in not increasing order of their probabilities. Then:

$$\zeta(n, R, P) \geq \sum_{i \in [1,n]} p^i + 1 - \sum_{i \in [1,n_0]} p^i \geq n/n0.$$

This algorithm needs exploration and building a flow table R_a based on input flow table R. There is a minimal set of rules (n_0) and a maximum optimal correction ratio $\zeta(n, R, P)$.

5.3 Software solution

In this section, we introduce software workflow of our algorithm. First of all we need to add a new fields in tree structure nodes for our algorithm.

- A probability into tree node. It must be filled if node contains rule.
- A sum of probabilities of leaf nodes.

Let us introduce program operation for split tree.

- Generate a set of tree nodes.
- Sort this set in non-increasing order.
- Create a counter that stores a sum of node probabilities.
- Get the first node with maximum self-probability.
- Increase the counter.
- Add this node into another set and remove from first.
- Repeat last three operations while counter less than 0.95.
- Build tree from second set of rules.

After performing these operations, we get the set of nodes. We could build first tree from second set of rules and second tree from first set of nodes. After this, we need to translate the first tree into an assembly language program.

5.3.1 Notation used

Let $node_1$, $node_2$ – tree vertices, *value* – some feature value. Let's introduce the following notations:

- *Tree.root* – the root node of the tree *Tree*.
- $node_1(value)$ – the descendant of the node $node_1$, connected to it by an arc with the mark value.
- $node_1.rules$ – set of rules corresponding to node $node_1$.
- $node_1.edges$ — set of marks of arcs coming from node $node_1$.
- $node_1.prob$ – an amount of probabilities of rules.
- $copy(node_1, val, node_2)$ – a procedure that adds to the node node1 a descendant with an arc marked value, copying the tree that forms the node $node_2$.
- $equals(node_1, node_2)$ – function that returns true if the trees formed by nodes $node_1$ and $node_2$ are the same, otherwise it returns false. The comparison takes into account the rule sets and arc labels associated with the nodes.
- $same(rule_1, rule_2)$ – function that returns true if rules are **same**.
- $isleaf(node_1)$ – function that returns true if $node_1$ – tree leaf, otherwise it returns false.

5.3.2 Flow table optimization algorithm

Let us introduce procedure **Same** (Listing 1), it returns a set of rules that are a union of **same** rules in the sets of two nodes.

Let us introduce flow table optimization algorithm. It can be described by the procedure **Optimize** (Listing 2), where **node** – tree node. For optimizing flow table, we need to perform this procedure to *Tree.root*.

```
1  procedure Same(node_1, node_2):
2      rules = {}
3
4`     for all rule_1 in node_1.rules do
5          for all rule_2 in node_2.rules do
6`             if same(rule_1, rule_2) then
7`                 rules += {rule_1 union rule_2}
8`             endif
9
10`    return rules
```

Listing 1. Procedure for obtaining a set of rules derived from the same rules

```
1`procedure Optimize(node):
2      if not isleaf(node) then
3          for all val_1 in node.edges do
4              for all val_2 in node.edges do
5                  if val_1 not equal val_2 then
6                      node.rules += Same(node(val_1), node(val_2))
7                  endif
8          for all val in node.edges do
9              Optimize(node(val))
10     endif
```

Listing 2. Procedure for optimizing the tree

6. Evaluation

6.1 Evaluation methodology

In this section, we describe evaluation methodology. Flow table compression algorithms can be assessed by an assembly language program evaluation. This is so because flow tables translator with

implemented data compression algorithms translates flow table into an assembly language program. We used the following parameters in our evaluation:

- An assembly language program memory usage%Memory usage by assembly language program.
- An assembly language program average number of instructions requires for one packet processing.

The described analysis requires doing the following actions for each flow table.

1) Choose a flow table for this experiment.
2) Translate the flow table using flow tables translator implementation based on: LPM tree, AVL tree, Our flow table compression algorithm.
3) Execute simulation on the NP simulation model.
4) Evaluate results.

6.1.1 Memory usage calculation method

The flow tables translator tool uses intermediate flow table representation as trees. Each node of the tree is translated into an assembly language program part. Fully assembled from parts assembly language program has *N* instructions. Every instruction uses 128 *bits* of memory. Therefore, memory usage defined as *M* can be calculated as:

$$M = 128 * N.$$

In our evaluation results we use *Kbytes* to represent memory usage units.

6.2 Evaluation data

Several variants of the flow tables should be used for the evaluation. These variants cover most usable network protocols. In this section, we will introduce the flow table templates.

- The first pattern – a flow table rule pattern contains the values of three attributes: an input port number, a destination MAC address and a source MAC address.
- The second pattern – a flow table rule pattern contains the values of two attributes: an IPv4 destination address and an IPv4 source address.
- The third pattern – a flow table contains five attributes: an input port number, a destination MAC address, a VLAN ID, a L3-level header ID (EtherType) and a destination IPv4 address.

An example of input data represented in Listing 4.

```
{SRC_MAC , DST_MAC , INSTR}
{SRC_MAC , DST_MAC , INSTR}
+---+-----------+-----------+--------------+
| 1 | :12       | :10:1     | goto_table 1 |
+---+-----------+-----------+--------------+
| 1 | :23:45    | :20       | goto_table 1 |
+---+-----------+-----------+--------------+
| 1 | :0        | :1        | goto_table 1 |
+---+-----------+-----------+--------------+
```

Listing 4. Example of input data for input of a flow table

6.3 Evaluation results

We performed an evaluation on the simulation model of the NP. This evaluation shows that our solution allowed to use several times less memory of the NP.

We carried out our research for different sized flow tables. Currently, the maximum size of a flow table is 6000 rules with compression. Flow table without compression can contain only about 1500 rules.

Optimal caching has the best compression rate (fig. 1a) but the worst average number of instructions required to processing one packet (fig. 1b).

This can be explained by necessity to use many instructions to make the CPU call.

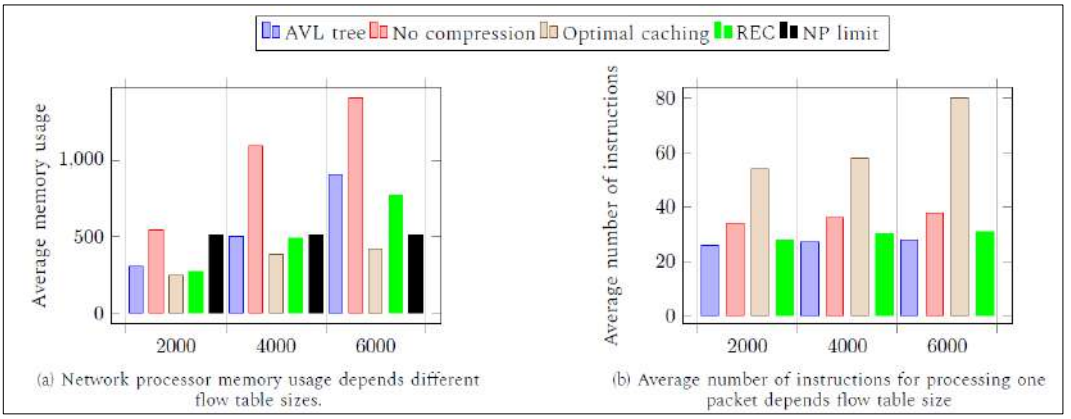


Fig. 1. Results of evaluation

7. Future work

In the future works, we will refine evaluation data. We expect less memory usage with our compression algorithm implemented into flow table translator. In the first experiments conducted, we obtained results showing a significant reduction in the amount of memory usage with the help the data compression algorithm. After this we could check possibility of TCAM memory implementation and use this compression algorithm for it.

References / Список литературы

[1] Smeliansky R.L. System Defined networks. Open Systems. DBMS, issue 9, 2012, pp. 15-26 (in Russian) / Смелянский Р.Л. Программно-конфигурируемые сети. Открытые системы. СУБД, вып. 9, 2012 г., стр. 15-26.

[2] Open Networking Foundation. OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04). 2012.

[3] Markoborodov A., Skobtsova Y., and Volkanov D. Representation of the OpenFlow Switch Flow Table. In Proc. of the International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTeC). 2020, pp. 1–7.

[4] Braun Wolfgang and Menth Michael. Wildcard compression of inter-domain routing tables for OpenFlow-based software-defined networking. In Proc. of the Third European Workshop on Software Defined Networks, 2014, pp. 25–30.

[5] Rottenstreich Ori and Tapolcai János. Optimal rule caching and lossy compression for longest prefix matching. IEEE/ACM Transactions on Networking, vol. 25, issue 2, 2016, pp. 864–878.

[6] Chang Yeim-Kuan and Chen Han-Chen. Fast packet classification using recursive endpoint-cutting and bucket compression on FPGA. Computer Journal, vol. 62, no. 2, 2019, pp. 198–214.

Information about authors / Информация об авторах

Nikita Igorevitch NIKIFOROV is a Master's student at the department of Computer Systems and Automation of the Faculty of Computational Mathematics and Cybernetics. Research interests include software-defined networks, data compression algorithms.

Никита Игоревич НИКИФОРОВ – студент магистратуры кафедры АСВК ф-та ВМК. Научные интересы включают программно-определяемые сети, алгоритмы сжатия данных.

Dmitry Yuryevitch VOLKANOV – candidate of physical and mathematical sciences, associate professor at the department of Computer Systems and Automation of the Faculty of Computational Mathematics and Cybernetics. Areas of research: analysis and design of network processing unit architecture.

Дмитрий Юрьевич ВОЛКАНОВ – кандидат физико-математических наук, доцент кафедры АСВК ф-та ВМК. Направления исследований: анализ и разработка архитектуры сетевого процессора.

DOI: 10.15514/ISPRAS-2021-33(4)-7

Оценка влияния различных неархитектурных изменений предсказательной модели на качество классификации ЭКГ

[1,2] *В.В. Ананьев, ORCID: 0000-0002-5070-8117 <novisp53@ispras.ru>*
[3] *С.Н. Скорик, ORCID: 0000-0002-8316-7302 <skorik@ispras.ru>*
[2] *В.В. Шаклеин, ORCID: 0000-0002-4239-0807 <shaklein@ispras.ru>*
[4] *А.А. Аветисян, ORCID: 0000-0002-7066-6954 <a.a.avetisyan@ispras.ru>*
[5,6,7] *Ю.Э. Терегулов, ORCID: 0000-0001-9120-142X <tereg2@mail.ru>*
[1,4] *Д.Ю. Турдаков, ORCID: 0000-0001-8745-0984 <turdakov@ispras.ru>*
[8] *В. Глинер, ORCID: 0000-0003-2900-3291 <vadim.gliner@gmail.com>*
[8] *А. Шустер, ORCID: 0000-0002-3311-6937 <assaf@technion.ac.il>*
[1] *Е.А. Карпулевич, ORCID: 0000-0002-6771-2163 <karpulevich@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Новгородский Государственный Университет им. Ярослава Мудрого, 173003, Россия, г. Великий Новгород, ул. Большая Санкт-Петербургская, д. 41*
[3] *Московский физико-технический институт, 141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*
[4] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1.*
[5] *Казанский государственный медицинский университет, 420012, Россия, Республика Татарстан, г. Казань, ул. Бутлерова, д. 49.*
[6] *Казанская государственная медицинская академия - филиал РМАНПО МЗ РФ, 420012, Россия, Республика Татарстан, г. Казань, ул. Бутлерова, д. 36.*
[7] *Республиканская клиническая больница МЗ РТ, 420064, Россия, Республика Татарстан, г. Казань, Оренбургский тракт, д. 138.*
[8] *Факультет компьютерных наук, Технион, 3200003, Израиль, г. Хайфа*

Аннотация. Запись и расшифровка электрокардиограммы в 12 отведениях является наиболее распространенной процедурой для определения сердечных заболеваний. В последнее время предлагаются различные методы машинного обучения для автоматической постановки диагноза по электрокардиограмме. Их задача – предоставить второе мнение для врача и помочь обнаружить патологию на ранней стадии. В статье рассматриваются методы улучшения качества автоматического определения патологий по ЭКГ: добавление метаданных пациента, уменьшение шума электрокардиограммы и самоадаптивное обучение. Также представлены результаты экспериментального исследования влияние различных ЭКГ отведений, значимости длины электрокардиограммы и объема обучающей выборки на результаты работы алгоритмов. Проведенные эксперименты показывают релевантность описываемых подходов, а также предлагают оптимальную оценку параметров входных данных.

Ключевые слова: классификация ЭКГ; сверточная нейронная сеть; глубокое обучение; шумоподавление; самоадаптивное обучение

Для цитирования: Ананьев В.В., Скорик С.Н., Шаклеин В.В., Аветисян А.А., Терегулов Ю.Э., Турдаков Д.Ю., Глинер В., Шустер А., Карпулевич Е.А. Оценка влияния различных неархитектурных изменений предсказательной модели на качество классификации ЭКГ. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 87-98. DOI: 10.15514/ISPRAS–2021–33(4)–7

Благодарности: Исследование выполнено при финансовой поддержке РФФИ и МНТИ в рамках научного проекта № 19-57-06004

Assessment of the impact of non-architectural changes in the predictive model on the quality of ECG classification

[1,2] *V.V. Ananev, ORCID:* 0000-*0002-5070-8117 <novisp53@ispras.ru>*
[3] *S.N. Skorik, ORCID: 0000-0002-8316-7302 <skorik@ispras.ru>*
[2] *V.V. Shaklein, ORCID: 0000-0002-4239-0807 <shaklein@ispras.ru>*
[4] *A.A. Avetisyan, ORCID: 0000-0002-7066-6954 <a.a.avetisyan@ispras.ru>*
[5,6,7] *Y.E Teregulov, ORCID: 0000-0001-9120-142X <tereg2@mail.ru>*
[1,4] *D.Yu. Turdakov, ORCID: 0000-0001-8745-0984 <turdakov@ispras.ru>*
[8] *V. Gliner, ORCID: 0000-0003-2900-3291 <vadim.gliner@gmail.com>*
[8] *A. Schuster, ORCID: 0000-0002-3311-6937 <assaf@technion.ac.il>*
[1] *E.A. Karpulevich, ORCID: 0000-0002-6771-2163 <karpulevich@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the RAS,*
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia
[2] *Yaroslav-the-Wise Novgorod State University,*
173003, Russia, Veliky Novgorod, st. Bolshaya St. Petersburg, 41
[3] *Moscow Institute of Physics and Technology,*
9, Institutskiy per., Dolgoprudniy, 141701, Russia
[4] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation
[5] *Kazan State Medical University,*
49, Butlerova Str., Kazan, Republic of Tatarstan, 420012, Russian Federation
[6] *Kazan State Medical Academy - Branch Campus of the RMACPE MOH Russia,*
36, Butlerova Str., Kazan, Republic of Tatarstan, 420012, Russian Federation
[7] *Republican Clinical Hospital of the Ministry of Health of the Republic of Tatarstan,*
138, Orenburgskiy trakt, Kazan, Republic of Tatarstan, 420064, Russian Federation
[8] *Computer Science Department, Technion-IIT,*
Haifa, 3200003, Israel

Abstract. Recording and analyzing 12-lead electrocardiograms is the most common procedure for detecting heart disease. Recently, various deep learning methods have been proposed for the automatic diagnosis by an electrocardiogram. The proposed methods can provide a second opinion for the doctor and help detect pathologies at an early stage. Various methods are proposed in the paper to improve the quality of prediction of ECG recording pathologies. Techniques include adding patient metadata, ECG noise reduction, and self-adaptive learning. The significance of data parameters in training a classification model is also explored. Among the considered parameters, the influence of various ECG leads, the length of the electrocardiogram and the volume of the training sample is studied. The experiments carried out show the relevance of the described approaches and offer an optimal estimate of the input data parameters.

Keywords: ECG Classification; Convolutional Neural Network; Deep Learning; Denoising; Self-Adaptive Learning

For citation: Ananev V.V., Skorik S.N., Shaklein V.V., Avetisyan A.A., Teregulov Y.E, Turdakov D.Y, Gliner V., Schuster A., Karpulevich E.A. Assessment of the impact of non-architectural changes in the predictive model on the quality of ECG classification. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 4, 2021. pp. 87-98 (in Russian). DOI: 10.15514/ISPRAS–2020–32(2)–7

Acknowledgments: The reported study was funded by RFBR and MOST according to the research project № 19-57-06004

1. Введение

Общепринятая процедура для диагностики сердечных патологий — электрокардиография в 12 отведениях. Снятая электрокардиограмма представляет собой кривые, которые отражают изменение биоэлектрических потенциалов, генерируемых мыш-цами работающего сердца, во времени. 12 отведений ЭКГ принято делить на две группы:

- 6 отведений от конечностей: 3 униполярных (I, II, III) и 3 биполярных (aVR, avL, aVF);
- 6 грудных отведений (V1, V2, V3, V4, V5, V6).

Несмотря на больше количество работ, посвященных построению эффективных алгоритмов автоматической классификации электрокардиограмм, эта проблема по-прежнему остается актуальной. Модели, основанные на вручную созданных методах извлечения признаков, могут не учитывать скрытые зависимости, необходимые для более точной классификации. В то же время модели глубоких нейронных сетей не требуют человеческих знаний о предметной области, в которой они применяются и могут быть обучены независимо от них, выделяя наиболее важные закономерности в сигналах для извлечения признаков или классификации. Глубокие нейронные сети были впервые использованы для классификации ЭКГ относительно недавно, но они сразу же показали многообещающие результаты, продемонстрировав качество распознавания некоторых патологий, сопоставимое с человеческим. Тем не менее, такие модели чувствительны к типу входного сигнала, качеству и количеству разметки, а также к различным метаданным.

В представленной статье мы рассматриваем неархитектурные способы улучшения качества классификации ЭКГ с использованием глубокой нейронной сети. В статье рассматриваются такие способы, как включение метаданных пациента, устранение шума и обучение самоадаптивной модели. Поставлены различные эксперименты, оценивающие важность различных параметров для обучения модели, таких как длина ЭКГ записи, количество и выбор отведений, объем обучающей выборки. Эксперименты, проведенные на полученных объемах данных, свидетельствуют о релевантности предложенных методов и важности выбора параметров обучения.

2. Обзор существующих решений

В этом разделе кратко описываются недавно появившиеся алгоритмы классификации ЭКГ.

Широкое распространение цифровых систем диагностики здоровья привело к накоплению больших массивов медицинских данных и, как следствие, к возможности улучшения существующих систем автоматической диагностики за счет использования нейронных сетей [1] [2].

Для решения задачи классификации электрокардиограмм были предложены различные архитектуры глубоких нейронных сетей. Например, в [3] рекуррентные нейронные сети использовались для распознавания аритмий. В работе сравнивались архитектуры на основе долгой краткосрочной памяти (LSTM) и управляемых рекуррентных блоков (GRU). В качестве входных данных использовался сигнал только одного отведения. Кроме того, авторы ограничились задачей бинарной классификации. Рекуррентная нейронная сеть на базе LSTM также использовалась в качестве алгоритма для извлечения глобальных признаков [4].

В [5] распознавание аритмий осуществлялось с помощью вероятностной нейронной сети (PNN). На вход ей подавался вектор признаков, полученный с помощью цифровой обработки сигнала, которая состояла из вейвлет-преобразования, детектирования R-пика, QRS-комплекса, а также P и T волн. В [6] классифицировались только три типа сердечных ритмов и для них использовалась нейронная сеть с многослойным персептроном (MLP NN). На вход

ей также подавался вектор признаков, однако помимо обработки сигналов и извлечения признаков в работе также использовался независимый компонентный анализ (ICA) для понижения размерности. Вектор признаков подавался и на вход двойной полносвязной нейронной сети [7]. Применение такой архитектуры было обусловлено борьбой с несбалансированностью четырех классов. Они были разделены на две группы и первая нейронная сеть училась определять сигнал в одну из этих групп. Вторая нейронная сеть использовалась для классификации сигнала внутри первой группы. Внутри второй группы классификация проводилась с помощью оценки RR-интервала.

Проблема несбалансированности классов является типичной для электрокардиографии и вызывает трудности, связанные с обучением нейронных сетей. Одним из способов преодоления дисбаланса являются классические методы увеличения данных, таких как сдвиги, повороты изображения вдоль горизонтальной или вертикальной осей, вращение, которые стали важным шагом в задачах компьютерного зрения [8]. Однако такие преобразования могут нарушить зависимость между некоторыми признаками в сигналах, что отрицательно скажется на качестве этих данных. Алгоритм SMOTE (Synthetic Minority Oversampling Technique) также является методом увеличения количества примеров в наименее распространенных классах, однако в отличие от классических приемов аугментации, SMOTE создает «близкие» к исходным примеры, которые не являются преобразованием какого-то одного сигнала и не нарушают зависимость между признаками. Такие примеры являются промежуточными, между двумя соседними в наборе данных. Этот алгоритм использовался, например, в [9] и [10]. Тем не менее, алгоритм SMOTE никак не учитывает плотность распределения менее распространенных классов. Если они равномерно с низкой плотностью распределены в мажоритарных классах, то SMOTE нарушит это распределение.

Впрочем, существуют генеративные модели, которые учатся восстанавливать такое распределение и по нему синтезировать новые примеры. Одним из видов генеративных моделей являются генеративно-состязательные сети (GAN), которые показали свою эффективность при синтезе изображений в нескольких областях [11]. Так в работе [12] показали, что сигналы, синтезированные с помощью генеративно-состязательных сетей вносят больший вклад в качество классификации, чем другие распространенные методы увеличения данных.

Наиболее часто используемой для классификации ЭКГ архитектурой глубокой нейронной сети является сверточная нейронная сеть (CNN), так как CNN действуют одновременно как алгоритм извлечения признаков и как классификатор. Для обнаружения фибрилляции предсердий Ся (Yong Xia) и др. [13] используют двумерные сверточные сети для классификации электрокардиограмм по спектрограммам, полученным с по-мощью оконного преобразования Фурье или дискретного вейвлет-преобразования. Глинер (Vadim Gliner) и др. [14] предложили две разные архитектуры CNN, одну, обученную с использованием цифровых сигналов, а другую с помощью изображений. Эти методы показали высокие результаты в выявлении фибрилляции предсердий (AF).

В [15] и [16] CNN применялись непосредственно к сигналу ЭКГ. Наборы данных, используемые в этих статьях, были значительно больше, чем во всех ранее упомянутых наборах, поскольку они не являются общедоступными. В рамках исследования [15] был получен набор записей ЭКГ в одном отведении. Собранные данные были аннотированы сертифицированными кардиологами. Затем, одномерная CNN с остаточными блоками была обучена предсказывать тип ритма каждые 1.21 секунды на основе соответствующего сигнала ЭКГ без предварительной обработки.

Рибейро (Antônio H. Ribeiro) и др. [16] использовали данные системы теледиагностического мониторинга ЭКГ. Использованная выборка состояла из 2 470 424 записей ЭКГ в 12 отведениях, что соответствует 1 676 384 пациентам. Как и в [15], модель представляет собой

одномерную CNN с остаточными блоками. Обученная в этом исследовании модель достигла более высокого качества классификации патологий, чем человек, для некоторых из рассматриваемых патологий.

В работе [17] проводится ряд результатов сравнительного анализа недавно опубликованного набора данных PTB-XL, охватывающих различные задачи машинного обучения для ЭКГ. Важно отметить, что авторы коснулись сравнения различных архитектур нейронных сетей для задачи классификации ЭКГ и предоставили исходный код для их тестирования в общий доступ. Текущая работа построена на их реализации архитектуры нейронной сети resnet1d50.

3. Методология

3.1 Модель

Для классификации ЭКГ использовалась сверточная нейронная сеть с остаточными блоками resnet1d50, адаптированная для работы с одномерными данными (сигнал ЭКГ в исходном виде). Модель была разработана с использованием фреймворка Pytorch, на основе реализации, представленной в [17]. Архитектура модели представлена на рис. 1.

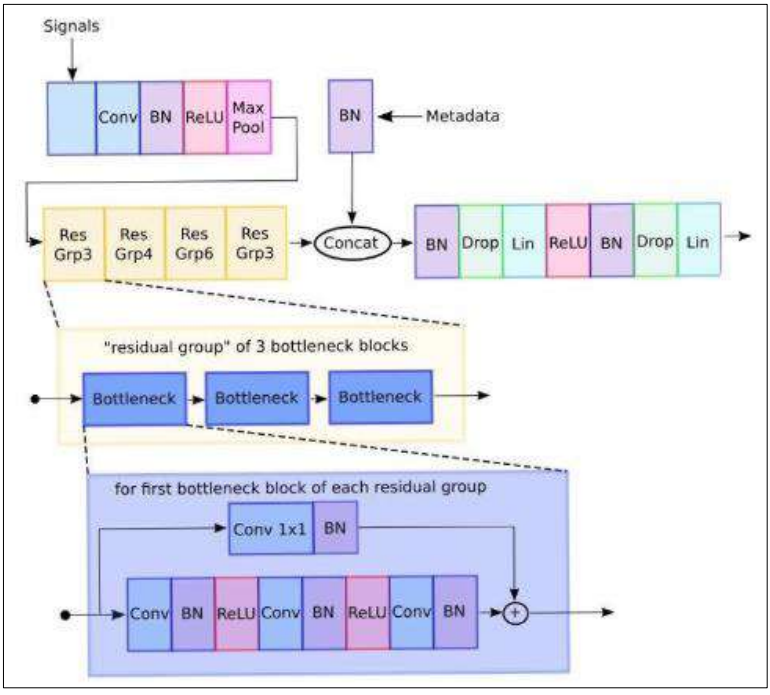


Рис. 1. Архитектура нейронной сети. Conv - операция одномерной свертки, BN - батч-нормализация, ReLU - функция нелинейности, Drop - операция исключения, MaxPool - объединение по функции максимума, Lin - полносвязный линейный слой, Concat - конкатенация векторов

Fig. 1. The neural network architecture. Conv - unidimensional convolution operation, BN - batch normalization, ReLU - nonlinearity function, Drop - dropout operation, MaxPool - pooling by the maximum function, Lin - fully connected linear layer, Concat - vector concatenation, Sigm - element-wise sigmoid function

В качестве входных данных для модели выступают ЭКГ в 12 отведениях, представленные в виде матрицы 10000x12, а также метаданные (возраст пациента в виде натурального числа от 18 до 96). 10000 - это количество измерений, которое соответствует 10 секундам записи при частоте дискретизации 1000 Гц. В наборе данных также были представлены данные с

частотой ниже 1000 Гц. Для таких данных была проведена процедура передискретизации к частоте 1000 Гц. Чтобы уменьшить влияние некорректно размеченных образцов ЭКГ на процесс обучения, была реализована недавно предложенная техника самоадаптивного обучения [18]. Обученная модель позволяет предсказывать наличие на ЭКГ патологий следующих классов: желудочковые экстрасистолы (PVC), синусовая брадикардия (SBRAD), синусовая тахикардия (STACH), блокада левой ножки пучка Гиса (LBBB), блокада правой ножки пучка Гиса (RBBB), фибрилляция предсердий (AFIB). Выходные данные модели представлены в виде набора значений в интервале (0, 1), где каждое значение показывает вероятность присутствия соответствующей патологии на рассматриваемой ЭКГ.

3.2 Данные

Набор данных, содержащий записи ЭКГ в 12 отведениях, используемых в этой работе, был получен от ООО «Телемедицинские информационные системы». Данные были собраны с разных географических точек Татарстана с помощью телемедицинской системы и размечены более 200 врачами Татарстана в режиме обследования. Полезной особенностью системы сбора данных является стандартизованная классификация патологий с применением дерева патологий, что облегчает работу с данными. Распределение ЭКГ по исследуемым патологиям (п. 3.1) представлено на рис. 2.

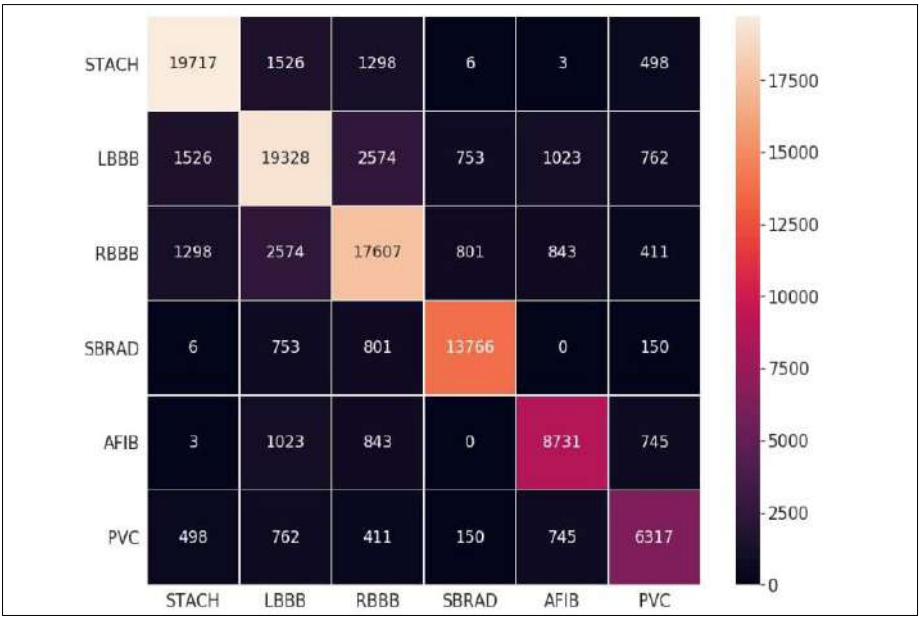


Рис. 2. Распределение образцов ЭКГ по классам. Число на пересечение i-й строки и j-го столбца соответствует количеству записей, в которых одновременно представлены i-я и j-я патологии, i-е число на диагонали обозначает общее количество записей, соответствующих i-ой патологии

Fig. 2. Distribution of ECG samples by class. The number at the intersection of the i-th row and j-th column corresponds to the number of records in which the i-th and j-th pathologies are simultaneously represented, the i-th number on the diagonal indicates the total number of records corresponding to the i-th pathology

Длина каждой записи в наборе варьируется от 4 до 69 секунд. Для экспериментов были отобраны сигналы длиной 10 секунд, а также более длительные записи, которые нарезались на 10 секундные отрезки. В результате было получено 74931 анонимизированных записей ЭКГ, соответствующих пациентам в возрасте от 18 лет. Распределение записей по возрасту пациентов приведено на рис. 3.

На многих записях присутствуют помехи в виде «блуждающей изолинии». Для того чтобы скорректировать изолинию, каждая запись была предварительно обработана методом локально-взвешенного сглаживания LOWESS.

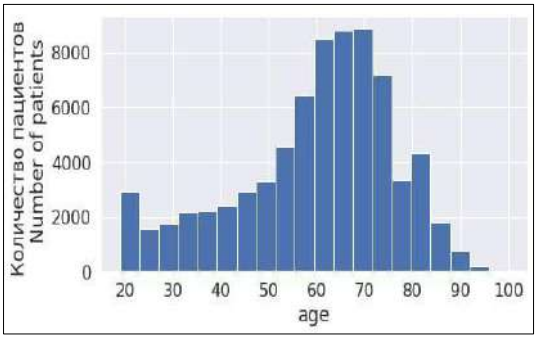


Рис. 3. Распределение записей ЭКГ по возрасту пациентов
Fig. 3. Distribution of ECG records by age of patients

4. Эксперименты

В разделе описаны эксперименты, оценивающие влияние различных неархитектурных изменений модели на качество предсказания. В поставленных экспериментах изучается зависимость качества классификации от предобработки сигнала, количества отведений, длины записи, объема обучающей выборки и использования техники самоадаптивного обучения. В большинстве представленных экспериментов была проведена 5-кратная перекрестная проверка. В качестве показателей эффективности классификации были выбраны: площадь под ROC-кривой (ROC-AUC), средняя точность (AP) и F-мера (F1-score).

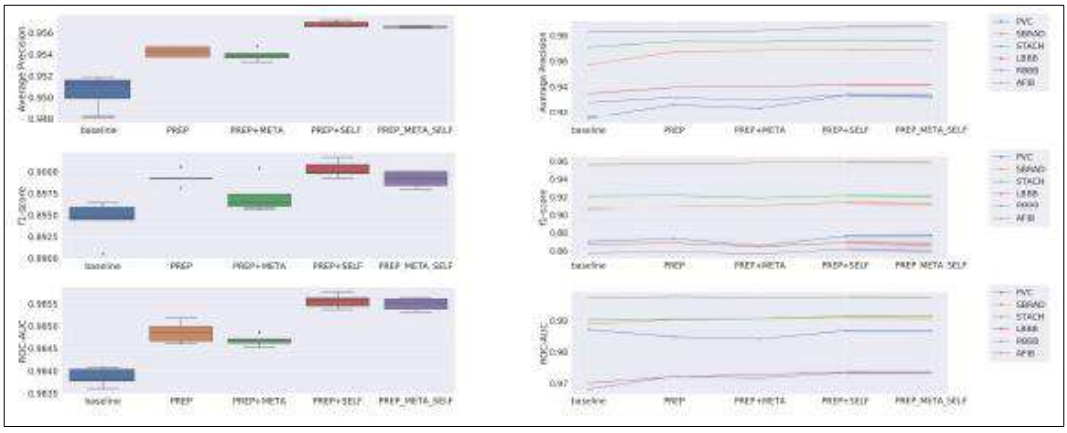


Рис. 4. Зависимость показателей метрик качества от методологии, используемой для классификации ЭКГ. (a) - усредненные по классам; (b) - для каждого класса. baseline - исходная модель (resnet1d50) с необработанными сигналами в качестве входных данных, PREP - предварительная обработка сигналов (LOWESS), MET - включение метаданных в модель, SELFAD - техника самоадаптивного обучения
Fig. 4. Dependence of quality metrics indicators on the methodology used for ECG classification. (a) - class averaged; (b) - for each class. baseline - original model (resnet1d50) with raw signals as input, PREP - signal preprocessing (LOWESS), MET - including metadata in the model, SELFAD - self-adaptive learning technique

4.1 Зависимость качества классификации от методов предобработки данных

В процессе изучения подходов к улучшению качества классификации, которые не затрагивают архитектуру модели, был проведен эксперимент, в котором сравнивались модели с различными входными данными и разметкой. Для эксперимента были отобраны 5 моделей:

- модель resnet1d50 с необработанными ЭКГ в качестве входных данных;
- модель с предварительно обработанными ЭКГ (метод LOWESS);
- модель с предварительно обработанными ЭКГ + метаданные (возраст пациента);
- модель с предварительно обработанными ЭКГ + техника самоадаптивного обучения;
- модель с предварительно обработанными ЭКГ + метаданные + техника самоадаптивного обучения.

Положительная динамика изменения метрик качества (рис. 4) свидетельствует о том, что использование предложенных методов позволяет повысить качество классификации. Эксперименты показали, что предобработка ЭКГ сигнала и техника самоадаптивного обучения модели улучшают качество работы модели. При этом выяснилось, что добавление возраста пациента не дает существенного прироста качества для рассматриваемых патологий (см. 3.1).

4.2 Зависимость качества классификации от размеров обучающей выборки

Важную роль в машинном обучении играет размер обучающей выборки. Для изучения его влияния был поставлен эксперимент, в котором модель PREP обучалась на различных объемах данных. Чтобы более объективно отразить зависимость качества классификации от размера выборки, вместо использования 5-кратной перекрестной проверки, набор данных был поделен на непересекающиеся множества тренировочной, валидационной и тестовой выборки. Далее из тренировочной выборки отбиралась некоторая часть, на которой производилось обучение модели.

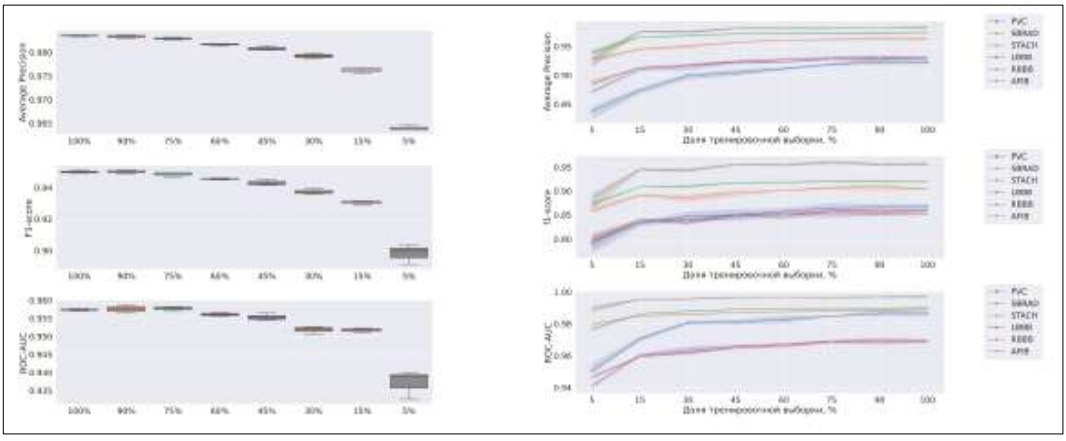


Рис. 5. Зависимость метрик качества от размера использованной обучающей выборки выраженного в долях от 75 тысяч образцов (максимально доступный размер обучающей выборки на кросс-валидации). (a) - усредненные по классам; (b) - для каждого класса

Fig. 5. Dependence of quality metrics on the size of the used training sample expressed in fractions of 75 thousand samples (the maximum available size of the training sample for cross-validation). (a) - class averaged; (b) - for each class

По графикам (рис. 5) видно, что качество классификации снижается при умень-шением размера обучающей выборки, но данная зависимость имеет нелинейный характер. Отсюда можно сделать следующие выводы.

- Дополнение тренировочной выборки новыми образцами потенциально пр-ведет к дальнейшему улучшению качества классификации некоторых патологий(PVC, LBBB, RBBB, SBRAD), в то время как для других патологий(STACH, AFIB) при использовании размера обучающей более 45% наблюдается ситуация с выходом на плато.
- Для более быстрого подбора гиперпараметров можно использовать 45% от объема имеющейся тренировочной выборки.

4.3 Зависимость качества классификации от длины входного сигнала

Еще одним фактором, влияющим на качество классификации, является длина входного сигнала, используемого при тренировке. В связи с этим был проведен эксперимент с варьированием длины входного сигнала ЭКГ при обучении модели. В рамках эксперимента длина сигнала варьировалась в соответствии с сеткой [2, 4, 6, 8, 10] секунд. Стоит отметить, что длина сигналов в тестовой выборке при этом оставалась равной 10 секундам, что позволяет проводить прямое сравнение результатов с другими экспериментами.

По результатам эксперимента (см. рис. 6) можно сделать вывод, что среди патологий из п. 3.1 существенную зависимость от длины сигнала на этапе тренировки имеет только PVC. Качество классификации остальных патологий выходит на плато при тренировке на сигналах длиной более 4 секунд.

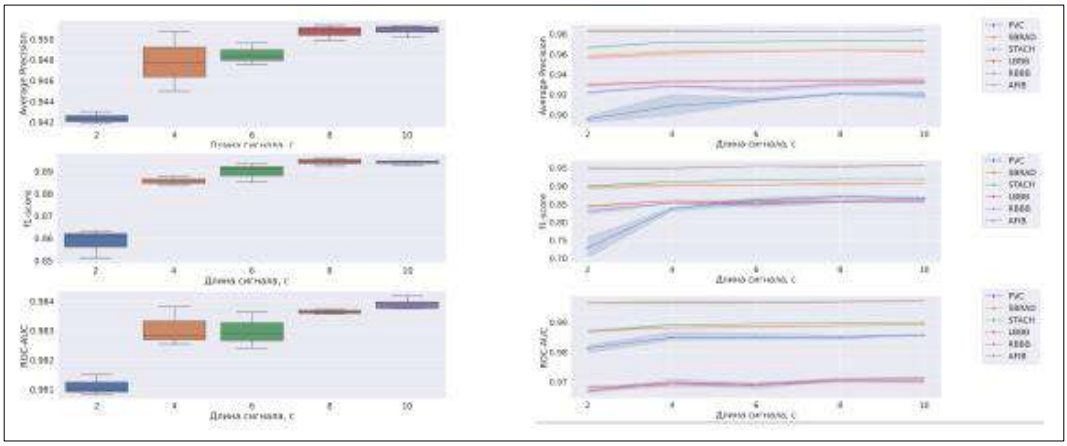


Рис. 6. Зависимость метрик качества от длины входного сигнала. (a) - усредненные по классам; (b) - для каждого класса

Fig. 6. Dependence of quality metrics on the length of the input signal. (a) - class averaged; (b) - for each class

4.4 Зависимость качества классификации от набора отведений

В рамках представленного исследования было изучено влияние различных наборов отведений на качество предсказания выбранных патологий. Оценка характера зависимости качества классификации от количества отведений на ЭКГ проводилась по методологии схожей с методологией, использованной в 4.1. В эксперименте рассматривается 5 наборов отведений:

- I, II;
- I, II, V1;
- V1, V2, V3, V4, V5, V6;
- I, II, III, aVL, aVR, aVF, V1;

- I, II, V1, V2, V3, V4, V5, V6;

Результаты эксперимента (рис. 7) показывают, что при использовании комбинаций 'three_lead' и 'eight_lead' достигается наибольшее значение метрик качества. Модели 'two_lead' и 'six_lead', которые предсказывают патологии только по отведениям на конечностях и грудным отведениям соответственно, уступают по качеству другим представленным моделям. Для более высокого качества классификации ЭКГ по выбранным патологиям необходимо изучать и грудные отведения, и отведения на конечностях. Использование комбинации 'three_lead' для рассматриваемых патологий позволяет существенно сократить объем входных данных, и, как следствие, ускорить процесс обучения модели, без существенных потерь в качестве.

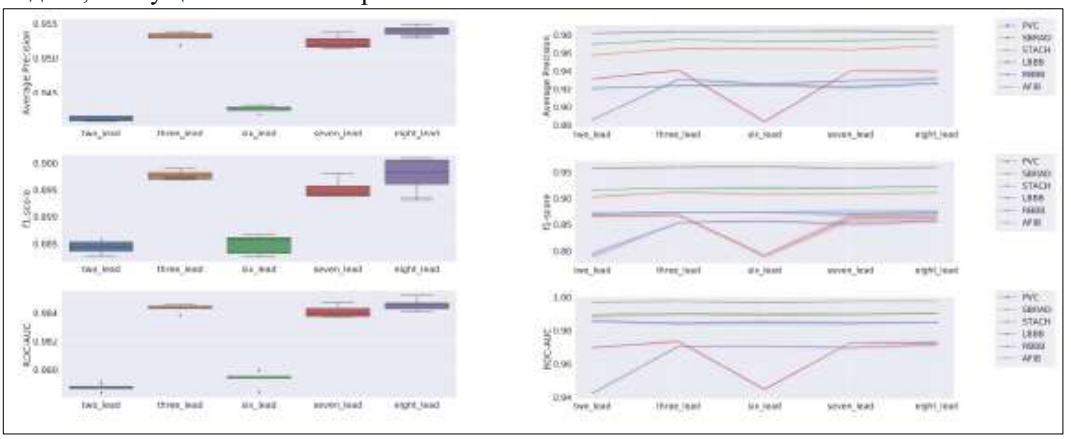


Рис. 7: Зависимость метрик качества от использованного набора отведений. (a) - усредненные по классам; (b) - для каждого класса. 'two_lead' - отведения I, II; 'three_lead' - отведения I, II, V1; 'six_lead' - отведения V1, V2, V3, V4, V5, V6; 'seven_lead' - отведения I, II, III, aVL, aVR, aVF, V1; 'eight_lead' - отведения I, II, V1, V2, V3, V4, V5, V6

Fig. 7: Dependence of quality metrics on the set of leads used. (a) - class averaged; (b) - for each class. 'Two_lead' - leads I, II; 'Three_lead' - leads I, II, V1; 'Six_lead' - leads V1, V2, V3, V4, V5, V6; 'Seven_lead' - leads I, II, III, aVL, aVR, aVF, V1; 'Eight_lead' - Leads I, II, V1, V2, V3, V4, V5, V6

5. Заключение

В представленной статье были предложены методы улучшения классификации ЭКГ по патологиям, не меняющие архитектуру модели. Кроме того, было оценено влияние формата данных обучаемой модели на качество предсказания. Эксперименты, проведенные на большом наборе данных, доказали релевантность предложенных методов и предложили оптимальные значения параметров входных данных, таких как длина ЭКГ записи, выбор отведений для предсказания и объем обучающей выборки. Предложенные оценки формата данных и неизменяемые архитектуру подходы, которые улучшают качество классификации, дают возможность применять их в других глубоких нейронных сетях.

Список литературы / References

[1] V. Gulshan, L. Peng et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. Jama, vol. 316, issue 22, 2016, pp. 2402-2410.

[2] A. Esteva, B. Kuprel et al. Dermatologist-level classification of skin cancer with deep neural networks. Nature, vol. 542, issue 7639, 2017, pp. 115-118.

[3] S. Singh, S.K. Pandey et al. Classification of ecg arrhythmia using recurrent neural networks. Procedia computer science, vol. 132, 2018, pp. 1290-1297.

[4] T. Teijeiro, C. A. Garcıa et al. Arrhythmia classification from the abductive interpretation of short single-lead ecg records. In Proc. of the 2017 Computing in Cardiology (CinC), 2017, pp. 1-4.

[5] J.A. Gutiérrez-Gnecchi, R. Morfin-Magaña et al. Dsp-based arrhythmia classification using wavelet transform and probabilistic neural network. Biomedical Signal Processing and Control, vol. 32, 2017, pp. 44-56.
[6] M. Ramkumar, C.G. Babu et al. ECG cardiac arrhythmias classification using DWT, ICA and MLP neural networks. Journal of Physics: Conference Series, vol. 1831, 2021, article no. 012015.
[7] H. Wang, H. Shi et al. A high-precision arrhythmia classification method based on dual fully connected neural network. Biomedical Signal Processing and Control, vol. 58, 2020, article no. 101874.
[8] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, vol. 25, 2012, pp. 1097-1105.
[9] S. Mousavi и F. Afghah. Inter-and intra-patient ecg heartbeat classification for arrhythmia detection: a sequence to sequence deep learning approach. In Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019, pp. 1308-1312.
[10] K. N. Rajesh и R. Dhuli. Classification of imbalanced ecg beats using re-sampling techniques and adaboost ensemble classifier. Biomedical Signal Processing and Control, vol. 41, 2018, 242-254.
[11] I. Goodfellow, J. Pouget-Abadie et al. Generative adversarial nets. Advances in neural information processing systems, vol. 27, 2014, pp. 2672-2680.
[12] A.M. Shaker, M. Tantawi et al. Generalization of convolutional neural networks for ecg classification using generative adversarial networks. IEEE Access, vol. 8, 2020, pp. 35592-35605.
[13] Y. Xia, N. Wulan et al. Detecting atrial fibrillation by deep convolutional neural networks. Computers in biology and medicine, vol. 93, 2018, pp. 84-92.
[14] V. Gliner, N. Keidar et al. Automatic classification of healthy and disease conditions from images or digital standard 12-lead electrocardiograms. Scientific Reports, vol. 10, issue 1, 2020, pp. 1-12.
[15] A.Y. Hannun, P. Rajpurkar et al. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. Nature medicine, vol. 25, issue 1, 2019, pp. 65-69.
[16] A.H. Ribeiro, M.H. Ribeiro et al. Automatic diagnosis of the 12-lead ecg using a deep neural network. Nature communications, vol. 11, issue 1, pp. 1-9.
[17] N. Strodthoff, P. Wagner, et al. Deep learning for ecg analysis: benchmarks and insights from ptb-xl. IEEE Journal of Biomedical and Health Informatics, vol. 25, issue 5, 2020, pp. 1519-1528.
[18] L. Huang, C. Zhang и H. Zhang. Self-adaptive training: beyond empirical risk minimization. Advances in neural information processing systems, vol. 33, 2020, pp. 19365-19376.

Информация об авторах / Information about authors

Владислав Валерьевич АНАНЬЕВ является выпускником и ассистентом кафедры информационных технологий и систем НовГУ, сотрудник ИСП РАН. Сфера научных интересов: анализ и разметка данных из различных сфер деятельности, глубокое обучение, компьютерное зрение и обработка изображений.

Vladislav Valerievich ANANEV is a graduate of the magistracy and assistant of the Department of Information Technologies and Systems, Novgorod State University, an employee of ISP RAS. Area of research interests: data labeling and analysis for various fields of activity, deep learning, computer vision and image processing.

Сергей Николаевич СКОРИК, студент бакалавриата МФТИ. Научные интересы: машинное обучение, методы оптимизации.

Sergej Nikolaevich SKORIK undergraduate student of Moscow Institute of Physics and Technology. Research interests: machine learning, optimization methods.

Всеволод Владиславович ШАКЛЕИН, студент бакалавриата НОВГУ. Научные интересы: машинное обучение, анализ данных.

Vsevolod Vladislavovich SHAKLEIN, undergraduate student of Novgorod State University. Research interests: machine learning, data analysis.

Арам Арутюнович АВЕТИСЯН, студент магистратуры факультета ВМК МГУ. Научные интересы: сбор данных, анализ информационных потоков в сети Интернет.

Aram Arutyunovich AVETISYAN, graduate student of the faculty of CMC at Moscow State University. Research interests: data collection, analysis of information flows on the Internet.

Юрий Эмильевич ТЕРЕГУЛОВ — доктор медицинских наук, доцент, заведующий кафедрой функциональной диагностики, доцент кафедры госпитальной терапии, заведующий отделением функциональной диагностики. Сферой научных интересов является изучение эндотелиальной дисфункции, гемодинамических особенностей артериальных гипертензий, оценка жесткости артериальной системы на основе математической модели сердечно сосудистой системы.

Yurij Emilevich TEREGULOV — D. Med. Sc., Associate Professor, Head of the Department of Functional Diagnostics, Associate Professor of the Department of Hospital Therapy. Research interests include endothelial dysfunction, hemodynamic features of arterial hypertension, assessment of the rigidity of the arterial system based on a mathematical model of the cardiovascular system

Денис Юрьевич ТУРДАКОВ – к.ф.-м.н., заведующий отделом «Информационные системы» ИСП РАН, доцент МГУ. Сфера научных интересов: машинное обучение, интеллектуальный анализ данных, извлечение информации, обработка естественного языка, сложные сети, анализ социальных сетей, большие данные.

Denis Yuryevich TURDAKOV, Ph.D. in Physics and Mathematics, Head of the Information Systems Department at ISP RAS, Associate Professor of the System Programming Department of Moscow State University. Research interests: natural language processing, machine learning, data mining, social network analysis, distributed data processing.

Вадим ГЛИНЕР, Ph.D., руководитель группы. Исследовательское интересы включают разработку и внедрение алгоритмов (комплексная цифровая обработка биосигналов, электромагнитное моделирование, алгоритмы локализации\навигации, алгоритмы искусственного интеллекта и машинного обучения), электромагнетизм и электрооптику.

Vadim GLINER, Ph.D., team leader. Research interests include algorithms development and implementation (complex digital signal processing of bio-signals and others, electromagnetic simulations, localization\navigation algorithms, artificial intelligence and machine learning algorithms), Hardware Engineering, & Physics (electromagnetism and electro-optics).

Ассаф ШУСТЕР, Ph.D., профессор. Исследовательские интересы Машинное и глубокое обучение, большие данные, кибербезопасность, параллельные и распределенные вычисления.

Assaf SCHUSTER, Ph.D., Professor. Research interests: Machine and Deep Learning, Big Data, Cyber Security, Parallel and Distributed Computing,

Евгений Андреевич КАРПУЛЕВИЧ является специалистом отдела «Информацион-ные системы». Сфера научных интересов: применение алгоритмов анализа данных к биомедицинскому домену, разработку систем распределенного хранения и анализа данных.

Evgeny Andreevich KARPULEVICH is a specialist of the Information Systems Department. Research interests: application of data analysis algorithms to the biomedical domain, development of systems for distributed data storage and analysis.

DOI: 10.15514/ISPRAS-2021-33(4)-8

Синтаксический анализ текстов предметной области при помощи онтологии

[1] *Б.И. Гельцер, ORCID: 0000-0002-9250-557X <boris.geltser@vvsu.ru>*
[2] *Т.А. Горбач, ORCID: 0000-0003-4380-6517 <tagorbachdv@gmail.com>*
[2] *В.В. Грибова, ORCID: 0000-0001-9393-351X <gribova@iacp.dvo.ru>*
[3] *О.В. Карпик, ORCID: 0000-0002-0477-1502 <parlak@mail.ru>*
[4] *Э.С. Клышинский, ORCID: 0000-0002-4020-488X <eklyshinsky@hse.ru>*
[4] *Н.А. Кочеткова, ORCID: 0000-0002-5346-0081 <nkochetkova@hse.ru>*
[2] *Д.Б. Окунь, ORCID: 0000-0002-6300-846X <okdm@iacp.dvo.ru>*
[2] *М.В. Петряева, ORCID: 0000-0002-1693-4508 <margaret@iacp.dvo.ru>*
[5] *К.И. Шахгельдян, ORCID: 0000-0002-4539-685X <carina.shahgeldyan@vvsu.ru>*

[1] *Дальневосточный Федеральный университет*
690922, Владивосток, о. Русский, п. Аякс, 10
[2] *Институт автоматики и процессов управления ДВО РАН*
690041, Владивосток, ул. Радио, д. 5
[3] *Институт прикладной математики им. М.В. Келдыша РАН*
125047, Москва, Миусская пл., д. 4
[4] *Национальный исследовательский университет «Высшая школа экономики»*
105066, Ст. Басманная ул., д.21/4, стр. 1
[5] *Владивостокский государственный университет экономики и сервиса*
690014, Владивосток, ул. Гоголя, д.41

Аннотация. В работе проводится сравнение трех методов синтаксического анализа текстов жалоб пациентов, извлеченных из электронных медицинских карт. В качестве контрольного теста используются существующие библиотеки синтаксического анализа текста. В качестве альтернативы предлагается использование онтологии для исправления ошибок, допущенных синтаксическим анализатором, либо полное формирование синтаксических зависимостей по данным, хранимым в онтологии. В статье показано что ограниченный набор правил, описывающих управление падежами зависимых слов, может показывать точность, сопоставимую с точностью современных синтаксических анализаторов, основанных на нейронных сетях.

Ключевые слова: синтаксический анализ; поверхностно-синтаксический анализ; онтологии; медицинские тексты

Для цитирования: Гельцер Б.И., Горбач Т.А., Грибова В.В., Карпик О.В., Клышинский Э.С., Кочеткова Н.А., Окунь Д.Б., Петряева М.В., Шахгельдян К.И. Синтаксический анализ текстов предметной области при помощи онтологии. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 99-116. DOI: 10.15514/ISPRAS–2021–33(4)–8

Благодарности. Данная работа поддержана грантом РФФИ 18-29-03131.

Ontology-based syntactic analysis of domain-specific texts

[1] *B.I. Geltser, ORCID: 0000-0002-9250-557X <boris.geltser@vvsu.ru>*
[2] *T.A. Gorbach, ORCID: 0000-0003-4380-6517 <tagorbachdv@gmail.com>*
[2] *V.V. Gribova, ORCID: 0000-0001-9393-351X <gribova@iacp.dvo.ru>*
[3] *O.V. Karpik, ORCID: 0000-0002-0477-1502 <parlak@mail.ru>*
[4] *E.S. Klyshinskiy, ORCID: 0000-0002-4020-488X <eklyshinsky@hse.ru>*
[4] *N.A. Kochetkova, ORCID: 0000-0002-5346-0081 <nkochetkova@hse.ru>*
[2] *D.B. Okun, ORCID: 0000-0002-6300-846X <okdm@iacp.dvo.ru>*
[2] *M.V. Petryaeva, ORCID: 0000-0002-1693-4508 <margaret@iacp.dvo.ru>*
[5] *K.I. Shakhgeldyan, ORCID: 0000-0002-4539-685X <carina.shahgeldyan@vvsu.ru>*

[1] *Far Eastern Federal University*
10 Ajax Bay, Russky Island, Vladivostok, Russia, 690922
[2] *Institute of Automation and Control Processes, Far Eastern Branch of RAS*
5 Radio st., Vladivostok, Russia, 690041,
[3] *Keldysh Institute of Applied Mathematics*
4 Miusskaya square, Moscow, Russia, 125047,
[4] *HSE University*
21/4 building 1, Staraya Basmannaya st., Moscow, Russia, 105066,
[5] *Vladivostok State University of Economics and Service*
41 Gogolya st., Vladivostok, Russia, 690014

Abstract. The paper compares three methods for parsing of patients' chief complaints extracted from electronic medical cards. We propose two methods which are based on usage of an ontology: either as a method for correction of mistake made by a parser, or for constructing syntactical dependencies according to this ontology and a limited set of rules of syntactical governance. As a control test, we use existing natural text parsing libraries. The paper demonstrates that such a simple approach could achieve a high accuracy, which is comparable to modern parsers.

Keywords: parsing; shallow parsing; ontology; medical texts

For citation: Geltser B.I., Gorbach T.A., Gribova V.V., Karpik O.V., Klyshinskiy E.S., Kochetkova N.A., Okun D.B., Petryaeva M.V., Shakhgeldyan K.I. Ontology-based syntactic analysis of domain-specific texts. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021, pp. 99-116 (in Russian). DOI: 10.15514/ISPRAS-2021-33(4)-8

Acknowledgments. This work was supported by the grant of RFBR no. 18-29-03131.

1. Введение

Медицинские информационные системы позволяют вывести общение доктора и пациента на новый уровень. Ситуация, когда информация об одном пациенте стекается из разных медицинских центров в одну точку, являющуюся сосредоточием самой полной информации об этом пациенте, удобна всем участникам. Становятся возможны консультации с врачами в других городах и странах без потери данных об истории болезни пациента. Одной из тем, активно развивающихся в рамках данного направления, является автоматическая обработка медицинских текстов, помогающая извлекать из медицинских карт пациентов информацию о течении заболевания. Полученные данные могут использоваться для решения различных задач: помощь в диагностике пациента по проявляемым симптомам и результатам обследования; поиск взаимосвязей между симптомами, диагнозом и прописанными лекарствами [1]; автоматизация оценки эффективности применения лекарств по результатам повторного обследования и т. д.

Одним из методов обработки медицинских текстов является извлечение фактов, суть которого состоит в выделении объектов и событий, а также взаимосвязей между ними.

Извлечение фактов, в свою очередь, базируется на извлечении сущностей (в том числе, именованных), которое помогает найти термины, названия лекарств, имена пациентов и врачей, единиц измерения и т.д. [2] Для нахождения связей между извлеченными сущностями используется синтаксический анализ, именно он помогает понять логику взаимодействия между извлеченными сущностями и суть производимых ими действий. Если отвлечься на другую предметную область, то недостаточно понять, что была произведена сделка между компаниями A, B и C, в которую были также вовлечены акции; необходимо понять, какая из компаний приобрела акции другой компании у третьей. Аналогичные проблемы ставятся и при анализе медицинских текстов. Например, требуется понять какая доза какого из лекарств была прописана при каком виде боли в какой именно части тела.

Обычно для синтаксического анализа применяют соответствующие программные библиотеки, точность работы которых достигает на реальных текстах порядка 95%. Такая точность достигается за счет использования нейронных сетей. Чаще используются ячейки LSTM [3], некоторые системы строятся по архитектуре трансформеров [4] с использованием уровня внимания [5].

Точность анализа медицинских текстов гораздо ниже. Это связано с тем, что медицинские тексты написаны весьма специфичным языком, с большим трудом поддающимся анализу: предложения могут строиться без применения глаголов; используемая терминология сложна, термины представляют собой длинные последовательности слов с большой глубиной подчинения; тексты содержат длинные перечисления подобных сложных терминов. В итоге, точность работы синтаксических анализаторов падает до 80-85%. Следствием низкой точности являются сложности в практическом применении методов анализа медицинских текстов. Извлекаемая из них информация служит входом для прочих этапов (диагностики, расчета статистики и т. д.). Получив на вход неточную информацию, эти этапы сами будут выдавать некорректные или неполные результаты, что вступает в противоречие высокой точностью анализа, который так важен в области медицины. Таким образом, требуется разработка новых методов синтаксического анализа медицинских текстов, которые позволили бы повысить его точность.

В данной статье мы хотели бы вернуться к старой идее из области автоматической обработки текстов, заключающейся в применении глубинной семантики в синтаксическом анализе. Текст содержит в себе некоторую семантику и логику повествования, которые выражаются в связях между словами. Аналогичные связи обычно закладываются в тезаурусы или онтологии. Следовательно, для синтаксического анализа текста может использоваться информация о семантических отношениях между терминами предметной области, описанных в онтологии этой предметной области. В чистом виде такой подход плохо применим на практике, так как для синтаксического анализа текста на любом языке необходимо иметь хотя бы некоторые представления о синтаксисе данного языка: преимущественном направлении связей между словами для разных конструкций и видов подчинения слов, согласовании слов в определенных конструкциях, глагольном и именном управлении слов и т. д. Особенности языков с развитой системой словоизменения, например, русского, польского или финского, требуют проведения морфологического анализа и снятия грамматической неоднозначности слов. Игнорируя эти особенности языка, мы не сможем получить качественную систему синтаксического анализа.

В данной работе мы предлагаем смешанную методику синтаксического анализа текста, опирающуюся на использование богатой онтологии предметной области и поверхностного синтаксического анализа (упрощенной и неполной его версии). Здесь мы используем тот факт, что два термина, связанные в тексте, должны быть связаны и в онтологии, а сам факт их связи будет выражен при помощи предлогов и грамматического согласования или управления. Мы утверждаем, что для проведения подобного анализа требуется короткий список правил и база информации об именном управлении зависимыми словами, однако сам метод подходит лишь для анализа коротких текстов узкой предметной области. Материалом для экспериментальной

части исследования послужили тексты жалоб пациентов. Тексты были записаны врачами и являлись частью истории болезни пациента.

2. Существующие решения

Прежде чем описать существующие решения в области анализа медицинских текстов, более внимательно рассмотрим сам этап синтаксического анализа. Целью данного этапа является построение дерева зависимостей, показывающего связи между словами в предложении. Так для фразы «Он видел их семью своими глазами» мы получим следующие связи: некто «он» производил действие «видеть», направленное на «семью», «семья» была «их», действие производилось при помощи инструмента «глаза», имеющего свойство «свои». Заметим, что фраза является неоднозначной, и альтернативный результат ее разбора будет следующим: действие направлено на «них» (видел их), действие производилось при помощи инструмента «глаза», количество глаз равно семи («семью глазами»), остальные связи совпадают. В обоих случаях семантическая информация об окружающем мире используется корректно («он» может производить действия, «семья» может являться объектом материального мира и ее можно увидеть, «семья» может относиться к кому-то, «глаза» могут кому-то принадлежать и быть в определенном количестве), равно как и правила языка (субъект действия предшествует глаголу, объект – следует за ним, свойства, выраженные прилагательным или местоимением, идут перед главным словом и согласовываются по грамматическим признакам). Если из полученных связей построить дерево, то мы получим структуру, называемую деревом зависимостей.

Как было сказано выше, современные системы синтаксического анализа строятся с использованием нейронных сетей. Заметим, что, применяя примерно одинаковый инструментарий, системы синтаксического анализа работают с разной эффективностью и скоростью. Альтернативой нейронным сетям являются поверхностный синтаксический анализ и сегментация [6, 9]. Они применяются, когда нет необходимости проводить полный синтаксический анализ всего предложения, а достаточно обнаружить границы фрагмента, например, при извлечении терминов, именованных сущностей или фактов. В такой ситуации может быть использована сегментация, которая даже не строит дерева зависимостей. Сегментация отличается более высокой скоростью работы, связанной с уменьшением числа правил или использованием более простых методов, например, скрытых Марковских моделей [7], условных случайных полей [8], контекстно-свободных грамматик или конечных автоматов [9]. В отличие от сегментации, поверхностный синтаксический анализ восстанавливает структуру зависимостей в фразе или предложении, используя сходные инструменты. Однако в его задачи не входит восстановление дерева зависимостей всего предложения или текста.

Применение поверхностно-синтаксического анализа позволяет сосредоточиться на определенных аспектах языка, не решая задачу анализа в общем случае. Это помогает упростить анализ конкретных языковых явлений, но требует написания или автоматической генерации набора правил. Такая работа занимает довольно продолжительное время и требует значительного ручного труда.

Как отмечалось выше, наша идея состоит в проведении синтаксического анализа с использованием знаний из онтологий. В целом, построение онтологий описано, например, в таких фундаментальных трудах, как [10, 11]. Обращаясь к онтологиям медицинской области, следует заметить, что успехи здесь достигли впечатляющих размеров: в связи с важностью и актуальностью предметной области, медицинские онтологии являются самыми проработанными среди всех. Самой большой из медицинских онтологий является Unified Medical Language System (UMLS) [12]. Она содержит в себе такие подсистемы, как Metathesaurus (иерархию понятий, собранных из различных словарей), Semantic Network (отношения между понятиями и категориями) и SPECIALIST Lexicon and Lexical Tools (большой словарь биомедицинских терминов и слов общего английского языка, используемый специально разработанными

инструментами для анализа текстов). Metathesaurus, называемый также MeSH (Medical Subject Headings), использовался, например, в системе MetaMap, предназначенной для извлечения данных из медицинских текстов [13]. Алгоритм MetaMap состоит из двух этапов: обработка медицинских текстов с извлечением фактов и уточнение извлеченных понятий. Первый этап содержит в себе такие стандартные шаги, как токенизация, синтаксический анализ, поиск сокращений и аббревиатур, поиск неоднословных терминов и проч. Результатом работы является размеченный медицинских текст, содержащий в себе ссылки на Metathesaurus. Сходный подход, основанный, однако, на поверхностном синтаксическом анализе, использован в работе [14]. Словарь Metathesaurus был переведен на 15 языков, считая русский [15]. Так система Exactus [16], использующая переведенную версию UMLS, проводит логический вывод относительно течения хронических заболеваний. Её алгоритм машинного обучения позволяет поднять точность извлечения фактов до 82% для определения тяжести заболевания и до 99% для течения заболевания.

Структура описанных выше онтологий предполагает объединение понятий в тематические группы или построение иерархии подобных групп, однако в них находит слабое отражение информация о связях между словами текста. В результате, такие онтологии хорошо подходят для выделения фактов или поиска сущностей, но плохо – для поиска связей между понятиями. В следующем разделе мы опишем структуру используемой нами онтологии, структура которой подходит для решения поставленных задач.

3. Используемые данные и инструменты

В нашем исследовании мы использовали «Базу медицинской терминологии и наблюдений» [17], которая обладает несколькими полезными для нас свойствами. Помимо описания отдельных понятий, данная онтология включает в себя три основных типа сущностей: признаки, характеристики и значения. Признаки объединяют характеристики в смысловые группы. Характеристики показывают текущий функциональный статус пациента и связаны с множеством принимаемых значений. Значения описывают течение заболевания и могут быть качественными, числовыми или интервальными. База медицинской терминологии и наблюдений имеет форму дерева, в котором значения являются листьями, подчиняющимися названию характеристики. Название характеристики подчиняется названию признака, все названия признаков подчинены общей вершине, описывающей жалобы пациентов. Например, *Боль в ноге → Локализация → правая нога*. Каждому термину может быть сопоставлен набор его синонимов или заменителей.

Онтология также содержит в себе другие разделы, например, раздел симптомов, в котором хранится информация о видах исследований. В данной работе мы использовали только секцию жалоб пациентов, описывающую субъективное мнение и ощущения пациентов. Этот раздел характеризует самочувствие пациентов и состояние систем организма: нервной, дыхательной, опорно-двигательной и др. Раздел жалоб содержит в себе подраздел «Общие жалобы», описывающие симптомы заболеваний: слабость, головокружение, тошнота, потливость и др., подраздел «Боли», включающий в себя головную боль, боль в спине, боль в шее, боль в горле и др., и другие подразделы. Для признаков определены такие характеристики как, например, «локализация», «причина», «частота», «время возникновения» и др.

Заметим, что среди прочих в онтологиях используется два вида связи: гипоним/гипероним и мероним/холоним. Гипоним выражает более частную сущность, чем данная. Гипероним, наоборот, показывает более общую сущность (то есть обратен гипониму). На некотором уровне абстракции можно сказать, что базовый класс является гиперонимом по отношению к наследуемому от него гипониму. Меронимы и холонимы отражают отношения часть-целое. Мероним является составной частью холонима (целое по отношению к мерониму). Так автомобиль является холонимом по отношению к меронимам двигатель, капот, руль и др.

Помимо этих двух видов отношений, вводится целый ряд других: функциональные, пространственные, временны́е, атрибутивные и др. [10] В используемой нами онтологии «База медицинской терминологии и наблюдений» используется несколько видов отношений. Так, например, связь между названием характеристики и его значением является атрибутивной (*Локализация → Правый глаз*), между признаком и характеристикой – функциональной (*Боль в глазу → Локализация*), между группой признаков и признаком – гиперонимической (*Боль → Боль в глазу*). Как это отмечалось выше, наша онтология представляет собой дерево (а не граф, как это бывает во многих онтологиях), содержащее в себе ссылки на другие ветви графа, используемые в разных местах. Так если локализация нескольких видов боли может совпадать, для них будет построено единственное поддерево, ссылки на которое будут размещены в соответствующих местах. Подобный подход помогает избегать дублирования информации.

Важным элементом наших экспериментов являлось использование программных библиотек синтаксического анализа, для которых предусмотрена возможность работы с русским языком: UDPipe версии 2.5 (выпущен в декабре 2019) [3] и spaCy 3.0 (выпущен в марте 2021) [18]. Оба синтаксических анализатора используют нейронные сети на основе LSTM. Языковая модель UDPipe была обучена только на русскоязычной части корпуса Universal Dependencies [19]. SpaCy обучалась на нескольких корпусах и обладает целым рядом моделей, лучше приспособленными для анализа текстов, написанных в разных стилях. Для своих экспериментов мы использовали модель «ru_core_news_sm», натренированную на новостной ленте и показывающую по отзывам создателей более точные результаты.

4. Алгоритмы синтаксического анализа медицинских текстов

В данной работе мы изложим алгоритмы работы трех методов разбора жалоб пациентов: синтаксический анализ при помощи стандарт ной библиотеки с коррекцией результатов при помощи данных из онтологии, поверхностно- синтаксический анализ на основании связей из онтологии и, в качестве контрольного метода, обычный синтаксический анализ без применения онтологии. В следующем разделе будет дана количественная оценка работы указанных методов.

4.1 Синтаксический анализ с коррекцией результатов при помощи онтологии

Как отмечалось выше, тексты жалоб пациентов пишутся весьма специфическим языком, анализ которого приводит к большому числу ошибок, связанных с соединением слов в дерево зависимостей. Причиной таких ошибок является некорректное взвешивание синтаксических связей анализатором, или следование наиболее вероятному решению, которое в данном конкретном тексте будет неверным. Неправильные синтаксические связи влекут ошибки в семантике, например, подчинение значения значению или характеристике из другой ветви онтологии. Итогом может быть дерево зависимостей, являющееся корректным с точки зрения синтаксиса (так как возможны несколько вариантов разбора данного текста), но некорректным с точки зрения семантики, так как подобные связи с ее точки зрения запрещены или противоречат связям онтологии.

Основная идея данного метода состоит в следующем. Требуется получить результаты синтаксического анализа текста, а потом откорректировать их в соответствии с иерархией терминов, хранимой в онтологии. Суть алгоритма состоит в последовательном перемещении вершин по дереву. Если есть связь между двумя терминами из разных ветвей онтологии, подчиненный термин должен быть перемещен выше по дереву с тем, чтобы найти там начало корректной ветви. Если мы видим некорректное подчинение терминов из одной ветви, следует исправить ситуацию хотя бы частично.

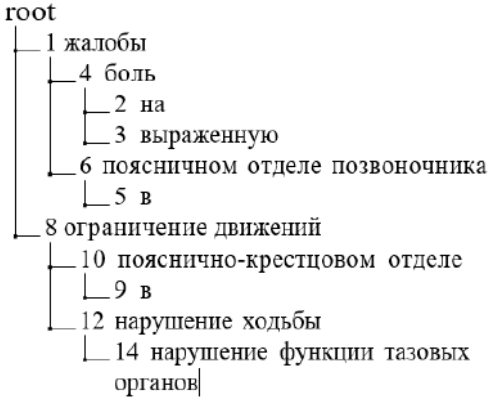
Предлагаемый алгоритм включает в себя следующие этапы: токенизация, морфологический анализ, извлечение терминов, синтаксический анализ, коррекция полученного дерева зависимостей. На этапе токенизации предложение разбивается на отдельные слова. Этап морфологического анализа приписывает этим словам грамматические характеристики (род, число и др.). Этап извлечения терминов анализирует последовательность токенов и выделяет из них непересекающиеся последовательности слов, являющиеся терминами. Далее все неоднословные термины будут анализироваться как единые элементы. В нашем случае нет необходимости решать проблему поиска новых терминов или снятия их многозначности, так как термином может являться только набор слов, присутствующий в онтологии. Все остальные слова и словосочетания рассматриваются как «заполнители», не имеющие отношения к предметной области. То есть здесь мы будем исходить из предположения о полноте онтологии предметной области, даже если это предположение неверно. Основными проблемами здесь являются возможное пересечение терминов, пропуск в них слов или написание слов с ошибкой. Под пересечением терминов мы понимаем ситуацию, когда начало и конец последовательности могут быть отнесены к разным терминам, а слова в середине относятся к ним обоим. В таком случае надо принять решение какой из терминов будет выделен из текста. Последними этапами алгоритма являются синтаксический анализ и коррекция дерева зависимостей по алгоритму, описанному ниже.

Здесь мы будем полагать, что два понятия из дерева зависимостей могут иметь общую дугу только случае, когда в онтологии есть прямой восходящий или нисходящий путь от одной вершины к другой. В противном случае дерево зависимостей должно быть исправлено при помощи следующих правил.

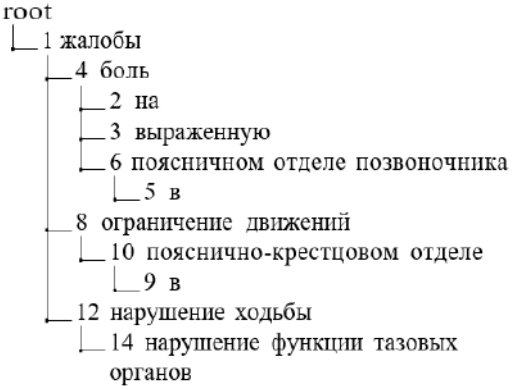
- Если родительская вершина в дереве зависимости имеет более низкий уровень в онтологии, чем дочерняя, следует поменять местами эти две вершины.
- Если в онтологии отсутствует прямой путь между двумя соединенными вершинами, следует подчинить дочернюю вершину следующему родителю, находящемуся на один уровень выше.
- Пусть две вершины дерева зависимостей подчинены одной родительской вершине, при этом одна из вершин является потомком другой вершины. В таком случае, первую вершину надо подчинить второй.

Эти правила применяются к дереву зависимо стей до тех пор, пока процесс не сойдется. После перестроения, дерево зависимостей должно соответствовать общей иерархической структуре онтологии, то есть родители и потомки должны относиться к одним и тем же ветвям онтологии и иметь корректный порядок подчинения. Часть вершин, которые отсутствуют в онтологии, должны оказаться на самом верхнем уровне дерева. Для остальных вершин факт связи между двумя вершинами будет означать, что в онтологии существует прямой путь вниз от родительской вершины к дочерней.

Рассмотрим пример работы алгоритма. Пусть дано следующее предложение: «*жалобы на выраженную боль в поясничном отделе позвоночника, ограничение движений в пояснично-крестцовом отделе, нарушение ходьбы, нарушение функции тазовых органов»*. После нахождения и объединения терминов мы получим следующую последовательность: «[***жалобы***] *на* [***выраженную***] [***боль***] *в* [***поясничном отделе позвоночника***], [***ограничение движений***] *в* [***пояснично-крестцовом отделе***], [***нарушение ходьбы***], [***нарушение функции тазовых органов***]». После синтаксического анализа будет получено следующее дерево зависимостей.



Вершины 1 и 8 находятся на одном уровне в дереве, но 8 является признаком в разделе 1. Следовательно, 8 необходимо подчинить 1. Вершина 6 является значением вершины 4, то есть первая должна быть подчинена второй. Вершина 12 является «братом» вершины 8, то есть должна быть поднята на один уровень вверх. После применения всех преобразований, дерево выглядит следующим образом.



4.2 Синтаксический анализ, основанный на применении онтологии

Второй рассматриваемый алгоритм предполагает, что онтология описывает все разрешенные связи между понятиями, за исключением некоторых связок, предписываемых синтаксисом. Итоговое дерево разбора должно полностью соответствовать связям в онтологии и может изначально строиться по ним. Для определения порядка следования слов и связей между ними используются правила поверхностного синтаксиса. Данный алгоритм использует некоторые предположения о структуре русского предложения, описывающего жалобы пациента. Сформулируем их в терминах нашей онтологии.

- Признак вводится до перечисления своих характеристик.
- Характеристика вводится до упоминания значений. Например, «*Жалобы на* ***боль в спине***[признак] *с* ***локализацией*** [характеристика] *в области* ***поясничного отдела позвоночника***[значение]».
- Так как в тексте жалобы могут опускаться названия характеристик и признаков, по умолчанию значение может быть синтаксически подчинено как характеристике, так и признаку, а характеристика может быть подчинена признаку. Например, «*Жалобы на* ***боль в спине*** *с* ***локализацией*** *в области* ***поясничного отдела***

позвоночника» имеет то же значение, что и *Жалобы на* ***боль в спине*** *в области* ***поясничного отдела позвоночника***.

- В последовательности неоднословных терминов, обладающих одним и тем же первым или последним словом (или словосочетанием), повторяющееся слово или словосочетание может быть записано только один раз (*спинная и поясничная область позвоночника* вместо *спинная область позвоночника и поясничная область позвоночника*)[1]. Назовем такую форму записи упакованной формой использования терминов. Объединение терминов в упакованную форму может производиться при помощи союза или без него (например, *грудная аорта и артерия, спинная, поясничная область позвоночника*).

Для того чтобы создавать только корректные синтаксические связи, нам потребуется список терминов, предлогов и падежей, разрешенных для создания определенных связей с этими терминами. Термины должны соответствовать как главному, так и подчиненному словам, и могут задаваться не только начальными формами, но и разделами онтологии (здесь можно использовать регулярные выражения для проверки пути от корня онтологии к термину на соответствие шаблону). Подобные ограничения могут быть выражены в виде кортежа, задающего главное и подчиненное слово, шаблон их пути в онтологии от корня, возможный предлог и падеж, при помощи которых проводится подчинение зависимого слова главному. В качестве примера приведем правило <«», «.+/Боли», «», «», «.+/Боли/.+/Локализация/.+», «», «в», «Case=Loc»>, где любому главному слову, находящемуся в разделе «Боли», может подчиняться любое слово из подраздела «Локализация» раздела «Боли» в предложном падеже, соединенное через предлог «в» (боли в спинном отделе позвоночника). Всего было написано 44 таких правила, причем 41 из них описывало предлоги и падежи, задающие подчинение конкретных терминов.

С учетом описанных правил и предположений о структуре текста, алгоритм анализа жалоб пациентов на основе онтологии выглядит следующим образом.

1. Провести токенизацию и морфологическую разметку текста жалобы.
2. В размеченном тексте найти все термины. Каждый неоднословный термин должен быть свернут к одному токену с приписанными к нему грамматическими параметрами, взятыми от главного слова. Последовательность терминов в упакованной форме должна быть развернута в соответствующее перечисление полных терминов.
3. Соединить оставшиеся слова при помощи множества правил.
   a. Создать пустой список правил кандидатов.
   b. Пройти по всем словам текста.
   c. Начиная с хвоста списка правил кандидатов, найти первое, помеченное как «активное», и чье подчиняемое слово может быть применено к текущему слову. Если такое правило было найдено, нужно: 1) удалить все правила, находящиеся после него; 2) создать соединение между главным и зависимым словами; 3) подчинить предлог зависимому слову, если он есть в правиле.
   d. Поместить в список правил-кандидатов все правила, чье главное слово может быть успешно применено к текущему слову. Если правило требует предлога, пометить его как «неактивное», в противном случае – как «активное».
   e. Если текущее слово является предлогом, пометить все неактивные правила, которые ожидают этого предлога, как «активные»; все правила, не обладающие предлогом, пометить как «неактивные».

[1] Подобное явление описано, например, в [20, с. 240]

4. Перебирать все прилагательные и причастия в размеченном тексте, которые не были включены в неоднословные термины и не были соединены с другими словами на предыдущих шагах. Эти слова следует соединить с соседними существительными, согласующимися с данным прилагательным или причастием. Предпочтение отдается существительному, находящемуся справа.
5. По полученному списку связей слов создать дерево зависимостей предложения.

Удаляя правила в пункте 3с алгоритма, мы пытаемся поддерживать проективность дерева, то есть свойство, при наличии которого слова двух синтаксических групп находятся компактно и не перемешиваются. При соединении прилагательных и существительных, мы придерживаемся соответствующей статистики для русского языка [21, 22]. Мы «переиспользуем» предлоги, так как мы обнаружили, что зачастую при перечислении терминов предлог пишется только при первом упоминании, а дальше опускается. Аналогичная ситуация наблюдается и в группах терминов в упакованной форме. Заметим, что при анализе мы игнорируем все знаки препинания, так как в них делается больше всего ошибок. Также большое количество ошибок делается и в написании терминов, в связи с чем мы приме няем при их поиске алгоритмы нечеткого сравнения.

Рассмотрим пример разбора текста жалобы при помощи предложенного алгоритма.

Жалобы на выраженную боль в грудном и поясничном отделе позвоночника, усиление боли при физической нагрузке, нарушение ходьбы.

На первом этапе мы токенизируем текст и обрабатываем термины, как однословные, так и и неоднословные. Ниже термины даны в квадратных скобках, а остальные слова – в фигурных. Результат разбора будет следующим.

[Жалобы] {на} [выраженную] [боль] {в} [грудном отделе позвоночника] и [поясничном отделе позвоночника], [усиление боли] при [физической нагрузке], [нарушение ходьбы].

После анализа первых двух слов (*Жалобы на*) в списке будет одно правило, помеченное как активное, которое будет описывать соединения признака «*жалобы*» с его характеристиками. Слово «*выраженную*» пропускается без применения правил. По имеющемуся в списке правилу, термин «*боль*» может быть соединен с «*жалобы на*». Также на этом слове будет добавлено несколько правил вида «боль + в + локализация в предложном падеже». Эти правила будут активированы при анализе следующего слова – предлога «*в*». Активированные правила будут применены к терминам «*грудном отделе позвоночника*» и «*поясничном отделе позвоночника*», по этим правилам будут созданы соответствующие соединения между терминами. Термин «*усиление боли*» создаст соединение между признаком «*боль*» и характеристикой «*усиление боли*». На этом же слове будет удалено правило «боль + в + локализация» и добавлено правило «усиление боли + при + значение в предложном падеже». Последнее правило будет активировано предлогом «*при*», а на термине «*физической нагрузке*» будет создана связь между терминами. Наконец, термин «*нарушение ходьбы*» будет присоединен к термину «*жалобы*», а все правила, относящиеся к усилению боли, будут удалены из списка.

В результате работы алгоритма будут созданы следующие связи между терминами:

жалобы → боль → на,

боль → грудной отдел позвоночника → в,

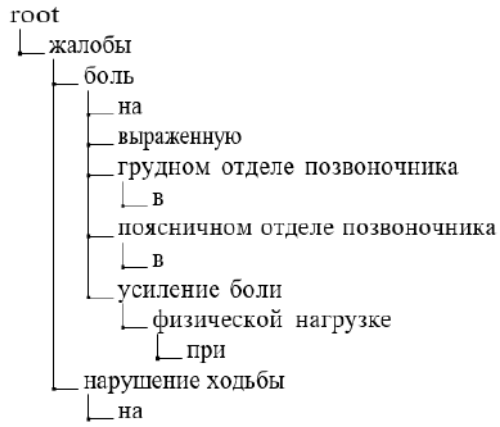
боль → поясничный отдел позвоночника → в,

боль → усиление боли,

усиление боли → физическая нагрузка → при,

жалобы → нарушение ходьбы → на

На следующем шаге мы соединим слова «*боль*» и «*выраженный*». После этого из полученных связей можно формировать итоговое дерево зависимостей.



Заметим, что правила глагольного и именного управления для терминов, которые мы писали вручную, могут быть извлечены из текстов жалоб в автоматизированном режиме. Для этого необходимо посчитать статистику совместной встречаемости терминов с учетом их связей в иерархии онтологии и выбрать наиболее вероятные варианты разбора. Однако детальная разработка такого метода сложна и является темой для отдельного проекта. Последовательности слов, которые не были найдены в онтологии, должны быть в нее добавлены вручную, так как для этого требуется определить их место в предметной области.

4.3 Разбор текста с использованием синтаксического анализатора

В качестве контроля мы использовали синтаксические анализаторы общего назначения. Заметим, что между деревьями зависимостей, которые строят наши алгоритмы, и деревьями зависимостей, построенными обычными анализаторами, имеется важное различие: последние строят деревья, вершины которых представляют собой отдельные слова, тогда как мы объединяем несколько слов неоднословного термина в одну вершину. Как следствие, прямое сравнение деревьев становится невозможным, и возникает необходимость ввести новый этап постобработки.

После разбора с использованием анализатора общего назначения мы находим термины в исходном тексте и вершины, соответствующие корням терминов. После этого мы удаляем все вершины, соответствующие словам каждого термина, оставляя лишь одну, находящуюся на самом верхнем уровне. Токен этой вершины заменяется на текст термина, а грамматические параметры заменяются на параметры слова, соответствующего корню дерева разбора термина. В случае, если в дереве зависимостей предложения находится не одно слово на верхнем уровне, а несколько, мы оставляем первое найденное. Такая ситуация означает, что синтаксический анализатор ошибся и разбил термин на несколько синтаксически не связанных фрагментов или некорректно определил связи термина с его потомками. Например, предложение *Жалобы на боль в поясничном отделе позвоночника с левой стороны* может разобраться *как жалоба → боль, боль → поясничный отдел* и *жалоба → позвоночник* (то есть позвоночник жалуется на боль в поясничном отделе). В этом примере термин был разорван на две части. Другим вариантом анализа является *боль → поясничный отдел позвоночника* и *боль → левая сторона* вместо *поясничный отдел позвоночника → левая сторона*. Здесь термины были выделены корректно, но связи построены неправильно. В первой ситуации наш алгоритм может создать некорректную связь (как мы это видели во втором примере), в результате чего будет сгенерирована ошибка (что является правильным поведением). Однако вероятна ситуация, когда наш алгоритм устранит ошибку и тем самым повысит точность работы синтаксического анализатора. Таким образом, полученная оценка для данного алгоритма будет являться скорее оценкой сверху, хотя и довольно точной.

5. Оценка точности методов

Итак, целью данной работы является сравнение трех описанных выше подходов к анализу текстов жалоб пациентов: традиционного синтаксического анализатора без каких-либо подсказок; традиционного синтаксического анализатора, результаты работы которого исправляются при помощи данных из онтологии; поверхностно-синтаксический анализ текста на основе определения связей между терминами из онтологии. Для экспериментов мы использовали два синтаксических анализатора: UDPipe 2.5 (выпущен в декабре 2019) и spaCy 3.0 (март 2021).

Мы вручную разметили «золотой стандарт», содержащий сто текстов жалоб пациентов, написанных на русском языке. Каждая жалоба была представлена в виде дерева зависимостей, всего жалобы содержали 1313 связей, соединяющих термины. Сравнивая результаты работы анализаторов с «золотым стандартом», мы не принимали во внимания слова, не являющиеся терминами, и не включили их в разметку «золотого стандарта». Несмотря на то, что предлоги в нашей нотации несут важную информацию о соединении терминов, они также были исключены из рассмотрения с тем, чтобы не вносить шум в статистику соединения терминов.

Так как нашей целью была оценка точности соединения терминов между собой, а не оценка правильности определения вида такого соединения, мы использовали метрику UAS (Unlabeled Attachment Score, доля правильных ответов при построении связей), а не LAS (Labeled Attachment Score, доля правильных ответов как для связей, так и их меток) [23]. Результаты наших экспериментов приведены в табл. 1.

Табл. 1. Результаты экспериментов
Table 1. Experimental results

Метод	**Правильных ответов**	**UAS**
UDPipe	975	0.743
Spacy	1080	0.823
UDPipe + онтология	930	0.708
Spacy + онтология	1021	0.778
Поверхностный синтаксический внализ + онтология	1084	0.826

Как видно из Табл. 1, применение онтологии ухудшает результаты синтаксического анализа, что связано с неполнотой применяемой нами онтологии. Наш анализ показал, что применение онтологии позволяет исправить некоторые ошибки анализатора, но привносит две большие проблемы. Первая проблема заключается в том, что если две вершины найдены в онтологии, но между ними нет прямой связи, то наш метод переносит подчиненную вершину выше к корню, где она и остается. Получается, что метод разрывает вершины, корректно соединенные синтаксическим анализатором, внося тем самым ошибку. Заметим, что такой же результат мы получим там, где синтаксический анализатор сам допустил ошибку. Второй проблемой является тот факт, что метод не включает в итоговое дерево те вершины, которые не были найдены в онтологии, тогда как синтаксический анализатор старается так или иначе соединить такую вершину с другими (причем может сделать это

вполне успешно). Получается, что неполнота онтологии, в которой отсутствуют как понятия, так и связи между ними, приводит к увеличению числа ошибок.

Поверхностный синтаксический анализ с применением информации из онтологии менее чувствителен к ее неполноте. Вместо того чтобы искать прямые соединения между вершинами в онтологии, метод проверяет семантические метки, которыми являются фрагменты путей от корня онтологии. Например, если термин имеется в онтологии и записан в ней как «Локализация», то он будет трактоваться как таковая вне зависимости от наличия связи с нужным термином. Получается, что метод подразумевает неполноту онтологии: локализация для одного признака может являться локализацией для другого, даже если прямая связь в онтологии не обозначена по тем или иным причинам. Более того, отсутствие подобной связи может рассматриваться как сигнал о необходимости пополнения онтологии.

Отметим разницу между результатами работы UDPipe и spaCy – 0.08 UAS. Эта разница является видимым прогрессом в области синтаксических анализаторов за последние полтора года. Мы сравнили результаты работа spaCy 2 и UDPipe 2.5 и обнаружили, что точность их работы являлась сравнимой.

Также отметим разницу между точностью, которую показывает spaCy, и точностью нашего метода. Среди прочего такая маленькая разница связана с тем, что нам не удалось реализовать обработку последовательностей терминов в упакованной форме при анализе в spaCy из-за сложностей редактирования и ручного построения генерируемого им дерева зависимостей. Мы обнаружили, что в «золотом стандарте» имеется 20 последовательностей с союзом «и» (спинной и поясничный отдел позвоночника) и несколько, объединенных через запятую (спинной, поясничный отдел позвоночника). То есть результаты работы spaCy должны быть выше.

Одной из самых важных проблем метода поверхностного синтаксического анализа с использованием онтологии является отсутствие правил, описывающих глагольное управление (которому также подчиняются и причастия). Заметим, однако, что онтология содержит в себе по большей части информацию о понятиях, то есть существительных, а не о процессах, выражаемых глаголами. Как следствие, на текущем этапе мы можем игнорировать глаголы, так как в жалобах пациентов они встречаются относительно редко и не играют той важной роли, которая им присуща в других текстах. Ещё одним решением может быть преобразование глаголов в существительные по словарю.

Мы можем утверждать, что наши эксперименты подтвердили старую идею о том, что анализ текстов может проводиться с использованием семантической информации (в нашем случае – онтологии) и небольшого числа простых правил поверхностного синтаксического анализа (хотя текущие правила скорее хранят информацию об именном управлении). Подобные правила можно извлекать из текстов в автоматизированном режиме, но такая работа служит темой для отдельного исследования. Для автоматического получения кандидатов в добавляемые термины можно использовать подходы, подобные предложенному в [24], извлекающие их из формализованных описаний предметной области.

6. Используемые свойства онтологии и требования к ней

В данном проекте мы использовали несколько свойств связей онтологии, относящихся к синтаксису русского языка. Часть из этих свойств была отражена при построении онтологии, использованной в данном проекте – Базы медицинской терминологии и наблюдений [17]. Опишем разницу между разными подходами в разработке онтологий.

Онтология WordNet в исходном виде [25], использует ограниченное количество семантических помет для связей между словами: гипоним/гипероним, мероним/холоним, специальные наборы связей для глаголов и прилагательных. Сама структура онтологии не запрещает использование других видов связей, однако на практике они встречаются

довольно редко. Этот недостаток зачастую исправляется при переводе онтологии или конвертации других онтологий и тезаурусов в формат WordNet. Так, например, онтология ruWordNet, созданная на основе тезауруса РуТез, [26] содержит в себе такие связи как ассоциации, синонимия и др. В отличие от WordNet и ruWordNet, наша онтология содержит в себе функциональные и атрибутивные связи (см. разд. 3).

Теперь кратко рассмотрим, как эти семантические связи выражаются в русском языке. Связь гипоним/гипероним вводится в тексте при помощи специальной конструкции, например, «А – это В», «А, вид А – В» и т.д. (подробнее описано в [27, 28]), в противном случае, он будет содержать в себе тавтологию. То же касается синонимических и антонимических связей. Отношение меронимии показывает принадлежность одного объекта или понятия другому. В русском языке такая связь выражается при помощи генетивной конструкции (холоним подчиняется мерониму и находится в родительном падеже) – «капот автомобиля». Та же генетивная конструкция может использоваться и для выражения функциональной связи – «локализация боли». Также функциональная связь может быть выражена при помощи связи через предлог – «боль с локализацией» или с использованием связывающего глагола в разных формах – «боль, локализующаяся в». Для атрибутивных связей также может использоваться прямое дополнение или предложная связь – «локализация в правом глазу», «иррадиация в правую ногу». В отличие от других видов, атрибутивная связь позволяет связывать между собой существительное, выражающее название атрибута, и прилагательное, выражающее его значение – «сильная боль в глазу».

Заметим, что функциональных и атрибутивных связей достаточно, чтобы описать большую часть связей между словами в медицинских текстах. При этом отсутствие таких связей не позволяет получить приемлемый уровень полноты при их анализе. Таким образом, мы можем утверждать, что онтология жалоб пациентов обязана включать в себя атрибутивные и функциональные связи для обеспечения успешной работы синтаксического анализатора. Именно это свойство наблюдается у «Базы медицинской терминологии и наблюдений» [17], и именно благодаря ему нам удалось достигнуть достаточно высокого уровня точности результатов (хотя, говоря о точности, следует говорить скорее об ошибках, полученных вследствие неполноты онтологии и системы правил поверхностно-синтаксического анализа).

Также заметим, что синонимические и гипонимические связи оказываются очень полезны, когда один термин оказывается заменен другим. Для нашей онтологии такая замена означает пропуск уровня в связях одного термина с другим. Именно поэтому мы предъявляли требование принципиального наличия прямого пути между терминами по иерархии, а не наличия непосредственной связи. Требования наличия прямого пути связано с тем, что переход между соседями означает переход к понятию хотя и близкому, но имеющему отличия. Так, например, «Боль в глазу» имеет набор характеристик и их значений, отличающийся от характеристик понятия «Боль в спине», хотя оба этих термина являются соседями и прямыми потомками термина «Боль».

7. Заключение

В данной статье мы показали, что поверхностный синтаксический анализ русских текстов с определением связей при помощи онтологии узкой предметной области может показывать точность, сравнимую с точностью современных синтаксических анализаторов на основе нейронных сетей. Предложенный нами подход находится на одном уровне с одним из лучших современных синтаксических анализаторов – spaCy 3.0 в области анализа жалоб пациентов, извлеченных из историй болезни (разница составила лишь 0.003 UAS). Заметим, что предыдущее поколение синтаксических анализаторов (UDPipe 2 и spaCy 2) показывает гораздо более низкие результаты.

Подобная разница объясняется тем, что медицинские тексты обладают рядом особенностей, критически снижающих точность при их анализе: отсутствие глаголов; длинные

последовательности именных групп, выражающих связи между терминами; сами термины. Из-за таких особенностей, точность синтаксического анализа снижается с примерно 0.95 до примерно 0.85.

Недостатком предложенного метода является наличие онтологии предметной области, построенной по определенным принципам и содержащей в себе иерархию терминов. Создание подобной онтологии является серьезным трудом и занимает годы. Ручное создание правил глагольного и именного управления также занимает много времени. Подобная ситуация оправдывает себя в случаях, когда исходных данных недостаточно для применения методов машинного обучения. В то же время, развитие нейронных сетей, являющихся основой современных синтаксических анализаторов, ещё не исчерпало своего ресурса, в связи с чем мы можем ожидать нового прогресса, в том числе и в синтаксическом анализе. Применение предложенного нами метода для широкого спектра текстов требует появления новых методов автоматического выделения словарей глагольного и именного управления, а также автоматизированного пополнения онтологий, так как полнота и точность существующих методов всё ещё не удовлетворяет практических потребностей. В связи с этим можно утверждать, что предложенный нами метод подходит для текстов узкой предметной области с простым синтаксисом, существенно отличающимся от общепринятой нормы, и хорошо проработанной онтологией, отвечающей описанным в статье требованиям. В подобной ситуации он позволит получить выигрыш в скорости разработки системы и точности ее работы.

Список литературы / References

[1] Nugmanov R., Alimova I., Tutubalina E. Adverse drug reactions identification in social media posts and electronic health records with neural networks. European Journal of Clinical Investigation, vol.49, 2019, pp. 116-117.

[2] Chapman W.W., Gundlapalli A.V. et al. Natural Language Processing for Biosurveillance. In Infectious Disease Informatics and Biosurveillance: Research, Systems and Case Studies. Springer, 2011. pp. 279-310.

[3] Straka M., Straková J., Hajic J. UDPipe at SIGMORPHON 2019: Contextualized Embeddings, Regularization with Morphological Categories, Corpora Merging. In Proc. of the 16th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, 2019. pp. 95-103.

[4] Astudillo R.F., Ballesteros M. et al. Transition-based Parsing with Stack-Transformers. arXiv:2010.10669, 2020.

[5] Wang Y., Lee H.-Y., Chen Y.-N. Tree Transformer: Integrating Tree Structures into Self-Attention. In Proc. of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019. pp. 1061-1070.

[6] Abney S. P. Parsing By Chunks. Studies in Linguistics and Philosophy, vol. 44, 1991. pp. 19-33.

[7] Molina A., Pla F. Shallow Parsing using Specialized HMMs. Journal of Machine Learning Research, vol. 2. 2002. pp. 595-613.

[8] Sha F., Pereira F.C. Shallow Parsing with Conditional Random Fields. In Proc. of 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, 2003. pp. 134-141.

[9] Кобзарева Т.Ю., Лахути Д.Г., Ножов И.М. Модель сегментации русского предложения. Труды Международного семинара Диалог 2001, 2001 г., стр. 185–194 / Kobzareva T.Yu., Lakhuti D.G., Nozhov I.M. Segmentation model of the Russian sentence. In Proc. of the International Seminar Dialogue 2001, 2001, pp. 185-194 (in Russoan).

[10] Sowa J.F. Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks Cole Publishing, 2000, 594 p.

[11] Лукашевич Н.В. Тезаурусы в задачах информационного поиска. Изд-во Московского университета, 2011 г., 512 стр. / Lukashevich N.V. Thesauri in information retrieval problems. Publishing house of Moscow State University, 2011, 512 p. (in Russian).

[12] Current Bibliographies in Medicine. URL: https://www.nlm.nih.gov/archive/20040831/pubs/cbm/umlscbm.html.

[13] Aronson A.R., Lang F.-M. An overview of MetaMap: historical perspective and recent Advances. Journal of the American Medical Informatics Association, 2010, vol. 17, issue 3, pp. 229-236.
[14] Valdez J. An Ontology-Enabled Natural Language Processing Pipeline for Provenance Metadata Extraction from Biomedical Text (Short Paper). Lecture Notes in Computer Science, vol. 10033, 2016, pp. 699-708.
[15] MSHRUS (MeSH Russian) – Statistics. URL: https://www.nlm.nih.gov/research/umls/sourcereleasedocs/current/MSHRUS/stats.html
[16] Shelmanov A. O., Smirnov I. V., Vishneva E.A. Information Extraction from Clinical Texts in Russian. In Proc. of the International Conference on Computational Linguistics and Intellectual Technologies (Dialog-2015), 2015, pp. 560–572.
[17] Грибова В.В., Москаленко Ф.М. и др. Концепция гетерогенного хранилища биомедицинской информации. Информационные технологии, том 27, no. 2, 2019 г., стр. 97-106 / Gribova V.V., Moskalenko Ph.M. et al. A Concept for a Heterogeneous Biomedical Information Warehouse. Information technologies, vol. 25, no. 2, 2019, pp. 97-106 (in Russian).
[18] spaCy: What's New in v3.0. URL: https://spacy.io/usage/v3.
[19] Nivre J., de Marneffe M.-C. et al. Universal Dependencies v1: A Multilingual Treebank Collection. In Proc. of the 10th International Conference on Language Resources and Evaluation (LREC 2016), 2016, pp. 1659-1666.
[20] Апресян Ю.Д. Избранные труды, т. I. Лексическая семантика: 2 изд. М, Школа, 1995, 472 стр. / Apresyan Yu.D. Selected works, vol. I. Lexical semantics: 2nd ed. M, Shkola, 1995, 472 p. (in Russian).
[21] Клышинский Э.С. Степень свободы русского синтаксиса несколько преувеличена. Сборник трудов 20-го научно-практического семинара «Новые информационные технологии в автоматизированных системах», 2017 г., стр. 112-116 / Klyshinskiy E.S. The degree of freedom of Russian syntax is somewhat exaggerated. In Proc. of the 20th Scientific-Practical Seminar on New Information Technologies in Automated Systems, 2017, pp. 112-116 (in Russian).
[22] Клышинский Э.С., Логачева В.К. и др. Количественная оценка грамматической неоднозначности некоторых европейских языков. Вестник НГУ. Серия: Лингвистика и межкультурная коммуникация, 2020, том 18, вып. 1, стр. 5-21 / Klyshinskiy E.S. Logacheva V.K. et al. Quantitative Estimation of Grammatical Ambiguity: Case of European Languages. NSU Vestnik. Series: Linguistics and Intercultural Communication, vol. 18. issue 1, 2020, pp. 5-21 (in Russian).
[23] Nivre J., Fang C.-T. Universal Dependency Evaluation. In Proc. of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017), 2017, p. 86-95.
[24] Захаров В.П., Хохлова М.В. Анализ эффективности статистических методов выявления коллокаций в текстах на русском языке. Труды международной конференции Диалог-2010, 2010 г., стр. 136-143 / Zakharov V.P., Khokhlova M.V. Study of effectiveness of statistical measures for collocation extraction on Russian texts. In Proc. of the International Conference Dialogue 2010, 2010, стр. 136-143 (in Russian).
[25] Fellbaum C. (ed.) WordNet: An Electronic Lexical Database. MIT Press, 1998, 449 p.
[26] Лукашевич Н.В., Лашевич Г. и др. Порождение тезауруса типа WordNet для русского языка. Труды Пятнадцатой национальной конференция по искусственному интеллекту с международным участием (КИИ-2016), 2016 г., стр. 89-97 / Loukachevitch N.V., Lashevich G. et al. Generating russian wordmet. In Proc, of the Fifteenth National Conference on Artificial Intelligence with International Participation (CAI 2016), 2016, pp. 89-97 (in Russian).
[27] Большакова Е.И., Васильева Н.Э., Морозов С.С. Лексико-синтаксические шаблоны для автоматического анализа научно-технических текстов. Труды Десятой национальной конференция по искусственному интеллекту с международным участием (КИИ-2006), 2006 г., стр. 506-524 / Bolshakova E.I., Vasilieva N.E., Morozov S.S. Lexicosyntactic patterns for automatic text processing. In Proc, of the Tenth National Conference on Artificial Intelligence with International Participation (CAI 2006), 2006, pp. 506-524 (in Russian).
[28] Большакова Е.И., Баева Н.В. и др. Лексико-синтаксические шаблоны в задачах автоматической обработки текстов. Труды международной конференции Диалог-2007, 2007 г., стр. 70-75 / Bolshakova E.I., Baeva N.V. et al. Lexicosyntactic patterns for automatic text processing. In Proc. of the International Conference Dialogue 2007, 2007, pp. 70-75 (in Russian).

Информация об авторах / Information about authors

Борис Израйльевич ГЕЛЬЦЕР – доктор медицинских наук, профессор, член-корреспондент РАН, директор Департамента клинической медицины Школы биомедицины ДВФУ. Научные интересы: доказательная медицина, клиническая медицина, методы машинного обучения в медицине, медицинские информационные системы.

Boris Israelevich GELTSER – Doctor of Medicine, professor corresponding member of RAS, head of Department of Clinical Medicine of School of Biomedicine FEFU. Research interests: evidence based medicine, clinical medicine, machine learning in medicine, medical information systems.

Татьяна Александровна ГОРБАЧ – кандидат медицинских наук, врач-невролог Медицинского центра ДВФУ. Научные интересы: неврология, когнитивные расстройства, системы представления знаний.

Tatiana Aleksandrovna GORBACH – PhD in Medicine, researcher at IACP FEB RAS. Research interests: neurology, cognitive disorders, knowledge representation.

Валерия Викторовна ГРИБОВА – доктор технических наук, заместитель директора по научной работе ИАПУ ДВО РАН. Научные интересы: Искусственный интеллект, принятие решений, экспертные системы, программные системы.

Valeriya Victorovna GRIBOVA – doctor of technical science, Research Deputy Director of IACP FEB RAS. Research interests: artificial intelligence, decision making, expert systems, software systems.

Олеся Владимировна КАРПИК – младший научный сотрудник ИПМ им. М.В. Келдыша. Научные интересы: лексикография, синтаксис, фонетика.

Olesya Vladimirovna KARPIK – junior researcher at Keldysh IAM RAS. Research interests: lexicography, syntax, phonetics.

Эдуард Станиславович КЛЫШИНСКИЙ – кандидат технических наук, доцент, доцент школы лингвистики НИУ ВШЭ. Научные интересы: искусственный интеллект, формальный синтаксис, автоматическая обработка текстов.

Eduard Stanislavovich KLYSHINSKIY – PhD in Computer Science, associated professor at School of Linguistics at NRU HSE. Research interests: artificial intelligence, formal syntax, natural language processing.

Наталия Александровна КОЧЕТКОВА – аспирант НИУ ВШЭ. Научные интересы: автоматическая обработка текстов, извлечение именованных сущностей, стилеметрия.

Natalia Aleksandrovna KOCHETKOVA – PhD student at NRU HSE. Research interests: natural language processing, named entities recognition, stylometrics.

Дмитрий Борисович ОКУНЬ – кандидат медицинских наук, научный сотрудник ИАПУ ДВО РАН. Научные интересы: онтологии, прикладные интеллектуальные системы, экспертные системы.

Dmitry Borisovich OKUN – PhD in Medicine, researcher at IACP FEB RAS. Research interests: ontology, application intelligent systems, expert systems.

Маргарита Вячеславовна ПЕТРЯЕВА – кандидат медицинских наук, научный сотрудник ИАПУ ДВО РАН. Научные интересы: биомедицина, прикладные интеллектуальные системы, экспертные системы.

Margaret Vyacheslavovna PETRYAEVA – PhD in Medicine, researcher at IACP FEB RAS. Research interests: biomedical system, applied intelligent systems, expert systems.

Карина Иосифовна ШАХГЕЛЬДЯН – доктор технических наук, профессор, директор института информационных технологий ВГУЭС. Научные интересы: системы представления знаний, машинное обучение, программные системы.

Carina Iosifovna SHAKHGELDYAN – doctor of technical science, professor, director of Institute of Information Technologies at VVSU. Research interests: knowledge representation, machine learning, software systems.

DOI: 10.15514/ISPRAS-2021-33(4)-9

Построение нейросетевых моделей морфологического и морфемного анализа текста

А.С. Сапин, ORCID: 0000-0002-9532-132X<alesapin@gmail.com>
Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. Морфологический анализ текстов на естественном языке является одним из важнейших этапов автоматической обработки текстов (АОТ). Традиционные и хорошо исследованные задачи морфологического анализа включают приведение словоформы к нормальной форме (лемме), определение ее морфологических характеристик, а также разрешение (снятие) морфологической омонимии (неоднозначности характеристик). К морфологическому анализу относится также задача морфемного разбора слов (т.е. сегментация слов на составляющие морфы и их классификация), которая востребована в некоторых приложениях АОТ. В последние годы разработан ряд программных моделей на основе машинного обучения, повышающих точность традиционного морфологического анализа и морфемного разбора, однако производительность таких моделей недостаточна для многих практических задач, а для задачи морфемного разбора высокоточные модели построены только для лемм. В данной работе описаны две новые высокоточные нейросетевые модели, реализующие морфемный разбор словоформ русского языка при достаточно высокой производительности. Первая модель основана на сверточной нейронной сети и показывает достойное качество морфемного разбора словоформ. Вторая модель, кроме морфемного разбора словоформы, позволяет предварительно уточнить её морфологические характеристики, решая задачу снятия омонимии. Производительность этой объединенной морфологической модели оказалась наилучшей среди рассмотренных моделей морфемного разбора, при сравнимой точности разбора.

Ключевые слова: морфологический анализ словоформ; автоматический морфемный разбор; нейросетевые модели морфемного разбора.

Для цитирования: Сапин А.С. Построение нейросетевых моделей морфологического и морфемного анализа текста. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 117-130. DOI: 10.15514/ISPRAS–2021–33(4)–9

Building neural network models for morphological and morpheme analysis of texts

A.S. Sapin, ORCID: 0000-0002-9532-132X <alesapin@gmail.com>
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. Morphological analysis of text is one of the most important stages of natural language processing (NLP). Traditional and well-studied problems of morphological analysis include normalization (lemmatization) of a given word form, recognition of its morphological characteristics and their morphological disambiguation. The morphological analysis also involves the problem of morpheme segmentation of words (i.e., segmentation of words into constituent morphs and their classification), which is actual in some NLP applications. In recent years, several machine learning models have been developed, which increase the accuracy of traditional

morphological analysis and morpheme segmentation, but performance of such models is insufficient for many applied problems. For morpheme segmentation, high-precision models have been built only for lemmas (normalized word forms). This paper describes two new high-accuracy neural network models that implement morphemic segmentation of Russian word forms with sufficiently high performance. The first model is based on convolutional neural networks and shows the state-of-the-art quality of morphemic segmentation for Russian word forms. The second model, besides morpheme segmentation of a word form, preliminarily refines its morphological characteristics, thereby performing their disambiguation. The performance of this joined morphological model is the best among the considered morpheme segmentation models, with comparable accuracy of segmentation.

Keywords: morpheme segmentation of wordforms; neural models for morphological analysis; morphological analysis of wordforms

For citation: Sapin A.S. Building neural network models for morphological and morpheme analysis of texts. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021, pp. 117-130 (in Russian). DOI: 10.15514/ISPRAS-2021-33(4)-9

1. Введение

Морфологический анализ является одним из базовых этапов автоматической обработки текстов (АОТ), результаты которого используются во многих прикладных задачах. К основным задачам морфологического анализа относится определение морфологических характеристик (часть речи, падеж, число, род и т.д.) словоформы [1]. Например, для словоформы “шоколада” распознаются характеристики: существительное, родительного падежа, единственного числа, мужского рода.

Важной задачей морфологического анализа является снятие (разрешение) морфологической неоднозначности (омонимии), т.е. выявление корректного для обрабатываемого текста варианта *морфологических характеристик* словоформы из всех возможных. Например, словоформа *“стали”* может быть как существительным множественного числа (“*виды стали”*), так и глаголом прошедшего времени (“*стали разгружать”*). Разрешение омонимии в этом примере сводится к выбору одного варианта из двух возможных *<сущ., мн. ч., ...>* и *<гл., пр. вр., ...>*. Качество морфологического анализа обычно оценивается с учетом снятия омонимии, для этого используется метрика аккуратности (точности) определения морфологических характеристик [2], которая рассчитывается как количество правильных ответов к количеству всех анализируемых словоформ текста.

Ещё одной задачей, относящейся к морфологическому анализу, является морфемный разбор [3], который заключается в анализе морфемного состава слова путем его разбиения (сегментации) на морфы (морфемы), например: *impossible → im-poss-ible*, *прекрасный → пре-крас-н-ый*. Морфемы являются наименьшими значащими единицами текста, и результаты морфемного разбора необходимы в ряде прикладных задач АОТ, таких как исправление словообразовательных и паронимических ошибок, распознавание смысла незнакомых и редких слов по более частотным родственным словам.

Задачи разрешения морфологической омонимии и морфемного разбора являются актуальными для высокофлективных языков со сложным словоизменением и словообразованием (большое количество суффиксов, префиксов, окончаний), к каковым относится русский язык. В последние годы продолжаются исследования по применению машинного обучения для задач морфологического анализа русского языка [4, 5, 6, 7], которые позволили улучшить качество разрешения морфологической омонимии до 95% точности для морфологических характеристик. Однако производительность таких машиннообученных моделей анализа является чрезвычайно низкой (всего лишь сотни слов в секунду на одном ядре CPU). Для задачи морфемного разбора на базе машинного обучения были построены высокоточные модели разбора лемм (нормальных форм) русского языка [8, 9, 10], однако их точность для различных словоформ русского языка недостаточна и их производительность не оценивалась.

Настоящая работа посвящена проблеме эффективности программных моделей морфологического анализа, в том числе морфемного разбора, для словоформ русского языка. Под эффективностью мы понимаем как высокую точность решения задач морфологического анализа, так и производительность, позволяющую быстрее обрабатывать современные корпуса текстов из сотен миллионов слов.

В работе реализованы и исследованы две новые нейросетевые модели морфемного разбора словоформ русского языка. Первая модель выполняет морфемный разбор словоформ, превосходя как по производительности, так и по точности разбора известные модели морфемного разбора для лемм [8, 9, 10]. Поскольку для применения этой модели необходима такая морфологическая характеристика словоформы, как часть речи, дополнительно на основе этой модели построена вторая, объединенная модель морфологического анализа, выполняющая одновременно снятие морфологической омонимии словоформы и её морфемный разбор.

В следующем разделе кратко излагаются результаты в области традиционного морфологического анализа, применимые для построения процессоров русского языка. В третьем разделе рассматриваются подходы к автоматическому морфемному разбору и разработанные в последние годы высокоточные модели морфемного разбора лемм русского языка. В четвертом разделе описывается разработанная нами модель морфемного разбора словоформ русского языка, а также её экспериментальное исследование. В пятом разделе содержится описание объединенной модели морфологического анализа: ее архитектура, параметры обучения, оценки качества и производительности. В заключении кратко приводятся основные результаты настоящей работы.

2. Методы традиционного морфологического анализа

Для русского языка большинство применяемых в настоящее время морфологических процессоров (в том числе открытые анализаторы [11, 12, 13]) базируются на словарной информации, т.е. либо на словаре основ, либо на словаре словоформ (последние для русского языка используются значительно чаще). Определение морфологических характеристик анализируемой словоформы сводится к её поиску в соответствующем словаре и выдаче всех возможных вариантов морфологических характеристик (тегов) обрабатываемой словоформы.

Словарные морфологии показывают высокую производительность (до 120 тысяч слов в секунду на CPU [11]), однако не позволяют решать задачу снятия морфологической омонимии. Для её решения требуется последующее применение отдельной процедуры к полученным из словаря результатам. Эта процедура обычно строится на основе машинного обучения с учителем по размеченному текстовому корпусу и позволяет выбрать единственно верный вариант морфологических характеристик из нескольких возможных. В разных морфологических процессорах используются разные методы машинного обучения: в Диалинг-АОТ [11] – скрытые марковские цепи, в TreeTagger [14] – деревья решений, а в парсере UDPipe 1.0 [15] – полносвязная нейронная сеть. Подобные методы достигают точности определения морфологических характеристик с учетом снятия морфологической омонимии до 94.5% для известных слов и до 79% для слов отсутствующих в словарях [2]. Точность (аккуратность) определения морфологических характеристик рассчитывается как отношение количества словоформ, у которых характеристики определены верно, к количеству всех анализируемых словоформ:

$$A_{tags} = \frac{\sum_{i=0}^{len(dataset)} correct(word_i)}{len(dataset)},$$

где *len(dataset)* – количество словоформ в анализируемом тексте, $word_i$ – *i*-ое слово в тексте, а $correct(word_i) = 1$, когда все морфологические характеристики слова определены верно, и равно 0 в противном случае.

В последние годы были предложены модели морфологического анализа, в которых определение морфологических характеристик и снятие омонимии происходит одновременно, т.е. для каждой словоформы сразу же находится единственный вариант леммы и морфологических характеристик [4, 5, 6]. Особенностью такого подхода является использование векторных представлений слов из нейронных языковых моделей разного вида: FastText [16], ELmO [17], BERT [18]. В работе [5] использовались контекстуализированные векторные представления BERT и мультиклассовая логистическая регрессия и было достигнуто наилучшее качество решения задач морфологического анализа для русского языка: 95% точности определения морфологических характеристик. Такие показатели качества достаточны для прикладных задач АОТ, однако производительность подобных высокоточных моделей оказывается более чем на два порядка ниже словарных методов, поэтому их применение в практических приложениях ограничено. Открытые морфологические процессоры русского языка [11, 12, 13] по-прежнему базируются на словарях и более простых методах снятия морфологической омонимии.

3. Методы морфемного разбора лемм

Известны два варианта морфемного разбора слов:

- *морфемная сегментация*, когда требуется сегментировать слово на составляющие его морфы (морфемы), например, для слова *сетка – сет-к-а*;
- морфемная *сегментация с классификацией*, когда требуется не только сегментировать слово на морфы, но и определить их тип: приставка (*PREF*), корень (*ROOT*), суффикс (*SUFF*), окончание (*END*) и т.д., например, *сетка – сет:ROOT/к:SUFF/а:END*.

Морфемная сегментация с классификацией является наиболее полным вариантом задачи морфемного разбора и именно она рассматривается в настоящей работе.

Качество автоматической морфемной сегментации оценивается с помощью метрик точности (*Presicion*), полноты (*Recall*) и F-меры по границам морфем [19], рассчитываемых следующим образом:

$$Precision = \frac{TP}{TP + FP};\ Recall = \frac{TP}{TP + FN};\ F = \frac{2TP}{2TP+(FP+FN)},$$

где *TP* – количество верно обнаруженных границ между морфами, *FP* – количество ложно обнаруженных границ, *FN* – количество не обнаруженных границ. Для задачи сегментации с классификацией добавляются ещё точность (аккуратность) определения типа всех получившихся морфем в сегментированном слове (аккуратность по словам целиком):

$$A_{words} = \frac{\sum_{i=0}^{len(dataset)} correct(word_i)}{len(dataset)},$$

где, *len(dataset)* – количество словоформ в анализируемом тексте, $word_i$ – *i*-ое слово в тексте, а *correct(word)* = 1 только когда типы и границы всех морфов слова определены верно, и равно 0 иначе.

Первые методы автоматической морфемной сегментации [3] были чисто статистическими, основанными на неразмеченных данных и показывали 50-65% значения F-меры обнаружения границ морфем. Наиболее известное решение задачи морфемной сегментации было реализовано в системе Morfessor [20] на основе метода машинного обучения без учителя по большой неразмеченной коллекции текстов. Основная идея метода Morfessor состоит в поиске минимального набора морфем, с помощью которого можно сегментировать все слова обрабатываемой коллекции текстов. Для таких языков как английский, финский и турецкий система показывает около 70-80% F-меры для обнаруженных границ морфем.

Для применения машинного обучения с учителем нужны представительные наборы размеченных данных (*датасеты*) с сегментированными морфемами, но они трудоемки в создании и отсутствуют для большинства языков. Относительно недавно появились

несколько датасетов с морфемной разметкой (сегментация и классификация) для русского языка, наиболее представительный из них, RuMorphs-Lemmas[1], был получен на основе словообразовательного словаря Тихонова [21] и содержит около 96 тысяч размеченных лемм русского языка. Благодаря этому, на основе методов машинного обучения с учителем были разработаны несколько высокоточных методов (моделей) морфемной сегментации с классификацией для лемм русского языка [8, 9, 10]. В этих моделях использовались различные методы машинного обучения:

- сверточная нейронная сеть (CNN) [8];
- деревья решений с градиентным бустингом (GBDT) [9];
- двунаправленная нейронная LSTM-сеть (Bi-LSTM) [10].

Во всех моделях задача морфемного разбора рассматривалась как задача классификации букв, и помимо различий в методах машинного обучения модели различаются набором классов букв. CNN-модель применяет схему классификации BMES (используемую обычно в задаче выявления именованных сущностей), классифицируя каждую букву на 22 различных класса, а модели GBDT и Bi-LSTM используют сокращенный набор из 10 классов, но достаточный для решения рассматриваемой задачи. Во всех моделях буквы слова представляются в унитарной кодировке (*one-hot encoding*), а также учитывается информация о их гласности. Дополнительно, GBDT-модель использует значения морфологических характеристик сегментируемого слова: часть речи, род, число, падеж, время. Модель на основе двунаправленной LSTM-сети также применяет морфологическую информацию, но только часть речи. Важной особенностью CNN-модели является дополнительная корректирующая процедура на основе простых правил морфотактики (корень идет после приставки, суффикс после корня и т.п.), применяемая к результату нейронной сети, а также использование ансамбля из трех одинаковых CNN-моделей, что значительно повышает точность разбора, но увеличивает размер модели и снижает производительность.

Экспериментальная оценка [9] трёх указанных моделей на одних и тех же размеченных датасетах для русского языка (в том числе RuMorphs-Lemmas) показала их сравнимое качество: до 98-99% F-меры по границам морфем (в зависимости от обучающего датасета и параметров модели), а также 86-89% точности (аккуратности) морфемного разбора слов целиком – см. табл. 1 с оценками, полученными на датасете RuMorphs-Lemmas. Модель на основе Bi-LSTM слегка превосходит CNN-модель, возможно за счет дополнительного использования части речи разбираемого слова и сокращенного набора классов букв. В тоже время эта модель не требуют корректирующей процедуры. В работе [9] также показано, что наибольшее влияние на распознавание класса буквы оказывают не только соседние буквы, но и часть речи.

Табл. 1. Качество морфемного разбора для лемм русского языка (%)
Table 1. Quality of morphemic segmentation for Russian lemmas (%)

Модель	**F-мера по границам морфем**	**Точность разбора слов**
CNN + корректирующая процедура + ансамбль	98.10	88.62
GBDT + морфохарактеристики	98.01	86.54
Bi-LSTM + часть речи	**98.45**	**89.03**

Описанные модели морфемной сегментации с классификацией показывают высокую точность разбора лемм, однако их производительность не измерялась. Поскольку код моделей является открытым, мы произвели замеры их производительности на фрагменте

[1] https://github.com/cmc-msu-ai/NLPDatasets/blob/main/morphemic/dicts/tikhonov.txt

коллекции текстов lib.rus.ec[2], объемом 10 млн слов, в одноядерном режиме процессора Intel Core I7-8750H, без использования графического ускорителя. Измерялось количество слов в секунду, обрабатываемых моделью (с учетом времени на определение морфологических характеристик в моделях GBDT и Bi-LSTM и корректирующей процедуры в CNN-модели), результаты показаны в табл. 2.

Производительность моделей оказалась невысока, так что для обработки большой коллекции текстов (сотни миллионов слов), даже с учетом параллелизма потребуется несколько дней. Наибольшая производительность достигается моделью, построенной на базе сверточных нейронных сетей, она же имеет и наименьший размер.

Табл. 2. Производительность моделей морфемного разбора лемм русского языка
Table 2. Performance morphemic segmentation of Russian lemmas

Модель для лемм	**Слов в секунду**	**Размер модели (МБ)**
CNN + корректирующая процедура + ансамбль	**354**	**9.5**
GBDT + определение морфохарактеристик	269	2651
Bi-LSTM + определение части речи	64	203

Отметим, что все описанные выше модели морфемного разбора были обучены для морфемного разбора лемм (нормальных форм) русского языка, и выполненные замеры качества морфемного разбора для словоформ показали их непригодность для практики (менее 38% точности разбора по словам целиком). Причиной этого является существенное различие в морфемной структуре различных словоформ морфологически богатого русского языка, например:

разбор леммы: *расшить – рас:PREF/ши:ROOT/ть:END*
разбор словоформы: *разошьют – разо:PREF/шь:ROOT/ют:END*

Поскольку тексты состоят не из лемм, а из словоформ, необходима эффективная модель морфемного разбора, ориентированная на обработку словоформ.

4. Сверточная модель морфемного разбора словоформ

Известно, что сверточные нейронные сети являются одним из наиболее производительных видов нейронных сетей, как для обучения, так и для применения уже обученной модели. При сравнении моделей морфемного разбора лемм (табл. 2) модель на базе сверточных нейронных сетей также показала наилучшую производительность. Поэтому мы выбрали одномерные сверточные нейронные сети в качестве основы архитектуры модели морфемного разбора словоформ. Поскольку сверточные сети работают с последовательностями фиксированной длины, наша модель обрабатывает слова из 20 букв (подавляющее число слов русского языка содержит меньшее число букв). Более короткие слова дополняются пустыми символами, а более длинные делятся на части. Архитектура сети представлена на рис. 1.

На вход разработанной CNN-модели подается числовой вектор из закодированных букв словоформы в унитарной кодировке, признаков их гласности, а также закодированной части речи словоформы. Вход модели соединен со “сверточным блоком”, который состоит из одномерного сверточного слоя, слоя субдискретизации (*max pooling*) и слоя исключения (*dropout*). Слой субдискретизации позволяет значительно ускорить обучение и последующее применение модели, а слой исключения помогает бороться с переобучением. В качестве функции активации сверточного слоя взята ReLU, которая является одновременно вычислительно простой и хорошо зарекомендовавшей себя на практике. Всего в модели

[2] librusec.pro (фрагмент по ссылке https://bit.ly/3typZ57)

используются три последовательно соединённых “сверточных блока”, выход последнего подается на вход полносвязным слоям сети с функцией активации мягкий максимум (*softmax*), выступающим в роли классификаторов. Эксперименты показали, что увеличение числа сверточных блоков незначительно улучшает качество разбора, но снижает производительность модели.

Поскольку для обучения модели морфемного разбора словоформ необходим размеченный датасет с морфемным разбором словоформ (а не лемм), нами была разработана автоматическая процедура генерации размеченного датасета словоформ, исходя из известного датасета RuMorphs-Lemmas. Процедура последовательно принимает на вход морфемный разбор очередной леммы русского языка из этого датасета и на основе системы словоизменительных классов для русского языка и информации из морфологического словаря Opencorpora [22] генерирует разборы всех словоформ входной леммы. Построенный датасет RuMorphs-Words содержит более 1.7 млн различных словоформ с морфемной разметкой, для каждой словоформы указана её часть речи.

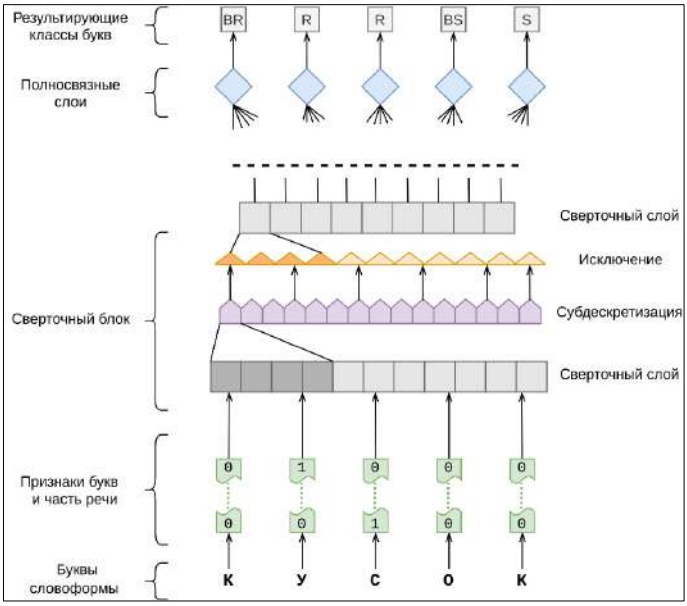


Рис. 1. Архитектура модели морфемного разбора словоформ
Fig 1. The architecture of the morphemic segmentation model of wordforms

При обучении модели буквы словоформы классифицируются на 10 классов, что достаточно для выделения соседних морфем, относящихся к одному и тому же типу (*ROOT*, *PREFIX*, *SUFFIX*). Ниже приведен пример, показывающий отличия более традиционной BMES-разметки (22 класса) от используемой нами BM-разметки (10 классов) на примере разбора словоформы *“мечтателя”, мечт:ROOT/а:SUFF/тел:SUFF/я:END*:

м	*е*	*ч*	*т*	*а*	*т*	*е*	*л*	*я*
B-ROOT	M-ROOT	M-ROOT	E-ROOT	S-SUFF	B-SUFF	M-SUFF	E-SUFF	S-END
B-ROOT	M-ROOT	M-ROOT	M-ROOT	B-SUFF	B-SUFF	M-SUFF	M-SUFF	B-END

Как видно, BM-разметка (нижняя строка) позволяет выделить границу последовательных суффиксов *“а”* и *“тель”*.

При обучении модели датасет разбивался в соотношении 70% для обучающего множества, 10% для валидационного и 20% для тестового (время обучения составило около 25 минут на Nvidia Tesla T4). Точность обученной модели морфемного разбора словоформ составила 91.06% по словам целиком для словоформ, а при проверке только на леммах – 90.03%, что является наилучшим достижимым качеством морфемного разбора для слов русского языка –

см. табл. 3, строка 1 (точность F-меры по границам морфем также высока, как и в моделях для разбора лемм, поэтому не показана).

Табл. 3. Точность моделей морфемного разбора словоформ русского языка (%)
Table 3. Accuracy of models for morphemic segmentation of Russian word forms (%)

Модель для словоформ	RuMorphs-Words	RuMorphs-Lemmas	Morphs-SynTagRus
CNN	90.03	91.06	-
Объединенная	-	85.90	88.54

Оценка производительности модели для словоформ выполнялась с помощью библиотеки tensorflow-lite [23], так как она включает большинство реализованных в tensorflow оптимизаций, а также обладает простым интерфейсом для применения моделей и поддерживается для нескольких языков программирования. Разработанная для словоформ модель показала наилучшую производительность среди рассмотренных моделей морфемного разбора: 4559 слов в секунду – см. табл. 4, строка 1. Однако, с учетом времени определения части речи (морфопроцессором[3] для русского языка) производительность снизилась до 2380 слов (строка 2 табл. 4). Тем самым, определение части речи негативно сказывается на производительности модели. Разработанная нами объединённая модель морфологического анализа позволяет добиться большей производительности за счёт одновременного определения части речи и морфемного разбора словоформы.

Табл. 4. Производительность моделей морфемного разбора словоформ
Table 4. Performance of models for morphemic segmentation of Russian word forms

Модель для словоформ	Слов в секунду	Размер (МБ)
CNN с известной частью речи	4559	1.1
CNN с определением части речи	2380	1.1
Объединенная морфологическая модель	1893	1.5
Комплекс объединенных морфологических моделей	3543	13.5

5. Объединенная модель морфологического анализа

Объединенная модель, так же, как и модель морфемного разбора словоформ, основана на сверточных нейронных сетях из-за их высокой производительности. В отличие от описанной выше CNN-модели для словоформ, объединённая модель обрабатывает текст по предложениям, последовательностям слов фиксированного размера.

Для каждой словоформы предложения берутся её возможные морфологические характеристики (варианты морфологического анализа), определяемые морфологическим процессором. В случае морфологической омонимии модель снимает её (уточняет часть речи, падеж, число, род, время) и использует уточненную часть речи для выполнения морфемного разбора.

Архитектура объединенной модели представлена на рис. 2, слева показана часть модели, отвечающая за разрешение морфологической омонимии, а справа – часть модели, отвечающая за морфемный разбор.

Поскольку использование векторных представлений слов, полученных из нейронных языковых моделей, значительно повышает качество морфологического анализа [4, 5, 6], на вход модели подаются вектора обрабатываемых словоформ из языковой модели FastText [16] (эта одна из вычислительно-простых языковых моделей для высокофлективного русского

[3] https://github.com/alesapin/XMorphy

языка). Эти вектора словоформ конкатенируются с векторами закодированных вариантов их морфологического анализа, полученными морфопроцессором.

Эти данные обрабатываются тремя сверточными блоками (их архитектура аналогична сверточным блокам вышеописанной модели морфемного разбора словоформ), полученный результат поступает в полносвязные слои (для каждого слова свой набор слоёв), выступающие в роли классификаторов и определяющие значения морфологических характеристик словоформ: часть речи, падеж, род, число, время.

Один выход полносвязного слоя для каждого слова, ответственный за часть речи, подается в ту часть модели, которая реализует морфемный разбор, вместе с закодированными буквами обрабатываемых словоформ и признаками их гласности (аналогично сверточной модели морфемного разбора словоформ). Морфемный разбор словоформ из обрабатываемой последовательности слов выполняется независимо.

Поскольку для обучения разрабатываемой модели необходим размеченный датасет, в котором будет одновременно и морфологическая, и морфемная разметка словоформ русского языка, а такие датасеты на данный момент не разработаны, то был взят и дополнительно размечен известный корпус с морфологической разметкой SynTagRus [24] (около 1.1 млн слов) – в нем была добавлена морфемная разметка каждой словоформы. Корпус SynTagRus был выбран, как представительный и в тоже время использованный в морфологическом соревновании [7], что позволяет сравнить разработанную нами модель с наилучшим достижимым качеством морфологического анализа. Морфемная разметка добавлялась в автоматизированном режиме с помощью нашей уже реализованной сверточной модели морфемного разбора словоформ и дополнительной ручной проверки результата.

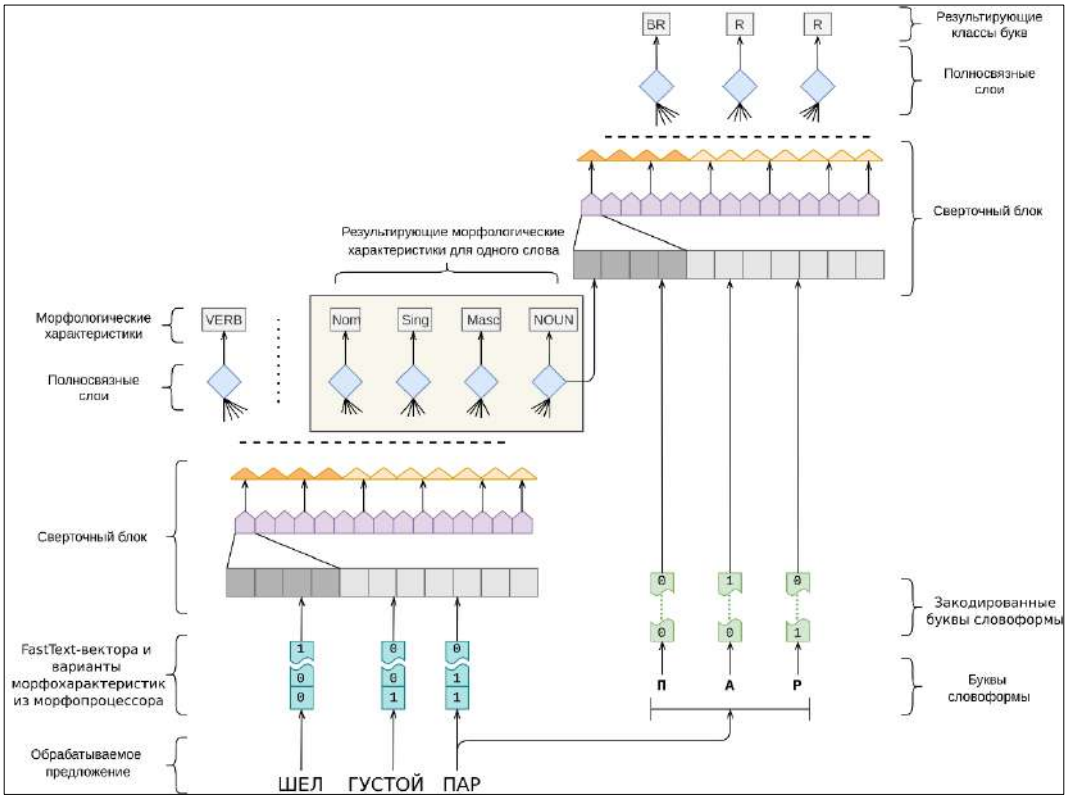


Рис. 2. Архитектура объединенной модели
Fig. 2. The architecture of the joined model

При обучении рассматриваемой объединенной модели использовалось следующее разбиение корпуса SynTagRus с морфемной разметкой, далее – Morphs-SynTagRus: 70% предложений для обучающего множества, 10% для валидационного и 20% для тестового. В ходе экспериментов с моделью было выяснено, что наилучшее качество морфологического анализа и морфемного разбора словоформ достигается при следующих гиперпараметрах: количество узлов в сверточных слоях равно соответственно 512, 256, 192, алгоритм градиентного спуска – Adam, со скоростью обучения равной 0.001. Величина исключения равна 0.3, а размер субдискретизации равен трём. Обучение такой модели занимает около 20 минут на видеокарте Nvidia Tesla T4.

Оценка модели, обученной для входных последовательностей из 9 слов, показала, что точность разрешения омонимии равна 94.2%, что несколько ниже наилучшего достижимого качества (96.5% [5]), а точность морфемного разбора по словам целиком достигает 96.5%, что значительно превосходит все предыдущие модели морфемного разбора. Заметим, что при оценке качества морфемного разбора не учитывались все слова из тестового множества короче трех букв, т.к. морфемный разбор таких слов тривиален, и оценка модели оказалась бы завышена. Однако при дополнительной валидации на датасете RuMorphs-Words модель показала значительно худший результат – 47.3% точности морфемного разбора целиком. Обнаруженная чрезмерная настройка модели на корпус Morphs-SynTagRus с морфемной разметкой объясняется в первую очередь тем, что слова в этом корпусе обладают очень низким “морфемным разнообразием”: в нем маленькое количество различных слов, большое количество коротких слов, в том числе повторяющихся или очень похожих по структуре.

Для преодоления обнаруженного недостатка был применен техника “переноса знаний” (*transfer learning*), часто используемая при создании нейронных моделей для обработки текстов. Обучение объединенной модели было разделено на 3 этапа. На первом этапе часть модели, отвечающая за морфемный разбор, обучалась отдельно на датасете RuMorphs-Words (с уже известными частями речи). На втором этапе веса в этой части нейронной модели замораживалась (т.е. исключались из обучения) и производилось обучение объединенной модели на доразмеченном корпусе Morphs-SynTagRus. На третьем этапе веса морфемной подмодели размораживались, скорость обучения устанавливалась на 2 порядка меньше, чем на втором этапе (для того, чтобы не потерять знания, полученные на этапе 1), и обучение всей объединенной модели производилось еще раз с максимальным количеством итераций равным 20 (по той же самой причине).

Таким образом, модель сохраняла знания о морфемных разборах, полученные на первом этапе обучения, и в тоже время обучалась разбирать словоформы из Morphs-SynTagRus. Это позволило добиться точности морфемного разбора 88.5% на словах из Morphs-SynTagRus и 85.9% для словоформ из RuMorphs-Words – см. табл. 3, строка 2. Последний показатель ниже наилучшего достижимого, однако заметим, что при уменьшении числа итераций на третьем этапе обучения точность морфемного разбора словоформ из RuMorphs-Words была более высокой, но при этом для Morphs-SynTagRus была ниже. Тем самым, изменяя количество итераций на третьем этапе обучения, модель можно настраивать на специфику одного или другого датасета.

Итоговое сравнение качества наилучшей модели для лемм (Bi-LSTM), CNN-модели для словоформ и объединённой модели по метрики точности сегментации с классификацией по словам целиком – см. табл. 5.

Табл. 5. Точность моделей морфемного разбора для русского языка (%)
Table 5. Accuracy of models for morphemic segmentation of Russian (%)

Модель	**RuMorphs-Lemmas**	**RuMorphs-Words**	**Morphs-SynTagRus**
Bi-LSTM (леммы)	89.03	38.57	34.49
CNN (словоформы)	90.03	91.06	-
Объединенная	85.11	85.90	88.54

Как видно из таблицы, модель Bi-LSTM для разбора лемм показывает плохое качество разбора словоформ. CNN-модель разбора словоформ показывает наилучшее достижимое качество на датасетах RuMorphs-Words и RuMorphs-Lemmas (на датасете Morphs-SynTagRus модель не оценивалась, так как с её помощью производилась разметка этого датасета). Объединенная морфологическая модель проигрывает по точности CNN-модели, хотя и не критично, однако её применение позволяет получить лучшую производительность.

Для тестирования производительности объединенной модели использовалась библиотека tensorflow-lite. Производительность модели оказалась равна 1893 слова в секунду – см. табл. 4, строка 3, что сравнимо с моделью морфемного разбора словоформ с учетом времени, затрачиваемого на определение части. Размер обученной объединенной модели составляет менее 1.5 мегабайт.

Описанная объединенная модель обучалась на входных последовательностях из девяти слов, до двадцати букв каждое. Поскольку в текстах часто встречаются короткие предложения, а также короткие слова, то при их обработке выполняются излишние вычисления (для дополненных до фиксированного размера концов таких предложений и слов). Для улучшения производительности предлагается использовать комплекс из 9 аналогичных объединённых моделей, для меньших размеров входных данных: 9 слов, 7 слов, 5 слов и, соответственно каждая из них для слов из 20 букв, 12 букв и 6 букв. Суммарный объем комплекса моделей составил около 13.5 мегабайт. При обработке входного текста делается выбор подходящей модели комплекса, т.е. размер слов в которой больше, чем во входном предложении, и количество букв в словах больше, чем у самого длинного слова. В этом случае производительность такого комплекса составила около 3543 слов в секунду, что является наилучшим результатом для морфемного разбора словоформ (табл. 4, строка 4).

6. Заключение

Разработаны и экспериментально исследованы две нейросетевые модели, реализующие морфемный разбор словоформ русского языка. Их эффективность оценивалась одновременно по двум аспектам: точности морфемного разбора и затратам по времени работы и памяти (по производительности, вычисляемой в словах в секунду, и по объему памяти). Сверточная модель морфемного разбора словоформ показывает наилучшее достижимое качество морфемного разбора при достаточно высокой производительности, но требует заранее определенной части речи словоформ. Объединенная модель морфологического анализа дополнительно уточняет морфологические характеристики словоформ, в том числе часть речи. Предлагаемый комплекс подобных моделей позволяет достичь более высокой производительности морфемного разбора, но с некоторой потерей точности. Выбор модели для конкретной прикладной задачи зависит от особенностей последней. Реализованные модели встроены в открытый морфологический процессор русского языка[4].

Заметим, что производительность описанных моделей изучалась только с точки зрения архитектуры моделей машинного обучения. Дополнительное использование таких техник, как кэширование результатов анализа, квантование и удаление лишних весов, а также параллелизм может увеличить производительность на порядок.

Для обучения разработанных моделей были построены необходимые размеченные наборы данных (датасеты) со словоформами русского языка. В открытый доступ выложены как сами датасеты, так и реализованные модели морфологического анализа.[5]

[4] https://github.com/alesapin/XMorphy

[5] https://github.com/alesapin/XMorphy/tree/trying_tensorflow/scripts

Список литературы / References

[1] Большакова Е.И., Воронцов К.В. и др. Автоматическая обработка текстов на естественном языке и анализ данных: учебное пособие. Изд-во НИУ ВШЭ, 2017 г., 269 стр. / Bolshakova E.I., Vorontsov K.V. et al. Automatic processing of texts: handbook. HSE, 2017, 269 p. (in Russian)

[2] Ляшевская О.Н., Астафьева И. и др. Оценка методов автоматического анализа текста: морфологические парсеры русского языка. Труды международной конференции Диалог-2010, 2010, стр. 318-327 / Lyashevskaya O.N., Astafieva I. et al. Evaluation of automatic text analysis: morphological parsers for Russian. In Proc. of the International Conference Dialogue 2010, 2010, pp. 318-327 (in Russian).

[3] Harris Z.S. Morpheme boundaries within words: Report on a computer test. In Transformations and Discourse Analysis Papers. Formal Linguistics Series, Springer, 1970, pp. 68-77.

[4] Kanerva J., Ginter F. et al. Turku neural parser pipeline: An end-to-end system for the CoNLL 2018 shared task. In Proc. of the CoNLL 2018 Shared Task: Multilingual parsing from raw text to universal dependencies, 2018, pp. 133-142.

[5] Anastasyev D.G. Exploring pretrained models for joint morpho-syntactic parsing of Russian. In Proc. of the International Conference Dialogue 2020, 2020, pp. 1-12.

[6] Sorokin A., Smurov I., Kirianov P. Tagging and parsing of multidomain collections. In Proc. of the International Conference Dialogue 2020, 2020, pp. 670-683.

[7] Lyashevskaya O.N., Shavrina T.O. et al. GRAMEVAL 2020 Shared Task: Russian Full Morphology and Universal Dependencies Parsing. In Proc. of the International Conference Dialogue 2020, 2020, pp. 553-569.

[8] Sorokin A., Kravtsova A. Deep convolutional networks for supervised morpheme segmentation of Russian language. Communications in Computer and Information Science, vol. 930, 2018, pp. 3-10.

[9] Bolshakova E., Sapin A. Comparing models of morpheme analysis for Russian words based on machine learning. In Proc. of the International Conference Dialogue 2019, 2019, pp. 104-113.

[10] Bolshakova E., Sapin A. Bi-LSTM Model for Morpheme Segmentation of Russian Words. Communications in Computer and Information Science, vol. 1119, 2019, pp. 151-160.

[11] Сокирко А.В. Морфологические модули на сайте www.aot.ru. Труды международной конференции Диалог-2004, 2004 г., стр. 559–564. / Sokirko A.V. Morphological components on www.aot.ru. In Proc. of the International Conference Dialogue 2004, 2004, pp. 559–564 (in Russian)

[12] Korobov M. Morphological analyzer and generator for Russian and Ukrainian languages. Communications in Computer and Information Science, vol. 542, 2015, pp. 320-332.

[13] Segalovich I. A fast morphological algorithm with unknown word guessing induced by a dictionary for a web search engine. In Proc. of the International Conference on Machine Learning; Models, Technologies and Applications, 2003, pp. 273-280.

[14] Schmid H.: Probabilistic part-of-speech tagging using decision trees. In Proc. of the International Conference on New Methods in Language Processing, 1994, pp. 44-49.

[15] Straka M., Straková J., Hajic J. Prague at EPE 2017: The UDPipe system. In Proc. of the 2017 Shared Task on Extrinsic Parser Evaluation at the Fourth International Conference on Dependency Linguistics and the 15th International Conference on Parsing Technologies, 2017, pp. 65-74.

[16] Bojanowski P., Grave E. et al. Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 2017, vol. 5, pp. 135-146.

[17] Peters M.E., Neumann M. et al. Deep contextualized word representations. In Proc. of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1 (Long Papers), 2018, pp. 2227–2237.

[18] Devlin J., Chang M.-W. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2019, pp. 4171–4186.

[19] Kurimo M., Virpioja S. et al. Morpho challenge 2005-2010: Evaluations and results. In Proc. of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology, 2010, pp. 87-95.

[20] Virpioja S., Smit P. et al. Morfessor 2.0: Python implementation and extensions for Morfessor Baseline. Aalto University publication series science + technology, 2013, p. 38.

[21] Тихонов А.Н. Словообразовательный словарь русского языка. Русский язык, 1990 г., 864 стр. / Tikhonov A.N. Word Formation Dictionary of Russian language. Moscow, Russkiy yazyk, 1990, 864 p. (in Russian)

[22] OpenCorpora. URL: http://opencorpora.org/.
[23] Tensorflow – Large-Scale Machine Learning on Heterogeneous Systems. URL: https://www.tensorflow.org/.
[24] SynTagRus – Russian data from the SynTagRus corpus. URL: https://github.com/UniversalDependencies/UD_Russian-SynTagRus

Информация об авторах / Information about authors

Александр Сергеевич САПИН – аспирант кафедры алгоритмических языков факультета ВМиК. Сфера научных интересов: автоматическая обработка текстов на естественном языке, морфологический анализ и морфемный разбор слов в языках с богатой морфологией, применение машинного обучения для задач автоматической обработки текстов.

Alexander Sergeevich SAPIN is a post-graduate student of Algorithmic Languages Department, CMC Faculty. Research interests: natural language processing, morphological analysis and morpheme segmentation of words in natural languages with rich morphology, machine learning for NLP applications.

DOI: 10.15514/ISPRAS-2021-33(4)-10

Алгоритм маркирования текстовых документов на основе изменения интервалов между словами, обеспечивающий устойчивость к преобразованию формата

[1] *А.В. Козачок, ORCID: 0000-0002-6501-2008<a.kozachok@academ.msk.rsnet.ru>*
[1] *С.А. Копылов, ORCID: 0000-0003-2841-5243 <gremlin.kop@mail.ru>*
[1] *П.Н. Горбачев, ORCID: 0000-0002-4511-0348 <pavel.gorbachev@list.ru>*
[2] *А.Е. Гайнов, ORCID: 0000-0003-3887-5374 <gae@mil.ru>*
[2] *Б.В. Кондратьев, ORCID: 0000-0003-0510-651X <gae@mil.ru>*

[1] *Академия Федеральной службы охраны Российской Федерации,*
302015, Россия, г. Орёл, ул. Приборостроительная, д. 35
[2] *Министерство обороны Российской Федерации,*
119160, г. Москва, ул. Знаменка, д.19

Аннотация. В статье представлен алгоритм маркирования электронных текстовых документов, основанный на внедрении идентификационной информации за счет изменения величин интервалов между словами. Разработка алгоритма направлена на повышение защищенности документов, содержащих текстовую информацию, от утечки по каналу, обусловленному передачей напечатанных на бумаге документов, а также соответствующих электронных копий бумажных документов. В процессе разработки алгоритма маркирования проведен анализ существующих средств защиты бумажных документов от утечки, рассмотрены практические решения в области защиты текстовых документов, определены их достоинства и недостатки. В качестве подхода к внедрению информации выступает алгоритм изменения величин интервалов между словами. Изменение величин интервалов между словами основано на встраивании удлиненного пробела в выделенные области строк текста и корректировке остальных значений величин интервалов между словами на рассчитанные значения. Для обеспечения инвариантности встроенного маркера к печати и последующему сканированию или фотографированию разработаны алгоритмы формирования областей встраивания и матрицы встраивания. В процессе формирования областей встраивания из строк текста исходного документа формируются массивы пробелов, состоящие из пар: по четыре и два пробела или по два пробела. Посредством встраиваемой информации в сформированных областях определятся места встраивания удлиненного пробела. В процессе встраивания маркера формируется матрица встраивания, содержащая значения смещения слов, и осуществляется ее встраивание в исходный документ при печати. Применение разработанного алгоритма маркирования позволяет внедрять в структуру текста электронного документа маркер, инвариантный к преобразованию формата электронного документа в бумажный посредством печати и обратно через сканирование или фотографирование. Представлены особенности и ограничения разработанного алгоритма маркирования. Определены направления дальнейших исследований.

Ключевые слова: защита от утечки информации; маркирование; распознавание образов; обработка изображений; текстовые документы

Для цитирования: Козачок А.В., Копылов С.А., Горбачев П.Н., Гайнов А.Е., Кондратьев Б.В. Алгоритм маркирования текстовых документов на основе изменении интервала между словами, обеспечивающий устойчивость к преобразованию формата. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 131-146. DOI: 10.15514/ISPRAS–2021–33(4)–10

Text documents marking algorithm based on interword distances shifting invariant to format conversion

[1] *A.V. Kozachok, ORCID: 0000-0002-6501-2008 <a.kozachok@academ.msk.rsnet.ru>*
[1] *S.A. Kopylov, ORCID: 0000-0003-2841-5243 <gremlin.kop@mail.ru>*
[1] *P.N. Gorbachev, ORCID: 0000-0002-4511-0348 <pavel.gorbachev@list.ru>*
[2] *A.E. Gaynov, ORCID: 0000-0003-3887-5374 <gae@mil.ru>*
[2] *B.V. Kondrat'ev, ORCID: 0000-0003-0510-651X <gae@mil.ru>*

[1] *Academy of Federal Guard Service,*
35, Priborostroitel'naya st., Orel, 302015, Russia.
[2] *Ministry of Defence of the Russian Federation,*
19, Znamenka st., Moscow, 119160, Russia

Abstract. The article presents an electronic text documents marking algorithm based on the identification information embedding by changing the values of the intervals between words (interwords distance shifting). The algorithm development is aimed at increasing the documents containing text information security from leakage through the channel due to the transfer of documents printed on paper, as well as the corresponding electronic copies of paper documents. In the marking algorithm developing process, an existing tools analysis of protecting paper documents from leakage was carried out, practical solutions in the field of protecting text documents were considered, their advantages and disadvantages were determined. The interwods distance shifting algorithm acts as an approach to the information embedding in electronic documents. Changing the values of interwords distance is based on embedding the normalized space in the selected areas of text lines and adjusting the remaining values of the spacing between words by the calculated values. To invariance ensure of the embedded marker for printing and subsequent scanning or photographing, formation algorithms of embedding regions and embedding matrix have been developed. In the embedding regions forming process from the text lines of the source document, arrays of spaces are formed, consisting of pairs: four and two spaces or two spaces. By means of the embedded information in the formed areas, the places where the normalized space is inserted is determined. In the embedding a marker process, an embedding matrix is formed, containing the values of the word displacement, and it is embedded in the original document in the process of printing. The developed marking algorithm usage makes it possible to introduce a marker into the electronic document text structure that is invariant to the format transformation of an electronic document into a paper one and vice versa. In addition, the developed marking algorithm features and limitations are presented. Directions for further research identified.

Keywords: information leakage protection; marking; pattern recognition; image processing; text documents

For citation: Kozachok A.V., Kopylov S.A., Gorbachev P.N., Gaynov A.E., Kondrat'ev B.V. Text documents marking algorithm based on interword distances shifting invariant to format conversion. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021. pp. 131-146 (in Russian). DOI: 10.15514/ISPRAS–2021–33(4)–10

1. Введение

Совершенствование средств защиты информации является одним из наиболее актуальных направлений развития подходов к обеспечению информационной безопасности. Стремительный переход всех сфер общественных отношений в информационное пространство позволил не только повысить степень автоматизации предоставляемых услуг и сервисов конечным пользователям, но и выявить существующие ограничения и дефекты, присущие средствам защиты.

Анализ инцидентов информационной безопасности, связанных с утечкой конфиденциальной информации и персональных данных показал, что за 2020 год в мире зарегистрировано 2395 случаев утечки информации ограниченного доступа, при этом на Россию приходится примерно 17 процентов (404 случая утечки) от общего числа [1, 2]. По типу нарушителя наибольшее число утечек осуществлено за счет действий внутренних нарушителей (79 %), которые в большинстве случаев осуществлены умышлено. Наибольшее распространение среди каналов утечки получили: сетевые (61%), службы мгновенных сообщений,

осуществляющих передачу текстовых, голосовых и видео сообщений (19,7%), а также канал утечки бумажных документов (15%). Реализация столь значительного числа нарушений информационной безопасности стала возможна ввиду ориентированности существующих средств защиты на противодействие вредоносным и противоправным действиям внешних нарушителей.

Для защиты от утечек широкое распространение получили средства межсетевого экранирования и системы предотвращения утечек информации (Data Loss/Leak Prevention, DLP-системы). Межсетевой экран (брандмауэр, firewall) – аппаратное и\или программное средство, осуществляющее анализ, мониторинг и фильтрацию проходящего сетевого трафика согласно установленных правил безопасности [3–5]. Применение методов блокировки и анализа передаваемого трафика на разных уровнях модели OSI средствами межсетевого экранирования позволяет снизить количество утечек, осуществляемых по сетевым каналам, а также по каналам утечки, использующим службы мгновенных сообщений, осуществляющих передачу текстовых, голосовых и видео сообщений. В то же время указанные средства защиты не способны противодействовать утечкам конфиденциальной информации по каналу, обусловленному преобразованием формата электронного документа, содержащего текстовую информацию, в бумажный документ посредством печати и обратно – в цифровое изображение посредством сканирования распечатанного документа.

Для устранения указанного недостатка могут быть применены DLP-системы, осуществляющие идентификацию конфиденциальных данных, направляемых в другие сети, находящиеся в распределенных репозиториях (банках данных), а также хранящихся у конечного пользователя [6–9]. В зависимости от используемого метода защиты DLP-системы могут быть разделены на: сигнатурные методы (технология цифровых отпечатков пальцев), методы поиска по словарю, а также методы маркирования. Применение технологии цифровых отпечатков пальцев позволяет добиться высокой точности обнаружения в исследуемом трафике конфиденциальной информации. К недостаткам данной технологии относится отсутствие стойкости сигнатуры (шаблона) к применению преобразований и внесению искажений (в том числе и преднамеренных), а также невозможность выделения (формирования) сигнатуры для некоторых типов данных. Кроме того, сигнатурные методы и методы поиска по словарю накладывают на процесс обнаружения требование по созданию и ведению актуальной базы сигнатур или шаблонов.

В отличие от технологии цифровых отпечатков пальцев, в процессе использования технологии маркирования осуществляется встраивание маркера (идентификационной информации) в исходный документ [10]. Наличие встроенного маркера позволяет повысить стойкость защищаемой информации к осуществлению различных преобразований и внесению искажений. Указанная особенность позволяет использовать методы маркирования для повышения защищенности бумажных документов от утечки.

Анализ существующих решений в области маркирования документов, выводимых на печать, показал, что для защиты бумажных документов от утечки существуют следующие программные продукты Trace Doc [11], EveryTag [12] и SafeCopy [13]. Однако существенным недостатком всех продуктов является необходимость создания защищенной базы данных, содержащей или оригиналы электронных текстовых документов, подготовленных к печати, или их подписанные копии. Наличие указанного недостатка не позволяет предотвратить утечку бумажного документа, не имеющего подписанного электронного оригинала.

Для устранения указанного недостатка и повышения защищенности бумажных документов, а также соответствующих электронных копий от утечки необходимо разработать алгоритм маркирования текстовых документов, подготовленных к печати, обеспечивающий инвариантность встроенной информации к осуществлению преобразования формата и возможность извлечения встроенного маркера только из подписанного документа.

2. Алгоритм маркирования текстовых документов на основе изменения интервалов между словами

В качестве предложенного подхода к маркированию электронных документов, содержащих текстовую информацию, в процессе печати выступает подход к внедрению информации в структуру и формат текста за счет горизонтального сдвига слова внутри строки [14].

Подход к маркированию электронных документов, основанный на изменении положения (горизонтального сдвига, изменения величины интервала между словами) слов внутри текстовых строк, используется в работах Хуанга (Ding Huang) и др. [15], Алаттара (Adnan M. Alattar) и др. [16], Яна (Huijuan Yang) и др. [17].

В работе [15] в качестве электронного документа, в который осуществляется встраивание, выступает электронное текстовое изображение (изображение, содержащее исходный текстовый документ). Встраивание информации реализуется посредством изменения расстояния между словами и длин каждого слова таким образом, чтобы среднее расстояние между словами в каждой строке представляло собой синусоидальную волну определенной фазы и частоты. К достоинствам предложенного подхода относятся: возможность извлечения встроенной информации только из подписанных изображений, стойкость к осуществлению преобразования "печать-сканирование", фильтрация изображения и поворот на небольшие углы. Существенным недостатком предложенной схемы маркирования является возможность внедрения информации только в тексты, выравненные по ширине, а также требование по наличию информации о фазе и частоте синусоидальной волны на этапе извлечения встроенной информации.

По аналогии с исследованием [16], в работе Алаттара [15] представлена схема маркирования электронных текстовых документов за счет изменения величин интервалов между словами, позволяющая осуществлять извлечение встроенной информации, как из электронных документов, так и из соответствующих им бумажных копий. В процессе встраивания информации осуществляется увеличение или уменьшение величины интервала между словами на установленное значение в зависимости от значения символа встраиваемой последовательности. Предложенная схема позволяет повысить величину предельно достижимой емкости встраивания до 300 бит на страницу текста формата Letter. В свою очередь, процесс извлечения характеризуется наличием "пачек" ошибок (последовательно идущих ошибок), что накладывает требование по использованию помехоустойчивых кодов, обладающих свойством перемежения. Применение подобных кодов, в свою очередь, снижает размер встраиваемой информации.

В отличие от исследований [15, 16], в работе [17] предложена схема изменения величин интервалов между словами, основанная на измерении не только интервалов между словами, но и интервалов между символами на краях слов. В процессе внедрения осуществляется детектирование смежных друг с другом слов, а также интервалов, как между словами, так и между соседними символами, находящихся на краях данных слов. Посредством полученных значений осуществляется увеличение или уменьшение интервала между словами на установленное значение. Разработанный алгоритм позволяет осуществлять встраивание и извлечение встроенной информации из текстовых изображений и характеризуется высокой точностью извлечения. При этом в случае извлечения встроенной информации из отсканированных или сфотографированных изображений возрастает количество ошибок, связанных с ошибочным детектированием положения краевых символов. Указанные ошибки возникают из-за объединения соседних слов или слова и предлога воедино.

Проведенный анализ разработанных подходов к маркированию электронных документов, основанных на изменении интервалов межу словами, позволяет сделать вывод о наличии ограничений в процессе защиты бумажных документов от утечки, обусловленной преобразованием формата бумажного документа в электронный вид посредством применения операции "печать-сканирование" или "печать-фотографирование". В связи, с чем

для повышения точности извлечения и невидимости встроенных данных предложена схема маркирования $4BLK+2$. В качестве областей встраивания выступают последовательности из четырех или двух последовательно идущих пробелов. Встраивание информации осуществляется за счет изменения величины пробелов между словами следующим образом: последовательности из четырех пробелов соответствует два символа встраиваемой информации, двум пробелам – один символ. Применение данной схемы позволяет учесть недостатки подходов маркирования, описанных в работах [15–16].

Разработанный алгоритм маркирования электронных документов представлен на рис. 1.

Data: Документ D_O, встраиваемая информация L
Result: Подписанный документ D_M

$Im \leftarrow$ **DocumentConverting** (D_O)
$Lines \leftarrow$ **TextDetection** (Im)
$N \leftarrow |Lines|$
for $l \leftarrow 1$ **to** N **do**
  if $(|Lines[l]| > 1)$ **then**
    $SP[l] \leftarrow$ **CalcSpaceInLine**$(Lines[l])$
    $Long[l] \leftarrow$ **LongSpace**$(\frac{\sum SP[l]}{|SP[l]|})$
    $BLK = \max(\{bl \in \overline{0, |SP[l]|} : bl \mod 4 \equiv 0\})$
    $\{CS^4[l]\}, CS^2[l] \leftarrow$ **DevideSpaces**$(SP[l])$,
    $\forall cs_a \in \bigcup CS^4[l]\ \exists ss_b \in SP[l]: cs_a \equiv ss_b, a = b, b \leqslant BLK$
    $\forall cs_a \in CS^2[l]\ \exists ss_b \in SP[l]: cs_a \equiv ss_b, a = b, b > BLK, b \leqslant BLK + 2$
    $\forall cs \in CS^4[l]: |cs| = 4$
    $|CS^2[l]| \subset \{0, 2\}, |\bigcup CS^4[l] \cup CS^2[l]| \subset \{|SP[l]|, |SP[l]| - 1\}$
  else
    $SP[l], CS^4[l], CS^2[l] \leftarrow \varnothing$
$E \leftarrow$ **ReedSolomon** (L)
$m \leftarrow 0$
for $l \leftarrow 1$ **to** N **do**
  for $k \leftarrow 1$ **to** $|CS^4[l]|$ **do**
    if $(CS^4[l][k] \neq \varnothing)$ **then**
      switch $(E_{\mathbf{GF}(m,|E|)}, E_{\mathbf{GF}(m+1,|E|)})$ **do**
        case $0, 0$ **do**
          $CS^4[l][k] \leftarrow [Long[l], 0, 0, 0],\ m = m + 2$
        case $0, 1$ **do**
          $CS^4[l][k] \leftarrow [0, Long[l], 0, 0],\ m = m + 2$
        case $1, 0$ **do**
          $CS^4[l][k] \leftarrow [0, 0, Long[l], 0],\ m = m + 2$
        otherwise do
          $CS^4[l][k] \leftarrow [0, 0, 0, Long[l]],\ m = m + 2$
  if $(CS^2[l] \neq \varnothing)$ **then**
    if $(E_{\mathbf{GF}(m,|E|)} == 0)$ **then**
      $CS^2[l] \leftarrow [Long[l], 0],\ m{+}{+}$
    else
      $CS^2[l] \leftarrow [0, Long[l]],\ m{+}{+}$
  $av[l] \leftarrow 0$
  if $(SP[l] \neq \varnothing)$ **then**
    $av[l] \leftarrow \frac{\sum SP[l] - \sum \sum SP^4[l] - \sum SP^2[l]}{|SP[l]|}$
  $MLines[l] \leftarrow (CS^4[l], CS^2[l], av[l])$
$D_m \leftarrow$ **ChangeSpaces**$(D_O, Lines, MLines)$
return D_M

Рис. 1. Алгоритм маркирования текстовых электронных документов, основанный на изменении интервалов между словами

Fig. 1. Electronic text document marking algorithm based on interword distances shifting

Применение указанного подхода позволяет осуществить перцептивно невидимое внедрение информации, не внося изменений в структуру, семантику и синтаксис исходного документа. Кроме того, алгоритмы данной группы характеризуются высокой невидимостью встроенной

информации к визуальному анализу, а также емкостью встраивания достаточной для внедрения 50-200 бит информации.

Алгоритм маркирования электронных документов, содержащих текстовую информацию – электронных текстовых документов, подготовленных к печати, состоит из:

- преобразования формата электронного текстового документа в изображение;
- выделения строк текста из текстовых областей изображения;
- выделения областей встраивания;
- кодирования встраиваемой информации;
- формирования матрицы встраивания на основе закодированной информации и областей встраивания;
- смещения областей изображение соответствующих словам в выделенных текстовых строках согласно рассчитанной матрице встраивания.

На первом этапе (шаг 1 алгоритма маркирования) электронный текстовый документ D_O подвергается конвертации формата в изображение Im. Под электронным текстовым документом в контексте работы понимается документ, содержащий сплошной текст, текст, разбитый на графы, а также другую текстовую информацию, представленную в электронной форме [18, 19].

На втором этапе (шаг 2 алгоритма маркирования) осуществляется формирование текстовых областей изображения и выделение строк текста *Lines* посредством функции **TextDetection**.

2.1 Выделение строк текста в текстовых областях изображения

Алгоритм выделения строк текста *Lines*, реализуемый функцией **TextDetection**, представлен на рис. 2.

Function TextDetection(Im)
1 $Im_{rast} \leftarrow$ **Rasterization**(Im)
2 $k = 1$
3 **if** $(width(Im_{rast}) < 1654)$ **then**
4 $\quad k \leftarrow \frac{1654}{width(Im_{rast})}$
5 $\quad Im_{rast} =$ **Resize**(Im_{rast}, k)
6 **if** $(width(Im_{rast}) > 9920)$ **then**
7 $\quad k \leftarrow \frac{9920}{width(Im_{rast})}$
8 $\quad Im_{rast} \leftarrow$ **Resize**(Im_{rast}, k)
9 **for** $i \leftarrow 0$ **to** 4 **do**
10 $\quad Im_{filt} \leftarrow$ **BlurGaussianFilter**($Im_{rast}, [5 \times 5]$)
11 $Im_{filt} \leftarrow$ **Dilate**($Im_{filt}, [3 \times 3]$)
12 $Im_{mono} \leftarrow$ **Binarization**(Im_{filt})
13 $Areas \leftarrow$ **DetectConnectedAreas**(Im_{mono})
14 $Links \leftarrow$ **DetectAreaLink**($Areas$)
15 $RawLines \leftarrow$ **DetectLines**($Links$)
16 **for** $l \leftarrow 1$ **to** $|RawLines|$ **do**
17 $\quad Lines[i] \leftarrow$ **SplitToWords**($RawLines[i]$)
return $Lines$

*Рис. 2. Функция выделение строк текста **TextDetection***
*Fig. 2. Text lines detection function **TextDetection***

В процессе извлечения строк текста *Lines* из изображения функцией **TextDetection** осуществляется детектирование текстовых областей. Из полученных текстовых областей формируются текстовые последовательности, составляющие строки текста исходного

документа D_O. На шаге 1 осуществляется растеризация векторных элементов изображения Im, при их наличии в соответствии с заданным разрешением печати.

Растеризованное изображение Im_{rast} подвергается процедуре нормализации изображения (шаги 2-8). В случае если ширина $width(Im_{rast})$ превышает значение в 9920, то $width$ уменьшается до значения 9920, в случае если $width(Im_{rast})$ меньше 1654, то $width$ увеличивается до этого значения. В результате нормализации формируется нормализованное растрированное изображение Im_{rast} (рис. 3а). Процедура нормализации разрешения, с одной стороны, является оптимизацией, поскольку задачи детектирования текстовых областей не требуют сверхвысокого разрешения изображения, но в то же время сложность таких методов растет нелинейно, и работа над высокодетализированным изображением приведёт к взрывному росту вычислительной сложности. С другой стороны, ограничение возможного разрешения изображения некоторым диапазоном значений позволяет на последующих шагах заложить эвристики и фиксированные значения матриц, используемых в операциях над изображением.

Рис. 3. Результаты выделения строк из изображения
Fig. 3. Lines detection results from image

Нормализованное изображение Im_{rast} подвергается фильтрации, включающую гауссовский фильтра размытия (шаги 10-11) и операцию растягивания светлых областей (шаг 12). Функция **BlurGaussianFilter** фильтрует изображениеIm_{rast} гауссовским Фильтром размытия матрицей свертки 5×5 (рис 3б). Функция **Dilate** осуществляет растягивание (расширение) светлых областей изображения Im_{filt}, используя матрицу свертки 3×3 (рис 3в). Фильтрация изображения позволяет снизить шум и повысить точность детектирования текстовых областей. Отфильтрованное изображение Im_{filt} подвергается бинаризации – формированию черно-белого изображения Im_{mono} посредством функции **Binarization** (рис. 3г). Совокупность такой последовательности трех операций так же приводит к уменьшению различий между бинаризинованными изображениями одного и того же документа, полученных из разных источников. Фактическим результатом являются остовы текстовых символов, которые необходимо объединить в совокупности, принадлежащие к одной строке, одному слову.

Выделение множества смежных областей одного цвета в изображении Im_{mono} реализовано процедурой **DetectConnectedAreas** (шаг 14), путём поэтапного обхода всех пикселей изображения с маской отсечения для уже посещенных координат. На выходе формируется система множеств, элементы которой соответствуют координатам пикселей, формирующих остовы символов. При этом, в силу особенностей языковой записи, шрифта и искажений изображения, некоторые символы могут распознаваться как последовательность из более чем одной области, например, символы "ы", ";", "й". И наоборот, когда одной области могут принадлежать последовательности из нескольких символов, к примеру, сливающиеся декоративные элементы последовательности символов «ны» для шрифтов с засечками. В процедуре **DetectConnectedAreas** реализована эвристика отбрасывания тех элементов системы множеств, которые имеют слишком малую мощность для распознаваемого

изображения, что позволяет качественно отфильтровывать лишние шумы (рис. 3г, 3д). С теоретической точки зрения такая эвристика обоснована минимальным читаемым разрешением, поскольку совокупность из 2-3 пикселей шума, оставшаяся после этапов фильтрации изображения, не может представлять собой часть изображения пригодную для интерпретации человеком в качестве символа.

Рассчитанная система множеств, передается в процедуру **DetectAreaLink**, где с каждым элементом сопоставляются его базовые характеристики, включая центр области (центроид), высоту, ширину. Затем, для близких пар символов рассчитываться информация об их связанности, включая расстояние между ними, соотношение их сторон относительно друг друга. Текстовая область представляет собой совокупность связанных контуром отдельных символов, принадлежащих одной прямой, проходящей через центр символьных контуров. Для каждого символьного контура посредством функции **DetectAreaLink** рассчитывается его центр (центроид), который соединяется с центрами соседних символьных контуров, принадлежащих одной прямой линии (рис 3е). Символьные контуры, сформированные на шаге 15, представляют собой связанные области, из которых функцией **DetectLines** формируются строки текста *RawLines* (рис 3ж). На шагах 17-18 каждая текстовая строка разделяется на слова (рис. 3з). Полученный массив текстовых строк *Lines* представляет собой результат выполнения функции **TextDetection**, на основании которого формируются области встраивания и встраиваемая последовательность сдвигов слов в тексте.

2.2 Формирование областей встраивания

На третьем этапе (шаги 3-15 алгоритма маркирования) из массива строк текста *Lines* формируются области встраивания исходного документа D_O. Область встраивания представляет собой систему множеств, элементы которой эквивалентны словам строки текста и, в свою очередь, состоят из наборов координат пикселей, образующих эти слова. Предлагаемый алгоритм требует не менее трех слов в каждой кодируемой строке, поскольку крайние слова остаются на своих позициях для сохранения выравнивания текста и минимизации визуальных искажений. Таким образом, строки из одного-двух при встраивании метки остаются без изменений и игнорируются при её извлечении.

В процессе коррекции искажений требуется выполнить сдвиг слов внутри строки, обеспечивающий изменение соотношения длин обрамляющих их пробелов, при неизменности суммы всей совокупности, что создаёт особые требования к таковой операции. Незначительное изменение пробелов между словами может оказаться недостаточным для их различия на этапе извлечения информации, в то же время использование чрезмерно длинных пробелов, потребует равного их длине сокращения остальных. В свою очередь, указанная особенность может привести к недостаточности длины пробела для разделения двух и более слов и на этапе детектирования, и они будут обнаружены как одно.

Для решения этой проблемы была предложена схема кодирования "$4BLK+2$". В соответствии с этой схемой пробелы в строке, начиная с самого первого последовательно объединяться в *BLK* (шаг 8) блоков CS^4 по четыре пробела в каждом (шаг 9). При наличии в строке остатка, из двух-трех пробелов, не вошедших в блоки CS^4, первая пара формирует один блок CS^2. Для примера, строки из трех, четырех слов формирует $BLK = 0$ блоков CS^4 и один блок CS^2, пять-шесть слов дают $BLK = 1$ блок CS^4 и ноль блоков CS^2 и так далее. Формальные условия, которым должна удовлетворять формируемая совокупность $\{CS^4[l], CS^2[l]\}$, определены в строках 10-12. В такой схеме, для обычного текста имеется существенное преобладание блоков типа CS^4, что позволяет формировать сдвиги с одним удлиненным (удлиняющимся) пробелом, при вычитании этого удлинения из трех укорачивающихся. Подобный подход уменьшает ошибки в ходе детектирования текстовых областей с чрезмерно малыми пробелами, а также связывает схему кодирования с кодовой

структурой с четным числом блоков, снижая вероятность влияния ошибок вставки или выпадения одиночного пробела в строке.

В результате на третьем этапе алгоритма маркирования для каждой строки текста l формируется область встраивания – массив значений, состоящий из следующих элементов:

- $SP[l]$ – последовательность интервалов между словами (пробелов);
- $\{CS^4[l], CS^2[l]\}$ – область встраивания, состоящая из последовательностей по 4 или 2 пробела;
- $Long[l]$ – значение удлиненного пробела.

На четвертом этапе (шаг 16 алгоритма маркирования) осуществляется помехоустойчивое кодирование встраиваемой информации L в качестве маркера E. Использование помехоустойчивого кода позволяет компенсировать ошибки, возникающие в результате искажений маркируемого документа: в результате его подписи, добавления в отпечатанный документ печатей, утраты его части или ошибок в работе детектора текста.

2.3 Особенности формирования встраиваемого маркера

В качестве встраиваемой информации L может выступать следующая информация:

- идентификатор пользователя L_1;
- метка конфиденциальности документа L_2;
- метка, содержащая информацию о времени встраивания L_3;
- идентификатор средства вычислительной техники L_4;
- информация о целостности документа L_5.

Встраиваемая метка зависит структуры и размера встраиваемой информации L. Исходя из цели разработанного алгоритма маркирования – повышение защищенности конфиденциальной информации и персональных данных от утечек по каналу утечкой бумажных документов, структура встраиваемой информации L имеет вид: $L = L_1, L_2, L_3$.

Размер встраиваемой метки зависит от емкости встраивания электронного документа D_O и не может превышать величину предельно достижимой емкости встраивания. Под емкостью встраивания понимается то количество информации, которое может быть встроено в электронный документ выбранным подходом к внедрению информации. Для электронного текстового документа D_O величина предельно равна значению $|SP[l]|, \forall l \in 1 \dots N$ и определяется количеством интервалов между словами (числом пробелов) в тексте.

Исходя из особенностей электронных документов, которые могут быть заполнены текстом полностью, наполовину или могут содержать всего несколько строк текста, целесообразно ограничить размер встраиваемого маркера 48 битами. Учитывая тот факт, что в разработанном алгоритме маркирования для внедрения 1 бита информации необходимо два интервала между словами, то для внедрения 48 бит информации требуется не менее 96 интервалов между словами (пробелов) в тексте и соответствует документу, содержащему порядка 100-120 слов.

Для обеспечения стойкости встраиваемого маркера к осуществлению преобразования формата электронного текстового документа в бумажный вид, а также повышения точности извлечения встроенного маркера на шаге 16 алгоритма маркирования происходит помехоустойчивое кодирование встраиваемой информации L кодом Рида-Соломона. Применение указанного кода к последовательности маркера E размером 48 бит снижает информационную часть до значения в 32 бита, остальные 16 бит приходятся на проверочную часть.

Для формирования маркера E исходная информация L кодируется в двоичную систему счисления, при этом под каждый компонент L отводится определенное число бит: L_1 – 14 бит, L_2, – 4 бита, L_3 – 14 бит. Полученная последовательность кодируется кодом Рида-Соломона и передается на этап формирования встраиваемой последовательности.

2.4 Формирование матрицы встраивания

На пятом этапе (шаги 17-38 алгоритма маркирования) посредством закодированной информации E и областей встраивания ($SP[l]$, $\{CS^4[l], CS^2[l]\}$, $Long[l]$) осуществляется формирование матрицы встраивания *MLines*, содержащей позиции интервалов между словами и соответствующие значениям в пикселях. Для формирования последовательности *MLines* осуществляется последовательное считывание элементов областей встраивания $\{CS^4[l], CS^2[l]\}$. Для блока из четырех пробелов $CS^4[l]$ из закодированной последовательности E извлекается очередная пара бит, определяющая номер нормального пробела $Long[l]$ в блоке $CS^4[l]$. В зависимости от последовательно считанных символов $E_{GF(m,|E|}$, $E_{GF(m+1,|E|}$ $CS^4[l]$ формируется следующим образом:

- если два первых символа последовательности E имеют значение (0,0), то первый пробел в последовательности $CS^4[l]$ заменяется на нормированный пробел $Long[l]$ строки l, остальные элементы $CS^4[l]$ остаются неизменными;
- если два первых символа последовательности E имеют значение (0,1), то значение $Long[l]$ присваивается второму элементу, при этом остальные элементы $CS^4[l]$ остаются без изменений;
- если два первых символа последовательности E имеют значение (1,0), то значение $Long[l]$ присваивается третьему элементу, при неизменности остальных элементов $CS^4[l]$;
- в остальных случаях – только четвертый элемент $CS^4[l]$ заменяется на $Long[l]$;
- после любой из замен осуществляется переход к следующему символу кодовой последовательности E.

Для блоков из двух бит $CS^2[l]$ матрицы встраивания $\{CS^4[l], CS^2[l]\}$, в зависимости от символа $E_{GF(m,|E|}$, пробел $Long[l]$ подставляется следующим образом:

- если символ последовательности E имеет значение (0), то значение $Long[l]$присваивается первому элементу, при этом второй элемент $CS^2[l]$ не изменяется;
- в противном случае – второй элемент $CS^2[l]$ заменяется на $Long[l]$, при отсутствии изменений в первом;
- после любой из замен осуществляется переход к следующему символу кодовой последовательности E.

Стоит отметить, что в случае наложения сформированной матрицы встраивания на исходный текст без осуществления компенсации остальных пробелов в строке возможно появление артефактов вида: выход строки текста за поля, искажение структуры текста, сдвиг знаков пунктуации и прочие изменения в структуре исходного текста.

Наличие указанных искажений делает встроенный маркер заметным для визуального анализа и нарушает структуру текста. Для устранения указанного недостатка используется процедура компенсации пробелов в строке. На шагах 35-37 для каждой строки рассчитываются размеры всех остальных пробелов $av[l]$. Перерасчет размеров пробела позволяет добиться того, что все нормированные пробелы приняли одинаковое значение, а сумма всех пробелов в строке осталась неизменной относительно исходного документа.

Полученные значения пробелов $av[l]$ и массив значений пробелов областей встраивания $\{CS^4[l], CS^2[l]\}$ представляют собой матрицу встраивания $MLines$. Полученная матрица

используется на этапе встраивания встраиваемой информации в величины интервалов между словами функцией **ChangeSpaces**.

2.5 Внедрение матрицы встраивания в текстовые области исходного электронного документа

На заключительном этапе алгоритма маркирования осуществляется внедрение встраиваемой информации в электронный текстовый документ посредством изменения величин интервалов между словами (значений пробелов) исходного текстового документа D_O. При этом все пробелы в тексте электронного документа заменяются по следующему правилу:

- пробелы в строке l, соответствующие позициям нормального пробела $Long[l]$ в областям встраивания $\{CS^4[l], CS^2[l]\}$, заменяются на значение равное $Long[l]$;
- значение остальных пробелов строки l корректируются (уменьшаются или увеличиваются) до значения равного $av[l]$.

Изменение величин пробелов сформированной матрицы встраивания осуществляет функция **ChangeSpaces**, представленная на рис. 4.

```
Function ChangeSpaces(D_O, Lines, MLines)
Src ← Rasterization(D_O)
Lines' ← LinkedForegroundObjects(Src, Lines)
Dst ← ClearForeground(Src, Lines')
fastMin ← 0
fastMax ← 0
for l ← 0 to |MLines| do
    CS^4, CS^2, av ← MLines[l]
    CS ← CS^4 ⋃ CS^2
    SP ← CalcSpaceInLine(Lines[l])
    cur ← 0
    for k ← 1 to |CS| do
        if (CS[k] == 0) then
            CS[k] ← av
        if (k > 0 and k < |CS| − 1) then
            sp ← SP[k]
            cur ← (CS[k − 1] − (sp − cur))
            offset[k] ← Round(cur)
        else
            offset[k] ← 0
        fast[offset[k]] ← fast[offset[k]] ⋃ GetWordPixelMask(Lines'[l], k)
        fastMin ← min(fastMin, offset[k])
        fastMax ← max(fastMax, offset[k])
for l ← fastMin to fastMax do
    if (fast[l] ≠ ∅) then
        mask ← [Height(Src) × Width(Src)]
        for l ← fastMin to fastMax do
            if (Width(fast[l][k]) + l < Width(Src)) then
                mask[fast[l][k]] ← 1
        mask ← Dilate(mask, [5 × 5])
        locs ← {pix ∈ mask : mask[pix] ≠ 0}
        Dst[locs + l] ← Src[locs]
    TD_M ← Dst
return TD_M
```

Рис. 4. Функция изменения величин между словами ***ChangeSpaces***
Fig. 4. Interword distances shifting function ***ChangeSpaces***

В начале процесса внедрения матрицы встраивания проводится этап дополнительного детектирования шумов, на котором все объекты, не соотнесенные со строками детектором текста, связываются с ближайшим словом и включаются в его состав. На шаге 1 исходный текстовый документ D_O конвертируется в изображение и растеризуется. Из растеризованного

изображения Src и извлеченного на предыдущих этапах массива строк текста $Lines$ посредством функции **LinkedForegroundObjects** выделяется массив строк $Lines'$, содержащий объекты, не соотнесенные со строками $Lines$. Этот шаг позволяет соотнести со словами любые элементы изображения, которые на этапах детектирования текстовых областей могли быть отфильтрованы в качестве шума, и обеспечивает минимизацию визуальных искажений при изменении местоположения слов.

После этого на шаге 3 рассчитывается фоновое изображение, т.е. исходное множество пикселей Src приводится к такому виду, что значение цвета для всякого элемента множество Src, входящего в состав одного из подмножеств $Lines'$, замещается усреднением цвета ближайших к нему по матрице смежности элементов Src не принадлежащих какому либо подмножеству $Lines'$. Для каждой строки l из маски встраивания $Mlines$ выделяются значения CS^4, CS^2 и av. CS^4 и CS^2 формируют маску встраивания строки CS (шаг 8). Функция **CalcSpaceInLine** вычисляет значения величин интервалов между словами в массиве строк $Lines$ (шаг 9). Исходя из полученных значений областей встраивания, на шагах 11-19 осуществляется расчет величин интервалов между словами для каждой строки текста с учетом сформированной матрицы встраивания следующим образом:

- если в последовательности CS имеются еще не рассчитанные значения, то значениям интервалов между словами (пробелов) присваивается значение $av[l]$ (шаги 13-14);
- начиная со второго и заканчивая предпоследним словом в строке, осуществляется расчет смещения относительно исходной позиции слова *cur*, полученный результат округляется до целого функцией **Round** (шаги 15-20). В результате округления для каждой строки текста формируется массив смещений $offset$;
- во всех остальных случаях формируется нулевое смещение, что соответствуют формированию немаркированного документа D_O.

Из сформированного массива смещений и рассчитанной посредством функции **GetWordPixelMask** маски слов всех строк текста формируется маска заполнения, содержащая координаты формирующих слово пикселей и соответствующие им смещения. Кроме того, вычисляется наименьшее значение массива смещения $fastMin$ и наибольшее $fastMax$ (шаги 22 и 23 соответственно).

Для осуществления внедрения матрицы встраивания в текстовые области исходного электронного документа реализуется процедура морфологического изменения границы посредством использования ядра фильтра, кратного размеру изображения (шаги 25-30). После чего изображение заполняется значениями пикселей с координатами, сдвинутыми на рассчитанное значение смещения (шаг 31), включая нулевые смещения для первых и последних слов строк. В результате работы алгоритма маркирования формируется подписанный документ D_M, подготовленный к печати, т.е. документ, содержащий встраиваемую информацию L (маркер).

Предлагаемый на шагах 20-31 подход не только реализует разработанный алгоритм, но и обеспечивает оптимизацию, связанную с внедрением специальных переменных $fast$, $fastMin$, $fastMax$, позволяющих сгруппировать операции сдвига на идентичные значения, что актуально в виду их дискретности. Такая оптимизация уменьшает число дорогостоящих операций морфологического искажения (шаг 29), извлечения координат по маске (шаг 30), а также локализует операции доступа к памяти (шаг 31), увеличивая вероятность попадания в процессорные кэши.

Разработанный алгоритм маркирования, основанный на изменении величин интервалов между словами, позволяет осуществлять внедрение идентификационной информации в различные документы, содержащие текстовую информацию и подготовленные к печати. Стоит отметить, что, как и любой алгоритм маркирования, разработанный алгоритм имеет некоторые особенности.

3. Особенности разработанного алгоритма маркирования

Отличительная особенность разработанного алгоритма маркирования состоит в том, что алгоритм может быть применим только к документам, содержащим текст или текст, разбитый на графы, а также другую текстовую информацию. Помимо описанной, разработанный алгоритм характеризуется следующими особенностями [20, 21]:

- наличие стойкости встроенного маркера к осуществлению печати документа, а также применения операции сканирования или фотографирования к полученному бумажному документу;
- возможность маркирования любого используемого формата документа (.doc, docx, .odt, .pdf, .png, .jpg, и др.), а также отсутствие зависимости процесса внедрения от используемого текстового редактора (при работе с текстовыми документами);
- высокой емкостью встраивания информации, т. к. для встраивания 1 бита информации требуется только 2 интервала между словами (пробела) или 3 слова;
- отсутствие стойкости встроенного маркера к применению средств оптического распознавания символов.

Кроме рассмотренных особенностей необходимо описать процесс излечения из бумажных документов встроенной информации. Извлечение встроенной информации реализуется аналогичным образом, что и встраивание. Стоит отметить, что извлечение встроенной информации реализовано для цифровых изображений, полученных посредством сканирования или фотографирования бумажных документов.

При этом применение указанных операций имеет следующие особенности:

- процесс сканирования характеризуется внесением в конечное изображение таких искажений как геометрические преобразования (поворот, масштабирование, сдвиг, обрезка и др.) и шумы сканера (дополнительные пиксели или области изображения, не присутствовавшие в бумажном документе);
- процесс фотографирования помимо внесения геометрических искажений в трех плоскостях (поворот, отклонение и наклон камеры) характеризуется различными видами искажений, вносимых объективом фотоаппаратуры и условиями съемки (дисторсия, прогрессирующее снижение освещенности, хроматические аберрации, муар и прочие).

Наличие этапа предварительной обработки в процессе выделения строк текста, реализуемого функцией **TextDetection**, позволяет устранить лишь некоторые из представленных искажений. Для устранения указанного недостатка необходимо разработать алгоритм извлечения встроенной информации из изображений соответствующим бумажным документам. Разработка алгоритма извлечения встроенной информации позволит осуществить количественную оценку точности извлечения, устойчивости встроенного маркера к осуществлению преобразований и внесению искажений и осуществить сравнительный анализ полученных результатов с аналогами.

4. Заключение

Предложенный подход к маркированию электронных документов, выводимых на печать, позволяет повысить защищенность бумажных документов, содержащих конфиденциальную информацию, а также персональные данные, в случае неконтролируемого распространения или передачи документов лицам, не имеющим к ним легитимного доступа. Кроме того, наличие встроенной информации позволяет однозначно установить факт утечки, а также идентифицировать нарушителя. Разработанный алгоритм может применяться в DLP-системах и системах расследования инцидента информационной безопасности. При этом

разработка дополнительных этапов процесса извлечения, направленных на снижение влияния процесса сканирования (фотографирования) на точность извлечения встроенной информации, а также экспериментальная оценка основных параметров алгоритма маркирования и сравнение полученных результатов с существующими аналогами является направлением дальнейших исследований.

Список литературы / References

[1]. Россия: утечки информации ограниченного доступа, 2020 год. InfoWatch. 2021, 30 стр. / Russia: Restricted Information Leaks, 2020. InfoWatch. 2021, 30 p. Available at: https://www.infowatch.ru/analytics/analitika/rossiya-utechki-informatsii-ogranichennogo-dostupa-2020-god, accessed 10.08.2021 (in Russian).

[2]. Исследование утечек информации ограниченного доступа в 2020 году. InfoWatch. 2021, 40 стр. / Research on restricted information leaks in 2020. InfoWatch. 2021, 40 p. Available at: https://www.infowatch.ru/analytics/analitika/issledovanie-utechek-informatsii-ogranichennogo-dostupa-v-2020-godu, accessed 10.08.2021 (in Russian).

[3]. Mukkamala P. P., Rajendran S. A survey on the different firewall technologies, International Journal of Engineering Applied Sciences and Technology, vol. 5, issue 1, 2020, pp 363-365.

[4]. Neupane K., Haddad R., Chen L. Next Generation Firewall for Network Security: A Survey. In Proc. of the SoutheastCon 2018, 2018, pp. 1-6.

[5]. Sharma R. K., Kalita H. K., Issac B. Different firewall techniques: A survey. In Proc. of the Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT). 2014, pp. 1-6.

[6]. Lopez G., Richardson N., Carvajal J. Methodology for Data Loss Prevention Technology Evaluation for Protecting Sensitive Information. Revista Politécnica, vol. 36, no. 3, 2015, pp. 60-69.

[7]. Alneyadi S., Sithirasenan E., Muthukkumarasamy V. A survey on data leakage prevention systems. Journal of Network and Computer Applications, vol. 62, 2016, pp. 137-152.

[8]. Jadhav P., Chawan P. M. Data Leak Prevention system: A Survey. International Research Journal of Engineering and Technology, vol. 6, no. 10, 2019, pp. 197-199.

[9]. Kozachok A.V., Kopylov S.A. et al. Text marking approach for data leakage prevention. Journal of Computer Virology and Hacking Techniques, vol. 15. no. 3, 2019, pp. 219-232.

[10]. Козачок А.В., Кпылов С.А. и др. Подход к извлечению робастного водяного знака из изображений, содержащих текст. Труды СПИИРАН, вып. 5(60), 2018 г., стр. 128-155 / Kozachok A.V., Kopylov S.A. et al. An Approach to a Robust Watermark Extraction from Images Containing Text. SPIIRAS Proceedings, issue 5(60), 2018, pp. 128-155 (in Russian).

[11]. Trace Doc. Available at: https://secretgroup.ru/trace-doc, Accessed 10.08.2021.

[12]. Unique Interface. EveryTag. Available at: https://everytag.ru/ui, Accessed 10.08.2021.

[13]. Safe Copy. Available at: https://www.niisokb.ru/products/safecopy, Accessed 10.08.2021.

[14]. Jalil Z., Mirza A.M. A Review of Digital Watermarking Techniques for Text Documents. In Proc. of the International Conference on Information and Multimedia Technology, 2009, pp. 230-234.

[15]. Huang D., Yan H. Interword distance changes represented by sine waves for watermarking text images, IEEE Transactions on Circuits and Systems for Video Technology, vol. 11, no. 12, 2001, pp. 1237-1245.

[16]. Alattar A.M., Alattar O.M. Watermarking electronic text documents containing justified paragraphs and irregular line spacing, In Proc. of the Conference on Security, Steganography, and Watermarking of Multimedia Contents, 2004, pp. 685-695.

[17]. Yang H., Kot A.C. Text document authentication by integrating inter character and word spaces watermarking, In Proc. of the IEEE International Conference on Multimedia and Expo (ICME), 2004, pp. 955-958.

[18]. Национальный стандарт Российской Федерации. Система стандартов по информации, библиотечному и издательскому делу. Делопроизводство и архивное дело. Термины и определения, ГОСТ Р 7.0.8–2013, Стандартинформ, 2013 г., 16 с. / National standard of the Russian Federation. System of standards on information, librarianship and publishing. Records management and organization of archives. Terms and difinitions, GOST R 7.0.8–2013, Standartinform, 2013, 16 p. (in Russian).

[19]. Национальный стандарт Российской Федерации. Единая система конструкторской документации. Общие требования к текстовым документам, ГОСТ Р 2.105–2019, Стандартинформ, 2019 г., 35 с. /

National standard of the Russian Federation. Unified system for design documentation. General requirements for textual documents, GOST R 2.105–2019, Standartinform, 2019, 35 p. (in Russian).

[20]. Библиотека маркирования текстовых документов при печати за счет горизонтального смещения слов, Свидетельство о государственной регистрации программ для ЭВМ № 2020667592 от 24.12.2020, Россия, заявка № 2020666902 от 17.12.12020 / Library for marking text documents when printing due to horizontal displacement of words, Certificate of state registration of computer programs № 2020667592 dated 12.24.2020, Russia, application № 2020666902 dated 17.12.12020 (in Russian).

[21]. Модуль маркирования текстовых документов при печати для ОС семейства Windows, Свидетельство о государственной регистрации программ для ЭВМ № 2020667579 от 24.12.2020, Россия, заявка № 2020666721 от 17.12.12020 / Module for marking text documents when printing for Windows family OS, Certificate of state registration of computer programs № 2020667579 dated 12.24.2020, Russia, application № 2020666721 dated 12.17.12020 (in Russian).

Информация об авторах / Information about authors

Александр Васильевич КОЗАЧОК – доктор технических наук, доцент, сотрудник Академии Федеральной службы охраны Российской Федерации. Его научные интересы включают: информационная безопасность, защита от несанкционированного доступа, математическая криптография, теоретические проблемы информатики.

Alexander Vasilievich KOZACHOK – Doctor of Technical Sciences, Associated Professor. Employer of the Academy of Federal Guard Service. His research interests include: information security, unauthorized access protection, mathematical cryptography and theoretical problems of computer science

Сергей Александрович КОПЫЛОВ является сотрудником Академии Федеральной службы охраны Российской Федерации. Его научные интересы включают: методы машинного обучения, обработка цифровых изображений, текстовая стеганография.

Sergey Alexandrovich KOPYLOV is an employer of the Academy of Federal Guard Service. His research interests include machine learning methods, digital image processing, text steganography.

Павел Николаевич ГОРБАЧЕВ является сотрудником Академии Федеральной службы охраны Российской Федерации. Его научные интересы включают: информационная безопасность, методы машинного обучения, распознавание образов, текстовая стеганография, обработка изображений.

Pavel Nikolaevich GORBACHEV is an employer of the Academy of Federal Guard Service. His research interests include: information security, machine learning methods, pattern recognition text steganography and image processing.

Артур Евгеньевич ГАЙНОВ является сотрудником Министерства обороны Российской Федерации. Сфера научных интересов: безопасность информации, защита информации от несанкционированного доступа, построение информационных систем в защищённом исполнении.

Artur Evgenevich GAYNOV is an employer of the Ministry of Defence of the Russian Federation. Research interests: information security, information unauthorized access protection, information systems construction in a secure design.

Борис Владимирович КОНДРАТЬЕВ является сотрудником Министерства обороны Российской Федерации. Сфера научных интересов: информационная безопасность, защита информации от утечки по техническим каналам, сертификация программного обеспечения по требованиям безопасности информации, методы сокрытия информации.

Boris Vladimirovich KONDRAT'EV is an employer of the Ministry of Defence of the Russian Federation. Research interests: information security, protection against information leakages through technical channels, software certification in accordance with information security requirements, information hiding methods.

DOI: 10.15514/ISPRAS-2021-33(4)-11

Маркирование текстовых документов на экране монитора посредством изменения яркости фона в областях межстрочных интервалов

[1] *А.Ю. Якушев, ORCID: 0000-0001-6089-6505 <yakushev@ispras.ru>*
[1] *Ю.В. Маркин, ORCID: 0000-0003-1145-5118 <ustas@ispras.ru>*
[1] *С.А. Фомин, ORCID: 0000-0002-1151-2189 <fomin@ispras.ru>*
[1] *Д.О. Обыденков, ORCID: 0000-0002-9296-6333 <obydenkov@ispras.ru>*
[2] *Б.В. Кондратьев, ORCID: 0000-0001-6348-117X <gae@mil.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*
[2] *Министерство обороны Российской Федерации, 119160, г. Москва, ул. Знаменка, д.19*

Аннотация. Утечка значительной части электронных документов происходит путем фотографирования экрана. Для расследования таких случаев применяются технологии *data leakage prevention* (DLP), в частности, внедрение цифровых водяных знаков (ЦВЗ) в изображение на экране. В статье представлен краткий обзор существующих методов внедрения ЦВЗ. Предложен подход к маркированию изображений текстовых документов, выведенных на экран. Цифровая метка внедряется в межстрочные интервалы текста путем незначительного изменения яркости. ЦВЗ неразличим для восприятия человеком, но может быть запечатлен на цифровую камеру. Разработан алгоритм извлечения цифровой метки из фотографии экрана. Алгоритм не требует изображение исходного документа для успешного извлечения ЦВЗ. Результаты тестирования показали, что цифровая метка устойчива к преобразованиям (атакам), возникающим в ходе фотографирования экрана. Предложен метод, позволяющий оценить корректность извлечения цифровой метки при отсутствии информации о встроенной цифровой метке.

Ключевые слова: защита от утечек информации; цифровой водяной знак; маркирование документов на экране монитора; слепое извлечение ЦВЗ; устойчивость к screen-cam атакам

Для цитирования: Якушев А.Ю., Маркин Ю.В., Фомин С.А., Обыденков Д.О., Кондратьев Б.В. Маркирование текстовых документов на экране монитора посредством изменения яркости фона в областях межстрочных интервалов. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 147-162. DOI: 10.15514/ISPRAS–2021–33(4)–11

Text documents screen watermarking by changing background brightness in the interline spacing

[1] *A.Yu. Yakushev, ORCID: 0000-0001-6089-6505 < yakushev@ispras.ru >*
[1] *Yu.V. Markin, ORCID: 0000-0003-1145-5118 <ustas@ispras.ru>*
[1] *S.A. Fomin, ORCID: 0000-0002-1151-2189 <fomin@ispras.ru>*
[1] *D.O. Obydenkov, ORCID: 0000-0002-9296-6333 <obydenkov@ispras.ru>*
[2] *B.V. Kondrat'ev, ORCID: 0000-0001-6348-117X <gae@mil.ru>*

[1] *Ivannikov Institute for System Programming of the RAS,*
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia
[2] *Ministry of Defence of the Russian Federation,*
19, Znamenka Str., Moscow, 119160

Abstract. One of the most common ways documents leak is taking a picture of document displayed on the screen. For investigation of such cases data leakage prevention technologies including screen watermarking are used. The article gives short review on the problem of screen shooting watermarking and the existing research results. A novel approach for watermarking text images displayed on the screen is proposed. The watermark is embedded as slight changes in luminance into the interline spacing of marked text. The watermark is designed to be invisible for human eye but still able to be detected by digital camera. An algorithm for extraction of watermark from the screen photo is presented. The extraction algorithm doesn't need the original image of document for successful extraction. The experimental results show that the approach is robust against screen-cam attacks, that means that the watermark stays persistent after the process of taking a photo of document displayed on the screen. A criterion for watermark message extraction accuracy without knowledge about the original message is proposed. The criterion represents the probability that the watermark was extracted correctly.

Keywords: data leakage prevention; text documents screen watermarking; screen-cam robust watermarking; blind watermarking method

For citation: Yakushev A.Yu., Markin Yu.V., Fomin S.A., Obydenkov D.O., Kondrat'ev B.V. Text documents screen watermarking by changing background brightness in the interline spacing. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021. pp. 147-162 (in Russian). DOI: 10.15514/ISPRAS–2021–33(4)–11

1. Введение

С развитием информационных технологий происходит постоянное увеличение объемов мирового электронного документооборота. Значительная часть документов содержит конфиденциальную информацию, доступ к которой предназначается ограниченному кругу лиц. Однако компаниям приходится сталкиваться с проблемой утечки таких документов. Зачастую виновниками утечек являются сотрудники компаний, случайно или умышленно способствующие передаче конфиденциальных документов третьим лицам. Так, исследование компании InfoWatch [1] за 2020 год показало, что более 79% утечек информации в российских компаниях происходит по причине действий сотрудников, а не злоумышленников извне. Немалую роль в росте числа утечек документов сыграла пандемия вируса COVID-19 в 2020 году. Многие предприятия перевели большую часть своих сотрудников на удаленный режим работы, что усложнило возможность отслеживания оборота конфиденциальных документов. Значительная доля утечек документов осуществляются путем фотографирования или взятия скриншота экрана.

Для предотвращения утечек конфиденциальной информации используются технологии Data Leakage Prevention (DLP). DLP-решения представляют собой программно-аппаратные комплексы, ведущие мониторинг действий сотрудников, а также определенным образом реагирующие на эти действия. Специализированное ПО на рабочем месте сотрудника может вести журнал используемых приложений, блокировать доступ к сети Интернет, запрещать использование съемных USB-накопителей, записывать действия сотрудника на веб-камеру и т.д. В то же время существующие DLP-системы не позволяют предотвращать утечки,

возникающие в результате фотографирования экрана монитора, на котором отображается содержимое конфиденциального документа.

Для реализации такого рода утечек не нужно обладать специализированными знаниями — достаточно использовать возможности современных смартфонов. Цифровые камеры смартфонов позволяют быстро делать снимки высокого качества, а современные сотовые сети — отправлять снимки любому получателю.

Снимок экрана в дальнейшем может оказаться в публичном доступе. В этом случае его можно использовать для расследования утечки. Один из подходов к последующему проведению расследования состоит во встраивании дополнительной информации в изображение, выводимое на экран. Если эта информация попадет на снимок, эксперт по расследованию сможет ей воспользоваться. Полезными для расследования могут оказаться сведения об устройстве, на котором был открыт документ, учетная запись пользователя устройства, дата и время создания снимка. В совокупности с данными видеонаблюдения и записями в журналах эта информация позволит установить обстоятельства произошедшей утечки и выяснить, какой сотрудник ее допустил.

Встраивание дополнительной информации в изображение относится к технологии *цифровых водяных знаков* (ЦВЗ). ЦВЗ представляет собой незаметное для пользователя внедрение в файл дополнительной информации, которую впоследствии можно будет из этого файла извлечь. В настоящий момент технология внедрения ЦВЗ широко применяется в области защиты авторских прав цифровых данных различных форматов: изображений, аудио- и видеофайлов.

Далее будем отождествлять понятия ЦВЗ и *цифровых меток*, а процесс встраивания ЦВЗ в изображение будем называть *маркированием изображения*. Цифровые водяные знаки на изображениях могут быть видимыми или невидимыми. Видимый ЦВЗ представляет собой изображение, накладываемое на маркируемое изображение так, что ЦВЗ становится заметным для наблюдателя. Такие ЦВЗ используются с целью запрета копирования для сохранения авторских прав. Невидимые ЦВЗ обычно незаметны для невооруженного глаза, но может быть распознан соответствующим алгоритмом извлечения цифровой метки. Невидимые ЦВЗ применяются, когда необходимо скрыть от пользователя факт маркирования. Такие ЦВЗ относятся к стеганографии — способу передачи информации, при котором скрывается сам факт передачи. В данной статье рассматриваются именно невидимые ЦВЗ.

2. Сценарии использования систем внедрения ЦВЗ

Типовой сценарий использования системы, предотвращающей утечки посредством маркирования изображений, как правило состоит из трёх этапов: встраивание ЦВЗ в изображение, преобразование маркированного изображения, извлечение ЦВЗ из преобразованного маркированного изображения. Общая схема представлена на рис. 1.

На этапе встраивания используется алгоритм, формирующий изображение с ЦВЗ по исходному изображению и сообщению, внедряемому в ЦВЗ. В некоторых системах маркирования изображений используются схемы шифрования для защиты цифровой метки. В таких системах при встраивании ЦВЗ применяется криптографический ключ. Результатом работы алгоритма встраивания ЦВЗ является маркированное изображение. Если система маркирования встраивает невидимый ЦВЗ, маркированное изображение должно быть трудно отличимым от исходного, при визуальном восприятии человеком.

На втором этапе маркированное изображение подвергается преобразованиям. Будем называть атакой на цифровую метку любое преобразование изображения со встроенным ЦВЗ. Цифровая метка считается устойчивой к атаке, если после применения этой атаки

сохраняется возможность определения наличия ЦВЗ и его корректного извлечения. Атаки на ЦВЗ делятся на два класса: преднамеренные и непреднамеренные.

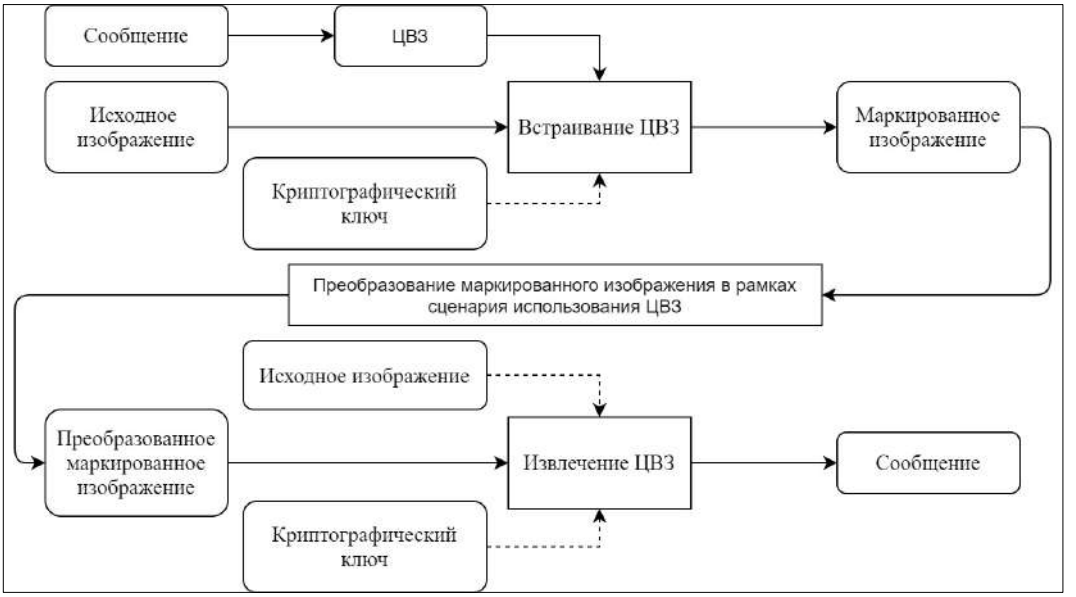


Рис. 1. Общая схема использования ЦВЗ
Fig. 1. General watermark usage scheme

Под преднамеренными атаками подразумеваются преобразования изображения человеком, знающим о наличии встроенного ЦВЗ, с целью удалить цифровую метку или изменить закодированные в ЦВЗ данные. В данной работе такие атаки рассматриваться не будут. Непреднамеренные атаки включают в себя искажения изображения, возникающие в ходе сценариев использования этого изображения. Обычно рассматриваются три сценария: сканирование напечатанных изображений («print-scan»), фотографирование напечатанных изображений («print-cam»), фотографирование изображений, выведенных на экран («screen-cam»).

На последнем, третьем этапе изображение с ЦВЗ, подвергнутое атакам, подается на вход алгоритма извлечения ЦВЗ. В зависимости от того, с какой целью применяется система маркирования изображений, результатом работы алгоритма извлечения ЦВЗ может быть:

- сообщение, внедренное в исходное изображение с помощью ЦВЗ;
- проверка корректности ЦВЗ, показывающая, было ли изображение модифицировано в ходе передачи;
- проверка факта наличия ЦВЗ.

На вход алгоритма извлечения дополнительно могут подаваться: криптографический ключ, исходное изображение, встроенный ЦВЗ. Если при извлечении ЦВЗ исходное изображение не требуется, говорят, что метод работает в режиме *слепого извлечения*.

Цифровая метка обладает рядом свойств, однако в статьях, посвященных технологии внедрения ЦВЗ, акцент главным образом делается на трех свойствах: емкость (*capacity*), незаметность (*imperceptibility*), устойчивость (*robustness*) [2]. Емкость цифровой метки означает количество битов информации, встраиваемой в исходное изображение. Незаметность цифровой метки показывает, насколько сложно человеку определить наличие ЦВЗ в маркированном изображении. Устойчивость цифровой метки характеризует сопротивляемость ЦВЗ изменениям изображения, происходящим между процессом встраивания ЦВЗ и процессом извлечения ЦВЗ. Эти три свойства цифровой метки

взаимоисключают друг друга, поэтому при разработке/применении метода маркирования необходимо достичь компромисса между ними.

3. Атаки на ЦВЗ в сценарии «screen-cam»

Изображение, полученное путем фотографирования экрана, существенно отличается от изображения, выведенного на экран. Такое различие обусловлено большим количеством непреднамеренных атак, возникающих в сценарии «screen-cam». Эти атаки можно разделить на три группы [3] в зависимости от того, в какой момент они возникают: вывод изображения на экран, фотографирование экрана монитора, обработка фотографии (рис. 2).

На качество вывода изображения на экран влияют настройки монитора: яркость, контрастность, цветопередача, гамма-коррекция.

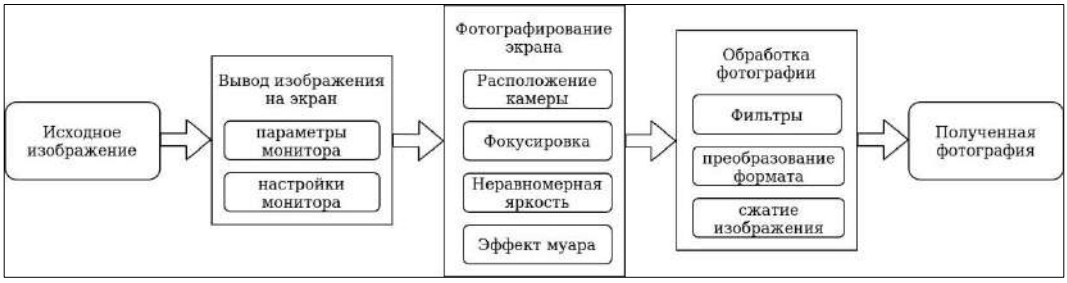


Рис. 2. Атаки на ЦВЗ в сценарии «screen-cam»
Fig. 2. Screen-cam attacks on watermark

В процессе фотографирования экрана возникают дополнительные искажения.

- Расположение камеры относительно экрана: влияет на изменение перспективы, сдвиг, изменение масштаба изображения.
- Фокусировка: если камера расположена под большим углом к плоскости экрана, разные части экрана находятся на разном расстоянии от нее, поэтому часть изображения может быть не в фокусе.
- Неравномерная яркость на фотографии: на яркость фотографии экрана влияет несколько факторов. Помимо того, что сам экран является источником света, на него может падать свет других источников. На экране также могут оказаться тени других объектов. Удаленные части экрана на фотографии будут менее яркими по сравнению с частями, расположенными близко к камере.
- Эффект муара: возникает из-за того, что пиксели экрана и сенсоры камеры расположены периодично. Плоскость экрана и плоскость матрицы камеры во время съемки обычно не строго параллельны друг другу, из-за чего на фотографии появляется нерегулярный узор, распространяющийся по всему изображению.

Полученный снимок подвергается дополнительной обработке на устройстве. Современные смартфоны в процессе обработки применяют к фотографии ряд изменяющих ее фильтров. Также производится преобразование формата и сжатие изображения (например, алгоритмом JPEG). Сжатие уменьшает размер снимка в памяти устройства, но негативно влияет на качество изображения. Оно может быть выполнено автоматически при пересылке фотографии в сети Интернет.

4. Методы маркирования, устойчивые к «screen-cam» атакам

В настоящий момент известно большое число подходов к маркированию изображений. В то же время большинство из них предназначены для использования в сценариях «print-scan» или «print-cam» и требуют значительных изменений для успешной работы в сценарии «screen-

cam». Далее будут рассмотрены методы маркирования изображений, показавших высокую устойчивость к «screen-cam» атакам на ЦВЗ.

В основу метода, описанного в статье [4], положено дискретное косинусное преобразование. Встраивание цифровой метки осуществляется в два этапа. На первом этапе определяются области встраивания ЦВЗ. Для этого используются ключевые точки, полученные с помощью алгоритма I-SIFT [5]. Около каждой ключевой точки выделяется прямоугольная область, в которую внедряется цифровая метка. Кодируемая информация встраивается в домен дискретного косинусного преобразования каждой области. Множественное встраивание применяется для того, чтобы после съемки на камеру была возможность извлечь ЦВЗ из области, наименее пострадавшей от атак на ЦВЗ. В процессе извлечения цифровой метки с фотографии алгоритмом I-SIFT (как и при встраивании) определяются ключевые точки, расположение которых устойчиво к съемке. Цифровая метка извлекается из областей около ключевых точек, определенных на фотографии.

В работе [3] применяется схожая идея поиска наиболее подходящих областей для встраивания ЦВЗ. Центрами этих областей являются точки, определенные модифицированным детектором Харисса-Лапласа [6]. Для каждой такой точки определяется вектор, задающий поворот и размер квадрата, выступающего в качестве области для встраивания. Цифровая метка внедряется в домен дискретного преобразования Фурье найденных квадратных областей с использованием псевдослучайной последовательности, расположенной на окружности, соответствующей средним частотам домена Фурье. Положение центров квадратов, направление и размер векторов, задающих эти квадраты (длину стороны и угол поворота), а также средние частоты в домене Фурье обладают высокой устойчивостью к атакам сценария «screen-cam», что позволяет успешно извлекать встроенный ЦВЗ.

Рассмотренные методы обладают общей чертой: цифровая метка, внедряемая в изображение, встраивается в домене преобразований. В статье [7] также предлагается подход, выполняющий встраивание ЦВЗ в домене преобразования Фурье. Цифровая метка, встроенная в домене преобразования, создает характерный шум на изображении. Этот шум незаметен на богатых цветами и деталями изображениях, но хорошо различим на однотонных изображениях, в частности, на текстовых документах. В работе [4] показано, что предложенный в [4] ЦВЗ, встроенный в домене дискретного косинусного преобразования, хорошо заметен на изображении текстового документа. Похожий эффект возникает после внесения изменений в домене Фурье изображения текста на белом фоне (рис. 3).

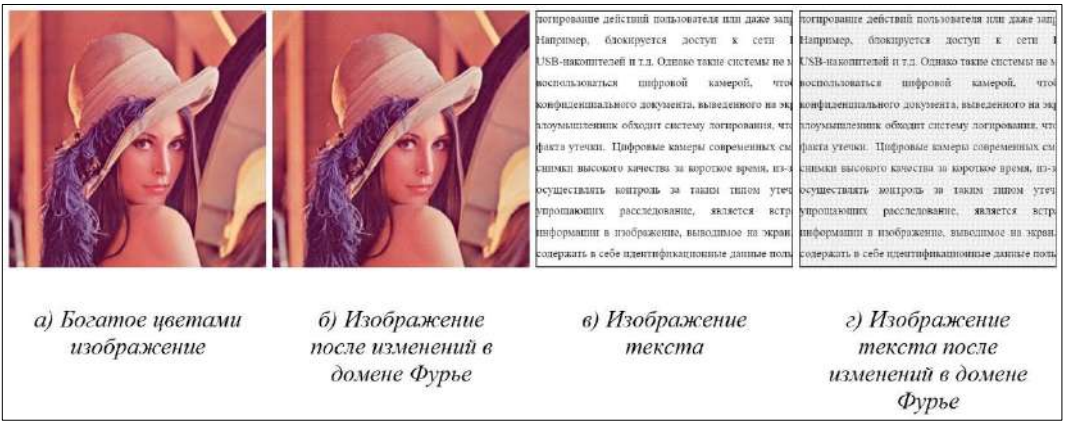


Рис. 3. Сравнение изображений после внесения изменений в домене Фурье
Fig. 3. Images comparison after changes in DFT domain

Метод, предложенный в статье [8], предназначен для маркирования любых изображений, выведенных на экран, в том числе текстовых документов. Идея подхода основана на том, что

зрительная система человека слабо восприимчива к небольшому непрерывному изменению яркости, в то время как камера способна его различить. Цифровая метка встраивается путем уменьшения или увеличения яркости областей на экране. Маска яркости, накладываемая на изображение на экране, рассчитывается заранее и зависит от кодируемой информации, но не от содержимого изображения на экране. Такой подход позволяет использовать маску в режиме реального времени, не расходуя при этом вычислительные ресурсы системы на расчет ЦВЗ. Авторам удалось добиться работы метода в режиме слепого извлечения. Каждому биту информации сопоставляется круговой маркер в накладываемой маске. Область в центре круга делается ярче или темнее чем область у его границы – в зависимости от значения кодируемого им бита. Незаметность ЦВЗ достигается за счет плавного изменения яркости в круге. Однако маркеры остаются заметными на белом фоне (наиболее распространенном для документов). В ходе извлечения ЦВЗ из фотографии экрана определяется положение кругов и для каждого круга вычисляется разность яркости в центре и на границе.

5. Разработка метода маркирования текстовых документов на экране

Проведенный анализ существующих методов маркирования текстовых документов указывает на необходимость разработки собственных алгоритмов внедрения и извлечения ЦВЗ. Под текстовым документом далее подразумевается изображение, содержащее несколько строк текста, расположенных периодично на однотонном фоне (как правило, черный текст на белом фоне). При этом строки текста могут быть ориентированы не горизонтально, а под некоторым углом.

На основе анализа существующих методов маркирования были выделены следующие требования к системе:

- Цифровая метка в документе должна быть незаметной для пользователя.
- ЦВЗ должен встраиваться в документ в режиме реального времени: это значит, что метка должна соответствовать содержимому изображения на экране в момент получения снимка экрана.
- ЦВЗ должен быть устойчивым к атакам, возникающим в сценарии «screen-cam».
- Цифровая метка должна содержаться во всех текстовых документах, отображенных на экране, вне зависимости от формата файла документа и приложения, посредством которого осуществляется работа над документом (например, документы Microsoft Word, документы в формате PDF, изображения сканированных документов).
- Цифровая метка должна извлекаться в случае, когда на фотографии запечатлена только часть экрана, содержащая текстовый документ.
- Алгоритм извлечения ЦВЗ должен быть способен извлекать метку, получив на вход только фотографию экрана (режим слепого извлечения).

5.1 Структура цифровой метки

Исходя из требований к ЦВЗ, был разработан подход к маркированию текстовых документов, выводимых на экран монитора. Емкость сообщения, внедряемого в ЦВЗ, составляет 32 бита. Цифровая метка встраивается в межстрочные интервалы маркируемого текста, в качестве последовательности светлых и темных прямоугольных областей (рис. 4). Будем называть эти области *маркерами*. В начало, середину и конец межстрочного интервала встраиваются специальные двойные маркеры, состоящие из двух частей разных цветов (выделены синей рамкой на рисунке), причем маркер середины «противоположен» маркерам начала и конца.

Эти маркеры используются для определения положения маркеров, кодирующих биты сообщения, в процессе извлечения ЦВЗ из фотографии экрана.

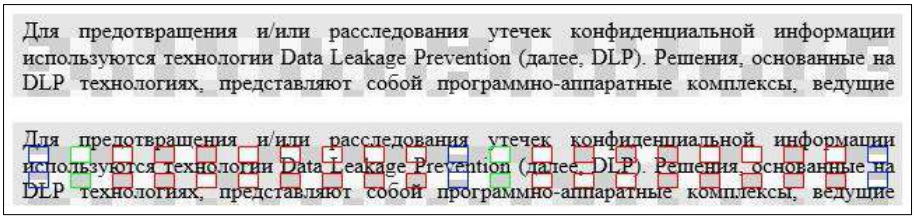


Рис. 4. Структура цифровой метки
Fig. 4. Watermark structure

Каждому биту сообщения ставится в соответствие кодирующий его маркер (выделены красной рамкой на рисунке). Темные маркеры кодируют «1», светлые – «0». Внедряемое сообщение делится на 4 части по 8 бит. Каждая четверть сообщения встраивается межу соседними двойными маркерами.

Перед четвертью сообщения добавляется дополнительный бит (маркер, выделенный зеленой рамкой), указывающий, к какой половине сообщения относится следующая за ним часть сообщения. Таким образом, двойной маркер и дополнительный бит, предшествующие последовательности из 8 маркеров, кодирующих биты сообщения, однозначно определяют, к какой четверти сообщения относится эта последовательность.

Между каждой парой маркеров, кодирующих биты сообщения, добавляется промежуточный маркер. Если соседние значимые маркеры одного цвета, промежуточный маркер между ними заполняется цветом, дополняющим до белого, а если разного — средним значением цветов соседних значимых маркеров. Такое чередование цветов используется в процессе извлечения метки.

Рис. 5. Пример маркированного текста
Fig. 5. Example of watermarked text image

Для повышения незаметности метки переходы цветом между маркерами сглаживаются посредством фильтра Гаусса, размер ядра которого подбирается исходя из размера (высоты и ширины) межстрочного интервала. Поскольку фон текстовых документов преимущественно белого цвета, общая яркость изображения понижается так, чтобы разность яркости между светлыми маркерами и белым фоном совпадала с разностью яркости между белым фоном и темными маркерами. При этом цифровая метка становится менее заметной на белом фоне. На незаметность цифровой метки влияет интенсивность маркеров, задаваемая уровнем непрозрачности. Так, при уровне непрозрачности 0% ЦВЗ будет отсутствовать, а при непрозрачности 100% темные маркеры будут абсолютно черными, а светлые – белыми (рис. 5). При промежуточных значениях непрозрачности цвет маркера будет зависеть от фона текста.

Предложенная структура ЦВЗ имеет ряд преимуществ.

- Цифровая метка, встроенная в межстрочный интервал, не влияет на качество отображаемого текста.
- Плавное изменение яркости слабо заметно для восприятия, но хорошо различимо цифровой камерой.
- Для полного встраивания 32 битного сообщения достаточно трех строк текста.
- Если текст состоит из большого числа строк, метка может быть встроена несколько раз, а ЦВЗ может быть извлечен из фотографии только части текста.
- Допустимы отклонения высоты определенного межстрочного интервала от его фактической высоты.
- Поскольку информация, встроенная в один межстрочный интервал, однозначно соответствует части сообщения и не зависит от других межстрочных интервалов, допустимы ошибки определения межстрочных интервалов на этапах встраивания/извлечения (разбиение межстрочного интервала на несколько интервалов, объединение межстрочных интервалов и строки между ними в один межстрочный интервал, определение ложных межстрочных интервалов).

5.2 Алгоритм встраивания цифровой метки

Для наложения маски с областями измененной яркости на изображение на экране используется подход, описанный в [9]: создается окно, обладающее свойствами частичной визуальной прозрачности и «прозрачности» нажатия клавиш. Это окно всегда находится на вершине стека отображаемых окон, что позволяет встраивать ЦВЗ в любой момент времени. Далее будем называть это окно оверлей. С помощью оверлея можно отобразить любое изображение, содержащее 4 канала: 3 цветовых канала красного, зеленого и синего цветов, а также альфа-канал, задающий непрозрачность каждого пикселя изображения в оверлее. Итоговые значения цветовых каналов изображения, выводимого на экран, получаются как линейная комбинация изображения на оверлее и изображения, составленного другими окнами.

Положение окон на экране, а также их содержимое меняется в зависимости от действий пользователя. Содержимое оверлея необходимо регулярно обновлять, чтобы цифровая метка соответствовала измененному изображению. Процесс встраивания ЦВЗ состоит из следующих этапов:

1) получение списка окон и их положения в стеке;
2) получение скриншотов видимых частей окон, подлежащих маркированию;
3) определение областей с текстом на полученных изображениях;
4) определение углов поворота текстовых областей;
5) определение положения межстрочных интервалов;

6) отображение цифровой метки на оверлее в областях, соответствующие межстрочным интервалам нижележащего маркируемого текста.

На первом этапе посредством оконного менеджера ОС определяется список открытых окон, их положение в стеке окон, координаты прямоугольников окон. Из этого списка исключаются окно-оверлей, а также окна, заведомо не содержащие текст. На основе полученной информации определяется, какие окна видны пользователю. Для дальнейшей обработки получаются скриншоты видимых частей окон.

Второй этап – определение областей с текстом на полученных скриншотах. Для этого используется предварительно обученная нейронная сеть. В основу нейронной сети положена архитектура U-Net [10]. С целью ускорения работы алгоритма было сокращено число слоев нейронной сети, а также число каналов в промежуточных слоях.

Угол поворота текстовых областей определяется при помощи преобразования Хафа [11], применяемого для идентификации прямых на изображении. Направление линий, найденных таким образом на изображении текста, преимущественно совпадает с направлением строк текста. Это позволяет достаточно быстро определить угол наклона текста с достаточно высокой точностью.

Области межстрочных интервалов определяются по особым точкам алгоритма FAST [12]. Метод схож с методом определения горизонтального профиля страницы, описанным в [13]. При подсчете горизонтального профиля учитываются не черные пиксели, а особые точки. Такой подход позволяет определять области межстрочных интервалов вне зависимости от цвета текста и цвета фона.

5.3 Алгоритм извлечения цифровой метки

В сценарии «screen-cam», для которого была разработана предлагаемая система, изображение маркированного документа выводится на экран монитора, после чего экран фотографируется. Полученный снимок подвергается процедуре извлечения цифровой метки. Извлечение ЦВЗ проводится в несколько этапов:

1) коррекция перспективы и обрезка фотографии экрана;
2) определение областей с текстом на фотографии;
3) определение межстрочных интервалов текста;
4) поиск двойных маркеров в межстрочных интервалах;
5) извлечение битов сообщения, закодированных в маркерах ЦВЗ.

В задаче расследования утечек текстовых документов время и вычислительные ресурсы, затрачиваемые на процесс извлечения цифровой метки, имеют меньшее значение, чем точность извлеченного сообщения. Это позволяет выполнять извлечение ЦВЗ многократно с подбором параметров с целью получения наиболее успешных результатов. Некоторые этапы извлечения могут быть проведены как автоматически, так и вручную. К ним относятся этапы 1–3. При фотографировании экрана камера может быть расположена под углом к плоскости экрана, поэтому на первом этапе извлечения необходимо произвести коррекцию перспективы и обрезку фотографии. На следующих двух этапах на скорректированной фотографии определяются области с текстом и межстрочные интервалы. На этих этапах может быть применен такой же метод, что и при встраивании ЦВЗ.

Для определения положения цифровой метки в межстрочном интервале производится поиск двойных маркеров. При этом учитывается, что верхняя и нижняя половины всех маркеров, кроме двойных, одинакового цвета. Путем сравнения нижней и верхней половин межстрочного интервала на фотографии определяется, в каких точках межстрочного интервала расположены маркеры начала, середины и конца ЦВЗ. По ним можно точно определить координаты центров маркеров, кодирующих биты сообщения.

В процессе извлечения отдельных битов сообщения используется тот факт, что соседние с кодирующим промежуточные маркеры отличаются от него цветом. Функция яркости от горизонтальной координаты в межстрочном интервале на фотографии имеет локальные минимумы в темных маркерах, кодирующих бит «1», и локальные максимумы в светлых маркерах, кодирующих бит «0». Характер экстремума можно определить по знаку второй производной: положительное значение для локального минимума и отрицательное для локального максимума. Пусть $I(x, y)$ — функция яркости скорректированной фотографии экрана, (x_i, y_i) — координаты центра маркера i-го бита сообщения $M = m_1 \dots m_N$. Значение извлекаемого бита определяется по правилу:

$$s_i = \frac{\partial^2}{\partial x^2} I(x_i, y_i),$$

$$m_i = \begin{cases} 0, \text{если } s_i < 0 \\ 1, \text{иначе} \end{cases}.$$

5.4 Алгоритм извлечения цифровой метки

Обычно в процессе извлечения ЦВЗ отсутствует возможность сравнения извлеченного сообщения с встроенным посредством ЦВЗ. В таком случае, если результаты, полученные в ходе извлечения с различными параметрами, не совпадают, невозможно определить, какое из извлеченных сообщений содержит меньше ошибок по отношению к встроенному сообщению. В данном подразделе предложена метрика оценки корректности результата работы алгоритма извлечения цифровой метки. Её построение основано на следующих предположениях.

- $s_i, i = 1 \dots N$ — независимые одинаково распределенные случайные величины;
- Значения второй производной в точках межстрочного интервала на фотографии без встроенного ЦВЗ удовлетворяют нормальному распределению $\mathcal{N}(0, \sigma^2)$;
- Если i-ый бит m_i встроенного сообщения равен 0, что равносильно встраиванию светлого маркера, то значение второй производной в центре соответствующего маркера смещается на константу $-\mu, s_i \sim \mathcal{N}(-\mu, \sigma^2)$. Если же бит сообщения равен 1, то $s_i \sim \mathcal{N}(\mu, \sigma^2)$.

Приведенные предположения были проверены экспериментально. Результаты эксперимента представлены на рис.6.

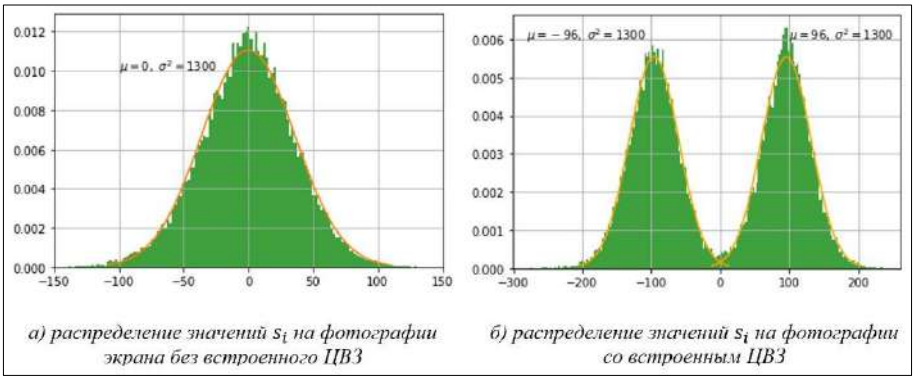


Рис. 6. Распределение значений s_i на фотографии экрана
Fig. 6. Experimental distribution of s_i

По фотографии экрана, поданной на извлечение, определяется выборка значений $\widehat{s_i}, i = 1 \dots N$. По этой выборке можно оценить значения μ и σ^2. Для всех i производится выбор из двух гипотез:

$$\begin{cases} H_0: s_i \sim \mathcal{N}(-\mu, \sigma^2), m_i = 0 \\ H_1: s_i \sim \mathcal{N}(\mu, \sigma^2), m_i = 1 \end{cases}.$$

Решающее правило строится на основе равноценности гипотез, а именно равенстве ошибок первого и второго рода (минимаксное решающее правило). В силу симметрии распределений относительно 0:

$$m_i = \theta(\widehat{s_i}) = \begin{cases} 0, f(\widehat{s_i}|H_0) > f(\widehat{s_i}|H_1) \\ 1, \text{иначе} \end{cases} = \begin{cases} 0, \widehat{s_i} < 0 \\ 1, \text{иначе'} \end{cases}$$

где

$$f(x|H_0) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x+\mu)^2}{2\sigma^2}},$$

$$f(x|H_1) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Для каждого полученного бита сообщения можно оценить вероятность, с которой выбранная гипотеза справедлива. По формуле Байеса, при условии $\mathbb{P}(H_0) = \mathbb{P}(H_1)$:

$$\mathbb{P}(H_0|\widehat{s_i}) = \frac{f(s_i|H_0)\cdot\mathbb{P}(H_0)}{f(s_i|H_0)\cdot\mathbb{P}(H_0)+f(s_i|H_1)\cdot\mathbb{P}(H_1)} = \frac{f(s_i|H_0)}{f(s_i|H_0)+f(s_i|H_1)}.$$

Определим вероятности p_i, соответствующие решающему правилу θ:

$$p_i = \max\{\mathbb{P}(H_0|\widehat{s_i}), \mathbb{P}(H_1|\widehat{s_i})\}.$$

Из предположения о независимости s_i, получаем вероятность того, что сообщение было извлечено верно:

$$\mathbb{P}(m_1, \cdots, m_n|\widehat{s_1}, \cdots, \widehat{s_N}) = \prod_{i=1}^{N} p_i.$$

Полученная вероятность позволяет оценить, насколько точно прошло извлечение ЦВЗ: если значение близко к 1 (0.9 и выше), можно считать, что извлеченное сообщение совпадает с сообщением, встроенным при помощи ЦВЗ. Приведенная оценка была получена экспериментально. Если значение вероятности меньше 0.9, извлечение ЦВЗ следует считать неудачным: рекомендуется изменить параметры алгоритма и повторить процедуру извлечения цифровой метки.

6. Тестирование разработанного метода маркирования

Для проверки устойчивости предложенного метода маркирования текстовых документов к атакам сценария «screen-cam» был проведен ряд экспериментов по встраиванию цифровой метки в изображение документа на экране и извлечению этой метки из фотографии экрана. Цифровая метка встраивалась в текст, состоящий из 15 строк, что соответствует 14 межстрочным интервалам. Кегль шрифта 14 пт, множитель межстрочного интервала 1.15, масштаб текста 100%. Текст отображался на экране монитора Acer VG272U диагональю 27 дюймов, с разрешением 2560×1440 пикселей, типом матрицы IPS. Снимки экрана выполнялись на камеру смартфона Samsung Galaxy S8, обладающую характеристиками: 12 мегапикселей, апертура f/1.7, фокусное расстояние 26 мм. В межстрочные интервалы текста внедрялась цифровая метка, состоящая из 32 бит. Всего метка была встроена 7 раз, в пары подряд идущих интервалов. Фотографии подвергались процедуре извлечения метки. Извлеченная метка сравнивалась со встроенной на экран.

Для оценки извлекаемости ЦВЗ применяется мера Bit Error Rate (BER). BER вычисляется как отношение числа неверно извлеченных битов к общему числу битов цифровой метки. Если при извлечении эта величина равна нулю, это означает, что ЦВЗ был извлечен без ошибок. Извлекаемость ЦВЗ сравнивалась тремя оценками: оценкой BER-32, вычисляемой после объединения значений цифровых меток, встроенных в разные межстрочные интервалы, оценкой BER-224, подсчет которой проводился при предположении, что цифровые метки, встроенные в разные межстрочные интервалы, независимы, и оценкой вероятности $\mathbb{P}(M|\widehat{s_1}, \cdots, \widehat{s_{32}})$, рассчитываемой без учета знания встроенного сообщения. Если значение

последней оценки близко к 1 в случае, если извлечение проходит успешно, и близко к 0, если сообщение извлечено с ошибками, это означает, что она применима для принятия решения об успешности извлечения в условиях отсутствия информации о встроенном сообщении.

В каждом эксперименте было сделано по 10 фотографий. В приведенных ниже таблицах указано число фотографий, на которых цифровая метка была обнаружена, а также средние значения оценок по всем таким фотографиям.

6.1 Непрозрачность маркеров

Ранее отмечалась необходимость достижения компромисса между устойчивостью цифровой метки к атакам и ее незаметностью. На незаметность ЦВЗ оказывает влияние непрозрачность маркеров: чем больше непрозрачность маркеров, тем они более заметны. При уменьшении непрозрачности маркеров снижается заметность цифровой метки, но при этом снижается и различимость маркеров на фотографии на этапе извлечения. В табл. 1 представлены результаты эксперимента по изменению непрозрачности маркеров цифровой метки. В этом эксперименте камера располагалась на расстоянии 60 см от экрана параллельно его плоскости. В следующих экспериментах маркеры метки встраивались с непрозрачностью 3%.

Табл. 1. Извлекаемость цифровой метки в зависимости от непрозрачности маркеров
Table 1. Extraction rate with different watermark opacity

Непрозрачность маркеров	Метка обнаружена	BER-32	BER-224	$\mathbb{P}(\boldsymbol{M}\|\widehat{\boldsymbol{s_1}},\cdots,\widehat{\boldsymbol{s_{32}}})$
6%	10/10	0%	0%	1
5%	10/10	0%	0.2%	1
4%	10/10	0%	0.8%	1
3%	10/10	0%	2.3%	1
2%	6/10	0.5%	7%	0.94
1%	0/10	-	-	-

6.2 Расположение камеры относительно экрана

Предложенный метод был протестирован на устойчивость к атакам пространственного расположения камеры относительно экрана. Было проведено два эксперимента. В первом эксперименте изучалось влияние расстояния от камеры до экрана на извлекаемость цифровой метки при расположении камеры параллельно плоскости экрана. Результаты представлены в табл. 2. Метка извлекается с ошибками, когда камера расположена на расстояниях 40 см и 25 см от экрана. Это связано с тем, что на таких расстояниях эффект муара оказывает большое влияние на извлекаемость ЦВЗ.

Табл. 2. Влияние расстояния от камеры до экрана на извлекаемость ЦВЗ
Table 2. Extraction rate with different capture distances

Расстояние, см	Метка обнаружена	BER-32	BER-224	$\mathbb{P}(\boldsymbol{M}\|\widehat{\boldsymbol{s_1}},\cdots,\widehat{\boldsymbol{s_{32}}})$
100	10/10	0.6%	9.6%	0.7
80	10/10	0%	6.9%	1
60	10/10	0%	2.3%	1
50	10/10	0%	2.5%	1
40	9/10	3.3%	13%	0.61
30	10/10	0%	1.9%	1
25	8/10	12%	24%	0.27

Во втором эксперименте снимки экрана были сделаны под углом к плоскости экрана на фиксированном расстоянии. С целью повышения точности оценки влияния величины угла, съемка проводилась на расстоянии 50 см от камеры до центра экрана. Результаты представлены в табл. 3. Цифровая метка успешно извлекается при углах не более 45º, а также сохраняет возможность частичного извлечения вплоть до угла 60º между камерой и плоскостью экрана.

Табл. 3. Извлекаемость ЦВЗ при разных углах между камерой и плоскостью экрана
Table 3. Extraction rate with different capture angles

Величина угла	Метка обнаружена	BER-32	BER-224	$\mathbb{P}(\boldsymbol{M}\|\widehat{\boldsymbol{s_1}},\cdots,\widehat{\boldsymbol{s_{32}}})$
0º	10/10	0%	2.3%	1
15º	10/10	0%	2.5%	1
30º	10/10	0%	5.4%	0.89
45º	10/10	0.9%	7.6%	0.89
60º	10/10	1.5%	10%	0.64
75º	4/10	7.5%	24%	0.17

6.2 Сжатие фотографии

Выше упоминалось, что в сценарии «screen-cam» фотографии экрана могут подвергаться воздействию алгоритмов сжатия.

Одним из распространенных алгоритмов сжатия изображений является алгоритм JPEG.

Был проведен эксперимент по проверке предложенного метода на устойчивость сжатию изображения.

Фотографии экрана, полученные при расположении камеры на расстоянии 60 см параллельно плоскости экрана, подвергались разной степени сжатия по алгоритму JPEG. Результаты представлены в табл. 4. Предложенный метод показывает высокую устойчивость к атаке сжатия фотографии.

Табл. 4. Влияние степени сжатия JPEG на извлекаемость ЦВЗ
Table 4. Extraction rate with different JPEG compression

Коэффициент качества JPEG	Метка обнаружена	BER-32	BER-224	$\mathbb{P}(\boldsymbol{M}\|\widehat{\boldsymbol{s_1}},\cdots,\widehat{\boldsymbol{s_{32}}})$
100	10/10	0%	2.1%	1
60	10/10	0%	2%	1
40	10/10	0%	3.2%	1
20	10/10	0%	7.6%	1
15	10/10	0%	9.9%	0.98
10	8/10	0.4%	22%	0.82

7. Заключение

В статье представлен метод маркирования текстовых документов, выводимых на экран монитора – алгоритмы встраивания и извлечения ЦВЗ. Описан подход, позволяющий оценить точность извлеченной цифровой метки при отсутствии встроенного сообщения. В ходе проведенного тестирования разработанного метода оценивалась точность ЦВЗ при различных параметрах: заметность, положение камеры, степень JPEG-сжатия. Результаты экспериментов показали, что разработанный метод устойчив к основным атакам сценария «screen-cam» и может использоваться на практике.

Список литературы / References

[1]. Утечки информации ограниченного доступа: отчет за 9 месяцев 2020 г. Экспертно-аналитический центр InfoWatch, 2020 г. / Restricted information leaks: report for 9 months of 2020, InfoWatch Analytical Center, 2020 (in Russian).
[2]. Pramila A. Reading watermarks with a camera phone from printed images. PhD Thesis. University of Oulu, Oulu, 2018, 86 p.
[3]. Chen W., Ren N. et al. Screen-Cam Robust Image Watermarking with Feature-Based Synchronization. Applied Sciences, vol. 10, no. 21, 2020, article no. 7494.
[4]. Fang H., Zhang W. et al. Screen-Shooting Resilient Watermarking. IEEE Transactions on Information Forensics and Security, vol. 14, no. 6, 2019, pp. 1403–1418.
[5]. Song J., Lu X., Wang W., Chen C. ISIFT: Improving the Performance of SIFT for Mirror Images. In Proc. of the 2nd IEEE International Conference on Computer and Communications (ICCC), 2016, pp. 742-745.
[6]. Mikolajczyk K., Schmid C. An Affine Invariant Interest Point Detector. Lecture Notes in Computer Science, vol. 2350, 2002, pp. 128–142.
[7]. Chen W., Ren N. et al. Joint Image Encryption and Screen-Cam Robust Two Watermarking Scheme. Sensors, vol. 21, no. 3, 2021, article no. 701.
[8]. Gugelmann D., Sommer D. et al. Screen Watermarking for Data Theft Investigation and Attribution. In Proc. of the 10th International Conference on Cyber Conflict (CyCon), 2018, pp. 391-408.
[9]. Piec M., Rauber A. Real-Time Screen Watermarking Using Overlaying Layer. In Proc. of the Ninth International Conference on Availability, Reliability and Security, 2014, pp. 561-570.
[10]. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. Lecture Notes in Computer Science, vol. 9351, 2015, pp. 234–241.
[11]. Hough P.V.C. Method and Means for Recognizing Complex Patterns. U.S. Patent 30696541962, 1992.
[12]. Mathur G., Rikhari M.S. Text Detection in Document Images: Highlight on Using FAST Algorithm. International Journal of Advanced Engineering Research and Science, vol. 4, no. 3, 2017, pp. 275–284.
[13]. Low S., Brassil J.T., Maxemchuk N.F., O'Gorman L. Document Marking and Identification Using Both Line and Word Shifting. In Proc. of the INFOCOM'95, 1995, pp. 853-860.

Информация об авторах / Information about authors

Алексей Юрьевич ЯКУШЕВ – студент. Научные интересы: стеганография, обработка цифровых изображений, алгоритмы машинного обучения.

Aleksey Yur'evich YAKUSHEV is a student. Scientific interests: steganography, digital image processing, machine learning algorithms.

Юрий Витальевич МАРКИН – научный сотрудник, кандидат технических наук. Область научных интересов: информационная безопасность, анализ сетевого трафика, обработка изображений, алгоритмы машинного обучения.

Yury Vital'evich MARKIN is a researcher, PhD in Technical Sciences. Scientific interests: information security, network traffic analysis, image processing, machine learning algorithms.

Станислав Александрович ФОМИН – ведущий программист. Область научных интересов: теория сложности, алгоритмы дискретной оптимизации, верификация ПО, архитектура информационных систем.

Stanislav Alexandrovich FOMIN – leading programmer. Research interests: complexity theory, discrete optimization algorithms, information systems architecture.

Дмитрий Олегович ОБЫДЕНКОВ – аспирант. Научные интересы: методы сокрытия и защищённой передачи информации, компьютерные сети, технологии анализа сетевого трафика.

Dmitry Olegovich OBYDENKOV is a graduate student. Scientific interests: methods for information hiding and secure transmission, computer networks, technologies of network traffic analysis.

Борис Владимирович КОНДРАТЬЕВ – сотрудник МО РФ. Сфера научных интересов: безопасность информации, защита информации от несанкционированного доступа и утечки по техническим каналам, построение информационных систем в защищённом исполнении, сертификация программного обеспечения по требованиям безопасности информации.

Boris Vladimirovich KONDRAT'EV is an employee of the Ministry of Defence of the Russian Federation. Scientific interests: information security, protection of information from unauthorized access and leakage through technical channels, building information systems in a secure design, certification of software for information security requirements.

DOI: 10.15514/ISPRAS-2021-33(4)-12

A Multilayer Approach to Subgraph Matching in HP-graphs

N.M. Suvorov, ORCID: 0000-0003-2871-9757 <SuvorovNM@gmail.com>
L.N. Lyadova, ORCID: 0000-0001-5643-747X <LNLyadova@gmail.com>
HSE University,
20, Myasnitskaya Ulitsa, Moscow, 101978, Russia

Abstract. Visual modeling is widely used nowadays, but the existing modeling platforms cannot meet all the user requirements. Visual languages are usually based on graph models, but the graph types used have significant restrictions. A new graph model, called *HP*-graph, whose main element is a set of poles, the subsets of which are combined into vertices and edges, has been previously presented to solve the problem of insufficient expressiveness of the existing graph models. Transformations and many other operations on visual models face a problem of subgraph matching, which slows down their execution. A multilayer approach to subgraph matching can be a solution for this problem if a modeling system is based on the *HP*-graph. In this case, the search is started on the higher level of the graph model, where vertices and hyperedges are compared without revealing their structures, and only when a candidate is found, it moves to the level of poles, where the comparison of the decomposed structures is performed. The description of the idea of the multilayer approach is given. A backtracking algorithm based on this approach is presented. The Ullmann algorithm and VF2 are adapted to this approach and are analyzed for complexity. The proposed approach incrementally decreases the search field of the backtracking algorithm and helps to decrease its overall complexity. The paper proves that the existing subgraph matching algorithms except ones that modify a graph pattern can be successfully adapted to the proposed approach.

Keywords: DSM platform; visual model; subgraph matching; isomorphism; graph model; HP-graph; algorithms on graphs.

For citation: Suvorov N.M., Lyadova L.N. A Multilayer Approach to Subgraph Matching in HP-graphs. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021, pp. 163-176. DOI: 10.15514/ISPRAS-2021-33(4)-12

Многослойный подход к поиску изоморфных подграфов в HP-графах

Н.М. Суворов, ORCID: 0000-0003-2871-9757 <SuvorovNM@gmail.com>
Л.Н. Лядова, ORCID: 0000-0001-5643-747X < LNLyadova@gmail.com>
Национальный исследовательский университет «Высшая школа экономики»,
101978, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. Визуальное моделирование на данный момент широко распространено, однако существующие платформы, предназначенные для моделирования, не могут удовлетворить все требования пользователей. Визуальные языки, как правило, основаны на графовых моделях, однако графовые формализмы, используемые для представления моделей, обладают существенными ограничениями. Для решения проблемы недостаточной выразительности существующих графовых моделей ранее была представлена новая графовая модель (*HP*-граф), основным элементом которой является множество полюсов, подмножества которых объединены в вершины и гиперребра. Многие операции над визуальными моделями, включая трансформацию моделей, сталкиваются с проблемой поиска изоморфного подграфа, что оказывает значительное влияние на скорость их выполнения.

Многослойная структура *HP*-графа позволяет снизить временную сложность алгоритмов поиска. Количество операций может быть снижено благодаря тому, что поиск изначально осуществляется на слое вершин и гиперребер, и только в случае нахождения подграфа с желаемыми характеристиками алгоритм переходит на более детальный уровень, где сравниваются наборы соответствующих полюсов и обыкновенных связей отобранных подграфов. Представлено описание идеи многослойного подхода. Предложен алгоритм поиска с возвратом, основанный на этом подходе. Алгоритмы Ульмана и VF2 адаптированы к данному подходу, выполнена оценка их временной сложности. Предложенный подход постепенно сокращает область поиска алгоритмов и помогает уменьшить их общую сложность. В статье доказывается, что существующие алгоритмы сопоставления подграфов, за исключением тех, которые изменяют шаблон графа, могут быть успешно адаптированы к предлагаемому подходу.

Ключевые слова: DSM платформа; визуальная модель; поиск изоморфного подграфа; изоморфизм; графовая модель; HP-граф; алгоритмы на графах

Для цитирования: Суворов Н.М., Лядова Л.Н. Многослойный подход к поиску изоморфных подграфов в HP-графах. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 163-176 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(4)–12

1. Introduction

The study of any objects and processes, as well as their design, can barely be done without modeling; that is why software tools that allow specialists to build various models and formalize descriptions of objects and processes, or use modeling as a method of analysis, are becoming more popular. Models are described and built with the help of a visual modeling language, which is a fixed set of graphical symbols and rules for constructing visual models by using these symbols [1]. Visual languages can be represented as various types of graphs, including oriented graphs [2], hypergraphs [3], hi-graphs [4], meta-graphs [5] and *P*-graphs [6].

Previously, a new graph model, called *HP*-graph, was proposed as a formalism for representing visual languages [7]. This model unites expressive possibilities of all the mentioned graph types and, thus, it can be used for building more complicated models than those which can be built with the help of the other graph models. The paper [7] proved that this graph model allows the creation of a flexible visual model editor based on it.

This model is proposed as a basis for domain-specific modeling, one of the key aspects of which is model transformations. Such transformations allow users to move from one level of abstraction to another (a vertical transformation) or from one modeling language to another (a horizontal transformation) [5]. Different approaches can be used to transform visual models, but the current standard is the algebraic approach which is based on the graph grammars [9]. Based on this approach, a transformation $r = (L, R)$ includes the left and the right part, where L is a subgraph to be found in a source graph, and R is a subgraph replacing L in the source graph.

As for the *HP*-graph, only main operations, including operations of adding and removing graph elements and operations of decomposition, were described for this model, and no algorithm were proposed to perform an isomorphic subgraph search operation. The structural complexity of the model requires modifying the existing algorithms to adapt them to this model. The *HP*-graph has a multilayer structure which consists of the layer of vertices and hyperedges and the layer of poles and links, sets of which are combined into the elements of the former layer. The multilayer structure of the graph model allows to reduce time complexity of search algorithms. The number of operations can be decreased due to the fact that the first search and matching is performed on the layer of vertices and hyperedges, and only after finding a subgraph with the desired characteristics, the algorithm moves to a more detailed level, where the already selected sets of corresponding poles and ordinary edges are compared.

In practice, a task of finding an isomorphic subgraph has a wide range of applications, including chemical compound search [10], social network analysis [11], pattern recognition [12], and protein interaction analysis [13]. However, subgraph matching is a bottleneck in the overall performance for most of these applications due to the fact that this task is *NP*-hard [14]. For instance, nodes count

for protein structure analysis can reach up to tens of thousands [15]; that is why active efforts are currently being made to find an optimal algorithm for subgraph matching.

In visual modeling the problem is the same. The thesis [5] proposes to represent all the models in the form of a single graph, which allows users to maintain links between the models and automatically propagate changes from the source model to the target ones associated with it. For instance, a change in the metamodel of the subject area should be propagated to all the models built on this metamodel. However, storing all the models as a single graph increases the computational complexity of the algorithms on this graph, which requires developing an efficient subgraph search algorithm for the graph model used.

The contributions of these paper are:

1) a new multilayer approach to decrease complexity of subgraph matching algorithms,
2) a backtracking algorithm based on this approach,
3) applications of this approach in several existing subgraph matching algorithms.

The paper is organized as follows. Section 2 discusses related work and the main algorithms for finding subgraph isomorphism. Section 3 presents the proposed graph model, definitions of the *HP*-subgraph and isomorphism of the *HP*-graphs, and the multilayer approach to subgraph matching. Section 4 introduces a backtracking algorithm based on this approach. Section 5 presents several applications of the approach in the existing subgraph matching algorithms. Section 6 describes the obtained results. Section 7 concludes the paper.

2. Related work

The problem of subgraph matching has been investigated for many years. The works of many scientists, such as [16]-[18], are dedicated to exploring applicability, time complexity and limitations of the existing subgraph matching algorithms. These algorithms are generally divided into two classes:

- Algorithms that observe many graphs $\{G_1, \ldots, G_n\}$ and retrieve those which contain a query graph Q.
- Algorithms that observe a single graph G and retrieve all its subgraphs which are isomorphic to a query graph Q.

In both of these approaches, algorithms can either return a correct and complete answer (having an exponential time complexity) or return an approximate answer (having a polynomial time complexity). While the complete answers describe all subgraphs exactly isomorphic to a pattern, the approximate answers are generally obtained using specific similarity measures and, thus, may also contain false positive subgraphs.

This work belongs to the second class of the algorithms. Most of these algorithms use *backtracking* to move through the built search tree and find appropriate combination of corresponding vertices of the source graph and the graph-pattern. Algorithms in this class include Ullmann algorithm [19], VF2 [20] (and also VF2 Plus [21] and VF3 [22]), TurboISO [23], CFL-Match [24], QuickSI [25], SPath [26] and others. These algorithms implement various techniques to decrease time needed for the matching process.

Exploiting Pruning Rules. The Ullmann algorithm uses *refining procedure* on each step of the algorithm by comparing degrees of corresponding neighbors of the added pair of vertices. VF2 [20] provides *feasibility rules* that are checked before a vertex is added to a graph-candidate. There rules check consistency of graph-candidates with this vertex and check for a sufficient number of vertices-neighbors of these graph-candidates. SPath [26] uses *neighborhood signature* for each vertex to store information about the surrounding vertices. These signatures are compared with the corresponding signatures of the query graph and are used for search space pruning before subgraph matching. TurboISO [23] compares quantity of *neighborhood labels* of corresponding vertices and prune out unpromising ones. CFL-Match [24] proposes a *compact-path-index* (CPI) structure

presented as a tree which is built from the source graph vertices with the same labels as query graph vertices and then refined by exploiting matching operations.

Graph Pattern Modification. The Ullmann algorithm and VF2 [20] do not modify graph pattern and search its embeddings in the source graph. SPath [26] changes the way of graph query processing from vertex-at-a-time to *path-at-a-time*, which tends to be more cost-effective than traditional graph matching methods. TurboISO [23] presents a *NEC-tree* structure which merges similar vertices together and present a query graph as a tree. CFL-Match [24] transform a query into a set of *dense subgraphs*, *forests*, and *leaves*. The source graph in this algorithm is only probed for non-tree edge validation, whereas other query parts are checked in the CPI structure.

Optimizing Matching Order. The Ullmann algorithm [19] does not specify the matching order of the vertices, whereas VF2 [20] starts from a random query vertex and then recursively adds those vertices that are connected with the already matched ones. QuickSI [25] exploits an order which is based on the vertex label frequency, and the algorithm starts a process of matching from the least frequent ones. TurboISO [23] implements a concept of candidate region exploration and produces a matching order for every region where a NEC-tree was found. CFL-Match [24] present all candidates as a CPI-structure, where all the pattern embeddings are filtered and validated by traversing this tree structure.

The most of theoretical research of this problem was conducted specifically for ordinary graphs [18]; that is why the approaches of these algorithms have to be adapted to an *HP*-graph model. In particular, this paper presents an adaptation of a standard backtracking algorithm for subgraph matching, the Ullmann algorithm [19] and the VF2 algorithm [20], which are optimized for the multilayer structure of this graph model.

3. Graph-Matching Approach for HP-graphs

Let *Pol* be a set of all poles of the graph, including external poles and internal poles of vertices and hyperedges. Then, an *HP*-graph is an ordered triple $G = (P, V, W)$, where $P = \{\pi_1,\ldots,\pi_n\}$ is a set of external poles, $V = \{v_1,\ldots,v_m\}$ is a non-empty set of vertices, $W = \{w_1,\ldots,w_l\}$ is a set of hyperedges [7]. An example of the graph model is demonstrated on fig. 1.

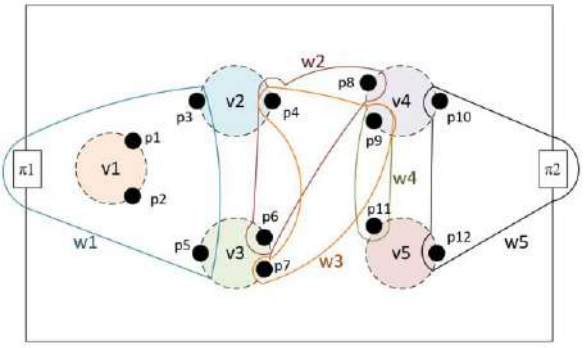


Рис. 1. Пример HP-графа
Fig. 1. Example of an HP-graph

In this figure external poles are represented by a set $P = \{\pi_1, \pi_2\}$, hyperedges by a set $W = \{w_1,\ldots,w_5\}$, and vertices by a set $V = \{v_1,\ldots,v_5\}$. A set *Pol* includes of the poles of the graph and is presented as $\{p_1,\ldots,p_{12}\}\cup\{\pi_1, \pi_2\}$.

Every hyperedge w of the *HP*-graph G can be presented by ordinary links, which are defined as a set $E_w = \{e_1,\ldots,e_n\}$, where every link ($e \in E_w$) is a pair of connected poles (p, r), where p is a source pole and r is a target pole of a link. An example of this decomposition is presented in Fig. 2. The hyperedge w_2 defines a set $E_{w_2} = \{(p4, p8), (p4, p6), (p6, p8)\}$. Every vertex and hyperedge can also be decomposed by a new *HP*-graph, which is described in detail in [7].

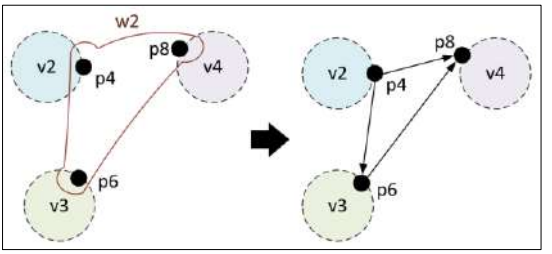


Рис. 2. Декомпозиции гиперребра w_2
Fig. 2. Decomposition of the hyperedge w_2

3.1 Definitions of a Subgraph and Isomorphism

To determine subgraph matching operations, it is needed to give a definition to a subgraph of the *HP*-graph. An *HP*-graph $G' = (P', V', W')$ is a subgraph of an *HP*-graph $G = (P, V, W)$ iff G' is a part of the graph G ($P' \subset P$ & ($\forall v' \in V' \; \exists v \in V$: $[v' \subset v]$) & $W' \subset W$) and meets the condition (1) to make transformation operations possible [7]. A subgraph can contain vertices called *incomplete* whose sets of poles are only part of the sets of poles of the vertices of the original graph:

$$\forall w \in W(\exists v \in V' \setminus V'_{partial} \; ([Pol(w) \cap Pol(v) \neq \varnothing]) \rightarrow w \in W'), \quad (1)$$

the set $V'_{partial}$ is a set of the *incomplete vertices* in the graph, where $V'_{partial} \subset V'$.

To define the isomorphism mapping, it is necessary to establish one-to-one correspondences between the same type elements of graphs that preserve the incident relations. This, two *HP*-graphs $G = (P, V, W)$ and $G' = (P', V', W')$ are isomorphic iff there exists a bijection f: $2^{Pol(G)} \rightarrow 2^{Pol(G')}$ such that for $\forall t \in 2^{Pol(G)}$:

$$(t \in W \leftrightarrow f(t) \in W') \& (t \in V \leftrightarrow f(t) \in V') \& (t \in P \leftrightarrow f(t) \in P').$$

3.2 A Multilayer Approach to Graph Matching

As the graph model is proposed to store all the models together, search algorithms for this formalism have to be optimized for this task. A possible solution to this problem is to divide the *HP*-graph into two main levels: the level of vertices and hyperedges, and the level of poles and ordinary links between them. In this case, the search is started on the higher level, and when a candidate is found, it moves to the lower level, where a more detailed comparison of graph elements is performed.

Fig. 3(a) illustrates an example of a query graph *Q*, which is a pattern for subgraph matching for a data *G* from Fig. 1. As is seen, it contains 4 vertices, 2 hyperedges and 4 poles. Its higher (or first) level is presented in fig. 3(b). It contains only 4 vertices and 2 hyperedges, whereas all the poles are eliminated. This layer is compared with the first layer of the graph *G* (fig. 4), and when a potential subgraph is found, the matrix of vertex correspondence is built.

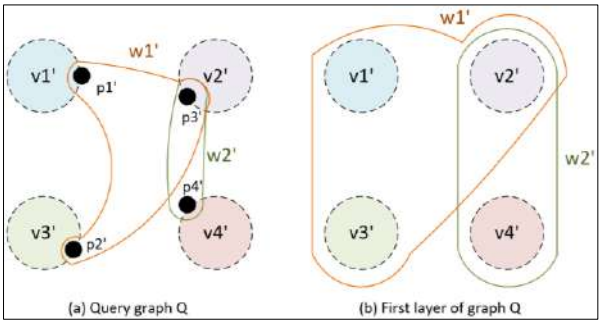


Рис. 3. Граф-паттерн Q и его верхний уровень
Fig. 3. Query graph Q and its first level

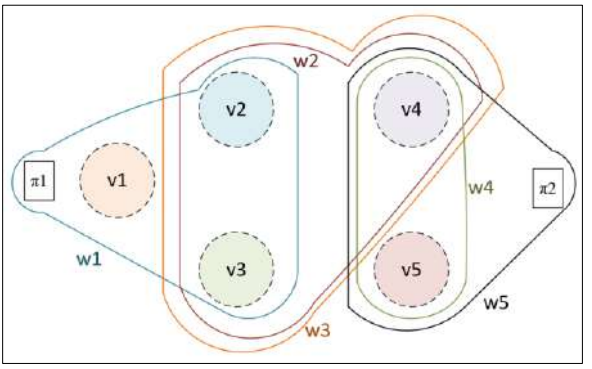


Рис. 4. Верхний уровень графа G
Fig. 4. First level of the graph G

The found correspondences between vertices of *Q* and *G* can be presented as a set {(*v*1', *v*2), (*v*3', *v*3), (*v*2', *v*4), (*v*4', *v*5)}. If a subgraph is found, the algorithm moves to the next level, where the corresponding hyperedges and their poles are compared.

All the candidate hyperedges are grouped by their incidence with each other depending on the poles which they consist of. For instance, hyperedges *w*1' and *w*2' are presented as a single group because of the pole *p*3' which both of them own. Thus, a corresponding pair (*w*3, *w*4) is also presented as a single group. All these groups are compared for exact isomorphism on the layer of poles and ordinary links. Fig. 5 demonstrates this layer for a pair of candidate groups (*w*1', *w*2') and (*w*3, *w*4). All these hyperedges are decomposed and only their poles and links are considered on this stage. As these graphs are identical, the found correspondences between poles of incident hyperedges of graphs *Q* and *G* can be presented as a set {(*p*3', *p*9), (*p*4', *p*11), (*p*2', *p*7), (*p*1', *p*4)}.

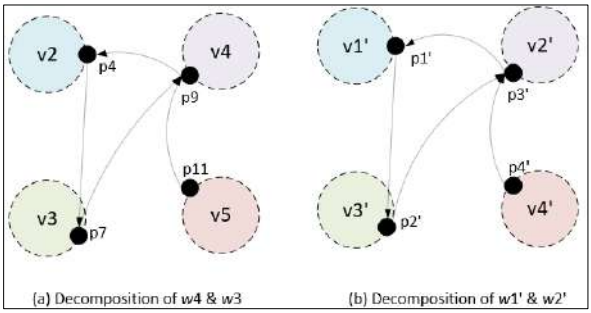


Рис. 5. Сопоставление гиперребер (w3, w4) и (w1', w2')
Fig. 5. Comparison of hyperedges (w3, w4) and (w1', w2')

If a validation on this hyperedge group is succeeded, the algorithm moves to the next group of hyperedges and validate them, until all the hyperedges are traversed. If a validation fails, the algorithm moves to the upper level and tries to find new pairs of vertices and hyperedges and validate them.

Lastly, the algorithm verifies that for every pole of the pattern graph only one pole of the source graph has been found. Otherwise, the found subgraph is considered as not isomorphic and the search continues.

4. Backtracking Graph Matching Algorithm based on the Multilayer Approach

The algorithm presented in this section uses as a basis a backtracking algorithm presented in [19]. This algorithm traverses a search tree using DFS until an isomorphic subgraph is found. If a pair of corresponding elements cannot be found at a certain step, a transition to an earlier step is carried out.

Considering the division of the subgraph matching into several levels, the search algorithm should be modified to perform the isomorphism search operation separately at the vertex level, separately at the hyperedge level, and separately at the level of poles and links.

Let *CompElems* define a set of compared elements: vertices, hyperedges or poles. Then, an algorithm for matching the corresponding sets of graph elements can be presented as follows (listing 1):

```
Function FindIsomorphism(G, Q, CompElems, args):
  M⁰, M, H, F, k, d = InitializeValues(G, Q, CompElems, args);
  do:
     k = GetNextNonVisitedColumn(M, F, k);
     if (k = -1):
        if (d = 1):
           return null;
        else
           MakeStepBack(F, d, M, k);
           continue;
     M = ChangeRowElementsToZerosExceptChosen(M, d, k);
     MakeStepForward(k, d, F, H, M);
  while (d ≤ |CompElems(Q)|);
  return ValidateIsomorph(M', CompElems(G), CompElems(Q));
```

Листинг 1. Псевдокод алгоритма сопоставления соответствующих множеств элементов графа
Listing 1. Pseudocode of the algorithm that matches the corresponding sets of graph elements

This algorithm at the beginning initializes a matrix M^0 which defines possible candidates between corresponding elements of graphs. If $m^0_{ij} = 1$ then the *i*-th element of the first graph is a candidate for isomorphism for the *j*-th element of the second graph. Otherwise, they cannot form a pair of corresponding elements. At each step, the modification of this matrix is used to determine appropriate pairs of elements. Thus, it is needed to define rules for building this matrix for each set of *HP*-graph elements.

For vertices matching, external poles and vertices can be combined into one set and named as vertices (for simplification). Thus, the matrix $M^0 = |Q_V \cup Q_P| \times |G_V \cup G_P|$ is filled according to the rule (2); if this condition is not met, $m^0_{ij} = 0$:

$$m^0_{ij} = \{1 |\ Deg(v_{Gj}) \geq Deg(v_{Qi})\ \&\ Count(v_{Gj}) \geq Count(v_{Qi})\}, \quad (2)$$

$Deg(v)$ is a number of hyperedges incident to the vertex v, $Count(v)$ is a number of the vertex poles.

For hyperedges matching, the matrix $M^0 = |Q_W| \times |G_W|$ is filled according to the rule (3):

$$m^0_{ij} = \{1 |\ Vertices(w_{Gj}) \cong Vertices(w_{Qi})\}, \quad (3)$$

$Vertices(w)$ is a set of vertices incident to the hyperedge w.

For poles matching, the matrix M^0 is created for each pair of grouped hyperedges; thus $M^0 = |Pol(W_{Ql})| \times |Pol(W_{Gm})|$. The matrix is filled according to the rule (4), considering that graphs *G* and *Q* on this stage only contain those hyperedges that are presented in the current groups:

$$\begin{aligned} m^0_{ij} = \{1 |\ & vertex(p_{Gj}) \cong vertex(p_{Qi})\ \&\ deg(p_{Gj}) \geq deg(p_{Qi})\ \& \\ & \&\ \forall edge(p_{Qi})\ \exists edge(p_{Gj})\ [edge(p_{Gj}) \cong edge(p_{Qi})]\ \& \\ & \&\ \forall edge(p_{Gj})\ \exists edge(p_{Qi})\ [edge(p_{Gj}) \cong edge(p_{Qi})]\ \}, \end{aligned} \quad (4)$$

$vertex(p)$ is a vertex which contains a pole p, $edge(p)$ is an edge which is incident to a pole p, $deg(p)$ is a degree of a pole (a number of ordinary links incident to a pole).

Listing 2 illustrates how an isomorphic subgraph for the proposed graph structure can be found. Vectors *VCorr*, *WCorr* and *PolCorr* contain pairs of corresponding elements of the graphs. *FindIsomorphism* method is presented above and is assumed to have a possibility to continue the

search from the position where the last candidate was found. For this purpose, the last argument for vertices and hyperedges isomorphism search is given to the algorithm (*VCorr* and *WCorr* respectively). *GroupByIncidence* combines the given hyperedges into groups, which represent incident edges.

```
Function FindHPGraphIsomorphism(G, Q):
   VCorr = [|V(Q)∪P(Q)|], WCorr = [|W(Q)|]; PolCorr = [|Pol(Q)|];
   do:
      VCorr = FindIsomorphism(G, Q, V(Q)∪P(Q), VCorr);
      if (VCorr = ∅):
         continue;
      do:
         WCorr = FindIsomorphism(G, Q, W(Q), VCorr, WCorr);
         if (WCorr = ∅ & |W(Q)| > 0):
            break;
         incidentHyperedges = GroupByIncidence(WCorr, G, Q);
         for ∀(W'Q, W'G) ∈ incidentHyperedges:
            polWCorr = FindIsomorphism(G, Q, Pol(W'Q), VCorr, WCorr);
            if (!PolCorr.TryAppend(polWCorr)):
               PolCorr = ∅;
               break;
         if (PolCorr ≠ ∅ or |W(Q)| = 0):
            unlinkedCorr = MatchUnlinked(G, Q, PolCorr, VCorr, WCorr);
            GenerateAnswer(PolCorr, unlinkedCorr, VCorr, WCorr);
      while (PolCorr = ∅);
   while (VCorr ≠ ∅ & PolCorr = ∅);
```

Листинг 2. Псевдокод алгоритма поиска изоморфного подграфа в HP-графе
Listing 2. Pseudocode of the algorithm that finds an isomorphic subgraph in HP-graph

The main idea of this algorithm is to incrementally shorten the search field. While the search for vertices traverses all the vertices of the original graph, the search for hyperedges only moves through those edges that are connected with the already chosen vertices and utilizes information about their correspondence with the vertices of the query graph. Pole matching is performed for each group of incident hyperedges, where a sufficient quantity of combinations is pruned out by exploiting information about the corresponding vertices and hyperedges. The algorithm also checks and matches the unlinked poles if they exist, which can be done in linear or close to linear time as all the corresponding vertices are already found. For simplicity, the algorithm is given for searching for the first isomorphic subgraph but can be transformed to searching for all embeddings of a pattern.

5. Exploiting Pruning Techniques of the Existing Algorithms

To optimize algorithms certain existing techniques can be used. Adaptation of the main techniques of the existing algorithms to the proposed graph model can prove the possibility of adapting these algorithms as a whole and improve the efficiency of the algorithm presented above.

5.1 Ullmann Algorithm

Ullmann algorithm [19] is one of the first algorithms for subgraph matching. This algorithm uses a backtracking algorithm presented above and at each step it performs a *refinement procedure* to prune out unpromising pairs.

This algorithm is performed at each node of the search tree. It traverses the matrix *M* and converts a certain part of values from ones to zeros. The condition for preserving 1 is that if a vertex *j* of the original graph is a candidate of a vertex *i* of the pattern graph, then each neighbor of the vertex *i* must have at least one candidate among the neighbors of the vertex *j*. Otherwise, *j* cannot be a candidate for a vertex *i*.

This algorithm can be implemented for both vertex matching and pole matching to eliminate unpromising element pairs. The refining algorithm for vertices can be presented as follows (listing 3):

```
Function RefineV(G, Q, M):
   do:
      anyChanges = false;
      for ∀i ∈ Range(|V(Q)|):
        if (¬∃j: [M_ij = 1]):
          return false;
        for ∀j ∈ Range(|V(G)|):
          for ∀x ∈ V(Q)\{v_Qi} where ∃w∈W(Q) [w∩v_Qi≠∅ & w∩x≠∅]:
            if (¬∃y∈V(G)\{v_Gj} where ∃w∈W(G) [w∩v_Gj ≠∅ & w∩y≠∅]& M_xy = 1):
                M_ij = 0;
        anyChanges = true;
  while (anyChanges);
  return true;
```

Листинг 3. Псевдокод алгоритма очистки для вершин HP-графа
Listing 3. Pseudocode of the algorithm that runs refining for vertices of the HP-graph

The algorithm goes through all the neighbors of the current query vertex, which have at least one common hyperedge with this vertex, and checks whether a source graph contains a corresponding neighbor-vertex. The algorithm for poles looks similarly but poles and ordinary links are used instead of vertices and hyperedges.

5.2 VF2 Algorithm

VF2 [20] has been proposed for performing subgraph matching on large graphs. Effective representation of data structures and the usage of feasibility rules significantly reduces both the average time complexity of the search and the amount of memory used.

The idea of the algorithm is to use special rules, called *feasibility rules*, at each node of the search tree to evaluate the feasibility of further progress on this branch of the tree before adding a pair of vertices to graph-candidates. There rules check consistency of graph-candidates and sufficiency of vertices-neighbors' quantity of the graph-candidate. If all the checks are passed, the algorithm can move to the next level of the tree.

An approach of checking the feasibility rules can be applied on both vertex and pole layers. As a pole layer is presented as an ordinary graph, the feasibility rules from [20] can be used without any significant modifications. However, feasibility rules for a vertex layer have to be defined.

The first rule checks the consistency of the existent candidate graphs by checking correctness of connections with the already added vertices. Let $core_G$ be a list of found pair vertices for the graph G and $core_Q$ be a list of found pair vertices for the graph Q. Accordingly, let $conn_G$ be a list of vertices which already have a pair or have a connection to the current graph-candidate G' and $conn_Q$ be a similar list for the graph-candidate Q'. Then, the first rule can be presented as follows:

$$\forall n'[core_G[n']\neq\varnothing \ \&\ n'\in Conn(G', n)]: \exists m'[m'\in Conn(Q', m) \ \&\ core_Q[m'] = n'] \ \&$$
$$\&\ \forall m'[core_Q[m']\neq\varnothing \ \&\ m'\in Conn(Q', m)]: \exists n'[n'\in Conn(G', n) \ \&\ core_G[n'] = m'].$$

$Conn(G, v)$ is a set of vertices of the candidate-graph G, which are connected to the vertex v.

Let PC define a set of vertices that can be connected to the vertex u, but the graph G does not include them; then it can be represented as follows:

$$PC(G, u) = \{v \mid v \in Conn(G, u) \ \& \ core_G[v] = \varnothing \ \& \ conn_G[v] \neq \varnothing\}.$$

Thus, a new rule, which compares numbers of newly added connections to graphs, appears:

$$|PC(G', n)| \geq |PC(Q', m)|.$$

The last rule performs a two-look-ahead in the searching process. Let N be a set of vertices which are connected to the target vertex but are not connected to the graph-candidate:

$$N(G, u) = \{v \mid v \in Conn(G, u) \ \& \ conn_G[v] = \varnothing\}.$$

Then, the last rule is presented by the condition:

$$|N(G', u)| > |N(Q', u)|.$$

The algorithm for traversing vertices can be presented as follows (listing 4):

```
Procedure RecurseV(G, Q, vectors):
   if (∀item ∈ vectors.core_Q[item≠∅]):
      poles_Q = RecurseW(vectors.core_G, vectors.core_Q, ∅, G, Q);
      if (poles_Q ≠ ∅):
         GenerateAnswer(poles_Q);
      else:
         vectors = RestoreVectors(vectors);
   else:
      P = GetAllCandidatePairs(vectors);
      for ∀p ∈ P:
         if (CheckVFisibilityRules(p, vectors, G, Q):
            vectors = UpdateVectors(vectors, G, Q);
            RecurseV(G, Q, vectors);
      vectors = RestoreVectors(vectors);
```

Листинг 4. Псевдокод алгоритма обхода вершин HP-графа на основе алгоритма VF2
Listing 4. Pseudocode of the algorithm that traverses vertices of the HP-graph based on the VF2 algorithm

5.3 Graph Pattern Modification Algorithms

The usage of algorithms such as TurboISO [23], CFL-Match [24] and other ones, that change a graph pattern, is complicated in the presented multilayer approach because these algorithms are made specifically for ordinary graphs. Their usage on the layer of vertices and hyperedges is a subject for the future research as it requires reformulation of their main aspects and ideas. Nevertheless, all these algorithms can be successfully used on the layer of poles and links and can find an isomorphic subgraph in the single-layer approach.

6. Complexity of the Algorithms

The presented algorithms can decrease the complexity of subgraph search by implementing matching on different graph layers. The search field shortens at each stage whereas the usage of pruning rules can also eliminate unpromising combinations of elements. Table 1 shows computational complexity of the backtracking algorithm at its main stages.

Табл. 1. Сложность алгоритма поиска с возвратом
Table 1. Complexity of the backtracking algorithm

Algorithm	Best Case	Worst Case
Isomorphic Vertices Matching	$O(N^2)$	$O(N \times N!)$
Isomorphic Hyperedges Matching	$O(N^2)$	$O(N \times N!)$
Isomorphic Poles Matching	$O(N^2)$	$O(N \times N!)$

The evaluation of the backtracking algorithms based on the Ullmann refinement is presented in Table 2. As the algorithm of hyperedge matching does not implement this technique, its complexity stays the same.

Табл. 2. Сложность алгоритма Ульмана
Table 2. Complexity of the Ullmann algorithm

Algorithm	Best Case	Worst Case
Isomorphic Vertices Matching	$O(N^3)$	$O(N^3 \times N!)$
Isomorphic Hyperedges Matching	$O(N^2)$	$O(N \times N!)$
Isomorphic Poles Matching	$O(N^3)$	$O(N^3 \times N!)$

The evaluation of the algorithms based on the VF2 approach is demonstrated in Table 3. The modification of the *GetAllCandidatePairs* procedure according to rules (2-4) slightly increases the worst-case complexity from $N \times N!$ to $N^2 \times N!$ and the best-case complexity from N^2 to N^3 but significantly shortens the search field.

Табл. 3. Сложность алгоритма VF2
Table 3. Complexity of the VF2 algorithm

Algorithm	Best Case	Worst Case
Isomorphic Vertices Matching	$O(N^3)$	$O(N^2 \times N!)$
Isomorphic Hyperedges Matching	$O(N^3)$	$O(N^2 \times N!)$
Isomorphic Poles Matching	$O(N^3)$	$O(N^2 \times N!)$

7. Conclusion

This paper proposed a solution to the problem of identifying isomorphic subgraphs in *HP*-graphs. The proposed approach is based on implementing matching on different graph layers of the graph model and incrementally shortening the search field at each layer.

The designed algorithms for subgraph matching based on the multilayer approach and evaluations of their complexity are presented above. The proposed approach incrementally decreases the search field of the algorithm and helps to decrease its overall complexity. The usage of pruning rules of the existing algorithms can eliminate unpromising candidates at each stage of the proposed algorithm and thus, significantly shorten the size of the search tree.

It is planned to evaluate actual time complexity of these algorithms on various data sets and develop a visual modeling system using the proposed approach to subgraph matching.

References / Список литературы

[1]. Koznov D.V. Methodology and tools for domain-specific modeling. Doctor Degree thesis. Saint-Petersburg, 2016, 430 p. (in Russian) / Кознов Д.В. Методология и инструментарий предметно-ориентированного моделирования. Диссертация доктора технических наук. СПб., 2016 г., 430 стр.

[2]. A formalism for describing software systems and computational processes for cyclic parallel processing of real time data. Information and control systems, 2006, no. 2, pp. 8-13 (in Russian) / Стручков И.В. Формализм для описания программных систем и вычислительных процессов циклической параллельной обработки данных реального времени. Информационно-управляющие системы, вып. 2, 2006, стр. 8-13.

[3]. Courcelle B. Recognizable Sets of Graphs, Hypergraphs and Relational Structures: A Survey. Lecture Notes in Computer Science, vol. 3340, 2005, pp. 1-11.
[4]. Power J., Tourlas K. Abstraction in Reasoning about Higraph-Based Systems. Lecture Notes in Computer Science, vol. 2620, 2003, pp. 392-408.
[5]. Sukhov A.O. Development of tools for creating visual subject-oriented languages. PhD thesis, Moscow, 2013, 256 p. (in Russian) / Сухов А.О. Разработка инструментальных средств создания визуальных предметно-ориентированных языков. Диссертация кандидата физико-математическиъх наук. М., 2013 г. 256 стр.
[6]. Mikov A.I. Performance evaluation: textbook. Krasnodar, Kuban State University, 2013, 89 p.
[7]. Suvorov N.M., Lyadova L.N. HP-Graph as a Basis of a DSM Platform Visual Model Editor. Suvorov N.M., Lyadova L.N. HP-Graph as a Basis of a DSM Platform Visual Model Editor. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 149-160. DOI: 10.15514/ISPRAS–2020–32(2)–12.
[8]. Parra F. Dean T. Survey of Graph Rewriting applied to Model Transformations. In Proc. of the 2nd International Conference on Model-Driven Engineering and Software Development, 2014, pp. 431-441.
[9]. Ehrig H., Ehrig K., Prange U., Taentzer G. Fundamentals of Algebraic Graph Transformation. Springer 2006, 403 p.
[10]. Yan X., Yu P.S., Han J. Graph Indexing: A Frequent Structure-based Approach. In Proc. of the ACM SIGMOD International Conference on Management of Data, 2004, pp. 335-346.
[11]. Fan W. Graph pattern matching revised for social network analysis. In Proc. of the 15th International Conference on Database Theory, 2012, pp. 8-21.
[12]. Liu C., Lio B., Kropatsch W, eds. Advances in Graph-based Pattern Recognition. Pattern Recognition Letters, vol. 87, 2017, 230 p.
[13]. Pržulj N., Corneil D.G., Jurisica I. Efficient Estimation of Graphlet Frequency Distributions in Protein–protein Interaction Networks. Bioinformatics, vol. 22, no. 8, 2006, pp. 974-980.
[14]. Han M., Kim H. et al Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together. In Proc. of the ACM SIGMOD International Conference on Management of Data, 2019, pp. 1429-1446.
[15]. Carletti V., Foggia P. et al. Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 4, 2018, pp. 804-818.
[16]. Ren X., Wang J. Exploiting Vertex Relationships in Speending up Subgraph Isomorphism over Large Graphs. Proceedings of the VLDB Endowment, vol. 8, no. 5, 2015, pp. 617-628.
[17]. Lee J., Han W. et ak. An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases. In: Proceedings if the VLDB Endowment, vol. 6, no. 2, 2012, pp. 133-144.
[18]. Seriy A.P., Lyadova L.N. An Approach to Graph Matching in the Component of Model Transformations. In Proc. of the 7th Spring/Summer Young Researchers' Colloquium on Software Engineering, 2013, pp. 41-46.
[19]. Ullmann J.R. An Algorithm for Subgraph Isomorphism. Journal of the ACM, vol. 23, no. 1, 1976, pp. 31-42.
[20]. Cordella L.P., Foggia P. et al. (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 10, 2004, pp. 1367-1372.
[21]. Carletti V., Foggia P., Vento M. VF2 Plus: An Improved version of VF2 for Biological Graphs. Lecture Notes in Computer Science, vol. 9069, 2015, pp. 168-177.
[22]. Carletti V., Foggia P. et al. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with VF3. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 4, 2018, pp. 804-818.
[23]. Han W., TurboISO: Towards UltraFast and Robust Subgraph Isomorphism Search in Large Graph Databases. In Proc. of the ACM SIGMOD International Conference on Management of Data, 2013, pp. 337-348.
[24]. Bi F., Chang L., Lin X., Qin L., Zhang W. Efficient Subgraph Matching by Postponing Cartesian Products. In Proc. of the ACM SIGMOD International Conference on Management of Data, 2016, pp. 1199-1214.
[25]. Shang H., Zhang Y., Lin X., Yu J.X. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. Proceedings if the VLDB Endowment, vol. 1, no. 1, 2008, pp. 364-375.
[26]. Zhao P., Han J. On graph query optimization in large networks. Proceedings if the VLDB Endowment, vol. 3, 2010, pp. 340-351.

Information about authors / Информация об авторах

Nikolai Mikhailovich SUVOROV – student. His research interests include language-oriented programming, modeling, language toolkits.

Николай Михайлович СУВОРОВ – студент бакалавриата НИУ ВШЭ-Пермь. Научные интересы включают языково-ориентированное программирование, моделирование, языковые инструментарии.

Lyudmila Nickolaevna LYADOVA – Candidate of Physical and Mathematical Sciences, associate professor of the Department of Information Technology in Business of the HSE (Perm). Research interests: modeling languages, modeling tools, domain specific modeling, language toolkits, semantic modeling.

Людмила Николаевна ЛЯДОВА – кандидат физико-математических наук, доцент, доцент кафедры информационных технологий в бизнесе НИУ ВШЭ (Пермь). Сфера научных интересов: языки моделирования, средства моделирования, предметно-ориентированное моделирование, языковые инструментарии, семантическое моделирование.

DOI: 10.15514/ISPRAS-2021-33(4)-13

Полная решающая процедура для теории ограниченной адресной арифметики

[1,2] *Р.Ф. Садыков, ORCID: 0000-0002-2792-2465 <sadykov@ispras.ru>*
[2] *М.У. Мандрыкин, ORCID: 0000-0002-9306-7719 <mandrykin@ispras.ru>*

[1] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*
[2] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*

Аннотация. Процесс разработки кода на Си довольно часто сопровождается появлением ошибок, связанных с использованием указателей и адресов памяти. В связи с этим возникает потребность создания инструментов автоматизированной проверки программ. Одним из методов, применяемых такими инструментами проверки, является использование решающих процедур на основе SMT-решателей. Но в то же время разрешимые логики (комбинации логических теорий), требуемые для адекватного моделирования указателей в языке Си, непосредственно не присутствуют в стандарте SMT-LIB и не реализованы в большинстве существующих SMT-решателей. Одним из возможных способов поддержки таких логик является непосредственная их реализация в каком-либо SMT-решателе, однако такой подход часто оказывается трудоемким (требуется изменение исходного кода решателя), негибким (трудно модифицировать сигнатуру и семантику новых теорий) и ограниченным (требуется отдельная реализация поддержки теории в каждом используемом решателе). Другим способом является реализация пользовательских стратегий конечного инстанцирования кванторов для сведения формул в неподдерживаемых логиках к формулам в широко распространенных разрешимых логиках, таких как QF_UFLIA. В данной статье представлена процедура инстанцирования лемм для трансляции формул в теории типизированной адресной арифметики в логику QF_UFLIA. Для процедуры трансляции даны доказательства корректности и полноты, а также описана формализация этих доказательств в системе Isabelle/HOL. Сама теория адресной арифметики сформулирована на основе известных ошибок использования адресной арифметики в языке Си, а также спецификаций семантики этих операций из стандарта языка. Аналогичные доказательства и процедура также могут быть сформулированы для фрагмента теории бит-векторов (монотонные формулы над равенствами между выражениями с побитовыми операциями, такими как исключающее “или”, сдвиг, конкатенация, экстракция, отрицание). Представленная в статье процедура трансляции, в частности, позволяет легко реализовать полный метод доказательства утверждений в теории адресной арифметики в системе Isabelle/HOL на основе существующей в этой системе поддержки SMT-решателей (Z3 или VeriT).

Ключевые слова: статическая верификация; задача выполнимости формул в теориях; адресная арифметика; решающие процедуры

Для цитирования: Садыков Р.Ф., Мандрыкин М.У. Полная решающая процедура для теории ограниченной адресной арифметики. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 177-194. DOI: 10.15514/ISPRAS–2021–33(4)–13

Благодарности. Работа выполнена при поддержке Минобрнауки России в рамках проекта №AAAA-A19-119110790086-3.

Complete decision procedure for the theory of bounded pointer arithmetic based on quantifier instantiation and SMT

[1,2] *R. Sadykov ORCID: 0000-0002-2792-2465 <sadykov@ispras.ru>*
[2] *M. Mandrykin ORCID: 0000-0002-9306-7719 <mandrykin@ispras.ru>*

[1] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. The process of developing C programs is quite often prone to errors related to the uses of pointer arithmetic and operations on memory addresses. This promotes a need in developing various tools for automated program verification. One of the techniques frequently employed by those tools is invocation of appropriate decision procedures implemented within existing SMT-solvers. But at the same time both the SMT standard and most existing SMT-solvers lack the relevant logics (combinations of logical theories) for directly and precisely modelling the semantics of pointer operations in C. One of the possible ways to support these logics is to implement them in an SMT solver, but this approach can be time-consuming (as requires modifying the solver's source code), inflexible (introducing any changes to the theory's signature or semantics can be unreasonably hard) and limited (every solver has to be supported separately). Another way is to design and implement custom quantifier instantiation strategies. These strategies can be then used to translate formulas in the desired theory combinations to formulas in well-supported decidable logics such as QF_UFLIA. In this paper, we present an instantiation procedure for translating formulas in the theory of bounded pointer arithmetic into the QF_UFLIA logic. We formally proved soundness and completeness of our instantiation procedure in Isabelle/HOL. The paper presents an informal description of this proof of the proposed procedure. The theory of bounded pointer arithmetic itself was formulated based on known errors regarding the correct use of pointer arithmetic operations in industrial code as well as the semantics of these operations specified in the C standard. Similar procedure can also be defined for a practically relevant fragment of the theory of bit vectors (monotone propositional combinations of equalities between bitwise expressions). Our approach is sufficient to obtain efficient decision procedures implemented as Isabelle/HOL proof methods for several decidable logical theories used in C program verification by relying on the existing capabilities of well-known SMT solvers, such as Z3 and proof reconstruction capabilities of the Isabelle/HOL proof assistant.

Keywords: static verification; quantifier instantiation; SMT formulas; SMT solvers; automated decision procedures; software verification

For citation: Sadykov R., Mandrykin M. Complete decision procedure for the bounded theory of pointer arithmetic based on quantifier instantiation and SMT. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021. pp. 177-194 (in Russian). DOI: 10.15514/ISPRAS–2021–33(4)–13

Acknowledgments. This work was supported by the Ministry of Education and Science of the Russian Federation within the framework of project No. AAAA-A19-119110790086-3.

1. Введение

Идея формулирования специфических логических теорий, разрешимых с помощью трансляции в одну из логик, поддерживаемых современными SMT-решателями, возникла у авторов данной статьи в контексте дедуктивной верификации Си-программ в среде Isabelle-HOL. Поэтому вначале рассмотрим проблемы, возникающие при автоматизации рутинных доказательств (часто необходимых при верификации программ) в системах интерактивной верификации, таких как Isabelle/HOL.

На сегодняшний день в области автоматизированной дедуктивной верификации Си-программ распространены два основных подхода к автоматизации доказательств – использование специфических тактик и использование общих решающих процедур, таких как SMT- и суперпозиционные решатели. При этом оба подхода на практике обладают достаточно существенными недостатками. Подход с использованием узко специфических тактик часто страдает из-за слишком узко определенного набора преобразований или

доказательств, выполняемых или поддерживаемых каждой конкретной тактикой, таких как, например, замена некоторого выделенного подтерма на эквивалентный (переписывание) или поддержка теории линейной целочисленной арифметики над неинтерпретируемыми константами. Хотя каждое конкретное применение такой узко специфичной тактики, как правило, приводит к хорошо определенному и вполне ожидаемому результату, число взаимодействий с интерактивным инструментом верификации, необходимое для доказательства типично возникающих целей при использовании специфичных тактик обычно оказывается весьма большим, что сильно увеличивает трудоемкость верификации. Использование же общих решающих процедур обычно характеризуется большой непредсказуемостью результата, так как лежащие в основе таких инструментов подходы в общем случае не обладают полнотой для формул из того класса, для которого они часто применяются. При этом в силу неполноты практическое значение обычно имеет только успешное применение решателя для полного разрешения какой-либо цели доказательства, возможность которого часто трудно предсказать заранее и которое часто оказывается неустойчивым к небольшим изменениям целевого утверждения.

Некоторым продвижением по сравнению с описанными практически используемыми подходами кажется применяемый в инструменте Isabelle/HOL [1] подход на основе структурированных доказательств на языке Isabelle/Isar, в котором каждая цель доказательства явно ставится пользователем с учетом возможностей реализованных в инструменте методов автоматизации логического вывода, что позволяет наилучшим образом подстраивать структуру доказательств под возможности инструментов автоматического доказательства. В сочетании с достаточно мощными методами, основанными на переписывании термов (*simp*, *auto*), методе табло (*blast*) и использовании суперпозиционных (*metis*, *meson*) и SMT-решателей (*smt*), такой подход позволяет существенно увеличить удобство использования инструмента верификации.

В данной статье мы рассматриваем одно из направлений развития такого подхода к верификации программ, позволяющего достичь полноты решающей процедуры для практически значимого класса целевых утверждений, а именно для формул, описывающих утверждения об адресной арифметике в языке Си. Вначале мы формулируем абстрактную аксиоматическую теорию адресной арифметики для языка Си с учетом как практического опыта исправления различных известных ошибок, связанных с адресной арифметикой, в индустриальном коде на языке Си, так и спецификации операций адресной арифметики, приведенной в последних версиях стандарта языка. Формулируемая теория, однако, не полностью формализует все аспекты корректного использования адресной арифметики в языке Си, но ограничивается достаточно значимым фрагментом тех свойств указателей, которые не связаны с состоянием адресуемой памяти. Таким образом, в частности, теория не включает операцию разыменования и полную формализацию понятия валидного указателя. Эти аспекты формализации модели памяти сильно зависят от конкретного используемого подхода к ее моделированию, такому как сепарационная логика или использование динамических фреймов и остаются за рамками данной работы. Однако, формализованная в данной статье модель совместима с различными методами моделирования памяти и полностью формализует понятия блока памяти (или происхождения (*provenance*) указателя), адреса, включая отсутствие переполнения, нулевого указателя, относительного выравнивания указателя на элемент массива и отчасти, понятие валидности, так как для невалидного указателя невозможно в общем случае показать отсутствие переполнения адреса.

1.1 Примеры известных ошибок при работе с указателями

Для формулирования теории ограниченной адресной арифметики рассмотрим вначале практические примеры известных ошибок, показывающие необходимость моделирования

блока памяти (происхождения) указателя, корректного соотнесения блока с адресом, а также проверки переполнения адреса в операциях адресной арифметики.

Вначале рассмотрим соотношение понятия указателя и адреса в языке Си, к формализации которого можно прийти, рассмотрев два важных примера работы оптимизирующих компиляторов, один из которых реально встречался в промышленном коде. Этот пример из исправления, включенного в одну из версий ядра ОС Linux [2]:

```
extern struct builtin_fw __start_builtin_fw[];
extern struct builtin_fw __end_builtin_fw[];
...
for (b_fw = __start_builtin_fw; b_fw != __end_builtin_fw; b_fw++) {
...
```

Здесь условие входа в цикл **for** оптимизировалось компилятором в тождественную истину, что впоследствии вызывало выход за границу выделенной страницы памяти. Такая семантика сравнения указателей в данном фрагменте совместима со стандартом языка Си, так как поведение компилятора при сравнении указателей на разные выделенные объекты в памяти (в данном случае массивы `__start_builtin_fw` и `__end_builtin_fw`), за исключением специального случая непосредственно следующих друг за другом объектов, является неопределенным поведением [3] и поддается произвольной трактовке в каждой конкретной реализации компилятора. Этот пример показывает, что понятие указателя в языке Си не сводится к понятию адреса, так как любой адрес в принципе может быть получен из любого другого с помощью операции адресной арифметики, а именно прибавления положительного или отрицательного смещения. Указатели же на элементы разных объектов в памяти с точки зрения компилятора рассматриваются как заведомо различные (не равные) независимо от прибавления к ним произвольного смещения. Это свойство указателей еще более наглядно демонстрирует следующий пример:

```
#include <stdio.h>
int main(void) {
    int a, b;
    int *p = &a;
    int *q = &b - 1;
    printf("%p %p %d", p, q, p == q);
    return 0;
}
```

В этом примере результатом работы такой программы в некоторых совместимых со стандартом ANSI C компиляторах со включенными оптимизациями (например, GCC 11.1.1 с опцией `-O3`) может быть, например, такая строка:

```
0x7ffe32e8ece8 0x7ffe32e8ece8 0
```

Видно, что хотя значения указателей `p` и `q`, представленные в виде адресов, совпадают, значение выражения `p == q` вычисляется компилятором в ложь (`0` в языке Си) вообще независимо от значений соответствующих адресов, потому что предполагается непересечение указателей на разные объекты памяти `a` и `b`, для которых не гарантируется никакое конкретное размещение (в том числе последовательное). Ориентируясь на приведенные примеры, можно предположить, что выделенный объект можно все же идентифицировать по его начальному адресу и представлять указатель в виде упорядоченной пары (a_0, a), где a — адрес, являющийся значением указателя, а a_0 — начальный адрес некоторого выделенного объекта, приписанного указателю компилятором. Однако с помощью небольшой модификации последнего примера можно получить еще один пример, опровергающий такую модель указателя:

```
#include <stdio.h>
int *f(int *u) {
```

```
    int b, *pb = &b;
    printf("%p %p %d", u, &b, u == &b);
    return pb;
}
int main(void) {
    int a, *b = f(&a);
    f(b);
    f(b);
    return 0;
}
```

При использовании компилятора CLANG 12.0.1 с опцией `-O3` результатом работы такой программы могут быть, например, строки:

```
0x7ffd0382fa1c 0x7ffd0382fa18 0
0x7ffd0382fa18 0x7ffd0382fa18 0
0x7ffd0382fa18 0x7ffd0382fa18 0
```

Здесь, в отличие от предыдущих примеров, не используется прибавление к указателю смещения, а начальные адреса объектов в памяти совпадают. Но один из объектов, на который указывает указатель из параметра функции, уже не является выделенным. Таким образом, указатель на ранее выделенный объект считается всегда отличным от указателя на любой другой выделенный (в текущем состоянии или ранее) объект, независимо от начального адреса каждого из этих объектов. Выразить такое свойство можно, в частности, с помощью модели указателя в виде упорядоченной пары (l, a), где a — как и ранее — значение адреса, содержащееся в указателе, а l — уникальный идентификатор выделенного блока, который присваивается при любом выделении памяти и никогда не повторяется.

1.2 Требуемые понятия

Выделим теперь понятия, которые могут использоваться для спецификации операций над указателями, и определим, какие из них будут представлены в разрабатываемой модели адресной арифметики и каким образом. Помимо хранимого в указателе адреса и идентификатора блока памяти, который не меняется при сдвиге указателя на любое смещение, при описании операций над указателями, как в стандарте ANSI C, так и в повседневной программистской практике, часто используются следующие понятия:

- *Размер* выделенного объекта (в смысле количества элементов). Это понятие часто используется неявно в виде упоминания «последнего элемента массива». Проблема формализации размера выделенного объекта в отрыве от соответствующей модели памяти в том, что этот размер зависит от рассматриваемого состояния программы и меняется при выделении или освобождении памяти, занимаемой объектом. Поэтому в теорию адресной арифметики, формализуемую в данной статье, это понятие не может быть включено явно. Однако, как будет ясно далее при рассмотрении семантики операций адресной арифметики, проверки переполнения адреса потребуют дополнительных предусловий вида $0 \leq a$ и $a \leq A$, где a — адрес, хранящийся в указателе. Эти предусловия могут выводиться, в частности, из условия выделенности адресуемой указателем памяти, которое формализуется уже в рамках выбранной модели памяти, а не адресной арифметики, и связано с текущим размером выделенного объекта.
- Начальный (*базовый*) адрес объекта. Это понятие возникает, например, при формализации спецификаций функций управления памятью (таких как `malloc` и `free`) и, в отличие от размера объекта, не меняется в ходе выполнения программы и, таким образом, не зависит от ее состояния. Так как в силу семантики вложенных объектов, принятой в языке Си и предполагающей непосредственное плоское размещение вложенных объектов одного за другим, в полной мере различными объектами в

программе могут считаться только объекты, изначально выделенные в разных точках программы (в разных объявлениях переменных либо при различных вызовах функций управления памятью), базовый адрес объекта фактически становится атрибутом соответствующего ему блока памяти. Поэтому в теории адресной арифметики базовый адрес формализуется как функция от идентификатора выделенного блока. Будем далее обозначать эту функцию от блока l как $\boldsymbol{base}\ (l)$.

- *Смещение* объекта относительно базового адреса. В ситуации, когда основные операции над указателями (сдвиг, вычитание, сравнение) определены только для указателей на элементы одного изначально выделяемого базового массива объектов и указателей на вложенные в элементы этого массива подобъекты, чаще всего имеет смысл говорить не об абсолютном адресе объекта в памяти, а о его смещении от начала (базового адреса) выделенного объекта (массива). При таком подходе адрес объекта может быть всегда представлен как сумма $\boldsymbol{address}(p) = \boldsymbol{base}(\boldsymbol{block}(p)) + \boldsymbol{offset}(p)$, где p — указатель, $\boldsymbol{block}(p)$ – идентификатор блока памяти, приписанного указателю p, $\boldsymbol{offset}(p)$ – смещение указателя p от базового адреса (целое число байт), а $\boldsymbol{address}(p)$ – значение адреса, хранящееся в указателе p.
- Еще одно понятие неявно возникает при формализации такого термина, как «элементы одного массива». Чтобы понять существенное отличие этого понятия от элементов одного блока памяти, рассмотрим для примера следующий массив:

```
struct s {
   int b[2];
   char c;
   int c[2];
} a[2];
```

В этом примере можно видеть по крайней мере четыре различных разновидности указателей типа `int *`: `&a[0].b[0]`, `&a[0].b[1]`, `&a[0].c[0]` и `&a[1].b[0]`. При этом про пары указателей (`&a[0].b[0]`, `&a[0].b[1]`) и (`&a[0].b[0]`, `&a[1].b[0]`) имеет смысл говорить как об указателях на элементы одного массива (внутреннего массива `b`, либо внешнего массива `a`, а про оставшиеся пары (`&a[0].b[0]`, `&a[0].c[0]`), (`&a[0].b[1]`, `&a[0].c[0]`), (`&a[0].b[1]`, `&a[1].b[0]`) и (`&a[0].c[0]`, `&a[1].b[0]`) — в общем случае нет. Для формализации такого понимания элементов одного массива хорошо подходит понятие *выравнивание* указателя, определенное как остаток от деления его абсолютного адреса, либо смещения относительно базового адреса, на размер типа указателя (в данном случае `int`). Если считать, что смещение поля `c` структуры `s` относительно ее поля `b` фиксировано некоторой неизвестной константой d, а размер структуры `s` равен также неизвестной фиксированной константе s, то выравнивания указателей в соответствующих парах будут выражаться как $(0, 0)$, $(s \,\%\, sizeof(int), s \,\%\, sizeof(int))$ соответственно для первых двух пар и как $(0, d \,\%\, sizeof(int))$, $(0, d \,\%\, sizeof(int))$, $(0, (s + d) \,\%\, sizeof(int))$ и $(d \,\%\, sizeof(int), (s + d) \,\%\, sizeof(int))$ соответственно для четырех остальных. В общем случае константы s и d могут быть подобраны таким образом, что выравнивания для последних четырех (неупорядоченных) пар указателей не совпадут. Таким образом, остаток от деления смещения на размер указателя дает желаемую семантику.

Перед формализацией операций над указателями во введенной модели указателей как упорядоченных пар (l, a) необходимо отдельно выделить понятие нулевого указателя. Это представляет собой некоторую трудность, потому что с одной стороны такой указатель должен быть отличен от любого указателя на выделенный объект, с другой, поскольку в реализации указатели представляются только хранимыми в них адресами, в принципе нулевой указатель может быть получен из указателей на различные выделенные объекты с помощью сдвига на соответствующее отрицательное смещение, а помимо этого любые два

нулевых указателя должны считаться равными друг другу. Чтобы обойти трудность формализации нулевого указателя, применим верхнее приближение, оставив поведение операции сдвига неопределенным при получении указателя с нулевым адресом, а также исключив из определений всех операций, кроме сравнения на равенство, операции над любыми указателями, имеющими нулевой адрес. Такая формализация не полностью покрывает все возможные поведения, разрешенные стандартом, в частности, она предполагает, что нулевой указатель хранит нулевой адрес, но на практике это соответствует поведению практически всех существующих реализаций языка. Кроме этого, такая формализация не исключает семантику, при которой помимо указателей на объекты (валидные и невалидные) и нулевого указателя могут существовать и другие указатели с нулевым адресом, которые тем не менее отличны от нулевого указателя.

1.3 Семантика основных операций

Определим теперь семантику основных операций с указателями с использованием введенных понятий и соответствующих им формальных обозначений:

1.3.1 Сдвиг указателя

Стандарт ANSI C (пункт 6.5.6(8)) определяет сложение указателей в предположении отсутствия переполнения адреса, полученного в результате сдвига. Это практически важное предположение, которое приводило к возникновению потенциально критических ошибок в промышленном коде [4]. Помимо отсутствия переполнения, операция сдвига указателя должна быть не определена для аргумента либо результата с нулевым адресом. Сдвиг указателя также порождает указатель на тот же объект памяти (без учета вложенности, то есть в нашей модели — блок), что и исходный указатель. Эти соображения приводят к следующей аксиоматической формализации операции сдвига указателя ($+_p$):

$$\forall p\ i.\ \boldsymbol{address}(p) \neq 0 \wedge \boldsymbol{range}(p, i) \longrightarrow p \triangle p +_p i,$$

$$\forall p\ i.\ \boldsymbol{address}(p) \neq 0 \wedge \boldsymbol{range}(p, i) \longrightarrow \boldsymbol{offset}\left(p +_p i\right) = \boldsymbol{offset}\ (p) + s \times i, \text{где}$$

$$\boldsymbol{range}(p, i) \equiv 0 < \boldsymbol{address}(p) + s \times i \wedge \boldsymbol{address}(p) + s \times i \leq A,$$

$$p \triangle q \equiv \boldsymbol{block}(p) = \boldsymbol{block}(q).$$

Несмотря на то, что такая формализация не полностью ограничивает операцию сдвига указателя в соответствии со стандартом, не требуя выделенности соответствующего объекта в памяти и не исключая выход за границу выделенного массива объектов более, чем на один элемент, на практике доказательство отсутствия переполнения (условие $\boldsymbol{range}(p, i)$) потребует использования условия выделенности аргумента или результирующего указателя, которое будет формализовано с использованием соответствующей модели памяти. Далее в статье будем рассматривать один тип указателей на некоторый фиксированный тип с ненулевым размером s и адресное пространство с максимально допустимым адресом A. Арифметика с указателями на типы других размеров может быть формализована аналогично, а приведение типа указателя может быть формализовано через комбинацию операций получения адреса ($\boldsymbol{address}(p)$), идентификатора блока $\boldsymbol{block}(p)$ (для исходного типа указателя) и операцию получения нового указателя $(l, a)_p$ (для целевого типа указателя), рассмотренную далее.

1.3.2 Разность указателей

Разность указателей определена в стандарте языка только для элементов одного массива. Учитывая приведенные соображения по поводу использования выравниваний для идентификации элементов одного массива и исключения из семантики операций нулевых

указателей, получим следующую аксиоматическую формализацию операции разности указателей ($-_p$):

$$\forall p\ q.\ p \triangle_0 q \wedge p \parallel q \longrightarrow s \times (p -_p q) = \boldsymbol{offset}(p) - \boldsymbol{offset}(q), \text{где}$$
$$p \triangle_0 q \equiv p \triangle q \wedge \boldsymbol{address}(p) \neq 0 \wedge \boldsymbol{address}(q) \neq 0,$$
$$p \parallel q \equiv \boldsymbol{align}(p) = \boldsymbol{align}(q).$$

Для формализации выравнивания ($\boldsymbol{align}(p)$) сразу учтем выразительные возможности целевой логики — комбинации теорий бесквантормой линейной целочисленной арифметики и конгруэнтности с неинтерпретируемыми функциями (логика QF_UFLIA). В этой теории операцию получения остатка от деления на константу s можно выразить с помощью следующих аксиом (вопрос о полноте инстанцирования кванторов рассмотрен далее и является основным предметом рассмотрения данной статьи):

$$\forall p.\ \boldsymbol{offset}(p) = s \times \boldsymbol{quot}(p) + \boldsymbol{align}(p),$$
$$\forall p.\ 0 \leq \boldsymbol{align}(p) \wedge \boldsymbol{align}(p) < s.$$

Здесь $\boldsymbol{quot}(p)$ – неинтерпретируемая функция, использованная для сколемизации квантора существования частного от деления.

1.3.3 Сравнение указателей на неравенство

В стандарте ANSI C сравнение указателей определено только для указателей на один и тот же выделенный объект памяти без учета вложенности, что соответствует условию $p \triangle q$. С учетом исключения нулевого указателя, для которого сравнение на неравенство не определено, получаем условие $p \triangle_0 q$. Без ограничения общности будем рассматривать только одну операцию сравнения указателей на неравенство ($\leq_p$):

$$\forall p\ q.\ p \triangle_0\ q \longrightarrow p \leq_p q = \boldsymbol{offset}(p) \leq \boldsymbol{offset}(q).$$

1.3.4 Сравнение указателей на равенство

Так как в стандарте разрешается сравнение указателей на два различных выделенных объекта памяти, а также сравнение на равенство с нулевым указателем (в том числе нулевых указателей друг с другом), для операции сравнения на равенство не удается формализовать никаких значимых предусловий. Условие выделенности обоих адресуемых объектов не может быть формализовано в отрыве от используемой модели памяти. Поэтому для сравнения указателей на равенство предлагается непосредственно использовать равенство соответствующих значений из теории конгруэнтности (=). Для операций сравнения указателей на равенство, используемых в коде программы, можно отдельно задать предусловие выделенности для случая сравнения указателей на различные объекты (с использованием модели памяти). В то же время формализованная теория адресной арифметики (даже без этого предусловия) дает достаточно точное приближение возможных поведений программ после компиляции, учитывая приписываемые компилятором указателям идентификаторы выделенных объектов в памяти. Кроме этого, точное определение (не)выделенности адресуемого объекта достаточно сложно в реализации (точнее, в общем случае алгоритмически неразрешимо), чтобы вероятность намеренного использования оптимизирующим компилятором неопределенного поведения, разрешенного стандартом для сравнения с указателем на невыделенный объект, была очень невысокой.

1.3.5 Получение нового указателя: по адресу и идентификатору блока

Эта операция необходима для полноты определения теории адресной арифметики, а также для выражения операции приведения типа указателя и операции приведения к указателю значения целочисленного типа. Несмотря на то, что в исходном коде программы такая операция фактически используется для одного аргумента — адреса, используемая в языке семантика указателей фактически требует аннотирования операций приведения типа блоком

памяти полученного указателя. В рассматриваемой формализации мы не требуем никакого конкретного способа определения результирующего блока памяти, оставляя возможность для различных конкретных реализаций в инструментах верификации. Формализация операции получения нового указателя ($(l, a)_p$) должна устанавливать два основных свойства полученного указателя — его адрес и блок памяти, а также устанавливать полноту этих свойств, полностью определяющих значение указателя:

$$\forall l\ a.\ 0 \le a \wedge a \le A \longrightarrow \boldsymbol{block}((l, a)_p) = l,$$
$$\forall l\ a.\ 0 \le a \wedge a \le A \longrightarrow \boldsymbol{address}((l, a)_p) = a,$$
$$\forall p.\ \left(\boldsymbol{block}(p), \boldsymbol{address}(p)\right)_p = p.$$

Для полноты аксиоматического определения теории адресной арифметики остается задать последнюю аксиому, отражающую ограниченность размера адресного пространства:

$$\forall p.\ 0 \le \boldsymbol{address}(p) \wedge \boldsymbol{address}(p) \le A.$$

Таким образом, в данной статье рассматривается аксиоматическая формулировка семантики теории ограниченной (то есть с ограниченным размером адресного пространства) адресной арифметики, включающей в себя 8 основных операций: $(l, a)_p$, $\boldsymbol{block}(p)$, $\boldsymbol{offset}(p)$, $\boldsymbol{align}(p)$, $\boldsymbol{base}(p)$, $p +_p i$, $p -_p q$ и $p \le_p q$. В статье рассматривается комбинация этой теории с логикой QF_UFLIA, определенной в формате SMT-LIB. Соответствующая задача о выполнимости формул в полученной комбинации теорий решается с помощью трансляции исходных формул в соответствующие равновыполнимые формулы в базовой логике QF_UFLIA, которая поддерживается как большинством современных SMT-решателей, так и процедурой воспроизведения доказательств, реализованной в системе Isabelle/HOL. Оставшаяся часть статьи практически целиком посвящена формулировке и доказательству корректности и полноты соответствующей процедуры трансляции формул из полученной теории ограниченной адресной арифметики (далее — BPA) в логику QF_UFLIA.

Сформулированная решающая процедура для теории BPA также была реализована в виде метода в системе автоматизированных доказательств Isabelle/HOL. Этот метод состоит в инстанцировании аксиом теории BPA термами из исходной формулы в соответствии с представленной в статье полной процедурой трансляции, интерпретации полученной формулы в логике QF_UFLIA (при этом символы теории BPA становятся неинтерпретируемыми), применении соответствующей существующей полной решающей процедуры с помощью вызова SMT-решателя и последующего восстановления доказательства средствами системы Isabelle/HOL. В свою очередь, доказательство полноты предложенного метода основывается на преобразовании полученной от SMT-решателя модели в логике QF_UFLIA в модель теории BPA.

Как отмечалось ранее в предыдущей статье [5], нам не требуется вносить изменения в инструменты воспроизведения доказательств системы Isabelle/HOL и SMT-решатель. Кроме этого, инстанцирование аксиом исходной теории в соответствии с предложенной процедурой увеличивает исходный размер анализируемой формулы не более, чем линейно. Основной целью нашей работы являлось доказательство полноты и корректности предложенной процедуры преобразования формул из теории BPA в логику QF_UFLIA.

2. Существующие модели адресной арифметики

Как правило, при моделировании семантики практически значимых фрагментов языка Си модель адресной арифметики рассматривается совместно с моделью памяти, то есть с операциями чтения и записи значения по указателю. Тем не менее, в тех моделях семантики языка (в частности, моделях памяти), которые учитывают возможность использования разрешенных стандартом оптимизаций, как правило, легко может быть выделен фрагмент, соответствующий формализованной в данной статье теории BPA. Среди четырех

рассмотренных нами работ [6-9] по моделированию памяти для языка Си фрагмент, соответствующий теории BPA, может быть выделен в двух работах — [7] и [9]. В обеих этих моделях для представления указателей используются пары вида (l, o), отличные от представленных в этой статье указателей вида (l, a) только непосредственным использованием смещения вместо адреса. В таких моделях можно определить соответствующие понятия смещения и базового адреса так, что аксиомы теории BPA будут выполнены (доказанны) как леммы (теоремы) соответствующей модели памяти. В таком случае теория BPA будет являться фрагментом модели памяти, и ее семантика будет надежно приближать семантику всех присутствующих в ней операций адресной арифметики, кроме сравнения указателей на равенство. Предусловие операции сравнения указателей на равенство не выражается в понятиях, формализованных в рамках самой теории BPA (в ней нет понятия валидности указателя), и поэтому приближается в ней снизу, то есть определяется семантически более сильная операция. Это не мешает, однако, формализовать также и корректное (верхнее) приближение операции сравнения указателей на равенство (с использованием формализованного в модели памяти понятия валидности), но для полученной в результате теории не будет в полной мере работать процедура, представленная в данной статье. Это означает, что в рамках моделей памяти, представленных в работах [7] и [9] представленная в этой статье решающая процедура для теории BPA будет неполной только для случаев сравнения на равенство указателей на два различных блока памяти, по крайней мере один из которых не является валидным. Тем не менее, так как такие случаи на практике достаточно редки, полная решающая процедура для теории BPA может быть существенно использована как для упрощения доказательств в подобных моделях памяти, так и для поиска контрпримеров утверждений в рамках этой теории, для которых сохраняется полнота.

Представленные в двух других работах ([6] и [8]) модели памяти делают предположения о семантике адресной арифметики, существенно отличающиеся как от семантики теории BPA, так и от семантики самого языка Си с точки зрения оптимизирующего компилятора. В работе [6] (использованной на практике при верификации кода микроядра L4[11]) не рассматриваются уникальные идентификаторы блоков, приписываемые различным объектам в памяти программы оптимизирующим компилятором. Указатели непосредственно соответствуют хранимым адресам. В работе [8] значения адресов объектов в памяти программы не ограничены сверху и могут быть представлены математическими целыми (предполагается отсутствие переполнений значения указателя). Таким образом, обе этих модели упускают существенные на практике аспекты семантики указателей в языке Си. Другие примеры неочевидного поведения программ при использовании операций вычитания и присваивания указателей, соответствующие тем не менее стандарту языка C11, приведены в работе [10].

3. Основные определения

В стандарте SMT-LIB [13], QF_UFLIA — это логика бескванторных формул с неинтерпретируемыми символами и равенством в комбинации теорий линейной целочисленной арифметики и неинтерпретируемых функций. Логику QF_UFLIA можно рассматривать как комбинацию логик QF_LIA и QF_UF. QF_LIA обозначают замкнутые бескванторные формулы с равенством в теории линейной целочисленной арифметики (LIA). В сигнатуру этой теории входят следующие функциональные символы: $\{+, c \times \cdot, \leq\}$, где c — целочисленная константа. QF_UF обозначает замкнутые бескванторные формулы с равенством в теории неинтерпретируемых функций.

Теорию ограниченной адресной арифметики BPA будем задавать аксиоматически как расширение логики QF_UFLIA. В сигнатуру данной теории включим следующие функциональные символы:

$\{ \cdot \leq_p \cdot,\ \cdot +_p \cdot,\ \cdot -_p \cdot,\ \boldsymbol{block}(\cdot),\ \boldsymbol{base}(\cdot),\ \boldsymbol{offset}(\cdot),\ \boldsymbol{align}(\cdot),\ (\cdot,\cdot)_p \}$.

Термы в этой теории могут быть двух различных сортов – указатели и целые. Семантика предполагает моделирование указателей как упорядоченных пар — (целое число, целое число), где первый элемент пары соответствует уникальному идентификатору блока памяти, а второй — хранимому в указателе адресу. Произвольный указатель можно таким образом получить при помощи соответствующей функции $(\cdot,\cdot)_p$. Неформальная семантика остальных функциональных символов была объяснена ранее во введении.

Для решения задачи о выполнимости формул в теории BPA будем использовать процедуру трансляции в логику QF_UFLIA. Процедура преобразования исходной формулы F состоит из последовательного инстанцирования аксиом теории BPA. Инстанцированием аксиомы A для некоторой формулы F будем называть взятие конъюнкции F с экземпляром аксиомы A, в котором вместо всех подкванторных переменных подставлены некоторые подтермы формулы F соответствующего сорта. Подтермы формулы F выбираются согласно правилам процедуры инстанцирования — триггерам, задаваемым отдельно для каждой аксиомы теории. Инстанцирование выполняется последовательно для всех аксиом и подтермов исходной формулы F, которые удовлетворяют заданным триггерам. В результате получается формула F^*, которая интерпретируется в логике QF_UFLIA. На формуле F^* запускается решающая процедура SMT-решателя для этой логики, которая является полной. В случае невыполнимости формулы F^* в результате получается доказательство, которое затем воспроизводится (сертифицируется) средствами системы Isabelle/HOL [14], [15]. В случае выполнимой формулы полнота решающей процедуры гарантирует существование модели формулы F^*.

Для доказательства полноты процедуры инстанцирования рассматриваем случай, когда мы получили некоторую модель R формулы F^*. Назовем модель R формулы F^* реализацией формулы F. В данной статье мы показываем, как из реализации R может быть восстановлена полная модель M исходной формулы F в теории BPA.

Дадим формальное определение теории BPA. Пусть $\mathbb{Z}$ – множество целых чисел, $\mathbb{P}$ — множество указателей, $\sum = \{+_p, \times_p, \leq_p, \boldsymbol{block}, \boldsymbol{base}, \boldsymbol{offset}, \boldsymbol{align}, (\cdot,\cdot)_p\}$ — сигнатура теории BPA. Аксиомы теории BPA могут быть записаны в логике QF_UFLIA. Они представлены на рис. 1.

4. Процедура инстанцирования

Зададим процедуру трансляции формулы F в теории BPA в равновыполнимую формулу F^* в логике QF_UFLIA. Данная процедура использует только аксиомы теории BPA, представленные на рис. 1, и определяет для их инстанцирования триггеры следующим образом:

- (A_p) инстанцируется всеми указательными термами p в формуле F;
- $(A_{\%})$ инстанцируется всеми указательными термами p в формуле F;
- $(A_{\in})$ инстанцируется всеми указательными термами p в формуле F;
- (A_o) инстанцируется всеми указательными термами p в формуле F;
- $(A_{\leq})$ инстанцируется термами p и q для любого терма вида $p \leq_p q$ в формуле F;
- (A_{-}) инстанцируется термами p и q для любого терма вида $p -_p q$ в формуле F;
- (A_{Δ}) инстанцируется термами p и i для любого терма вида $p +_p i$ в формуле F;
- (A_{+}) инстанцируется термами p и i для любого терма вида $p +_p i$ в формуле F;
- (A_a) инстанцируется термами l и a для любого терма вида $(l,a)_p$ в формуле F;
- (A_b) инстанцируется термами l и a для любого терма вида $(l,a)_p$ в формуле F.

Приведенные правила называются триггерами соответствующих аксиом.

$$\Sigma = \{+_p, -_p, \leq_p, \boldsymbol{block}, \boldsymbol{base}, \boldsymbol{offset}, \boldsymbol{align}, (\cdot,\cdot)_p\},$$
$$p, q \in \mathbb{P};\ l, a, i, j \in \mathbb{Z}.$$
$$\boldsymbol{block}(p) \in \mathbb{Z}, \boldsymbol{base}(p) \in \mathbb{Z}, \boldsymbol{offset}(p) \in \mathbb{Z}, \boldsymbol{align}(p) \in \mathbb{Z}, (l,a)_p \in \mathbb{P}.$$

Введем обозначения:

$$\boldsymbol{address}(p) = \boldsymbol{base}(\boldsymbol{block}(p)) + \boldsymbol{offset}(p),$$
$$p \parallel q \equiv \boldsymbol{align}(p) = \boldsymbol{align}(q),$$
$$p \triangle q \equiv \boldsymbol{block}(p) = \boldsymbol{block}(q),$$
$$p \triangle_0 q \equiv p \triangle q \wedge \boldsymbol{address}(p) \neq 0 \wedge \boldsymbol{address}(q) \neq 0$$
$$\boldsymbol{range}(p, i) \equiv$$
$$0 < \boldsymbol{address}(p) + s \times i \wedge \boldsymbol{address}(p) + s \times i \leq A, \text{ где } s, A \in \mathbb{Z} -$$
константы.

Далее определим аксиомы:

$$\forall p, q \in \mathbb{P}.\ p \triangle_0 q \Longrightarrow (p \leq_p q) \leftrightarrow \boldsymbol{offset}(p) \leq \boldsymbol{offset}(q), \quad (A_\leq)$$
$$\forall p \in \mathbb{P}.\ (\boldsymbol{block}(p), \boldsymbol{address}(p))_p = p, \quad (A_p)$$
$$\forall p \in \mathbb{P}.\ 0 \leq \boldsymbol{align}(p) \wedge \boldsymbol{align}(p) \leq s - 1, \quad (A_\%)$$
$$\forall p \in \mathbb{P}.\ 0 \leq \boldsymbol{address}(p) \wedge \boldsymbol{address}(\boldsymbol{p}) \leq A, \quad (A_\in)$$
$$\forall p, q \in \mathbb{P}.\ p \triangle_0 q \wedge p \parallel q \Longrightarrow$$
$$s \times (p -_p q) = \boldsymbol{offset}(p) - \boldsymbol{offset}(q), \quad (A_-)$$
$$\forall p \in \mathbb{P}.\ \boldsymbol{offset}(p) = s \times \boldsymbol{quot}(p) + \boldsymbol{align}(p), \quad (A_o)$$
$$\forall i \in \mathbb{Z}. \forall p \in \mathbb{P}.\ \boldsymbol{address}(p) \neq 0 \wedge \boldsymbol{range}(p, i) \Longrightarrow p +_p i \triangle p \quad (A_\Delta)$$
$$\forall i \in \mathbb{Z}. \forall p \in \mathbb{P}.\ \boldsymbol{address}(p) \neq 0 \wedge \boldsymbol{range}(p, i) \Longrightarrow$$
$$\boldsymbol{offset}(p +_p i) = \boldsymbol{offset}(p) + s \times i \quad (A_+)$$
$$\forall l, a \in \mathbb{Z}.\ 0 \leq a \wedge a \leq A \Longrightarrow \boldsymbol{address}((l,a)_p) = a \quad (A_a)$$
$$\forall l, a \in \mathbb{Z}.\ 0 \leq a \wedge a \leq A \Longrightarrow \boldsymbol{block}((l,a)_p) = l \quad (A_b)$$

Рис. 1. Аксиоматика теории ограниченной адресной арифметики
Fig. 1. Axioms defining the theory of bounded pointer arithmetic

Обозначим за F^* формулу F после выполнения алгоритма инстанцирования аксиом. Обозначим за $F^{\leq}$ множество экземпляров аксиом, полученных подстановкой термов вместо подкванторных переменных p и q в аксиоме $(A_\leq)$ для любого терма вида $p \leq_p q$ в формуле F. Аналогично определим множества F^p для (A_p), $F^\%$ для $(A_\%)$, $F^\in$ для $(A_\in)$, F^o для (A_o), F^Δ для (A_Δ), F^+ для (A_+), F^- для (A_-) и, наконец, F^a для (A_a) и F^b для (A_b). Полученная после инстанцирования формула F^* тогда может быть записана следующим образом:

$$F^* = F \wedge \bigwedge F^p \wedge \bigwedge F^\% \wedge \bigwedge F^\in \wedge \bigwedge F^o \wedge \bigwedge F^\Delta \wedge \bigwedge F^+ \wedge \bigwedge F^- \wedge \bigwedge F^\leq \wedge \bigwedge F^a \wedge \bigwedge F^b.$$

Формула F^* интерпретируется в логике QF_UFLIA, так что все символы теории BPA в ней не интерпретируются. Так как формула F^* получается из F только с помощью инстанцирования аксиом теории BPA, из выполнимости формулы F в BPA следует выполнимость F^* в логике QF_UFLIA, что соответствует корректности представленной процедуры трансляции.

5. Доказательство полноты

Рассмотрим произвольную формулу F в теории BPA, её трансляцию F^*, полученную через процедуру инстанцирования аксиом, и модель R, полученную для трансляции F^* в логике QF_UFLIA, которую мы называем реализацией. Указательные термы далее будем обозначать

буквами p и q, а целочисленные термы — буквами l, a, i и j. Доказательство полноты осуществляется с помощью восстановления модели M исходной формулы F в теории BPA из реализации R.

Для начала приведем несколько вспомогательных лемм.

Лемма 1. *Терм вида* $\boldsymbol{block}(p)$ *принадлежит формуле* F^* *тогда и только тогда, когда соответствующий указательный терм* p *принадлежит* F.

Лемма 2. *Терм вида* $\boldsymbol{address}(p)$ *принадлежит формуле* F^* *тогда и только тогда, когда соответствующий адресный терм* p *принадлежит* F.

Лемма 3. *Терм вида* $(l, a)_p$ *принадлежит формуле* F^* *тогда и только тогда, когда либо он исходно принадлежит* F*, либо подтерм* l *имеет вид* $\boldsymbol{block}(p)$*, а подтерм* a *— вид* $\boldsymbol{address}(p)$*, где соответствующий указательный терм* p *принадлежит* F.

Доказательство этих лемм осуществляется непосредственным перебором приведенных правил инстанцирования.

Далее дадим несколько дополнительных обозначений, которые мы будем использовать в доказательствах. Определим образ подмножества X множества A при отображении с помощью функции $f: A \to B$ как $f[X]$. Введем следующее обозначение:

$\{f(x) \mid P(x)\} \equiv f[\{x \mid P(x)\}]$, где $\{x \mid P(x)\}$ — множество термов x, удовлетворяющих предикату $P(x)$.

Далее определим следующие множества:

$P^R \equiv \{p^R \mid \boldsymbol{address}(t) \in F^*\}$,

$L^R \equiv \{(l^R,\ a^R) \mid (b, a)_p \in F^*\}$,

$L^A \equiv (\mathbb{Z} \times [0, A]) \backslash L^R$, где, p^R, l^R и a^R обозначают означивания соответствующих термов p, l и a в реализации R. Также введем функцию $(\cdot)_\perp$ преобразования указателя в пару, которая является следующим сокращением: $p_\perp \equiv (\boldsymbol{block}(p), \boldsymbol{address}(p))$.

Реализация R уже содержит частичные модели функций $(\cdot,\cdot)_p$ и $\cdot_\perp$ на соответствующих множествах L^R и P^R. Рассмотрим функции $(\cdot,\cdot)_p^R$ и $\cdot_\perp^R$, которые будем считать реализациями соответствующих функций $(\cdot,\cdot)_p$ и $\cdot_\perp$ на множествах L^R и P^R соответственно. Докажем для них следующие свойства:

Лемма 4. $[P^R]_\perp^R \subseteq L^R \cap (\mathbb{Z} \times [0, A])$. Показывается с помощью лемм 1–3.

Лемма 5. $[L^R]_p^R \subseteq P^R$. Показывается с помощью лемм 1 и 2.

Лемма 6. $\cdot_\perp^R$ *инъективна на множестве* P^R. Следует из лемм 1–5 и правила инстанцирования аксиомы (A_p).

Лемма 7. $\cdot_\perp^R$ *сюръективно отображает* P^R *на множество значений* $L^R \cap (\mathbb{Z} \times [0, A])$. Следует из лемм 1–5 и правила инстанцирования аксиомы $(A_\in)$.

Лемма 8. $\cdot_\perp^R$ *является биекцией между множествами* P^R *и* $L^R \cap (\mathbb{Z} \times [0, A])$*, а функция* $(\cdot,\cdot)_p^R$ *на множестве* $L^R \cap (\mathbb{Z} \times [0, A])$ *является обратной к ней.*

Первая часть утверждения о том, что $\cdot_\perp^R$ является биекцией непосредственно следует из лемм 6 и 7, а обратная функция может быть однозначна охарактеризована уравнением $(\cdot_\perp^R)^{-1}(p')_\perp^R = p'$ для любого $p' \in P^R$. Но для каждого такого p' верно, что $p' = p^R$, где $\boldsymbol{address}(p) \in F^*$. Из леммы 2 следует, что $p \in F$. Далее, согласно правилу инстанцирования аксиомы A_p верно, что $\left(\boldsymbol{block}(p), \boldsymbol{address}(p)\right)_p = p$. Таким образом, обратная к биективной $\cdot_\perp^R$ функция $(\cdot_\perp^R)^{-1}$ совпадает с $(\cdot,\cdot)_p^R$.

Получим ситуацию, изображенную на рис. 2, где $\cdot_\perp^R$ является биекцией между P^R и $L^R \cap (\mathbb{Z} \times [0, A])$ (на рис. 2 это множество обозначено двойной штриховкой), а $(\cdot,\cdot)^R$ определена на множестве L^R и является обратной к $\cdot_\perp^R$ на P^R. Рассмотрим теперь множество $L^A \equiv (\mathbb{Z} \times [0, A]) \backslash L^R$. Оно имеет не более, чем счетную мощность. Выберем произвольно

соответствующее (конечное, либо счетное) число различных элементов из какого-либо счетного домена и обозначим полученное таким образом множество элементов через $\mathbb{Z}'_p$. Теперь в качестве домена указателей в восстановленной модели M возьмем множество $\mathrm{P}^R \cup \mathbb{Z}'_p$. Обозначим его $\mathbb{P}^M$. Так как множества L^A и $\mathbb{Z}'_p$ по построению имеют одинаковую мощность, между ними существует биекция. Произвольно выберем такую биекцию, определенную на множестве $\mathbb{Z}'_p$ и назовем ее $\cdot'_\perp$, а обратную к ней функцию, определенную на множестве L^A — $(\cdot,\cdot)'_p$. Введем сокращения $p'_\perp = (\boldsymbol{block}'(p), \boldsymbol{address}'(p))$ и определим

$$\boldsymbol{offset}'(p) = \begin{cases} \boldsymbol{address}'(p) - \boldsymbol{base}^R(\boldsymbol{block}'(p)), & \boldsymbol{block}'(p) \in \{l^R \mid \boldsymbol{base}(l) \in F^*\} \\ \boldsymbol{address}'(p), & \text{иначе} \end{cases}$$

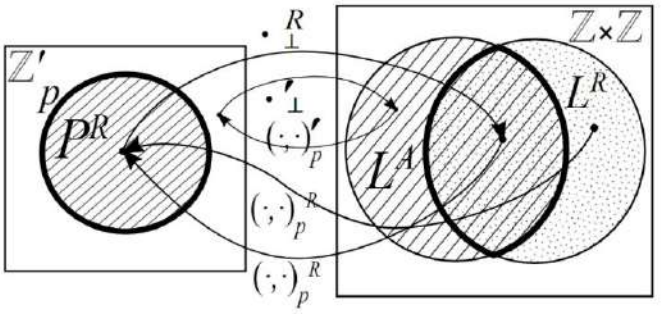


Рис. 2. Расширение биекции между множествами P^R и $L^R \cap (\mathbb{Z} \times [0, A])$ на множества $\mathbb{P}^M = P^R \cup \mathbb{Z}'_p$ и $\mathbb{Z} \times [0, A]$

Fig. 2. Extending the bijection between P^R and $L^R \cap (\mathbb{Z} \times [0, A])$ to the whole sets $\mathbb{P}^M = P^R \cup \mathbb{Z}'_p$ and $\mathbb{Z} \times [0, A]$

Теперь мы готовы ввести определение восстановленной из реализации R полной модели M исходной формулы F в теории BPA, которое представлено на рис. 3. Далее докажем необходимые свойства этого определения.

Лемма 11. *Восстановленная модель M на рисунке 3 однозначно определена.*

Восстановленная модель M, показанная на рис. 3, включает в себя определение домена $\mathbb{P}^M$ всех указателей, а также определения моделей для всех функций из сигнатуры теории BPA. Эти функции могут возвращать элементы двух сортов — целые числа и указатели.

Таким образом, мы должны показать, что функции, возвращающие указатели, определенные на рисунке, действительно отображают любую комбинацию своих аргументов, взятых из соответствующих доменов, в элементы домена указателей $\mathbb{P}^M$.

Рассмотрим случаи для функции $(\cdot,\cdot)^M_p$. В первом случае $(l, a)^R_p \in P^R$ для любой пары $(l, a) \in L^R$ согласно лемме 7. Во втором случае $(l, a)'_p \in \mathbb{Z}_p{}' \subseteq \mathbb{P}^M$ по построению домена $\mathbb{P}^M$. В третьем случае $\epsilon\mathbb{P}^M \subseteq \mathbb{P}^M$ по определению эпсилон-оператора Гильберта ϵ (выбор произвольного элемента из непустого множества), так как в общем случае либо $\mathbb{Z}_p{}'$, либо P^R непусты. Оставшаяся указательная функция $+^M_p$ в модели M однозначно определена с использованием функции $(\cdot,\cdot)^M_p$.

Лемма 12. *Аксиомы (A_p), ($A_\in$), ($A_\%$) и (A_o) теории BPA выполнены в модели M.*

Лемма 12. *Аксиомы ($A_\leq$) и (A_-) теории BPA сохраняются в модели M.*

Лемма 14. *Аксиомы (A_Δ) и (A_+) теории BPA сохраняются в модели M.*

Лемма 15. *Аксиомы (A_b) и (A_a) теории BPA сохраняются в модели M.*

Лемма 16. *Модель M может быть расширена неинтерпретируемыми константами, которые содержатся в F так, что для любого подтерма $t \in F$ его интерпретации в модели M и в реализации R совпадают, то есть $t^M = t^R$.*

$$\mathbb{P}^M = \mathrm{P}^R \cup \mathbb{Z}'_p,$$

$$(l,a)^M_p = \begin{cases} (l,a)^R_p, & (l,a) \in L^R \\ (l,a)'_p, & (l,a) \notin L^R, a \in [0,A] \\ \epsilon\mathbb{P}^M, & (l,a) \notin L^R, a \notin [0,A] \end{cases}$$

$$\boldsymbol{block}^M(p) = \begin{cases} \boldsymbol{block}^R(p), & p \in P^R \\ \boldsymbol{block}'(p), & p \notin P^R \end{cases}$$

$$\boldsymbol{offset}^M(p) = \begin{cases} \boldsymbol{offset}^R(p), & p \in P^R \\ \boldsymbol{offset}'(p), & p \notin P^R \end{cases}$$

$$\boldsymbol{base}^M(l) = \begin{cases} \boldsymbol{base}^R(l), & l \in \{l^R | base(l) \in F^*\} \\ 0, & l \notin \{l^R | base(l) \in F^*\} \end{cases}$$

$$\boldsymbol{align}^R(p) = \begin{cases} \boldsymbol{align}^R(p), & p \in P^R \\ \boldsymbol{offset}'(p)\ \%\ s, & p \notin P^R \end{cases}$$

$$\boldsymbol{quot}^M(p) = \begin{cases} \boldsymbol{quot}^R(p), & p \in P^R \\ \boldsymbol{offset}'(p) \div s, & p \notin P^R \end{cases}$$

$$p +^M_p i = \begin{cases} p +^R_p i, & (p,i) \in \{(p^R, i^R) | p +_p i \in F^*\} \\ (\boldsymbol{block}^M(p), (\boldsymbol{address}^M(p) + s \times i)\ \%\ A + 1)^M_p, & (p,i) \notin \{(p^R, i^R) | p +_p i \in F^*\} \end{cases}$$

$$p -^M_p q = \begin{cases} p -^R_p q, & (p,q) \in \{(p^R, q^R) | p -_p q \in F^*\} \\ \left(\boldsymbol{offset}^M(p) - \boldsymbol{offset}^M(q)\right) \div s, & (p,q) \notin \{(p^R, i^R) | p -_p i \in F^*\ \} \end{cases}$$

$$p \leq^M_p q = \begin{cases} p \leq^R_p q, & (p,q) \in \{(p^R, q^R) | p \leq_p q \in F^*\} \\ \boldsymbol{offset}^M(p) \leq \boldsymbol{offset}^M(q), & (p,q) \notin \{(p^R, q^R) | p \leq_p q \in F^*\} \end{cases}$$

Рис. 3. Определение восстановленной модели M формулы F в теории BPA
Fig. 3. Definition of the reconstructed model M for the formula F in BPA

Полнота процедуры инстанцирования напрямую следует из перечисленных лемм:

Теорема 1. *Каждая бесквантворная формула F выполнима в теории ИРА тогда и только тогда, когда её трансляция F^* выполнима в логике QF_UFLIA.*

6. Формализация

Доказательство полноты, описанное в предыдущем разделе, и было формализовано в системе Isabelle/HOL. С целью обобщения формализации на случай произвольных расширений логики QF_UFLIA различными аксиоматически заданными теориями для формального представления формул мы использовали нетипизированное глубокое погружение всего с двумя конструкторами — применение функции к аргументам (`App`) и подкванторная переменная (`Var`). Свободные переменные, или иначе неинтерпретируемые константы, представляются в этом случае как применения соответствующей неинтерпретируемой

функции к пустому списку аргументов. Кванторы явным образом не включались в представление, вместо этого для каждой формулы из множества аксиом теории неявно использовалась схематическая квантификация по всем подкванторным переменным, которые встречаются в теле аксиомы. Также для определения семантики этих кванторов было определено понятие означивания незамкнутого выражения, в котором вместо оценки свободных (подкванторных, но не связанных) переменных используется подстановка. Более подробно этот прием описан в нашей статье [5], которая использует аналогичный подход к интерпретации формул. Таким образом, в ходе данной работе мы подтвердили, что ранее сделанную формализацию из статьи [5] можно успешно использовать как основу в ходе доказательства полноты новой аксиоматической теории.

Также стоит заметить, что в первоначально сформулированной нами версии триггеров для аксиом $(A_{\%})$ и (A_o) использовалось инстанцирование не всеми указательными термами исходной формулы, а только термами p для подтермов вида $\boldsymbol{align}(p)$. Однако в ходе верификации выяснилось, что такая процедура трансляции не является полной, в частности, тавтология $\boldsymbol{address}(p) \neq 0 \Rightarrow p -_p (p +_p 0) = 0$ теории BPA не выводима с помощью такой процедуры. Такая ошибка могла бы в принципе найтись и при тестировании процедуры трансляции, но подбор достаточно полного набора формул для проверки процедуры трансляции с помощью тестов также достаточно сложен на практике. Это отчасти оправдывает применение с целью проверки корректности решающих процедур систем автоматизированного доказательства теорем, таких как Isabelle/HOL.

Соответствующая формализации теория Isabelle называется `TSMT_Pointers_Complete` [17].

7. Дальнейшие исследования

Основными направлениями будущей работы включают разработку некоторых предсказуемых (хотя и неполных) подходов к обработке кванторов, встречающиеся в цели без соответствующих триггеров (стратегии для обработки кванторов внутри существующих решателей очень эффективны, но редко предсказуемы). Еще одно важное направление — формализация и оценка решения, основанного на реализации процедуры для других разрешимых теорий, таких как эффективно разрешимый фрагмент теории бит-векторов — монотонные формулы над равенствами между выражениями с побитовыми операциями.

Аксиомы, формализующие семантику побитовых операций могут иметь общий вид $\forall b_1 \dots b_n\ l_1 \dots l_m\ i.\ P(i, f_1(\overline{b_{1..n}}, \overline{l_{1..m}})!!\, g_1(\overline{l_{1..m}}, i), \dots, f_k(\overline{b_{1..n}}, \overline{l_{1..m}})!!\, g_k(\overline{l_{1..m}}, i))$, где $!!$ — это взятие значения бита (как предикат), а $f_j(\overline{b_{1..n}}, \overline{l_{1..m}})$ и $g_j(\overline{l_{1..m}}, i)$ одинаковые (общие) для всех P, то есть в разных аксиомах одной f_j не может соответствовать разный g_j. То есть каждая функция (операция) от векторов может входить в аксиомы только с одним соответствующим индексом, зависящим от подкванторной переменной. Если целевая формула в теории битовых векторов является монотонным предикатом от равенств между побитовыми выражениями, то проверку существования ее модели можно свести к проверке существования некоторого набора индексов битов, в которых нарушены некоторые из равенств. Тогда все аксиомы можно инстанцировать соответствующими сколемовскими индексами, при этом для оставшихся индексов существование модели достаточно будет проверить один раз в общем случае, так как разные биты одной операции не зависят друг от друга в силу ограничения на однократное использование индексов. Побитовыми операциями являются, например, операции из языка программирования Си — `&`, `|`, `^`, `~`, `<<`, `>>` и целочисленные приведения типов. В саму теорию входят эти операции и еще операция взятия бита `!!`. Например, для операции `<<` нужны две аксиомы: $\forall a\ k.\ \forall i > 0.\ i < n - k \longrightarrow (a \ll k!!\, i \leftrightarrow a!!\, i + k)$ и $\forall a\ k.\ \forall i \geq n - k.\ \neg(a \ll k!!\, i)$.

Аналогичным образом можно сформулировать некоторые другие актуальные теории и их фрагменты, например, фрагмент теории операций со списками.

8. Заключение

В работе представлена теория ограниченной адресной арифметики (с Си-подобным разделением на непересекающиеся блоки памяти), позволяющая реализовать соответствующую решающую процедуру в Isabelle/HOL на основе SMT- решателей. Эта решающая процедура позволяет упростить дедуктивную верификацию программ на языке Си. Разработанная решающая процедура основана на эффективной с точки зрения размера термов трансляции формул теории адресной арифметики в существующую и широко поддерживаемую логику QF_UFLIA.

Список литературы / References

[1]. T. Nipkow, M. Wenzel, and L. C. Paulson. Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Springer-Verlag, 2002. 240 p.

[2]. Vegard Nossum. [PATCH] firmware: declare __{start,end}_builtin_fw as pointers. Available at: https://www.mail-archive.com/linux-kernel@vger.kernel.org/msg1176758.html.

[3]. Xi Wang, Haogang Chen et al. Undefined behavior: What happened to my code? In Proc. of the Asia-Pacific Workshop on Systems, 2012, Article No.: 9.

[4]. Chad R. Dougherty and Robert C. Seacord. C compilers may silently discard some wraparound checks. Available at: https://www.kb.cert.org/vuls/id/162289.

[5]. Р.Ф. Садыков, М.У. Мандрыкин. Верифицированная тактика Isabelle/HOL для теории ограниченных целых на основе инстанцирования и SMT. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 107-124 / R. Sadykov, M. Mandrykin. Verified Isabelle/HOL tactic for the theory of bounded integers based on quantifier instantiation and SMT. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 107-124 (in Russian). DOI: 10.15514/ISPRAS–2020–32(2)–9.

[6]. H. Tuch. Formal memory models for verifying C systems code. PhD Thesis. School of Computer Science and Engineering, The University of New South Wales, Australia, 2008, 249 p.

[7]. Sandrine Blazy and Xavier Leroy. Formal Verification of a Memory Model for C-Like Imperative Languages. Lecture Notes in Computer Science, vol. 3785, 2005, pp. 280-299.

[8]. Y. Moy. Automatic Modular Static Safety Checking for C Programs. PhD Thesis, Université de Paris-Sud 11, Paris, France, 2009, 249 p.

[9]. Jeehoon Kang, Chung-Kil Hur et al. 2015. A formal C memory model supporting integer-pointer casts. ACM SIGPLAN Notices, vol. 50, issue 6, 2015, pp. 326–335.

[10]. K. Memarian and P. Sewell. Clarifying the C memory object model. Available at: http://www.open-std.org/JTC1/SC22/WG14/www/docs/n2012.htm.

[11]. G. Klein, K. Elphinstone et al. seL4: Formal verification of an OS kernel. In Proc. of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, 2009, pp. 207-220.

[12]. N. Bjørner, A.Blass et al. Modular difference logic is hard. Tech. Rep. MSR-TR-2008-140, Microsoft, 2008. Available at: https://www.microsoft.com/en-us/research/publication/modular-difference-logic-is-hard/.

[13]. C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.6. Available at: https://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2017-07-18.pdf.

[14]. S. Böhme. Proof reconstruction for Z3 in Isabelle/HOL. In Proc. of the 7th International Workshop on Satisfiability Modulo Theories (SMT ’09), 2009, pp. 1-11.

[15]. S. Böhme and T. Weber. Fast LCF-style proof reconstruction for Z3. Lecture Notes in Computer Science, vol. 6172, 2010, pp. 179–194.

[16]. J. Dawson. Isabelle theories for machine words. Electronic Notes in Theoretical Computer Science, vol. 250, no. 1, pp. 55-70, 2009.

[17]. R. Sadykov and M. Mandrykin. Complete decision procedure for the bounded theory of pointer arithmetic based on quantifier instantiation and SMT. Available at: https://forge.ispras.ru/projects/tsmt/repository/, 2021.

Информация об авторах / Information about authors

Рафаэль Фаритович САДЫКОВ – стажер-исследователь в ИСП РАН, аспирант кафедры МаТИС механико-математического факультета МГУ им. М. В. Ломоносова. Сфера научных интересов: верификация программ, теория распределенных вычислений, теория графов, теория автоматов, математическая логика.

Rafael Faritovich SADYKOV – intern researcher of ISP RAS, PhD student of Faculty of Mechanics and Mathematics, Moscow State University. Research interests: program verification, distributed computing theory, graph theory, automata theory, mathematical logic.

Михаил Усамович МАНДРЫКИН – младший научный сотрудник ИСП РАН, кандидат физико-математических наук по специальности «математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей». Сфера научных интересов: формальные методы верификации программ, математическая логика, функциональное программирование, системы типов в языках программирования.

Mikhail Usamovich MANDRYKIN – researcher at ISP RAS, PhD. Research interests: formal methods, program verification, mathematical logic, functional programming, type systems.

DOI: 10.15514/ISPRAS-2021-33(4)-14

Предотвращение уязвимостей, возникающих в результате оптимизации кода с неопределенным поведением

Р.В. Баев, ORCID: 0000-0002-7999-7952 <baev@ispras.ru>
Л.В. Скворцов, ORCID: 0000-0002-1580-1244 <lvs@ispras.ru>
Е.А. Кудряшов, ORCID: 0000-0002-2361-7172 <kudryashov@ispras.ru>
Р.А. Бучацкий, ORCID: 0000-0001-8522-1811 <ruben@ispras.ru>
Р.А. Жуйков, ORCID: 0000-0002-0906-8146 <zhroma@ispras.ru>

Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. С развитием оптимизирующих компиляторов стали возникать случаи появления уязвимостей в программах во время оптимизации. Это связано с тем, что зачастую программисты используют конструкции с неопределенным поведением, опираясь на свое представление о том, в какой код такие конструкции транслировались знакомым им компилятором для определенной архитектуры. В то же время компилятор, руководствующийся стандартом языка, вправе проводить оптимизации так, как будто бы таких конструкций в коде не может существовать. В этой статье описываются подходы к обнаружению и устранению уязвимостей в программах, рассматривается применимость этих подходов для случая уязвимостей, появляющихся вследствие оптимизации, в условиях, когда возможность изменения исходного кода ограничена или отсутствует. В статье предлагается концепция безопасного компилятора, т.е. компилятора, обеспечивающего отсутствие внесения уязвимостей в программу во время оптимизации, и описывается реализация такого компилятора на базе компилятора GCC. Для безопасного компилятора приводится разделение реализованного функционала на три уровня защиты и описание применимости этих уровней, показывается применимость безопасного компилятора на практике, а также оценивается изменение производительности получаемой программы.

Ключевые слова: компилятор; уязвимость; неопределенное поведение

Для цитирования: Баев Р.В., Скворцов Л.В., Кудряшов Е.А., Бучацкий Р.А., Жуйков Р.А. Предотвращение уязвимостей, возникающих в результате оптимизации кода с неопределенным поведением. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 195-210. DOI: 10.15514/ISPRAS–2021–33(4)–14

Prevention of vulnerabilities arising from optimization of code with Undefined Behavior

R.V. Baev, ORCID: 0000-0002-7999-7952 <baev@ispras.ru>
L.V. Skvortsov, ORCID: 0000-0002-1580-1244 <lvs@ispras.ru>
E.A. Kudryashov, ORCID: 0000-0002-2361-7172 <kudryashov@ispras.ru>
R.A. Buchatskiy, ORCID: 0000-0001-8522-1811 <ruben@ispras.ru>
R.A. Zhuykov, ORCID: 0000-0002-0906-8146 <zhroma@ispras.ru>

Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Aggressive optimization in modern compilers may uncover vulnerabilities in program code that did not lead to bugs prior to optimization. The source of these vulnerabilities is in code with undefined behavior. Programmers use such constructs relying on some particular behavior these constructs showed before in their experience, but the compiler is not obliged to stick to that behavior and may change the behavior if it's needed for optimization since the behavior is undefined by language standard. This article describes approaches to detection and elimination of vulnerabilities arising from optimization in the case when source code is available but its modification is undesirable or impossible. Concept of a safe compiler (i.e. compiler that ensures no vulnerability is added to the program during optimization) is presented and implementation of such a compiler on top of GCC compiler is described. Implementation of safe compiler's functionality is divided into three security levels whose applicability is discussed in the article. Feasibility of using the safe compiler on real-world codebases is demonstrated and possible performance losses are estimated.

Keywords: compiler; vulnerability; undefined behavior

For citation: Baev R.V., Skvortsov L.V., Kudryashov E.A., Buchatskiy R.A., Zhuykov R.A. Prevention of vulnerabilities arising from optimization of code with Undefined Behavior. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021. pp. 195-210 (in Russian). DOI: 10.15514/ISPRAS–2021–33(4)–14

1. Введение

К появлению уязвимостей в программах могут приводить не только ошибки, допущенные программистом, или злой умысел, но и агрессивные оптимизации кода, выполняемые компилятором, как это показано в работе [1]. В таких случаях одинаковый исходный код может быть скомпилирован в корректно работающий исполняемый код, если компилятор по какой-то причине не использует некую конкретную оптимизацию (например, все оптимизации отключены, или в выбранной версии компилятора такая оптимизация отсутствует, или оптимизация не применяется для выбранной аппаратной платформы), и в исполняемый код, содержащий уязвимость, если эта оптимизация используется. При этом невозможно точно предсказать, сохранится ли ожидаемая программистом семантика программы при смене флагов компиляции, версии компилятора, самого компилятора (так, например, некоторые подобные ошибки проявлялись при замене GCC на Clang) или целевой архитектуры.

В большей части таких случаев речь идет о компиляции кода, содержащего конструкции с неопределенным поведением (undefined behavior). Неопределенным поведением называется результат выполнения конструкций языка, не регламентированный стандартом языка. В случае языка С примерами таких конструкций являются разыменование нулевого указателя, переполнение знаковых целых чисел и др. Компилятор действует, исходя из предположения, что компилируемая программа строго соответствует стандарту языка и, таким образом, не вызывает неопределенного поведения. Это позволяет ему более эффективно оптимизировать код в ряде случаев, однако может привести к появлению уязвимости в случае, если в компилируемой программе присутствовало неопределенное поведение.

Уязвимости, возникающие вследствие оптимизации кода компилятором, не ограничиваются случаями неопределенного поведения: есть пример корректной с точки зрения стандарта

языка оптимизации, вносящей уязвимость в код без неопределенного поведения. Перезапись буфера памяти без его последующего использования может быть удалена компилятором как не влияющая на результат выполнения программы, что приведет к тому, что чувствительные данные могут остаться в памяти, откуда их сможет получить злоумышленник.

Код, который может быть удален компилятором вследствие того, что компилятор полагается на отсутствие в программе неопределенного поведения, называется нестабильным кодом [2]. В рамках этой работы так же будет называться код, в котором удалена перезапись буфера памяти.

В разд. 2 демонстрируются примеры уязвимостей, возникающих в результате агрессивных оптимизаций кода компилятором. В разд. 3 обсуждаются подходы, применяемые на практике для обнаружения и предотвращения уязвимостей. В разд. 4 описывается безопасный компилятор и его область применимости. В разд. 5 приводятся некоторые подробности реализации безопасного компилятора и уровней защиты в нем. Разд. 6 посвящен результатам тестирования безопасного компилятора на реальных приложениях.

2. Примеры уязвимостей

В этом разделе приведены примеры некоторых типов уязвимостей, возникающих в результате агрессивных оптимизаций кода компилятором. В работе [1] приведено еще 4 типа уязвимостей, основанных на оптимизации неопределенного поведения. В рамках данного исследования было найдено еще несколько типов. Подробнее о поиске таких уязвимостей рассказывается в подразд. 6.1.

2.1 Неопределенное поведение

2.1.1 Слишком большой аргумент операции сдвига

На листинге 1 продемонстрирован фрагмент кода файловой системы ext4. Изначально этот фрагмент содержал уязвимость: в некоторых случаях значение переменной `groups_per_flex` становилось нулем, что приводило выполнению деления на ноль и, соответственно, исключению. Такая ситуация могла случиться, например, на 32-битной архитектуре PowerPC, если в `sbi->s_log_groups_per_flex` было значение 32 (на PowerPC для хранения аргумента сдвига отведено 6 бит, на x86 — только 5, поэтому выполнить сдвиг на 32 бита не получится). Исправление, добавляющее проверку `groups_per_flex` на равенство нулю, должно было устранить эту уязвимость, однако на самом деле этого не произошло. Причина в том, что с точки зрения стандарта языка C сдвиг знакового целого числа на число бит, большее или равное ширине его типа, является неопределенным поведением (п. 6.5.7. стандарта C11 [3]).

```
groups_per_flex = 1 << sbi->s_log_groups_per_flex;
/* There are some situations, after shift the
    value of 'groups_per_flex' can become zero
    and division with 0 will result in fixpoint
    divide exception */
if (groups_per_flex == 0)
    return 1;
flex_group_count = ... / groups_per_flex;
```

Листинг 1. Код из fs/ext4/super.c в ядре Linux 2.6.31
Listing 1. Code from fs/ext4/super.c in Linux kernel 2.6.31

Компилятор может доказать, что в отсутствие неопределенного поведения значение `groups_per_flex` строго положительно, поэтому выражение `groups_per_flex == 0` тождественно ложно, и проверку можно удалить. Именно так поступает компилятор Clang. Компиляторы GCC и Clang не предоставляют опции отключения подобной оптимизации.

2.1.2 Переполнение знакового целого

На листинге 2 приведен фрагмент реализации системного вызова `vfs_fallocate` в текущей (5.14) версии ядра Linux. Значения переменных offset и len приходят из пользовательского пространства, поэтому требуют проверки. Сначала проверяется, что значение `offset` неотрицательно, а значение len положительно. Кроме того, программист, писавший этот код, понимал (о чем свидетельствует комментарий), что при сложении может произойти целочисленное переполнение, что приведет к тому, что сумма `offset + len` будет меньше значения `inode->i_sb_s_maxbytes`, и выполнение функции продолжится вместо возврата со значением -EFBIG. Для устранения этой уязвимости он добавил в условие проверку на переполнение при сложении `offset + len`. Как и в случае, рассмотренном в предыдущем подразделе, добавленная проверка не приводит к устранению уязвимости.

```
int vfs_fallocate(..., loff_t offset, loff_t len)
{
    struct inode *inode = ...;
    if (offset < 0 || len <= 0)
        return -EINVAL;
    /* Check for wrap through zero too */
    if ((offset + len > inode->i_sb->s_maxbytes)
        || (offset + len < 0))
        return -EFBIG;
    …
}
```

Листинг 2. Код из fs/open.c в ядре Linux 5.14
Listing 2. Code from fs/open.c in Linux kernel 5.14

Компилятор GCC способен доказать, что проверка на переполнение происходит только в случае неотрицательного значения `offset` и положительного значения `len`. Переполнение знакового целого числа является неопределенным поведением (п. 6.5. стандарта C11 [3]), поэтому компилятор вправе рассчитывать, что этого не произойдет. Поскольку, с точки зрения компилятора, в проверке написано тождественно ложное выражение (некое положительное значение меньше нуля), он вправе удалить это выражение (`expr || false` ⇔ `expr`). Таким образом, уязвимость остается на месте. Подобное недопонимание между компилятором и программистом происходит достаточно часто, поэтому и в GCC, и в Clang была добавлена опция `-fno-strict-overflow`, которая позволяет запретить компилятору полагаться на отсутствие целочисленного переполнения. Ядро Linux, начиная с версии 2.6.31, и другие крупные проекты в обязательном порядке используют эту опцию при сборке.

2.1.2 Чтение неинициализированной переменной

На листинге 3 приведен код из функции `srandomdev` в библиотеке libc в FreeBSD 8.0. В случае компиляции без оптимизаций наблюдаемым поведением в этом случае будет хранение в переменной `junk` некого “мусорного” значения со стека. Судя по комментарию и сообщению в соответствующем коммите, автор этого фрагмента рассчитывал именно на такое поведение программы для использования в качестве источника энтропии. Здесь имеется сразу две проблемы: во-первых, стандарт языка ничего не говорит о размещении переменных, это решение принимает компилятор (GCC с включенными оптимизациями в данном случае размещает переменную на регистре), а во-вторых, чтение неинициализированной переменной является неопределенным поведением, что позволяет компилятору, например, заменить все выражение в аргументе функции srandom константой (именно таким образом поступает Clang).

```
struct timeval tv;
unsigned long junk; /* XXX left uninitialized on purpose */

gettimeofday(&tv, NULL);
srandom((getpid() << 16) ^ tv.tv_sec ^ tv.tv_usec ^ junk);
```

Листинг 3. Код из lib/libc/stdlib/rand.c из FreeBSD 8.0
Listing 3. Code from lib/libc/stdlib/rand.c in FreeBSD 8.0

На момент написания статьи в Clang было принято изменение, добавляющее опцию `-ftrivial-auto-var-init`. Эта опция инициализирует автоматические переменные указанным значением, не оставляя, таким образом, в программе неинициализированных переменных. Схожее изменение было предложено в GCC в феврале 2021 года, однако на момент написания статьи оно еще не было принято.

2.2 Удаление записи в память

На листинге 4 приведен пример кода, который не содержит неопределенного поведения, но тем не менее, в результате работы оптимизирующего компилятора в получающемся коде может появиться уязвимость. Стандарт языка C не обязывает компилятор сохранять побочные эффекты, которые не влияют на результат вычислений. В приведенном фрагменте кода таким побочным эффектом будет запись в память в результате вызова функции `memset`.

```
void func(void) {
    char password[1000] = {0};
    get_password(password);
    use_password(password);
    memset(password, 0, 1000); // delete password from memory
}
```

Листинг 4. Код, стирающий чувствительные данные из памяти
Listing 4. Code to erase sensitive data from memory

Компилятор может рассудить, что раз эти данные никогда не будут прочитаны, то эффективнее их не записывать, и удалить вызов `memset`. В результате такого решения чувствительные данные останутся в памяти, и злоумышленник сможет их прочитать, воспользовавшись другой уязвимостью (в том числе, возможно, в другой программе). Ни GCC, ни Clang не предоставляют надежного способа избежать подобной уязвимости.

3. Подходы к обнаружению и предотвращению уязвимостей

3.1 Инструменты статической проверки кода

Одним из основных способов поиска уязвимостей в программе является статический анализ. Статический анализ работает с исходным кодом программы. Статический анализ является мощным инструментом поиска ошибок, однако не лишен недостатков.

Во-первых, некоторые типы ошибок проявляются только во время исполнения, статический анализатор, который будет пытаться отмечать все возможные места таких ошибок, будет бесполезен на практике, поскольку ложноположительных срабатываний будет на несколько порядков больше, чем истинных.

Во-вторых, в общем случае, статический анализатор не имеет информации о компиляторе, который будет использоваться, и о целевой архитектуре, которая необходима для поиска уязвимостей, связанных с нестабильным кодом. В качестве исключения можно привести в пример инструмент STACK [4], который нацелен на поиск кода, нестабильного с точки зрения Clang. Этот инструмент основывается на знании устройства оптимизаций в Clang и способен находить случаи, подобные описанным в 2.1, но со временем оптимизатор может

научиться использовать другие типы неопределенного поведения, о чем может быть неизвестно инструменту STACK.

В-третьих, общим местом для всех инструментов статического анализа является тот факт, что статические анализаторы только сигнализируют о возможной ошибке, но не исправляют и не предотвращают ошибки. Это может быть недостатком в случаях, когда модификация исходного кода по каким-то причинам невозможна или затруднительна.

3.2 Инструменты динамической проверки кода

К инструментам динамического анализа относятся такие инструменты как Undefined Behavior Sanitizer (UBSan) [5] и Address Sanitizer (ASan) [6], входящие в состав GCC и Clang, интерпретатор tis-interpreter [7]. Санитайзеры вставляют в код программы проверки, обнаруживающие ошибки (в том числе случаи неопределенного поведения). При срабатывании этих проверок программа аварийно завершается, но в некоторых случаях UBSan позволяет продолжить выполнение программы после ошибки (так, например, есть возможность продолжить выполнение программы после целочисленного переполнения, в этом случае поведение программы совпадает с поведением программы, скомпилированной с флагом -fwrapv). Стоит отметить, что инструментация, используемая UBSan, не позволяет компилятору удалить нестабильный код, причиной нестабильности которого является неопределенное поведение. Инструмент tis-interpreter является интерпретатором языка C, который во время выполнения конструкций программы проверяет, способны ли они вызывать неопределенное поведение.

Преимуществом средств динамического анализа является большая точность по сравнению со средствами статического анализа. Общими недостатками средств динамического анализа являются заметное замедление анализируемой программы (замедление в 2-3 раза считается нормальным при использовании UBSan [8]), а также зависимость качества анализа от входных данных (динамический анализ не найдет ошибок на невыполнявшихся путях). Также упомянутые средства динамического анализа неспособны устранить уязвимость, приведенную в 2.2.

В отличие от средств статического анализа, UBSan способен предотвратить уязвимости вследствие наличия нестабильного кода, однако для большого числа реальных программ замедление, вызываемое использованием UBSan является неприемлемым.

3.3 Тестирование

Для обнаружения уязвимостей в скомпилированной программе может использоваться тестирование, в том числе автоматизированное. Генерация тестовых наборов также может производиться автоматически с помощью инструментов таких как KLEE [9]. Важно, чтобы используемые инструменты корректно моделировали неопределенное поведение с учетом выбранной целевой архитектуры. Такое тестирование должно проводиться для той же целевой архитектуры и версии компилятора, которые будут выбраны для финальной сборки. Важно отметить, что, как и статический анализ, тестирование способно сигнализировать об ошибке, но не предотвратить появление уязвимости.

3.4 Изменение стратегий оптимизации в компиляторе

Поскольку проблема, обсуждаемая в данной работе, состоит в том, что нестабильный код удаляется компилятором в ходе оптимизаций, одним из возможных способов борьбы с этим является отключение оптимизаций. Для некоторых случаев нестабильного кода существуют флаги компилятора, отключающие соответствующие оптимизации или меняющие их поведение (к таким флагам относятся `-fno-strict-overflow`, `-fno-delete-null-pointer-checks` и другие), но для некоторых (как для случая, приведенного в 2.1.1) таких

флагов не предусмотрено. При этом полное отключение оптимизаций (уровень оптимизации `-O0`) с одной стороны является крайне нежелательным с точки зрения производительности, а с другой стороны не гарантирует отсутствия уязвимостей (так, например, GCC с уровнем оптимизации `-O0` способен удалять такие проверки как `x + 100 < x`). К преимуществам этого подхода относится отсутствие необходимости модификации исходного кода.

3.5 Использование безопасных функций

Некоторые уязвимости могут быть предотвращены с помощью использования безопасных аналогов уязвимых функций. В языке C, начиная с версии C11, представлен набор функций (описаны в приложении K к стандарту языка C [3]) с суффиксом `_s`. В отличие от соответствующих функций без суффикса `_s`, они должны проверять ограничения во время выполнения (так, например, функция `memcpy_s` должна проверять, что указатели в ее аргументах не равны NULL, значения размеров не превосходят максимального значения, буферы не пересекаются и т.д.). Применительно к проблеме нестабильного кода интересна функция `memset_s`: ее спецификация требует обязательного сохранения записи в память (в отличие от спецификации функции `memset`). Это отличие позволяет избежать уязвимостей подобных описанной в 2.2.

Функции из приложения K к стандарту языка C не являются обязательными для реализации в библиотеке языка C для соответствия стандарту, многие реализации библиотеки (glibc и musl в их числе) не включают эти функции и не собираются их включать в обозримом будущем.

Многие реализации библиотеки языка C включают свои реализации функции со схожим функционалом (`explicit_bzero` в glibc, `SecureZeroMemory` в Windows), однако они не являются стандартными и не гарантируется, что компилятор никогда не сможет оптимизировать их аналогично `memset`.

Использование данного подхода требует модификации исходного кода, причем заметно более нетривиальной [10], чем задумывалось создателями функций из приложения K.

3.6 Проверка семантики

В статье [11] описывается "негативная" семантика языка C. Под "негативной" понимается семантика, описывающая неопределенное поведение. На основе негативной семантики реализован инструмент kcc, основывающийся на фреймворке K. kcc способен находить большое число случаев неопределенного поведения. kcc является академическим проектом и не является эффективным с точки зрения применимости, в первую очередь из-за ограниченной поддержки стандартных библиотек. Коммерческим продолжением kcc является RV-Match [12]. RV-Match поддерживает стандарт C11. В отличие от kcc, RV-Match не требует, чтобы всех используемых в проекте библиотек была определена семантика. По задумке авторов, kcc (и, соответственно, RV-Match) должно быть возможно использовать как прямую замену gcc без изменения системы сборки. Известно, что для некоторых программ, например, gzip, такая замена действительно возможна. Эффективность RV-Match показана в работе [13]. В этой же работе заявляется работоспособность RV-Match для приложений объемом ~300 тысяч строк кода. К недостаткам использования kcc/RV-Match можно отнести следующее:

- неполная поддержка стандартных библиотек;
- часть типов неопределенного поведения обнаруживается только в момент выполнения, т.е. эффективность зависит от полноты тестового покрытия;
- неизвестно качество работы инструмента на больших проектах; размер некоторых современных проектов (например, PostrgeSQL или ядро Linux) превосходит 1 миллион строк кода;

- позволяет обнаруживать неопределенное поведение, но не позволяет предотвратить уязвимость: kcc нельзя использовать в релизной сборке программы из-за низкой производительности, для предотвращения уязвимости придется исправлять исходный код программы;
- инструмент нацелен на поиск неопределенного поведения, поэтому не сможет найти нестабильный код, подобный приведенному в 2.2.

3.7 Использование стандартов безопасного программирования

При создании программ, используемых в областях, в которых безопасность критически важна (медицина, авиация и т.д.), применяются стандарты программирования, нацеленные на повышение безопасности, такие как MISRA [14], SEI CERT C Coding Standard [15] и другие. Эти стандарты ограничивают используемое подмножество языка и вводят указания по использованию этого подмножества. Это позволяет писать более безопасный код с меньшим числом уязвимостей. В рамках задачи, описанной в этой статье, использование стандартов безопасного программирования не может быть сочтено решением, поскольку подразумевается, что исходный код уже написан, а не пишется с нуля с соблюдением стандартов.

4. Концепция безопасного компилятора

В ситуациях, когда исходный код программы доступен, однако по каким-либо причинам не может быть модифицирован, или модификация потребует слишком больших усилий, большинство из способов, описанных в разделе 3, не приведут к желаемому результату: предотвращению уязвимостей. Инструменты динамического анализа в общем случае не способны заметить, что уязвимость, подобная описанным в 2.1, появилась в результате работы компилятора. Отдельно надо сказать об Undefined Behavior Sanitizer: при его использовании инструментация вставляется в промежуточное представление программы до этапа работы оптимизатора, что приводит к тому, что оптимизатор не будет удалять нестабильный код, причиной нестабильности которого является неопределенное поведение. Способ, описанный в 3.4, также в некоторой степени помогает: часть оптимизаций, способных удалить нестабильный код, можно отключить. Этот способ не является полным: некоторые оптимизации нельзя отключить, не установив уровень оптимизаций -O0, и даже в этом случае, если рассматривать GCC, некоторые оптимизации не отключаются. Также нельзя назвать этот способ надежным: потребуется вносить изменения в систему сборки, но эти изменения могут быть отменены другими флагами позднее, при этом сам факт изменений в системе сборки может быть нежелательным.

В таких ситуациях возможным способом предотвратить появление уязвимостей из-за нестабильного кода будет использование безопасного компилятора, не вносящего в код уязвимости при выполнении оптимизаций. Неудаление нестабильного кода без больших потерь в производительности скомпилированной программы является основным требованием к безопасному компилятору, при этом безопасный компилятор может предоставлять дополнительные возможности для обнаружения и предотвращения уязвимостей. Также безопасный компилятор должен работать при условии, что вносимые в исходный код модификации минимальны или вообще отсутствуют. Такой безопасный компилятор можно использовать в качестве замены штатного компилятора в системе сборки, что позволит ограничиться минимальными изменения в системе сборки или даже обойтись без них. При этом безопасный компилятор не должен предоставлять возможность отключения опций, контролирующих предотвращение уязвимостей.

5. Реализация безопасного компилятора

Реализованный безопасный компилятор разработан на базе GCC 9.3.0. Он реализует статические и динамические методы защиты, а также может работать вместе со средствами безопасности ОС. К статическим методам защиты относятся диагностики, настройки оптимизаций (как отключение конкретных оптимизаций, так и изменения их поведения). К динамическим методам относится использование Undefined Behavior Sanitizer, методов защиты от переполнения буфера (FORTIFY_SOURCE [16], stack protector [17]). Безопасный компилятор предоставляет возможность использовать улучшенную версию ASLR [18].

Очевидным следствием применения новых диагностик, проверок, оптимизаций и условия невозможности модификации исходного кода программ будет то, что некоторые программы не смогут быть собраны безопасным компилятором. Это может накладывать ограничения на применимость безопасного компилятора. Разделение реализованных в безопасном компиляторе методов защиты на несколько уровней позволяет более гибко подойти к вопросу применения безопасного компилятора для компиляции реальных программ. Было предложено разделение на три уровня защиты.

Использование безопасного компилятора с уровнем защиты 3 должно позволять собирать без ошибок большинство приложений. На этом уровне компилятор должен включать опции, относящиеся к “лучшим практикам”, используемым в больших проектах (ядре Linux, Firefox, PostgreSQL). К ним относятся такие опции как `-fPIE` (компиляция в позиционно-независимый код), `-fstack-protector-strong` (использование механизма `stack protector` для функций, которые содержат что-либо из перечисленного: локальный массив, вызов функции `alloca`, локальную переменную, у который берется адрес), `-fno-strict-aliasing` (запрет компилятору считать указатели различных типов обязательно различающимися) и другие, а также использование fortified-функций, включаемых опцией `FORTIFY_SOURCE`. Эти функции являются реализацией функций стандартной библиотеки с дополнительными проверками для обнаружения выходов за границы буферов. Fortified-функции были портированы из библиотеки Musl и представлены в заголовочных файлах в составе безопасного компилятора. Также диагностика -Wclobbered, присутствующая в GCC, была переработана (в частности, новая реализация использует другое внутреннее представление GCC) в рамках безопасного компилятора. Эта диагностика потребовала переработки, т.к. версия диагностики, присутствующая в GCC, имеет большое число ложных срабатываний [19]. Ожидается, что использование уровня защиты 3 может приводить к небольшому замедлению скомпилированной программы по сравнению с программой, скомпилированной GCC 9.3.0. В обоих случаях предполагается использование уровня оптимизации `-O2`, являющегося де-факто стандартным уровнем оптимизации при сборке реальных приложений.

На уровне защиты 2 добавляются флаги управления оптимизациями и настройки оптимизаций, разработанные специально для безопасного компилятора, а также включаются стандартные опции, необходимые для их работы. К добавленным на уровне 2 флагам управления оптимизациями относятся, в частности, флаги `-fkeep-oversized-shifts` и `-fpreserve-memory-writes`. Флаг `-fkeep-oversized-shifts` запрещает компилятору проводить сворачивание констант и продвижение констант в случаях, когда второй аргумент оператора сдвига больше или равен ширине типа. Это позволяет сохранить операцию сдвига в получающемся ассемблерном коде, повторяя поведение компилятора без оптимизаций. Флаг `-fpreserve-memory-writes` указывает компилятору сохранять все побочные эффекты всех операций записи, затрагивающих участок памяти из которого есть хотя бы одно чтение. Этот флаг позволяет устранить уязвимости вследствие нестабильного кода, подобные описанной в 2.2. Использование второго уровня защиты может приводить к тому, что некоторые приложения не будут собираться. Ожидается, что переход с уровня защиты 3 на уровень защиты 2 не приведет к сильному увеличению количества программ,

которые не смогут быть собраны безопасным компилятором, но скорость работы скомпилированных программ может уменьшиться.

Уровень защиты 1 отличается от предыдущих в первую очередь использованием UBSan. В безопасном компиляторе используется UBSan с добавленными опциями, в том числе опцией `-fsanitize=function`, добавляющей инструментацию непрямых вызовов функций проверками типов. Эта опция является реализацией аналогичной опции, присутствующей в Clang. Также на этом уровне добавлены опции `-fsanitize=null` (помимо других использований нулевого указателя, добавлен запрет вызова функции по нулевому указателю) и `-fsanitize=return` (достижение конца функции типа отличного от void без возврата значения, раньше такая проверка выполнялась только для кода на языке C++). Также на уровне защиты 1 используется модифицированная версия ASLR. В отличие от обычной реализации ASLR, позволяющей размещать загружаемую программу по рандомизированному адресу в памяти, реализация ASLR в безопасном компиляторе позволяет проводить рандомизацию расположения функций в файле программы, а также рандомизацию порядка размещения локальных переменных внутри фрейма функции. Уровень защиты 1 добавляет строгие проверки, многие приложения не смогут быть собраны на этом уровне без изменений в исходном коде. Использование этого уровня позволяет защититься от большего набора уязвимостей и снизить уровень угрозы, поскольку переведет часть уязвимостей из разряда “утечка данных” и “выполнение произвольного кода” в разряд “отказ в обслуживании”. Этот уровень предназначен для случаев, в которых безопасность является наивысшим приоритетом (в пользу чего можно пожертвовать производительностью, вплоть до замедления скомпилированной программы в 2-3 раза, и возможностью собирать недостаточно надежные приложения). Табл. 1 суммирует информацию о разделении методов защиты на уровни, приведенную в этом разделе.

Включение уровня защиты в безопасном компиляторе реализовано с помощью добавления опции `-Safe` с целочисленным аргументом, аналогично опции `-O`, контролирующей уровень оптимизации. Соответственно, безопасный компилятор может быть запущен с опциями `-Safe3, -Safe2, -Safe1` для включения соответствующего уровня защиты.

Табл. 1. Примеры используемых методов защиты по уровням
Table 1. Examples of used methods of protection by level

Методы защиты	Уровень защиты 3	Уровень защиты 2	Уровень защиты 1
-fPIE	X	X	X
-fstack-protector-strong	X	X	X
-fno-strict-aliasing	X	X	X
FORTIFY_SOURCE	X	X	X
-Wclobbered (улучш.)	X	X	X
-fkeep-oversized-shifts		X	X
-fpreserve-memory-writes		X	X
-fsanitize=undefined			X
-fsanitize=function			X
-fsanitize=null			X
-fsanitize=return			X
ASLR (улучш.)			X

6. Результаты

6.1 Исследование случаев неопределенного поведения

В ходе данной работы была проанализирована возможность появления в программах нестабильного кода в результате работы оптимизирующего компилятора. Для изучения поведения компиляторов при компиляции конструкций с неопределенным поведением был составлен тестовый набор, содержащий по одному и более фрагменту кода на каждый из 203 случаев, перечисленных в приложении J к стандарту языка C версии C11. Анализ показал, что знание о 17 из 203 типов неопределенного поведения используется компиляторами при проведении оптимизаций. Еще 9 типов могли бы влиять на работу оптимизаций, однако не удалось найти подтверждений что компиляторы (исследовались GCC и Clang) используют эту возможность. Остальные типы неопределенного поведения либо обнаруживаются диагностиками компилятора в виде предупреждения или ошибки, либо компилируются в код с некой предсказуемой семантикой. Под предсказуемой семантикой понимается, что либо существует единственный разумный способ скомпилировать конструкцию с таким случаем неопределенного поведения (ярким примером может служить неопределенное поведение из п. 6.4.2.1 стандарта C11 “Два идентификатора отличаются только в незначимых символах” — все современные компиляторы используют полные имена идентификаторов, а не только первые 16 символов), либо результатом компиляции будет код, содержащий ошибку, причем этот результат не будет зависеть от используемой версии компилятора, уровня оптимизаций и целевой архитектуры.

Проведение подобного исследования для языка C++ представляется заметно более трудоемким по следующим причинам: язык C++ более сложный, чем язык C (так, стандарт C++17 занимает 1448 страниц, а стандарт C11 — 696), причем для языка C++ не существует исчерпывающего списка неопределенного поведения, аналогичного приложению J к стандарту C11. Составление подобного списка само по себе является нетривиальной задачей. В 2019 году в рабочую группу по стандартизации языка C++ было внесено предложение [20] о составлении такого списка, однако на момент написания статьи оно не было принято.

6.2 Корректность

Для проверки корректности работы безопасного компилятора был составлен тестовый набор, включающий в себя 3 типа тестов: тесты, проверяющие наличие вывода компилятором требуемой диагностики, тесты, проверяющие срабатывание динамических проверок во время выполнения, и тесты, проверяющие результат кодогенерации. Тесты, проверяющие результат кодогенерации, построены на основе исследования случаев нестабильного кода. Тестовый набор разделен на 3 уровня соответственно уровням защиты. Безопасный компилятор, запущенный с опцией, устанавливающей уровень защиты, должен проходить все тесты соответствующего уровня. Разработанный в рамках этой работы безопасный компилятор успешно проходит тесты.

6.3 Сборка дистрибутивов Linux

Помимо сборки реальных приложений по отдельности с помощью безопасного компилятора, о чем будет подробнее рассказано в следующем разделе, применимость безопасного компилятора в реальных условиях оценивалась на сборке дистрибутивов Linux. Необходимо отметить, что сборка дистрибутива Linux компилятором, отличным от стандартного для данного дистрибутива, чаще всего является нетривиальной задачей — например, даже при обновлении версии GCC чаще всего требуется изменение исходного кода в некоторых пакетах, не говоря уже сборке каким-либо другим компилятором, например, Clang – на данный момент такая задача решена не для всех пакетов. Несмотря на то, что безопасный компилятор разработан на основе GCC, с рядом пакетов дистрибутива Linux (прежде всего,

ядра и некоторых системных) возникают трудности. Данный вопрос заслуживает подробного анализа и рассмотрения в рамках отдельной работы, а в данной статье мы приводим результаты предварительных экспериментов.

Для экспериментов использовались CRUX 3.5 [21] (версия ядра 4.19.48) и Debian 10.2.0. В случае CRUX безопасный компилятор с уровнем защиты 3 обнаружил проблемы в 28 пакетах из 100. Эти проблемы удалось устранить с помощью модификации исходного кода и собрать CRUX с уровнем защиты 3. Попытка собрать CRUX с уровнем защиты 2 показала, что необходима заметно большая модификация исходного кода, которая на данном этапе не выполнялась. В случае Debian возникли проблемы со сборкой 83 пакетов из 983 с уровнем защиты 3, устранение этих проблем потребовало бы слишком больших усилий, и в рамках данной работы также не проводилась.

При сборке реального дистрибутива сложности возникают прежде всего с ядром Linux, системными библиотеками и программами (например, grub и busybox), средами исполнения (python), а также программами, содержащими очень старый код, не соответствующий стандартам (cpio). Так, например, ядро не может быть собрано с опцией stack protector, системные программы используют в коде неподдерживаемые техники оптимизации, что противоречит настройкам безопасного компилятора для всех уровней безопасности. Работа по исправлению исходного кода пакетов, который не соответствует стандарту языка C, предполагает не только чисто технические исправления, но зачастую требует сложного анализа, для того чтобы переписать несовместимые части приложения, не повлияв на его функциональность. Такая работа должна вестись разработчиками дистрибутива с привлечением разработчиков приложений.

Работа, проведенная в рамках проекта безопасного компилятора, показала, что такая большая задача, как сборка дистрибутива операционной системы, является выполнимой, но требует взаимодействия с разработчиками операционной системы. На момент написания этой статьи эта работа была в начальной стадии.

6.4 Исследование производительности

Производительность программ, скомпилированных безопасным компилятором тестировалась в 5 сценариях:

- воспроизведение части партии в го с помощью приложения GNU Go 3.8 [22], являющегося частью набора тестов SPEC [23];
- перекодирование набора WAV-файлов в формат MP3 с помощью приложения LAME 3.100 [24], являющегося частью набора тестов Phoronix Test Suite [25];
- запуск реализации бенчмарка fannkuch, входящего в состав набора тестов The Computer Language Benchmarks Game [26];
- перекодирование набора видеофайлов из формата YUV в формат MKV при помощи библиотеки x264 [27], входящей в состав набора тестов SPEC (x264-snapshot-20190407-2245-stable);
- сжатие текстового файла при помощи библиотеки zlib 1.2.11 [28], являющейся частью набора SPEC.

Табл. 2. Результаты измерения производительности
Table 2. Performance evaluation results

Сценарий	Baseline	Safe3	Safe3 замедл.	Safe2	Safe2 замедл.	Safe1	Safe1 замедл.
GNU Go	4.67 с	4.85 с	3.85%	5.23 с	11.99%	9.32 с	99.57%
LAME	3.45 с	3.55 с	2.89%	3.55 с	2.83%	9.89 с	186.31%
fannkuch	2.03 с	2.42 с	19.21%	2.42 с	19.21%	3.51 с	72.91%

x264	7.91 с	7.90 с	-0.23%	8.06 с	1.78%	18.30 с	131.21%
zlib	2.25 с	2.29 с	2.00%	2.28 с	1.33%	2.80 с	24.44%

В табл. 2 приведены результаты измерения времени выполнения тестовых сценариев на компьютере с процессором Intel® Core™ i5-7600K (архитектура x86_64) и операционной системой Ubuntu 20.04.2. Столбец baseline соответствует запуску компилятора GCC версии 9.3.0 с уровнем оптимизации -O2. Столбцы Safe3, Safe2, Safe1 соответствуют запускам безопасного компилятора с уровнями защиты 3, 2 и 1 соответственно и уровнем оптимизации -O2. Для каждого из этих столбцов в столбце справа указано замедление соответствующего сценария в процентах. В столбцах указано среднее по 20 запускам время, округленное до сотой секунды, это приводит к тому, что для одинаковых значений времени может быть небольшое отличие в значении замедления. В двух случаях (между Baseline и Safe3 в сценарии x264, и между Safe3 и Safe2 в сценарии zlib) наблюдается ускорение в среднем на 0.01 с, что не является значимым результатом.

Представленные данные показывают, что в случае использования уровня защиты 3, замедление скомпилированной программы составляет не более 20%, причем в 4 сценариях из 5 не превышает 4%. Использование уровня защиты 2 ухудшает производительность в 2 сценариях из 5 относительно уровня защиты 3, но замедление также не превышает 20%. Нужно заметить, что замедление может усиливаться в случае программ, в которых заметный выигрыш от оптимизации достигается удалением лишних записей в память. При использовании уровня защиты 1 замедление составляет от 24% до 186%, что согласуется с ожиданием: в [29] указывается, что при использовании санитайзеров программа может замедляться в 2-3 раза.

7. Заключение

В рамках данной работы было проведено исследование возникновения уязвимостей в результате работы компиляторных оптимизаций. Основным видом уязвимостей, рассмотренных в этой работе, были уязвимости в результате наличия нестабильного кода, т.е. кода, который может быть удален в результате агрессивных оптимизаций. В работе были рассмотрены подходы, применяемые для обнаружения и устранения уязвимостей в коде программ. Для решения описанной проблемы была предложена концепция безопасного компилятора. Также в статье описана реализация безопасного компилятора на базе компилятора GCC, в нем реализовано три уровня защиты.

Реализованный безопасный компилятор был успешно применен для сборки реальных программ, а также для сборки дистрибутивов операционных систем CRUX и Debian. В случае Debian не удалось добиться полной сборки дистрибутива при сохранении условия минимальности вносимых изменений, задачу сборки дистрибутива планируется решить в дальнейшем во взаимодействии с разработчиками дистрибутива, разработав набор исправлений его исходного кода.

Список литературы / References

[1]. Wang X., Chen H. et al. Undefined behavior: what happened to my code? In Proc. of the Asia-Pacific Workshop on Systems, 2012, pp. 1-7.
[2]. Wang X., Zeldovich N. et al. Towards optimization-safe systems: Analyzing the impact of undefined behavior. In Proc. of the Twenty-Fourth ACM Symposium on Operating Systems Principles, 2013, pp. 260-275.
[3]. C11 Standard ISO/IEC 9899:2011, Programming languages – C. ISO/IEC, 2011.
[4]. STACK. Available at https://css.csail.mit.edu/stack/, accessed 25.08.2021.
[5]. Undefined Behavior Sanitizer. Available at https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html, accessed 25.08.2021.
[6]. Address Sanitizer. Available at https://clang.llvm.org/docs/AddressSanitizer.html, accessed 25.08.2021.
[7]. Tis-interpreter. Available at https://github.com/TrustInSoft/tis-interpreter, accessed 25.08.2021.

[8]. Serebryany K. Sanitize, Fuzz, and Harden Your C++ Code. In Proc. of the USENIX ENIGMA Conference, 2006, 35 p.
[9]. Cadar C., Dunbar D., and Engler D. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In Proc. of the 8th USENIX Conference on Operating Systems Design and Implementation, 2008, pp. 209-224.
[10]. Field Experience with Annex K – Bounds Checking Interfaces. Available at http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1967.htm, accessed 25.08.2021.
[11]. Hathhorn C., Ellison C., and Roşu G. Defining the undefinedness of C. ACM SIGPLAN Notices, vol. 50, issue 6, 2015, pp. 336–345.
[12]. RV-Match. Available at https://runtimeverification.com/match/, accessed 25.08.2021.
[13]. Guth D., Hathhorn C. et al. Rv-match: Practical semantics-based program analysis. Lecture Notes in Computer Science, vol. 9779, 2016, pp. 447-453
[14]. MISRA. Available at https://www.misra.org.uk/, accessed 25.08.2021.
[15]. SEI CERT C Coding Standard. Available at https://wiki.sei.cmu.edu/confluence/display/c, accessed 25.08.2021.
[16]. FORTIFY_SOURCE. Available at https://access.redhat.com/blogs/766093/posts/3606481, accessed 25.08.2021.
[17]. Stack Smashing Protector. Available at https://www.linuxfromscratch.org/hints/downloads/files/ssp.txt, accessed 25.08.2021.
[18]. Address Space Layout Randomization. Available at https://docs.oracle.com/en/operating-systems/oracle-linux/6/security/ol_aslr_sec.html, accessed 25.08.2021.
[19]. [9/10/11/12 Regression] "clobbered by longjmp" warning ignores the data flow. Bug 21161, GCC, 2005. Available at https://gcc.gnu.org/bugzilla/show_bug.cgi?id=21161, accessed 25.08.2021.
[20]. P1705R1 Enumerating Core Undefined Behavior. Available at http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1705r1.html, accessed 25.08.2021.
[21]. CRUX. Available at https://crux.nu/, accessed 25.08.2021.
[22]. GNU Go. Available at https://www.gnu.org/software/gnugo/, accessed 25.08.2021.
[23]. SPEC's Benchmarks. Available at https://www.spec.org/benchmarks.html, accessed 25.08.2021.
[24]. The LAME Project. Available at https://lame.sourceforge.io/, accessed 25.08.2021.
[25]. Phoronix Test Suite. Available at https://www.phoronix-test-suite.com/, accessed 25.08.2021.
[26]. The Computer Language Benchmarks Game. Available at https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html, accessed 25.08.2021.
[27]. x264. Available at http://www.videolan.org/developers/x264.html, accessed 25.08.2021.
[28]. Zlib. Available at http://www.zlib.net/, accessed 25.08.2021.
[29]. Song D., Lettner J. et al. SoK: Sanitizing for Security. In Proc. of the IEEE Symposium on Security and Privacy (SP), 2019, pp. 1275-1295.

Информация об авторах / Information about authors

Роман Вячеславович БАЕВ – старший лаборант отдела компиляторных технологий. Научные интересы: компиляторные технологии, оптимизации.

Roman Vyacheslavovich BAEV – Researcher in Compiler Technology department. Research interests: compiler technologies, optimizations.

Леонид Владленович СКВОРЦОВ – стажер-исследователь отдела компиляторных технологий. Научные интересы: компиляторные технологии, оптимизации.

Leonid Vladlenovich SKVORTSOV – Researcher in Compiler Technology department. Research interests: compiler technologies, optimizations.

Евгений Алексеевич КУДРЯШОВ – стажер-исследователь отдела компиляторных технологий. Научные интересы: компиляторные технологии, оптимизации.

Evgeny Alekseevich KUDRYASHOV – Researcher in Compiler Technology department. Research interests: compiler technologies, optimizations.

Рубен Артурович БУЧАЦКИЙ – научный сотрудник отдела компиляторных технологий. Научные интересы: компиляторные технологии, оптимизации.

Ruben Arturovich BUCHATSKIY – Researcher in Compiler Technology department. Research interests: compiler technologies, optimizations.

Роман Александрович ЖУЙКОВ – старший научный сотрудник отдела компиляторных технологий. Научные интересы: компиляторные технологии, оптимизации.

Roman Aleksandrovich ZHUYKOV – Researcher in Compiler Technology department. Research interests: compiler technologies, optimizations.

DOI: 10.15514/ISPRAS-2021-33(4)-15

Об особенностях фаззинг-тестирования сетевых интерфейсов в условиях отсутствия исходных текстов

[1] *И.В. Шарков, ORCID: 0000-0002-9767-142X <sharkov@ispras.ru>*
[1,2] *В.А. Падарян, ORCID: 0000-0001-7962-9677 <vartan@ispras.ru>*
[3] *П.В. Хенкин <Pkhenkin@yandex.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*
[3] *ПАО Сбербанк, 117312, Россия, Москва, ул. Вавилова, д. 19*

Аннотация. Цифровизация общества приводит к созданию большого количества распределенных автоматизированных информационных систем в различных областях современной жизни. Необходимость отвечать требованиям безопасности и надежности подталкивает к созданию инструментов их автоматизированного тестирования. Фаззинг-тестирование в рамках цикла безопасной разработки является необходимым инструментом решения указанной задачи, но не менее востребованы средства фаззинга бинарного кода, обеспечивающие поиск критических дефектов в уже функционирующих системах. Особенно остро этот вопрос стоит при исследовании безопасности проприетарных систем, функционирующих с использованием закрытых протоколов. В ходе проводимых исследований было выявлено, что для тестирования сетевых приложений в условиях отсутствия исходных текстов применение универсальных фаззеров затруднительно. Эти обстоятельства подталкивают к созданию простого в эксплуатации инструмента фаззинг-тестирования сетевых программ. В работе рассматриваются особенности фаззинг-тестирования такого рода программ и предлагаются возможные пути решения выявленных проблем.

Ключевые слова: фаззинг; тестирование; сетевые программы; спецификация протокола; DynamoRIO; протокольный автомат

Для цитирования: Шарков И.В., Падарян В.А., Хенкин П.В. Об особенностях фаззинг-тестирования сетевых интерфейсов в условиях отсутствия исходных текстов. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 211-226. DOI: 10.15514/ISPRAS–2021–33(4)–15

Features of fuzzing network interfaces without source codes

[1] *I.V. Sharkov, ORCID: 0000-0002-9767-142X <sharkov@ispras.ru>*
[1,2] *V.A. Padaryan, ORCID: 0000-0001-7962-9677 <vartan@ispras.ru>*
[3] *P.V. Khenkin <Pkhenkin@yandex.ru >*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[3] *PJSC Sberbank, 19, Vavilova Street, Moscow, 117312, Russia*

Abstract. The digital transformation of society is leading to the creation of a large number of distributed automated information systems in various areas of modern life. The need to meet security and reliability requirements prompts the creation of tools for their automated testing. Fuzzing within the security development lifecycle (SDL) is a strictly required tool for solving this problem. Tools for fuzzing binary-only applications are in demand too. These kind of fuzzing tools provide the search for critical defects in already functioning systems. It is especially acute when researching the security of proprietary systems operating using closed protocols. In the course of the research, it was found out that for fuzzing network applications in the absence of source codes, the use of universal fuzzers is complicated by many factors. These circumstances are pushing for the creation of an easy-to-use tool for network applications fuzzing. The paper discusses the features of fuzzing of this kind of programs and suggests possible solutions to the identified tasks.

Keywords: fuzzing; testing; network applications; protocol specification; DynamoRIO; protocol state machine

For citation: Sharkov. I.V., Padaryan V.A., Khenkin P.V. Features of network interfaces fuzzing without source codes. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021. pp. 211-226 (in Russian). DOI: 10.15514/ISPRAS–2021–33(4)–15

1. Введение

В условиях стремительной цифровизации общества крайне востребованы инструментальные средства анализа кода программного обеспечения (ПО) на наличие дефектов. Растет количество распределенных автоматизированных информационных систем, обеспечивающих функционирование объектов критической инфраструктуры, безопасность, медицинскую деятельность, видеонаблюдение, электронную коммерцию, хранение и обработку персональных данных. Целенаправленная эксплуатация программных дефектов способна вывести из строя или захватить управление такой системой, что влечет за собой очень серьезные последствия.

В состав современных инструментальных средств входят средства динамического анализа (тестирования), где для обеспечения безопасности наиболее популярным видом тестирования является фаззинг.

Фаззинг-тестирование — это способ автоматизированного тестирования программ, способный находить дефекты в их исполняемом коде вне зависимости от наличия исходных текстов. Инструменты фаззинг-тестирования входят в стек технологий цикла безопасной разработки, регламентированного нормативными документами ведомств-регуляторов, проведение процедур фаззинг-тестирования предусмотрено ГОСТ Р 56939-2016 как необходимой меры разработки безопасного ПО.

Существуют различные хорошо зарекомендовавшие себя в практической работе комплексные инструменты и подходы к решению данной задачи. Например, к подобным средствам относится инструмент динамического анализа Crusher [1] (ИСП РАН). Нормативные требования, необходимые с точки зрения разработки безопасного ПО, приводят к формированию типовых методик фаззинг-тестирования и стандартизации технологических цепочек. Данные условия вынуждают создавать универсальные и сложные

в эксплуатации инструменты фаззинг-тестирования, функционирование которых основано на типовых приемах, что затрудняет реализацию оригинальных и нестандартных методик фаззинг-тестирования. Стоит отметить, что многие серьезные уязвимости ПО были найдены не с помощью сложных многофункциональных систем, а с помощью простых программ, созданных для решения конкретной задачи с использованием оригинальных подходов [2-5].

Фаззинг, осуществляемый в рамах цикла безопасной разработки, предназначен для решения задач выявления дефектов до начала эксплуатации программного средства, т. е. нацелен на предупреждение выпуска ПО с дефектами и уязвимостями. Однако, учитывая факты выявления новых уязвимостей в «старых» компонентах ПО, в заимствованном коде, очевидно, что полностью решить задачу этим путем не удается, примером может служить уязвимость OpenSSL CVE-2021-3449 [6]. Данное обстоятельство является двигателем развития средств автоматизированного тестирования бинарного кода в условиях наличия минимума информации и отсутствия исходных текстов.

Фаззеры можно рассматривать не только как технологический элемент жизненного цикла безопасного ПО. Одновременно с этим они являются незаменимыми инструментами исследования и оценки надежности и стабильности бинарного кода различных классов ПО, их целесообразно применять, как один из инструментов в тестировании на проникновение (penetration test). В таких условиях средства фаззинга не могут использоваться для «тяжеловесной» верификации программных средств и должны обеспечивать «недорогой», легковесный метод поиска ошибок и выявления уязвимостей. При этом слово «недорогой» следует относить не столько к использованию вычислительных ресурсов в процессе функционирования инструмента, сколько к требованиям по полноте и доступности исходных данных на подготовительном этапе: исходных текстов, документации ПО, конфигураций CI/CD, функциональных тестов и тестовой обвязки, и т.д.

Основная задача данного исследования заключается в систематизации проблем, возникающих при фаззинге в условиях отсутствия исходных текстов, и определении возможности создания гибкого, легковесного и «недорогого» в использовании программного средства, или фреймворка для автоматизированного тестирования сетевых серверных и клиентских программ, доступ к исходным текстам которых невозможен.

2. Постановка задачи и способы решения

В качестве исходных данных рассмотрим ситуацию, когда в нашем распоряжении имеются настроенные и корректно функционирующие на контролируемых компьютерах серверная и клиентская программы. Среда их функционирования ОС Linux x86-64. Исходных текстов нет. Протокол сетевого взаимодействия неизвестен (не документирован). В качестве целевого объекта выберем серверную программу. Основная задача заключается в выявлении ошибок, приводящих к нарушению ее функционирования. Проведя декомпозицию задачи получается, что для осуществления фаззинг-тестирования целевых приложений необходимо решить следующие задачи:

- проанализировать алгоритмы, выявить доступные для воздействия блоки;
- изучить спецификацию сетевого протокола, включая служебную составляющую;
- определить поверхность атаки;
- сформировать систему оценки результатов поиска ошибок;
- реализовать систему контроля состояний целевой программы с интерфейсом прямой и обратной связи;
- подготовить начальную последовательность и создать способ автоматической модификации (мутации) данных, предназначенных для отправки исследуемой программе.

2.1. Анализ алгоритмов сетевых программ

Создание универсального фаззера упирается в использование большого количества сильно различающихся алгоритмов в разных программах. В большинстве случаев ход автоматизированного тестирования подразумевает, что исследуемая программа корректно завершается самостоятельно по завершению теста или аварийно в процессе его выполнения, в противном случае принимается решения о зависании, выполняется ее принудительное завершение и запускается очередной цикл тестирования.

Сетевые программы, как правило, самостоятельно не завершаются поскольку их алгоритмы основаны на бесконечных циклах ожидания подключений, обработки входных данных и отправки исходящих сообщений.

В этом состоит одна из трудностей их фаззинг-тестирования в условиях отсутствия исходных текстов. Модифицировать бинарный код с целью ограничения количества итераций хоть и возможно, но крайне затратно; требуется другой подход.

Если абстрагироваться от деталей конкретных реализаций, абстрактный алгоритм функционирования сетевого сервера (рис. 1) заключается в следующих шагах:

- создание сокета;
- инициализация сетевого подключения в режиме прослушивания;
- ожидание запроса клиента на подключение;
- порождения дочернего процесса/потока для осуществления сетевого взаимодействия;
- переход в состояние ожидания очередного запроса на подключение.

Обработка входных данных осуществляется в цикле получения-отправки данных в дочернем процессе/потоке.

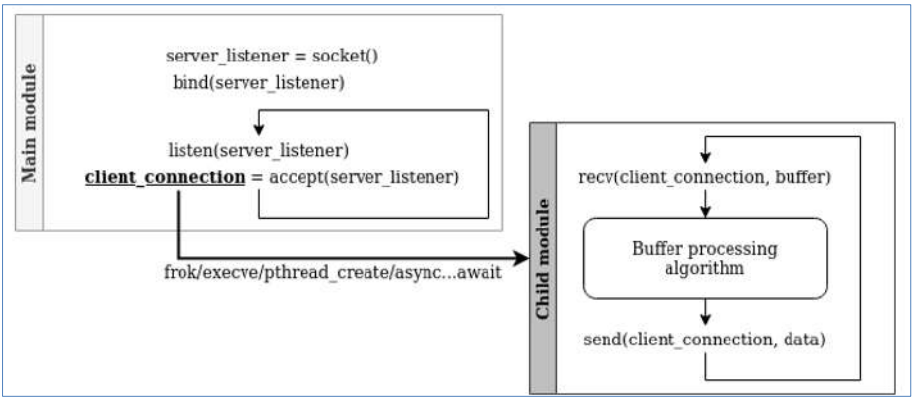


Рис. 1. Схема работы сетевого сервера
Fig. 1. Network server work scheme

Из описания алгоритма функционирования серверного приложения видно, что внешнее воздействие на его функционирование возможно оказать через сетевой интерфейс только в дочернем процессе/потоке после получения порции данных от подключенного клиента. Таким образом, изучение алгоритма серверной программы можно сузить до блока обработки входных данных (на схеме «Buffer processing algorithm», см. рис. 1), точкой входа следует рассматривать инструкцию следующую за функцией *recv(...)*, а точкой выхода – перед вызовом функции *send(...)*.

Клиентские сетевые программы функционируют по схожему алгоритму, за исключением того, что именно клиент инициирует создание канала вызовом функции *connect(...)*, после чего начинается цикл обмена информацией. Стоит обратить внимание на то, что блок

обработки входящих сообщений здесь может быть значительно больше, поскольку может включать часть кода, отвечающего за пользовательский интерфейс, журналирование и т.п.

2.2. Анализ спецификации сетевого протокола

Задача восстановления сетевого протокола разделяется на две дополняющие друг друга подзадачи: восстановление форматов сетевых сообщений и восстановление протокольного автомата. Для первой задачи были предложены различные подходы к ее автоматизированному решению, известные еще с нулевых годов, для второй задачи на данный момент эффективные автоматизированные решения авторам данной статьи неизвестны. Более того, даже в условиях наличия исходных текстов тестирование реализации протокола зачастую осуществляется посредством вручную написанной тестовой системы, которая в полной мере реализует ответную часть сетевого взаимодействия.

Методы восстановления сетевых протоколов можно разделить на четыре группы:

- запись и анализ сетевых трасс;
- дизассемблирование целевой программы и анализ полученного статического представления;
- отложенный анализ потока данных по записанной трассе выполнения тестируемой программы;
- динамический анализ потока данных непосредственно во время выполнения.

Запись сетевых трасс позволяет получить порядок обмена сетевыми сообщениями, просмотреть их содержимое, определить размер и назначение отдельных полей, но не дает возможность увидеть полную картину структур пакетов.

Дизассемблирование формирует статический код алгоритмов и потенциально позволяет решать задачу восстановления спецификации протокола, однако в случае анализа сложных громоздких программ этот процесс достаточно трудоемкий и требует значительного времени. Единственным средством автоматизации выступает декомпилятор, который восстанавливает переменные, в том числе – агрегатных типов. Но такой важный аспект, как семантику полей декомпиляция не захватывает, решать эту задачу приходится вручную, как и восстановление протокольного автомата.

Основная идея анализа потока данных заключается в контроле обращений к содержимому блока памяти и его распространению в процессе функционирования программы. В задаче восстановления спецификаций сетевых протоколов данный подход позволяет выделить основные элементы пакетов, их размеры, смещения относительно начала исходного блока. В результате формируется обобщенная структура сетевых пакетов, необходимая для рациональной модификации/мутирования сообщений в ходе осуществления фаззинг-тестирования. Опять же задача восстановления протокольного автомата в данном подходе не решается.

Запись и анализ трассы выполнения программы позволяет анализировать потоки данных, отслеживать их распространение, не только по самой программе, но и между процессами. Однако эта работа достаточно ресурсоемкая, особенно в случае записи трасс во время осуществления длительных сессий взаимодействия клиента и сервера, необходимых для воссоздания наиболее полных спецификаций.

Последний подход предполагает выполнение динамического анализа осуществляется «на лету», в процессе работы исследуемой программы. Данный подход может быть реализован путем динамической инструментации с использованием, например, систем Qemu, DynamoRIO или Pin и позволяет не только отслеживать поток управления, но контролировать его. Кроме того, динамический анализ позволяет осуществлять мониторинг только интересующей части алгоритма, что существенно уменьшает объем работы.

Сетевые протоколы разделяются на протоколы без состояния (stateless-протоколы) и требующие сохранения состояния (stateful-протоколы). При осуществлении взаимодействия между клиентом и сервером с помощью stateless-протокола каждое последующее сообщение не зависит от предыдущего, примером может служить HTTP, stateful-протоколы определяют строгий порядок отправки и получения сообщений, требуемый для перехода к следующему состоянию, нарушение которого приводит к откату к предыдущему состоянию или закрытию сетевого канала, например, протоколы FTP, TLS.

В условиях неизвестного сетевого протокола возникает отдельная задача классификации протокола как stateless или stateful. Решать такую задачу автоматизировано возможно при наличии представительных образцов трафика, по которым записывается трасса выполнения исследуемой программы. После выявления в ней мест, в которых принимаются входящие пакеты необходимо определить наличие связи в потоке данных между обработкой разных пакетов. В силу сложности и самодостаточности данной задачи в настоящей статье она не рассматривается.

Представляется целесообразным оценивать полноту достигнутого покрытия не только по покрытию кода, как того требуют действующие нормативные документы. При тестировании сетевого ПО не менее важно обеспечить максимально полное покрытие состояний протокольного автомата. Ниже по тексту будут использоваться понятия фаззинга «в ширину» и «в глубину», во избежание возникновения коллизий определим, что эти понятия в данном контексте рассматриваются применительно к спецификации сетевого протокола, а не к структуре бинарного кода исполняемой программы. Удлинение цепочки задействованных в процессе тестирования базовых блоков не указывает на увеличение глубины фаззинга, а переход на следующий уровень протокола как раз и является индикатором погружения и перехода протокола в новое состояние.

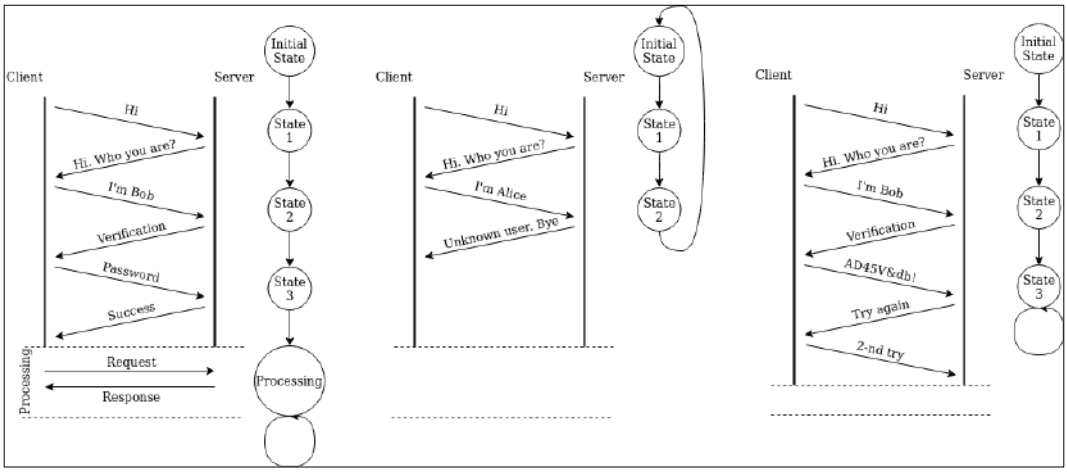
Фаззинг-тестирование сетевых программ, реализующих stateless-протоколы, представляет собой исследование «в ширину», поскольку порядок отправки пакетов неважен требуется максимально расширить количество возможных типов пакетов. Интересует процесс обработки входных данных, задача сводится к модификации входных данных таким образом, чтобы максимально увеличить количество задействованных в их обработке функций. При этом необходимо осуществлять контроль выполнения целевой программы путем отслеживания переходов между базовыми блоками.

Исследования ПО использующего stateful-протоколы сложнее, здесь важен и порядок и содержимое передаваемых сообщений, конечной целью можно назвать фаззинг «в глубину», т.е. осуществление результативной отправки наибольшего числа различных обрабатываемых (не отбрасываемых) целевой программой сообщений. Однако при достижении очередного уровня «погружения» в соответствии с имеющейся спецификацией протокола следует осуществлять фаззинг «в ширину», контролируя при этом состояние целевой программы и протокола для принятия решения об отправке очередного пакета. Для решения этой задачи может использоваться математический аппарат конечных автоматов или цепей Маркова [9].

Протокольный автомат определяет порядок и задает функцию перехода из состояния в состояние, как правило, является подсистемой инструмента фаззинг-тестирования, он задействован в процессе генерации набора входных данных и применяется для исключения случайных пакетов, не удовлетворяющих спецификации, и сокращения количества нерезультативных тестов. Наиболее удобный способ графического представления протокольного автомата – ориентированный граф, где вершины задают состояния, а ребра отражают возможные переходы.

Протокольный автомат строится итеративно. Рассмотрим пример абстрактной системы с авторизацией и аутентификацией. Осуществим подключение клиента к серверу. Проведем три таких эксперимента. В ходе первого выполним подключение с использованием валидной пары логин-пароль и реализуем после этого работу с системой (рис. 2а). Во втором

эксперименте используем неверное имя пользователя (рис. 2b. В третьем – неправильный пароль (рис. 2c). В результате будут получены три графа переходов состояний, каждый из которых описывает часть протокольного автомата данной системы.



a) сессия 1 *b) сессия 2* *c) сессия 3*

Рис. 2. Пример этапов построения протокольного автомата
Fig. 2. Stages of building a protocol state machine sample

Полученные графы переходов легко обобщаются в единый протокольный автомат (рис. 3), отражающий реализованную в рамках тестовых сессий часть протокола. Вместе с тем, важно отметить, что начинать фаззинг-тестирование в общем случае можно даже имея только часть автомата, полученную в рамках одной сессии, а его дополнение и модернизация может осуществляться в ходе тестирования с использованием вновь полученных данных.

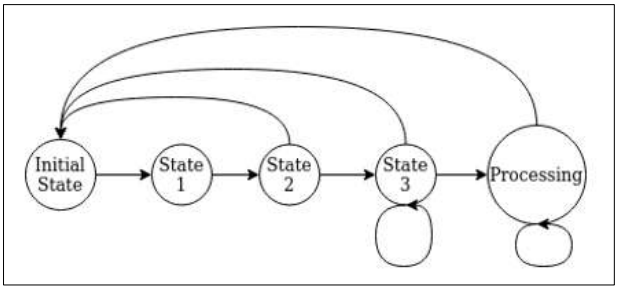


Рис. 3. Пример обобщенного протокольного автомата
Fig. 3. Composite protocol state machine example

Успешная практика применения протокольных автоматов в процессе фаззинга сетевых приложений наглядно отражена в инструменте AFLNet [12], основанном на широко известном фаззере AFL, позволяющем осуществлять фаззинг-тестирование программ без исходных текстов, для инструментации в этом случае используется эмулятор QEMU. Архитектура инструмента такова, что автомат строится вручную на основе анализа последовательности перехваченных сетевых пакетов исследуемого приложения и реализуется функцией на языке программирования C. Однако это вносит определенные неудобства в силу того, что для каждого нового протокола приходится перекомпилировать и отлаживать инструмент. Важно отметить, что в ходе написания функции, реализующей протокольный автомат, требуется спецификация протокола и понимание структуры сообщений. В противном случае результативность и успешность фаззинг-тестирования не гарантируется.

В ходе итерации фаззинга с использованием AFLNet выполняется последовательная отправка цепочки сообщений и анализируются ответы тестируемой программы, по завершению отправки осуществляется ее принудительное завершение командой *kill(...)*, что в некоторых случаях приводит к зависанию и необходимости прерывания фаззинга.

2.3. Определение поверхности атаки

При построении систем, функционирующих с использованием защищенных каналов связи, используется технология инкапсуляции протоколов, схематично это показано на рис. 4.

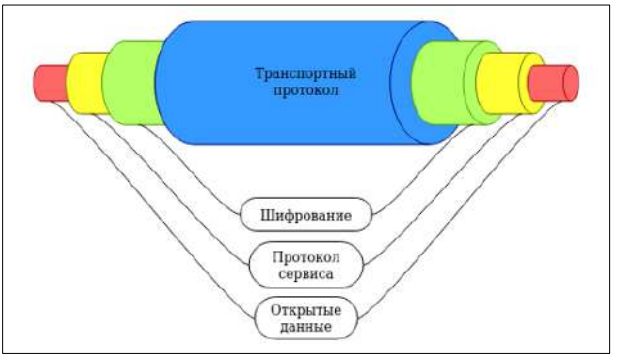


Рис. 4. Схематичный пример инкапсуляции протоколов
Fig. 4. Schematic example of protocol encapsulation

Открытые данные – это данные доступные в явном виде, например, сообщения, отображаемые с помощью элементов пользовательского интерфейса, или файлы, хранящиеся на носителе информации. Открытые данные кодируются и трансформируются в соответствии с формальным представлением протокола программы, формируется сообщение для отправки. Далее оно передается системе шифрования, формируется новое сообщение согласно спецификации используемого средства защиты. После чего транспортный протокол осуществляет его отправку адресату. На принимающей стороне для извлечения открытых данных осуществляется обратная процедура «снятия» слоев.

Программные дефекты, потенциально, могут присутствовать в программных модулях, обеспечивающих работу любого слоя, а значит проведение фаззинг-тестирования следует проводить для всех используемых программных модулей и задействованных протоколов. Причем чем больше «глубина погружения» между слоями тем больше требуется дополнительной информации.

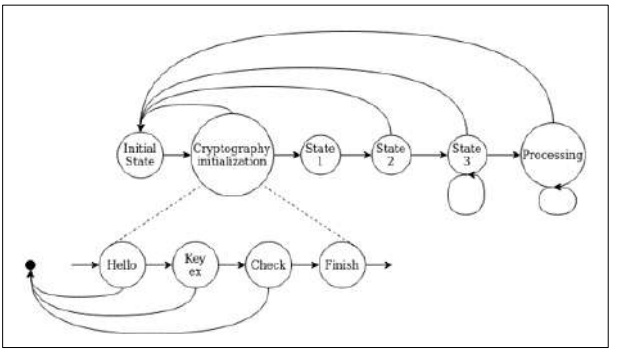


Рис. 5. Протокольный автомат программной системы с подсистемой защиты информации
Fig. 5. System with crypto protection subsystem protocol state machine example

Независимо от типа протокола, после инкапсуляции для обеспечения безопасности передаваемых данных (например, с использованием TLS), его следует рассматривать, как stateful-протокол.

Усложним рассмотренную ранее систему, введя в ее состав подсистему защиты передаваемой информации. Протокольный автомат новой системы получит дополнительное состояние – прохождения процедуры инициализации защищенного соединения (рис. 5).

Система построения защищенного канала связи имеет собственный протокол, значит, как показано на рис. 5, при необходимости возможно построить автомат, описывающий ее состояния. Однако осуществлять фаззинг-тестирование подобной системы лучше разделив ее на две: самой системы без защиты информации и подсистему защиты информации, - это позволит существенно упростить процесс.

2.4. Оценка результатов фаззинга и контроль состояния целевой программы

Современные фаззеры для оценки результатов тестирования используют измерение количества выполненных функций, базовых блоков, строк исполненного исходного текста, реберное покрытие графа переходов между базовыми блоками и т.д.

Прирост покрытия подсчитывается по результатам набора тестов, между которыми осуществляется перезапуск целевой программы и, как следствие ее переход в начальное состояние. В результате полученная оценка показывает, что та или иная часть кода достижима в ходе тестирования, однако, не гарантирует отсутствие в нем дефектов, которые могут проявиться в ходе длительной эксплуатации. Кроме того, в случае, когда исходные тексты недоступны и спецификация неизвестна, достигнуть 100% покрытие исполняемого кода невозможно.

В сетевых приложениях количество кода, отвечающего за непосредственное сетевое взаимодействие и обработку входных данных, может занимать лишь малую часть от его общего объема. В этом случае в процессе тестирования подсчитанное покрытие будет несущественным, а подход к оценке результатов на основе данной метрики мало информативен.

Два выше описанных фактора приводят к необходимости использования новой метрики для оценки результатов фаззинг-тестирования сетевых программ. Предполагается в качестве альтернативы использовать «покрытие» спецификации протокола. Подсчет реберного покрытия, вместе с тем, будет использоваться в качестве инструмента контроля процесса фаззинга, выявления условных переходов и подтверждения продвижения между базовыми блоками. Решить данную задачу возможно с применением средства динамической инструментации.

2.5. Подготовка начальной последовательности и модификация входных данных

В качестве начальной последовательности для инициализации процесса фаззинга целесообразно выбирать набор легитимных сетевых пакетов, отправленных целевой программе в процессе ее изучения и восстановления спецификации протокола. Кроме того, при проведении фаззинг-тестирования сетевых программ целесообразно не уменьшать корпус начальных данных, а максимально нагружать протокольный автомат.

Наличие спецификации позволяет формировать правильные с точки зрения назначения полей сообщения, однако, в случае использования в их структуре контрольных сумм возникают дополнительные трудности. Для правильного вычисления контрольной суммы сообщения необходимо знать алгоритм и понимать какие данные из сообщения используются в качестве параметров для функции ее вычисления. В условиях рассматриваемой задачи исходные

тексты целевой программы недоступны и алгоритм вычисления контрольной суммы неизвестен, указанную информацию возможно получить в ходе анализа алгоритмов целевой программы после дизассемблирования или в процессе анализа трассы выполнения программы, однако, как было отмечено ранее, эти подходы трудозатратные и ресурсоемкие.

Существуют подходы, основанные на генетических алгоритмах, позволяющие в ходе фаззинга генерировать сообщения, которые проходят проверку контрольной суммы и могут вызвать аварийное завершение тестируемой программы. Однако одним из основных недостатков этого пути можно назвать высокую вычислительную сложность и, соответственно, значительные временные затраты [7].

Ниже предлагается компромиссный способ, позволяющий решить задачу фаззинг-тестирования с некоторыми ограничениями, не меняющий ранее принятую парадигму о «легковесности» подхода.

Проверки контрольных сумм в бинарном коде реализуются инструкциями JZ или JNZ [8] обойти их можно инвертировав условные переходы, это позволит исключить необходимость подсчета контрольных сумм в процессе генерации входных данных и использовать случайное значение, поскольку пакеты с неверными контрольными суммами будут проходить проверку, а случайная генерация правильных контрольных сумм статистически маловероятна.

Для определения точек проверки контрольных сумм, подлежащих инверсии, можно обратиться к алгоритмам обработки входящих сетевых пакетов. Они подразумевают, что в первую очередь, во избежание выполнения ненужных, а возможно, опасных действий необходимо проверять целостность полученных данных, поэтому в общем случае можно предположить, что одна из первых инструкций условных переходов (JZ или JNZ) после получения сообщения и вычисления его контрольной суммы будет искомой, такое же предположение делается в исследовании [9].

При использовании данного подхода необходимо учитывать, что подтвердить наличие выявленных программных дефектов с использованием полученных последовательностей сообщений при работе с не инструментированной целевой программой напрямую не получится поскольку необходимо вычислить верные контрольные суммы. Для решения данной проблемы в ряде случаев возможно на основе имеющейся спецификации протокола выделять полезную нагрузку из сгенерированных в процессе фаззинга сетевых пакетов и использовать оригинальное клиентское приложение для передачи серверу, при этом валидные контрольные суммы будут вычислены автоматически.

3. Проблемы и перспективы развития инструментов фаззинг-тестирования сетевых программ

Осуществление фаззинг-тестирования сетевых программ связано с рядом проблем, часть из них обусловлена упрощениями алгоритмов и допущениями в процессе анализа программ:

- неверная инструментация инструкций может сломать алгоритм функционирования программы, в результате чего выявленные дефекты не будут проявляться в процессе работы оригинальной программы;
- фаззинг протоколов без обратной связи осложнен отсутствием информации о результатах обработки сообщений, для решения требуется синхронизировать протокольный автомат на стороне инструмента фаззинга и средство контроля целевого приложения в рамках инструментации;
- замедление работы, связанное с инструментацией, может провоцировать обрывы соединений;
- масштабирование фаззинг-тестирования. Основная идея масштабирования заключается в уменьшении времени, затраченного процесс тестирования. Достигаться это может за

счет параллельного выполнения нескольких экземпляров целевой программы или путем выполнения большого количества подключений к одной. Единого рецепта решения данной задачи нет. В некоторых случаях приходится довольствоваться осуществлением тестирования в режиме «one-to-one»;

- выбор стека технологий для разработки инструментов фаззинга. Разработка компонентов фаззера должна осуществляться на языке программирования получившем широкое распространение, обеспечивающем низкую стоимость, низкий порог входа и гибкость. С этой точки зрения в силу его широкого распространения подходит язык программирования C, однако, любое исправление требует перекомпиляции и отладки, что доставит немало неудобств. В случае фаззинга сетевого ПО, а особенно остро этот вопрос встает при исследовании программ, взаимодействующих по протоколу с закрытой спецификацией, необходимо иметь простой способ внесения изменений в исходные тексты и отладки модулей, осуществляющих сетевое взаимодействие с целевым объектом, модификацию данных, а также реализующих протокольный автомат statefull-протоколов. Это обстоятельство подталкивает к использования скриптовых языков, например, Python, несмотря на более низкую производительность программы.

Направления развития инструментов фаззинг-тестирования, в основном, направлены на автоматизацию подготовительного этапа и рутинных операций:

- совершенствование алгоритмов восстановления форматов;
- автоматизированный способ создания оптимальной начальной последовательности входных данных;
- оптимизация средств инструментации;
- «умный» in-memory фаззинг;
- автоматическое построение и расширение протокольного автомата;
- применение методов машинного обучения [10];
- создание коробочных решений и фреймворков.

4. Практические результаты предложенного подхода

В настоящее время исследования по данной тематике находятся в начальной стадии, однако, уже получены некоторые практические результаты. В процессе апробации предложенных подходов к решению задачи легковесного фаззинг-тестирования разработан макет клиента системы DynamoRIO [11] для инструментации сетевых серверных программ.

Реализованные алгоритмы позволяют отслеживать факты подключения клиентов к серверу, после выполнения которых наблюдение за потоком управления и контроль потока данных переносится в порожденные потоки или дочерние процессы, осуществляющие непосредственное взаимодействие между клиентом и сервером. Далее начинается анализ входящего потока согласно предложенной в п. 2.1 схеме, это позволяет в автоматическом режиме контролировать процесс обработки сообщений и выделять их составные блоки и формировать структуру согласно спецификации.

Для проведения экспериментов в качестве целевого объекта был выбран сервер баз данных PostgreSQL. Данная программа представляет интерес, во-первых, потому, что использует гибридный протокол, включающий как бинарную, так и текстовую части. С одной стороны, это существенно упрощает процесс оценки успешности экспериментов, поскольку часть структурных блоков сообщений представляют собой обычный текст, с другой – позволяет сделать предположение о потенциальной возможности анализа бинарных протоколов.

Во-вторых, сервер баз данных PostgreSQL, порождает процессы-потомки для каждого нового подключения и выполнения служебных операций. Это позволяет протестировать возможность переноса контроля потоков управления и данных в дочерние процессы.

Ход эксперимента:

- запуск сервера под инструментацией;
- подключение к серверу с использованием клиентской программы;
- выполнение запроса к базе данных.

Результаты анализа потока данных	Данные (HEX)	Описание
libc_recv(8, 0x7f5418292760, 81)	0000005100030000757365700747669737400646174616261736500706f7374677265730061707063696361746963617461e5f6e616d6500707371636c00636c69656e745f656e636f64696e67005554463800000	Размер сообщения 81 байт
from 0x7f5418292760 + 0 to 0x7fff5d6608bc size 4	00000051	000000Q
from 0x7f5418292760 + 4 to 0x7f541830c5a0 size 77	00030000757365720074766973740064617461626173650070677265737400617070706963617461696f6e5f6e616d65007073716c00636c69656e745f656e636f64696e67005554463800000	00030000user00tvist00database00postgres00application_name00psql00client_encoding00UTF800000
from 0x7f541830c5a0 + 9 to 0x7f5418336568 size 6	747669737400	tvist00
from 0x7f541830c5a0 + 24 to 0x7f5418336580 size 9	706f73746772657300	postgres00
from 0x7f541830c5a0 + 33 to 0x7f5418307700 size 17	6170706c69636174696f6e5f6e616d6500	application_name00
from 0x7f541830c5a0 + 50 to 0x7f54183365f0 size 5	7073716c00	psql00
from 0x7f541830c5a0 + 55 to 0x7f5418336628 size 16	636c69656e745f656e636f64696e6700	client_encoding00
from 0x7f541830c5a0 + 71 to 0x7f5418336668 size 5	5554463800	UTF800
from 0x7f54183365f0 + 0 to 0x7f5418327590 size 5	7073716c00	psql00
from 0x7f5418327590 + 0 to 0x7f5198cd6980 size 4	7073716c	psql
from 0x7f5418336668 + 0 to 0x7f5418327380 size 5	5554463800	UTF800
from 0x7f5418327590 + 0 to 0x7f5198cd6980 size 4	7073716c	psql
from 0x7f5418327590 + 0 to 0x7f5418336bb0 size 5	7073716c00	psql00
from 0x7f5418336bb0 + 0 to 0x7f5418307061 size 5	7073716c00	psql00
from 0x7f5418327380 + 0 to 0x7f5418336bb0 size 5	5554463800	UTF800
from 0x7f5418336bb0 + 0 to 0x7f5418307060 size 5	5554463800	UTF800
from 0x7f5418336bb0 + 0 to 0x7f5418307062 size 3	6f6e00	on00
from 0x7f5418336bb0 + 0 to 0x7f541830705d size 3	6f6e00	on00
from 0x7f5418336bb0 + 0 to 0x7f5418307060 size 5	5554463800	UTF800
from 0x7f5418336bb0 + 0 to 0x7f541830705f size 5	31312e3300	11.300
from 0x7f5418336bb0 + 0 to 0x7f5418307066 size 6	747669737400	tvist00
from 0x7f5418336bb0 + 0 to 0x7f541830706c size 3	6f6e00	on00

Листинг 1. Вывод анализа потока данных в ходе обработки пакета авторизации клиента СУБД
Listing 1. Data flow analysis output during processing of a client authorization package

Вывод результатов анализа отдельных сетевых сообщений с минимальной детализацией (с целью уменьшения его объема), отражающей копирование блоков данных и игнорирующий чтение отдельных байтов, полученный с помощью средства инструментации в ходе эксперимента, приведен в листингах 1 и 2; в столбце «Результаты анализа потока данных» приведены траектории движения блоков данных, выделенных из полученных сообщений; столбец «Данные HEX» отражает данные в шестнадцатеричном виде; столбец «Описание» показывает комментарий или печатные символы, соответствующие шестнадцатеричным

кодам. Результаты разбора тех же пакетов с помощью анализатора Wireshark приведены на рис. 6 и 7.

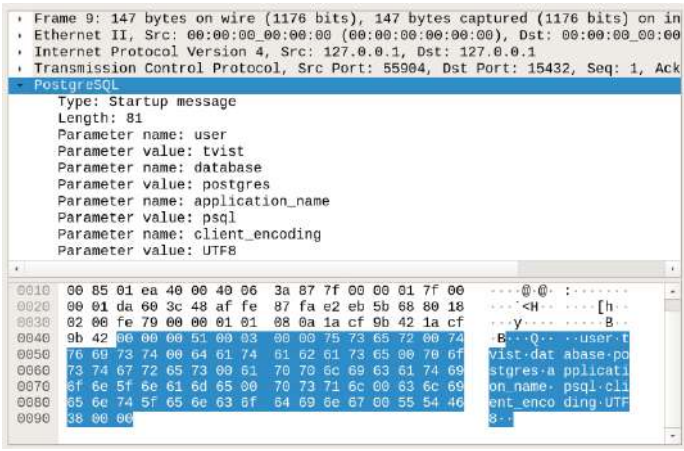


Рис. 6. Декодирование пакета авторизации клиента с помощью Wireshark
Fig. 6. Client authorization packet decoding using Wireshark

Результаты анализа потока данных	Данные (HEX)	Описание
libc_recv(8, 0x7f5418292760, 29)	510000001d73656c656374202a2066726f6d2070675f726f6c65733b00	Получено 29 байт
from 0x7f5418292760 + 1 to 0x7fff5d6605bc size 4	0000001c	0000001c
from 0x7f5418292760 + 5 to 0x7f541830b1c0 size 25	73656c656374202a2066726f6d2070675f726f6c65733b00	select * from pg_roles;00
from 0x7f541830b1c0 + 0 to 0x7f51994af380 size 24	73656c656374202a2066726f6d2070675f726f6c65733b	select * from pg_roles;
from 0x7f541830b1c0 + 0 to 0x7f541830baf0 size 24	73656c656374202a2066726f6d2070675f726f6c65733b	select * from pg_roles;

Листинг 2. Вывод анализа потока данных в ходе обработки пакета с запросом к базе данных
Listing 2. Data flow analysis output during processing of a client request to the database

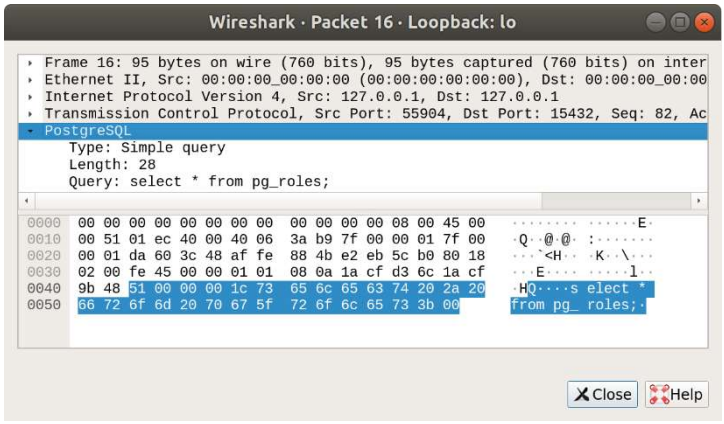


Рис. 7. Декодирование пакета с запросом клиента к базе данных с помощью Wireshark
Fig. 7. Client request to database decoding using Wireshark

Из приведенных листингов в сравнении с результатами декодирования тех же пакетов с использованием Wireshark видно, что определены все основные блоки полученных сервером

сообщений, их смещения относительно начала буфера, в который данные были записаны функцией *recv(…)*, и размеры блоков. Данное описание позволяет сформировать правила модификации данных в ходе фаззинг-тестирования.

Проведенные испытания подтвердили:

- эффективность предложенного подхода к анализу алгоритма сетевых программ на основе выделения блока, ограниченного функциями recv(…) приема и send(…) отправки данных;
- принципиальную возможность автоматического восстановления спецификации протокола в ходе динамического анализа алгоритма;
- возможность аналитическим итеративным путем восстанавливать протокольный автомат.

Ближайшие задачи, поставленные в рамках данной работы, это исследования:

- возможности и разработка методов построения автомата состояний исследуемой программы;
- способов автоматического построения и расширения протокольных автоматов;
- технологических подходов к реализации in-memory фаззинга.

Дальнейшие цели исследований по этой тематике определены в соответствии с перспективными направлениями развития инструментов фаззинг-тестирования.

5. Заключение

Активная цифровизация общества сопряжена с созданием большого количества распределенных автоматизированных систем в различных сферах жизни. Любая из данных систем может стать целью атак, реализуемых за счет уязвимостей входящего в их состав программного обеспечения, выявление которых может осуществляться с использованием инструментов фаззинга. В рамках данного исследования рассматриваются особенности тестирования сетевых приложений в условиях отсутствия исходных текстов, а также предлагается подход к разработке простой с точки зрения архитектуры и эксплуатации системы или фреймворка для решения указанной задачи.

Исследование находится на начальном этапе; в настоящее время определены основные задачи фаззинг-тестирования сетевых приложений, часть которых могут вылиться в отдельные исследования.

Во-первых, восстановление спецификаций и определение типов протоколов (stateful или stateless). В большинстве случаев в ходе решения данной задачи приходится полагаться на решение аналитика. Предложенный способ восстановления спецификации был опробован в ходе тестирования сервера баз данных PostgeSQL. Однако существует формальный способ – контроль отсутствия связей по данным между обработкой пакетов, при этом, что есть связь по данным и способы ее выявления – тема отдельных исследований.

Во-вторых, способ восстановления протокольного автомата. Современные техники подразумевают использование эмпирических подходов. Стоит все же отметить что задача восстановления автомата – тоже отдельная важная тема исследований.

И, наконец, вопрос оценки результатов фаззинга и принятия решения о необходимости продолжения или завершения. В большинстве случаев оценивается покрытие базовых блоков алгоритма программы или реберное покрытие, в рассматриваемом подходе предлагается оценивать «покрытие» спецификации протокола.

Список литературы / References

[1]. Мишечкин М.В., Акользин В.В., Курмангалеев Ш.Ф. Архитектура и функциональные возможности инструмента ИСП Фаззер. Открытая конференция ИСП РАН им. В.П. Иванникова,

2020 г. / Mishechkin M.V., Akolzin V.V., Kurmangaleev Sh.F. Architecture and functionality of the ISP Fuzzer tool. Ivannikov ISP RAS Open Conference, 2020. Slides are available at https://www.ispras.ru/technologies/docs/mishechkin-isprasopen2020.pdf (in Russian).
[2]. MsFontFuzz. Available at https://github.com/Cr4sh/MsFontFuzz, дата обращения 25.07.2021.
[3]. Уязвимости Windows, связанные с обработкой шрифтов / Windows font handling vulnerabilities. Available at blog.cr4.sh/2012/06/0day-windows.html, accessed 25.07.2021 (in Russian).
[4]. Ioctlfuzzer. Available at https://github.com/Cr4sh/ioctlfuzzer, accessed 25.07.2021.
[5]. Trinity. Available at https://github.com/kernelslacker/trinity, accessed 25.07.2021.
[6]. OpenSSL Security Adwisory. Available at https://openssl.org/news/secadw/20210325.txt, accessed 16.05.2021.
[7]. Embleton S., Sparks S. & Cunningham R. Sidewinder: An Evolutionary Guidance System for Malicious Input Crafting. Available at https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Embleton.pdf, accessed 15.08.2021.
[8]. Liu X., Wei Q. et al. CAFA: A Checksum-Aware Fuzzing Assistant Tool for Coverage Improvement, Security and Communication Networks, 2018, Article ID 9071065, 13 p.
[9]. Wang T., Wei T. et al. TaintScope: A Checksum-Aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection. In Proc. of the IEEE Symposium on Security and Privacy, 2010, pp. 497-512.
[10]. Yamaguchi F., Maier A. et al. Automatic Inference of Search Patterns for Taint-Style Vulnerabilities. In Proc. of the IEEE Symposium on Security and Privacy, 2015, pp. 797-812.
[11]. Bruening D., Zhao Q., and Amarasinghe S. Transparent dynamic instrumentation. In Proc. of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE '12), 2012, pp. 133-144.
[12]. AFLNet. Available at https://github.com/aflnet/aflnet, accessed 08.09.2021.

Информация об авторах / Information about authors

Иван Владимирович ШАРКОВ – ведущий инженер отдела компиляторных технологий. Сфера научных интересов: информационные технологии, системное программирование, фаззинг, компьютерные сети, исследования программ.

Ivan Vladimirovich SHARKOV is a leading engineer at the compiler technologies department. His research interests include information technologies, fuzzing, networks, software researching.

Вартан Андроникович ПАДАРЯН – кандидат физико-математических наук, ведущий научный сотрудник отдела технологий ИСП РАН; доцент кафедры системного программирования факультета ВМК МГУ. Его научные интересы включают компиляторные технологии, безопасность ПО, анализ бинарного кода, параллельное программирование, эмуляцию и виртуализацию.

Vartan Andronikovich PADARYAN is a candidate of physical and mathematical sciences, leading researcher at the compiler technologies department of ISP RAS; associate professor of the system programming department of the faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University. His research interests include compiler technologies, software security, binary code analysis, parallel programming, emulation, and virtualization.

Петр Владимирович ХЕНКИН – исполнительный директор - начальник отдела, департамент кибербезопасности ПАО Сбербанк. Сфера научных интересов: информационная безопасность, криптография, анализ исходных текстов, методы аутентификации и идентификации, биометрия.

Petr Vladimirovich KHENKIN is an executive director - head of department, cybersecurity department of Sberbank. His research interests include information security, source codes analyses, authentication, authorization and identification methods, and biometry.

DOI: 10.15514/ISPRAS-2021-33(4)-16

Калибровка k − ε модели турбулентности в пакете OpenFOAM с помощью методов машинного обучения для моделирования потоков на склонах гор на основе эксперимента

Д.И. Романова, ORCID: 0000-0002-5771-4114 <romanovadi@gmail.com>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. В работе проводится калибровка k − ε модели турбулентности для потоков со свободной поверхностью в русле и на склоне. Для калибровки модели проводится эксперимент в наклонном лотке постоянного уклона прямоугольного сечения. В эксперименте с помощью трубки Пито измеряются значения давления в потоке на различном расстоянии от дна, с помощью которых так же получается профиль скорости потока. На основании данных эксперимента с использованием алгоритма оптимизации Нелдера-Мида производится калибровка k − ε модели турбулентности. Откалиброванная модель турбулентности далее применяется для расчёта прорыва ледникового озера, вблизи ледника Малый Азау.

Ключевые слова: математическое моделирование; численное моделирование; снежная лавина; грязекаменный сель; склоновые потоки; OpenFOAM; interFoam; многофазный поток; турбулентный поток; k-ε модель турбулентности; оптимизация; метод Нелдера-Мидаб эксперимент; Малый Азау

Для цитирования: Романова Д.И. Калибровка k − ε модели турбулентности в пакете OpenFOAM с помощью методов машинного обучения для моделирования потоков на склонах гор на основе эксперимента. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 227-240. DOI: 10.15514/ISPRAS–2021–33(4)–16

Благодарности: Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 19-31-90105. Автор выражает благодарность сотрудникам НИИ Механики МГУ Иванову Олегу и Трифонову Владимиру, а также Коровиной Дарье, Гинзбург Нике, и Гинзбург Борису за предоставленные экспериментальные данные.

Experiment based calibration of k − ε turbulence model in OpenFOAM package for mountain slope flows using machine learning techniques

D.I. Romanova, ORCID: 0000-0002-5771-4114 <romanovadi@gmail.com>
Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. We calibrate the k − ε turbulence model for free surface flows in the channel or on the slope. To calibrate the turbulence model, an experiment is carried out in an inclined rectangular research tray. In the experiment, the pressure values in the flow are measured at different distances from the bottom using a Pitot tube; after transforming data, the flow velocity profile is obtained. The k − ε turbulence model is calibrated based on experimental data using the Nelder-Mead optimization algorithm. The calibrated turbulence model is then used to calculate the outburst of a lake near the glacier Maliy Azau on the Elbrus (Central Caucasus).

Keywords: mathematical modeling; numerical modeling; snow avalanche; mudflow; slope flow; OpenFOAM; interFoam; multiphase flow; turbulent flow; k-ε turbulence model; Nelder-Mead optimization; free surface flow; Maliy Azau.

For citation: Romanova D.I. Experiment based calibration of k − ε turbulence model in OpenFOAM package for mountain slope flows using machine learning techniques. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021, pp. 227-240 (in Russian). DOI: 10.15514/ISPRAS-2021-33(4)-16

Acknowledgements. The reported study was funded by RFBR, project number 19-31-90105. The author is grateful to the staff of the Research Institute of Mechanics of Moscow State University, Oleg Ivanov and Vladimir Trifonov, as well as Daria Korovina, Nika Ginzburg, and Boris Ginzburg for the experimental data provided.

1. Введение

Математическое моделирование динамики потоков на горных склонах позволяет прогнозировать опасные зоны, изучать размыв склонов или каналов потоками, а также рассчитывать напряжения, возникающие в защитных сооружениях и других объектах при ударе потока.

Большая часть используемого в настоящее время программного обеспечения для моделирования течений на горных склонах основывается на уравнениях мелкой воды (осреднённые по глубине уравнения механики сплошной среды). Такой подход дает информацию только о границах опасной зоны, толщине потока и средней скорости потока по глубине. Кроме того, осреднённые по глубине уравнения содержат коэффициенты, которые можно найти только при региональной калибровке. Трехмерное моделирование, основанное на полных уравнениях, а не на усредненных по глубине уравнениях, может предоставить полную информацию о распределении параметров в потоке. Это очень важно, особенно, в задаче взаимодействия потока с разными объектами.

Исторически первыми для исследования потоков на склонах гор использовались одномерные осреднённые по глубине модели в силу своей вычислительной простоты. Это такие работы как [2, 5, 11, 12, 14, 18, 34, 35, 39], датирующиеся начиная с 1967 года. Дальнейшим развитием осреднённых по глубине моделей было рассмотрение двумерного подхода [13, 25, 29, 40]. В настоящее время использование осреднённых по глубине уравнений можно видеть в работах [6–8, 17].

Несколько исследовательских групп в мире занимались моделированием потоков на склонах гор с использованием 3D-моделирования [3, 4, 9, 15, 21, 22, 24, 30–34, 37, 41], среди них Ямагучи (Yuya Yamaguchi) и соавторы [41], они использовали метод конечных элементов на

основе подхода Петрова-Галёркина для моделирования склонового потока (снежная лавина) как двухфазного потока (снег и воздух), где снег был представлен бингамовской жидкостью. Кеничи Ода (Kenichi Oda) и соавторы [30] также использовали двухфазный подход для описания динамики снежной лавины, только в отличие от работы [41] система уравнений решалась конечно-разностным методом. В настоящей работе будет использоваться многофазный односкоростной подход для моделирования потоков на склонах гор с использованием метода конечного объёма [31]. Течение считается турбулентным, используется k − ε модель турбулентности. Система уравнений решается методом конечного объёма.

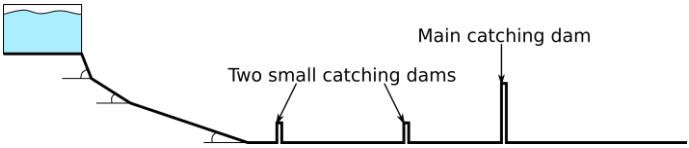


Рис. 1. Схема экспериментального лотка [1]
Fig. 1. Scheme of experimental chute [1]

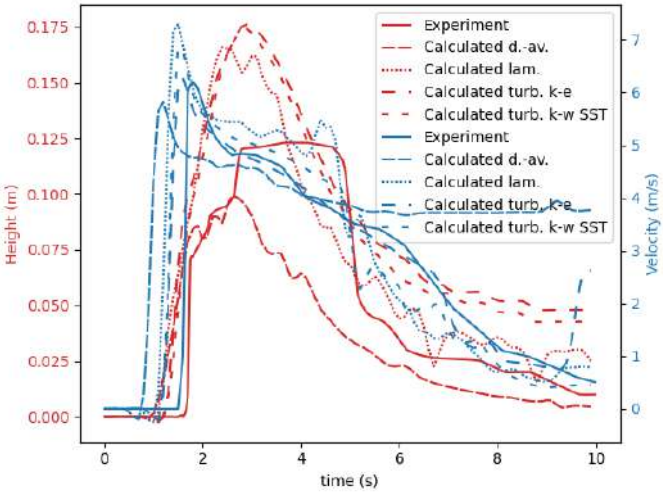


Рис. 2. Графики скорости (синий) и глубины потока (красный), замеренные на расстоянии 11.1 метра от начала установки. Показаны экспериментальные значения из работы [1] (Experiment), вычисленные с использованием осреднённого по глубине подхода (Calculated d.-av.), вычисленные с использованием трёхмерного подхода при ламинарном режиме течения (Calculated lam), вычисленные с использованием k − ε модели турбулентности в трёхмерном подходе (Calculated turb. k-e) и вычисленные с использованием k − ω SST модели турбулентности в трёхмерном подходе (Calculated turb. k-w SST)
Fig. 2. Depth-averaged velocity (blue) and flow depth (red) graphs measured at a distance of 11.1 meters from the start of the installation. Shown are the experimental values from [1] (Experiment), calculated using depth-avaraged approach (Calculated d.-av.), calculated for laminar flow (Calculated lam), calculated using the k − ε turbulence model (Calculated k-e) and calculated using the k − ω SST turbulence model (Calculated k-w)Scheme of experimental chute [1]

2. Актуальность

Потоки на склонах гор являются турбулентными потоками. Существующие модели турбулентности содержат в себе ряд констант, которые были откалиброваны при расчёте канонических течений таких, как обтекания потоком воздуха различных профилей, течения в трубах и каналах и др. [26, 27, 38]. Калибровки моделей турбулентности для расчёта

потоков неньютоновской жидкости со свободной поверхностью на склонах гор не проводилось.

На примере расчёта эксперимента по спуску водоснежного потока в экспериментальном лотке с двумя небольшими защитными дамбами и одной основной удерживающей дамбой (рис. 1), проведённого в университете Исландии [1,23], показана необходимость калибровки турбулентной модели с целью увеличения точности расчёта защитных сооружений (рис. 2 и табл. 1).

Данный эксперимент, поставленный в университете Исландии был спроектирован с целью воспроизвести взаимодействие водоснежного потока с различными препятствиями. Для проведения эксперимента использовалась чистая водопроводная вода, так как она является легко доступным материалом и схожа по плотности с насыщенным водоснежным потоком. В том числе вода позволяет воспроизвести отличия в динамике водоснежных потоков от сжимаемых потоков гранулированных сред с высоким числом Фруда. Эксперимент был спроектирован таким образом, чтобы число Фруда было близко к фактическому числу Фруда для водоснежных потоков: в диапазоне $3 < Fr < 5$. Глубина потока, высота барьеров и другие геометрические характеристики эксперимента моделировались с коэффициентом 10. Эксперимент проводился в длинном деревянном желобе, который состоял из четырех секций с разными углами наклона (55° , 27° , 11.7° , 0° — слева направо), см. рис. 2. Самые нижние 5.05 м желоба были горизонтальными, а верхние его части были покрыты ковром для увеличения трения и снижения скорости потока и, следовательно, числа Фруда для получения целевого диапазона чисел Фруда. Защитные конструкции были выполнены из фанеры.

Табл. 1. Сравнение измеренных и рассчитанных параметров потока
Table. 1. Comparison of measured and calculated flow parameters

Параметр сравнения	**Экспериментальные данные**	**Данные численного расчёта**		
		Ламинарный режим	**k − ε модель турбулентности**	**k − ω SST модель турбулентности**
Высота всплеска на основной дамбе	1.3 м	2.28 м	1.64 м	2.4 м
Время взаимодействия потока с основной дамбой	1.25 с	1.5 с	1.3 с	1.3 с
Объём, удержанный дамбой из 2.7 $м^3$	2.684 $м^3$	2.629 $м^3$	2.528 $м^3$	2.655 $м^3$

Расчёт эксперимента был проведён с использованием многофазного односкоростного подхода, реализованного в пакете OpenFOAM в решателе interFoam. На рис. 2 видно, что расчёты с использованием турбулентных моделей лучше воспроизводят эксперимент нежели ламинарный режим течения, незначительное преимущество у k − ε модели турбулентности. Однако по результатам сравнении параметров в таблице 1 вытекает, что оба турбулентных расчёта требуют доработки.

В данной работе с помощью методов машинного обучения производится калибровка k − ε модели турбулентности для такого класса задач, как потоки со свободной поверхностью на склоне под действием силы тяжести.

3. Калибровочный эксперимент

Для калибровки коэффициентов k − ε модели турбулентности используется эксперимент, поставленный в НИИ Механики МГУ. В процессе эксперимента в лотке постоянного уклона (эксперимент проведён для нескольких разных углов наклона) производился спуск турбулентного потока жидкости. Использовался лоток следующей геометрии: длина — 1 м, ширина — 12 см, высота бортиков — 10 см.

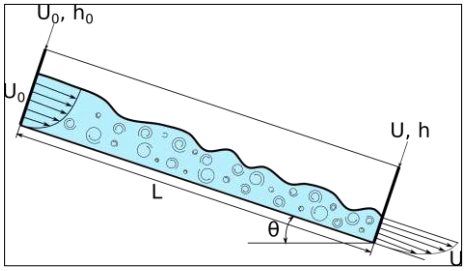


Рис. 4. Схема экспериментального лотка
Fig. 4. Experiment chute scheme

В расчётах использовалась область лотка между двумя точками замера скорости и глубины, находящимися на расстоянии 23 см и 82 см от верхнего края лотка. Схема эксперимента показана на рис. 4. Для проведения эксперимента используется водопроводная вода, как в эксперименте университета Исландии. Лоток выполнен полностью из акрилового стекла толщиной 4 мм. Вода подаётся из резервуара с возможностью регулировки высоты столба жидкости при подаче. При входе в исследуемый участок лотка установлен успокоитель потока. Эксперимент проводится в стационарном режиме с замкнутым жидкостным контуром. Стационарность обеспечивает насос перекачки жидкости из нижнего водосборника в верхний резервуар. Для замера профиля скорости потока используется трубка Пито, присоединённая к датчику давления.

Было проведено 3 серии экспериментов, в которых менялся угол наклона склона, начальная глубина потока и начальный профиль потока, как показано в табл. 2.

Табл. 2. Параметры расчётов
Table. 2. Experiments' parameters

u0 среднее по глубине, м/с	***h0 , мм***	***θ***
1.63	4.20	25°
2.00	4.95	28°
1.78	3.45	33°

4. Математическая модель

Используется трёхмерный подход для моделирования потока. При этом подходе поток рассматривается как многофазное течение, осреднение по глубине не используется. Одна из фаз — воздух, другая — материал потока (снег или грязекаменная смесь), третьей фазой может быть материал подстилающей поверхности. В данной работе рассматривается двухфазная модель — поток-воздух. Используется $k - \varepsilon$ модель турбулентности, основанная на работах [27], [38].

В работе используется метод VOF (объём жидкости) для отслеживания границы свободной поверхности, который был предложен Хиртом (C.W. Hirt) и Николсом (B.D. Nichols) в 1981

году [20]. Данный метод не отслеживает границу явно, она задаётся как пороговое значение для объёмной доли фазы.

Выпишем систему уравнений для описания двухфазной модели течения, в которой каждая из фаз считается несжимаемой и обе фазы имеют единую скорость (1). Данная модель состоит из следующих уравнений: уравнения неразрывности для смеси, уравнения переноса объёмной доли одной из фаз, уравнения сохранения количества движения, уравнения для расчёта турбулентной кинетической энергии и уравнения диссипации турбулентной кинетической энергии, а также замыканий для тензора напряжений, являющегося функцией эффективной вязкости и тензора скоростей деформаций, эффективная вязкость является суммой молекулярной и турбулентной вязкостей, а турбулентная вязкость представлена функцией турбулентной кинетической энергии и её диссипации, плотность и вязкость смеси в модели вычисляются по принципу весового среднего.

$$\begin{cases} \nabla \cdot \overline{u} = 0, \\ \dfrac{\partial \alpha}{\partial t} + \nabla \cdot (\overline{u}\alpha) = 0, \\ \dfrac{\partial (\rho \overline{u})}{\partial t} + \nabla \cdot (\rho \overline{u}\overline{u}) = -\nabla \overline{p} + \nabla \cdot \overline{\tau} + \rho \overline{f}, \\ \dfrac{\partial (\rho k)}{\partial t} + \nabla \cdot (\rho \overline{u} k) = \nabla \cdot (\mu \nabla k) - \nabla \cdot \left(\dfrac{\mu_t}{\sigma_k} \nabla k + P_k \right) - \rho \varepsilon, \\ \dfrac{\partial (\rho \varepsilon)}{\partial t} + \nabla \cdot (\rho \overline{u} \varepsilon) = C_{\varepsilon 1} P_k \dfrac{\varepsilon}{k} - \rho C_{\varepsilon 2} \dfrac{\varepsilon^2}{k} + \nabla \cdot \left(\dfrac{\mu_t}{\sigma_\varepsilon} \nabla \varepsilon \right), \\ \overline{\tau} = 2\mu_{eff} \overline{s}, \overline{s} = 0.5[\nabla \overline{u} + (\nabla \overline{u})^T], \mu_{eff} = \mu + \mu_t, \mu_t = \rho C_\mu k^2 / \varepsilon, \\ \rho = \rho_1 \alpha + \rho_0 (1 - \alpha), \mu = \nu \rho, \nu = \nu_1 \alpha + \nu_0 (1 - \alpha). \end{cases} \quad (1)$$

Здесь $\overline{u}$ — скорость смеси, горизонтальной чертой над буквами обозначается осреднение по Рейнольдсу; α — объёмная доля выбранной фазы; ρ — плотность смеси; $\overline{p}$ — давление; $\overline{f}$ — плотность массовых сил; k — плотность турбулентной кинетической энергии; ε — диссипация турбулентной кинетической энергии; P_k — скорость производства турбулентной кинетической энергии средним течением, $C_{\varepsilon 1}$, $C_{\varepsilon 2}$, C_μ, σ_k, σ_ε — коэффициенты модели турбулентности.

Объёмная доля фазы принимает значения в диапазоне $0 \le \alpha \le 1$. В случае, например, если $\alpha = 0$ в ячейке, то она полностью заполнена фазой 0, или, если $\alpha = 1$ в ячейке, то она полностью заполнена фазой 1.

Более детальное описание математической модели можно найти в книге Ферцигера (Joel Ferziger) и Перича (Milovan Peric) [16].

5. Вычислительная модель

Для решения гидродинамической задачи течения жидкости на склоне используется свободно распространяемый пакет с открытым исходным кодом OpenFOAM.

5.1. Расчётная область для эксперимента НИИ Механики МГУ

Для расчёта эксперимента НИИ Механики МГУ была рассчитана часть экспериментального лотка, находящаяся между двумя точками замера профиля скорости и глубины потока. Расчёт проводился для срединной полоски лотка толщиной 10 мм, где влияние боковых стенок незначительно. Первый измеренный профиль подавался на вход расчётной области, второй являлся объектом сравнения. Геометрия расчётной области представляет собой параллелепипед длинной 590 мм, шириной 10 мм, и глубиной 30 мм. Количество ячеек составило 590x10x60.

Были выделены следующие границы расчётной области: дно лотка, боковые стенки лотка, верхняя граница лотка, плоскость на входе в лоток, плоскость на выходе из лотка.

Были заданы следующие граничные условия:

- дно лотка: для скорости задано условие прилипания, для объёмной доли фазы –нулевой градиент, для турбулентной кинетической энергии установлено условие нулевого градиента, для диссипации турбулентной кинетической энергии установлена константа 0.1 $м^2/с^3$;
- боковые стенки лотка: для всех рассматриваемых величин задано условие нулевого градиента;
- верхняя граница лотка: смешанное условие с заданием атмосферного давления, и условия отсутствия притока среды через данную границу, отток происходит по принципу нулевого градиента;
- входная плоскость: заданы фиксированные значения объёмной доли фазы и профиля скорости потока, для других величин установлен свободный вход с условием нулевого градиента, позволяющий с течением времени установиться исследуемым параметрам потока;
- выходная плоскость: для всех величин установлено условие нулевого градиента.

Начальные условия в задаче таковы, что объём полностью заполнен неподвижным воздухом и подаётся жидкость через входную плоскость, спустя время поток устанавливается и снимаются замеры на выходной плоскости для сравнения с экспериментальными данными. Поток считается установившимся спустя 5 секунд.

5.2. Расчётная область для эксперимента, проведённого в университете Исландии

Для расчёта эксперимента, поставленного в университете Исландии использовалась двумерная расчётная область. Были выделены следующие границы:

- дно лотка, включающее в себя защитные сооружения, границу окончания экспериментальной установки и борта резервуара с жидкостью, для которых заданы аналогичные с экспериментом НИИ Механики МГУ граничные условия;
- верхняя граница экспериментальной установки, с соответствующими граничными условиями, описанными в предыдущем разделе;
- боковые границы расчётной области, заданные как нерасчётные для реализации двумерной постановки.

Всего расчётная область состоит из 257 тысяч ячеек. Начальные условия заданы таким образом, что вода, глубиной 0.9м располагается в стартовом резервуаре и удерживается виртуальной границей, исчезающий с началом расчёта.

6. Алгоритм оптимизации

Необходимость и эффективность калибровки турбулентных моделей продемонстрированы в таких работах, как [19,28]. Работа учёных Сержа Гийяса (Serge Guillas), Нины Гловер (Nina Glover) и Лиоры Малки-Эпштейн (Liora Malki-Epshtein) [19] демонстрирует эффективность применения калибровки с помощью байесовских методов для коэффициентов k − ε модели турбулентности при моделировании воздушных потоков в городе. В работе американских учёных Джулии Линг (Julia Ling), Эндрю Курзавски (Andrew Kurzawski) и Джереми Темплтона (Jeremy Templeton) [28] авторы используют глубокие нейронные сети для получения замыкающего члена для расчёта напряжений Рейнольдса, учитывающего анизотропию турбулентности потока жидкости в канале. В применении к задаче моделирования динамики снежной лавины или грязе-каменного селя точность моделирования динамики турбулентного потока имеет сильное влияние на расчёт защитного

сооружения. Как было показано в табл. 1, различие экспериментальных и расчётных параметров может достигать 25%.

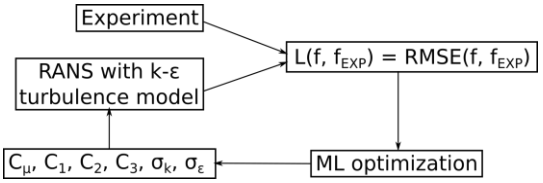


Рис. 3. Архитектура предлагаемого алгоритма для калибровки коэффициентов k − ε модели турбулентности

Fig. 3. The architecture of the proposed algorithm for calibrating the k − ε turbulence model coefficients

k − ε модель турбулентности содержит ряд констант: $C_\mu = 0.09$; $C_1 = 1.44$; $C_2 = 1.92$; $C_3 = 0.0$; $\sigma_k = 1.0$; $\sigma_\varepsilon = 1.3$. Использование данной турбулентной модели для расчёта потоков неньютоновских жидкостей на склонах требует калибровки коэффициентов модели. Для организации процесса калибровки используются алгоритмы машинного обучения.

Калибровка турбулентной модели проводилась с использованием следующего алгоритма, основанного на обучении с подкреплением, показанного на рис. 3:

1. Обучение на базе ряда расчётов, проведённых с использованием RANS модели с k−ε моделью турбулентности с различными значениями констант;
2. Получение новых значений коэффициентов турбулентной модели посредством алгоритма оптимизации;
3. Расчёт гидродинамики потока с использованием турбулентной модели с коэффициентами, полученными с помощью алгоритма оптимизации;
4. Дообучение алгоритма с использованием полученных данных расчёта гидродинамики потока.

При оптимизации коэффициентов турбулентной модели минимизируется корень из среднеквадратичного отклонения вычисленного профиля скорости потока на выходной плоскости от экспериментального профиля:

$$L_{RMSE} = \sqrt{\frac{\sum_{z=0}^{h}\left(v_{exp}(z) - v_{k-\varepsilon}(z)\right)^2}{h}}, \qquad (2)$$

где h — глубина потока на выходной плоскости.

Для минимизации используется метод Нелдера–Мида (метод безусловной оптимизации функции от нескольких переменных, не использующий градиентов), реализованный в библиотеке SciPy.

Для реализации приведённого выше алгоритма было разработан модуль, который позволил в автоматическом режиме связать OpenFOAM расчёты с алгоритмом оптимизации. В результате в процессе оптимизации генерировались обновлённые значения коэффициентов турбулентной модели, которые передавались в утилиту препроцессинга, готовящую расчётные кейсы. Далее запускались расчёты OpenFOAM кейсов в параллельном режиме, по результатам расчётов рассчитывалось значение loss-функции и передавалось в оптимизатор.

7. Результаты калибровки

Для оптимизации коэффициентов k − ε модели турбулентности было проведено 180 расчётов калибровочного эксперимента (рис. 5), что заняло 213 часов работы на 24 ядрах.

Начальные значения коэффициентов были заданы следующими:

$$C_\mu = 0.09;\ \ C_1 = 1.44;\ \ C_2 = 1.92;\ \ C_3 = 0.0;\ \ \sigma_k = 1.0;\ \ \sigma_\varepsilon = 1.3. \qquad (3)$$

После калибровки значения коэффициентов стали следующими:

$$C_\mu = 0.0902;\ \ C_1 = 1.5419;\ \ C_2 = 1.92;\ \ C_3 = 0.0001;\ \ \sigma_k = 1.0215;\ \sigma_\varepsilon = 1.3334. \quad (4)$$

Были получены профили скорости на выходной плоскости для различных углов наклона лотка, как показано на рис. 5

Калибровка привела к минимизации функции потерь (5), показанной в табл. 3

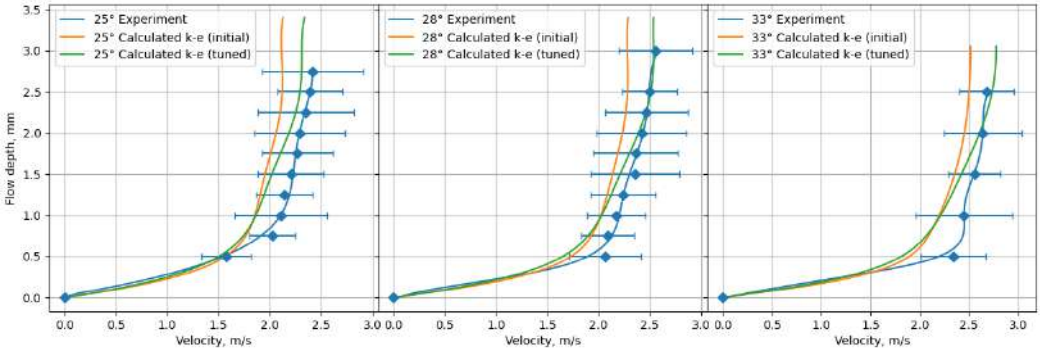


Рис. 5. Сравнение экспериментального профиля скорости с расчётным при использовании стандартных значений коэффициентов k − ε модели турбулентности и откалиброванных значений коэффициентов в лотках различного угла наклона к горизонту

Fig. 5. Comparison of the experimental velocity profile with the calculated one using the standard values of the k − ε turbulence model coefficients and the calculated velocity profile with calibrated values of the coefficients for different slope inclination angles (to the horizon)

Табл. 3. Минимизация функции потерь
Table 3. Loss function minimization

Угол наклона лотка	**Начальное значение функции потерь**	**Минимизированное значение функции потерь**
25°	0.242	0.218
28°	0.377	0.389
33°	0.236	0.168

8. Верификация на эксперименте университета Исландии

Был произведён расчёт эксперимента университета Исландии, показанного на рис. 1, с использованием откалиброванных коэффициентов k − ε модели турбулентности. Были получены результаты, показанные на рис. 6 и в табл. 4.

На рис. 6 показано, что для глубины потока уменьшилось различие экспериментальных данных от расчётных после калибровки коэффициентов турбулентной модели. Для параметров, перечисленных в таблице 4 можем видеть уменьшение расхождения для высоты первого всплеска на основной дамбе на 41%, для объёма, удержанного дамбой — на 60%. Из вышесказанного можно сделать вывод, что калибровка коэффициентов турбулентной модели для исследуемого класса задач увеличила точность расчётов. Однако, не будет лишней дальнейшая калибровка турбулентной модели для учёта анизотропии турбулентности, которая имеет существенное значение при исследовании потоков со свободной поверхностью на склонах.

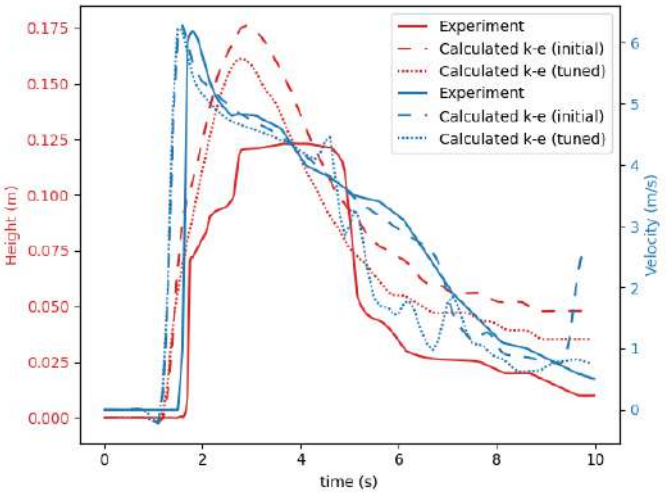


Рис. 6. Графики скорости (синий) и глубины потока (красный), замеренные на расстоянии 11.1 метра от начала установки. Показаны экспериментальные значения из [1] (Experiment), вычисленные с использованием k − ε модели турбулентности (Calculated k-e (initial)) с исходными значениями коэффициентов и вычисленные с использованием k − ε модели турбулентности (Calculated k-e (tuned)) с откалиброванными коэффициентами

Fig. 6. Depth-averaged velocity (blue) and flow depth (red) graphs measured at a distance of 11.1 meters from the start of the installation. Shown are the experimental values from [1] (Experiment), calculated using the k − ε turbulence model (Calculated k-e) with initial coefficients and calculated using the k − ε turbulence model (Calculated k-e tuned) with calibrated coefficients

Табл. 4. Сравнение измеренных и рассчитанных параметров потока
Table. 4. Loss function minimization

Параметр сравнения	**Эксперимен-тальные данные**	**k − ε модель турбулентности с исходными коэффициентами**	**k − ε модель турбулентности с откалиброванными коэффициентами**
Высота первого всплеска на основной дамбе	1.3 м	1.64 м	1.5 м
Время с начала взаимодействия потока с основной дамбой, до окончания переливания потока через дамбу	1.25 с	1.3 с	1.3 с
Объём, удержанный дамбой из 2.7 м^3	2.684 м^3	2.528 м^3	2.622 м^3

9. Расчёт возможного прорыва озёр вблизи ледника Малый Азау

С помощью полученной откалиброванной модели турбулентности произведём расчёт возможного прорыва ледникового горного озера вблизи ледника Малый Азау [10].

Группа озёр, расположенная около ледника Малый Азау на южном склоне Эльбруса, широко известна и ежегодно посещается туристами и альпинистами. В период 1957– 2021 гг. зафиксировано два случая прорывов озёр: озеро западное в 1978 г. и озеро восточное в 2011 г. Первый прорыв связан с процессами оползневых деформаций моренного массива,

слагающего часть озёрной котловины, второй — с деградацией ледяной плотины озера и переливом воды поверх неё. Существующие в настоящее время моренные плотины озёр и участки поверхностного стока воды из озёр находятся в стабильном состоянии, поэтому угроза прорыва озёр отсутствует. Потенциальная угроза прорыва остаётся в связи с высокой сейсмичностью и возможной вулканической деятельностью в этом районе.

Разработанное ПО позволяет получить точные данные о пути движения паводка, его профиле скорости, распределении давления в потоке, напряжении на дне потока в различных точках.

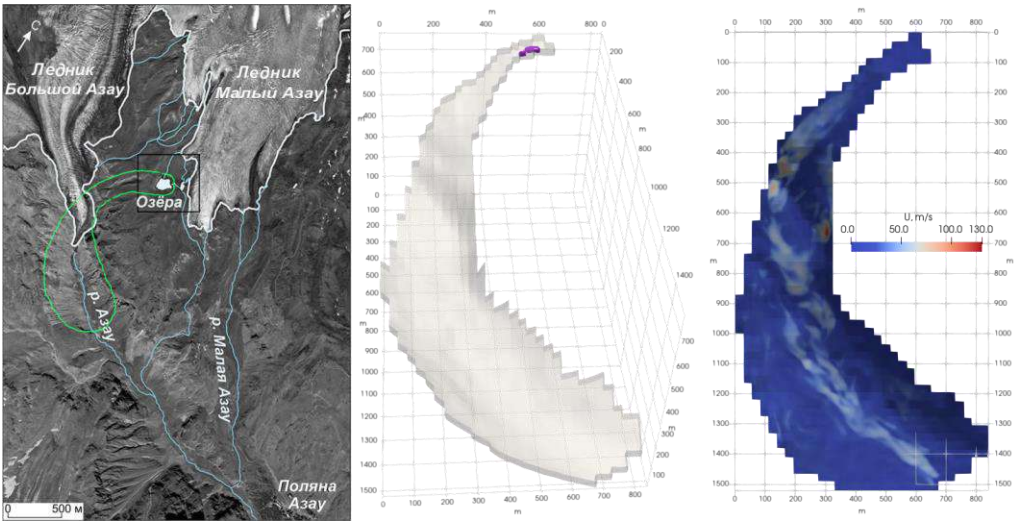


Рис. 7. Карта рельефа исследуемой местности (слева) [10]; трёхмерная расчётная область в начальный момент времени с начальным расположением озера (центр); картина потока в момент времени 100 минут от начала прорыва (справа)

Fig. 7. Maliy Azau glacial lake outburst flood map (left) [10]; three-dimensional computational domain at the initial time moment with the initial location of the lake (center); outburst flood at 100 minutes from the start (right)

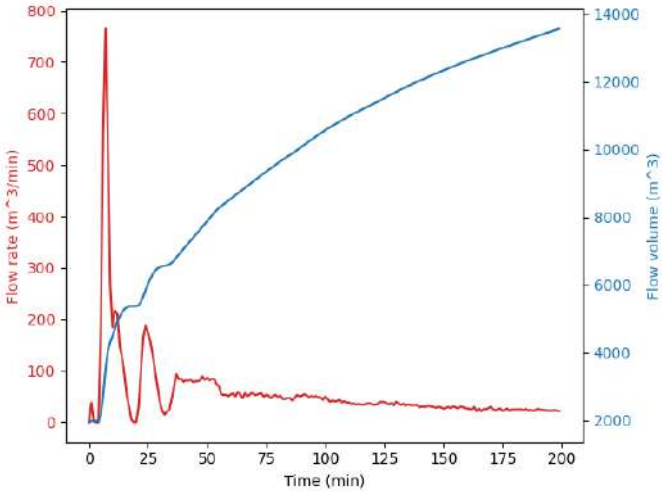


Рис. 8. Графики расхода жидкости (красный) и объёма жидкости слившегося с озера (синий)

Fig. 8. Graphs of the flow rate (red) and the volume of liquid poured out of the lake (blue)

В результате математического моделирования прорыва озера глубины 2 метра были получены картины течения, как показано на рис. 7. Слева на рис. 7 показана карта рельефа

исследуемой области, зелёным цветом схематически отмечен контур расчётной области; в центре рис. 7 показана расчётная область в начальный момент времени, фиолетовым цветом отмечена вода в озере; справа на рис. 7 показано распределение скорости потока спустя 100 минут после прорыва.

На рис. 8 показано, что прорыв озера продолжается более 200 минут, максимальный расход жидкости составляет порядка 760 м^3 /мин, средний — 58 м^3 /мин. Также на рис. 8 мы видим всплеск расхода жидкости сразу после прорыва, характерный для такого типа течений и дальнейшее выравнивание потока.

10. Результаты и выводы

В процессе работы был проведён эксперимент, по результатам которого была откалибрована k −ε модель турбулентности, произведён расчёт возможного прорыва ледникового горного озера вблизи ледника Малый Азау (Кавказ).

- Было получено уменьшение расхождения на обучающем эксперименте до 29%.
- При оптимизации коэффициентов турбулентной модели использовались три эксперимента, что позволило избежать переобучения алгоритма.
- При использовании турбулентной модели с откалиброванными коэффициентами при моделировании тестового эксперимента Университета Исландии получено уменьшение расхождения расчётных характеристик с экспериментальными.
- При использовании турбулентной модели с откалиброванными коэффициентами при моделировании тестового эксперимента Университета Исландии получено уменьшение расхождения расчётного объёма, удержанного комбинацией защитных сооружений и экспериментального на 60%.
- С использованием откалиброванной модели турбулентности было проведено моделирование возможного прорыва ледникового горного озера вблизи ледника Малый Азау (Кавказ).

Список литературы / References

[1]. Katrin Helga Agustsdottir. The design of slushflow barriers: Laboratory experiments. PhD thesis, Faculty of Industrial Engineering, Mechanical Engineering, and Computer Science, University of Iceland, 2019.

[2]. Бахвалов Н.С., Эглит М.Э. Исследование решений уравнения движения снежных лавин. Материалы гляциологических исследований. Хроника обсуждений, вып. 16, 1970 г., стр. 31-38 / N.S. Bakhvalov, M.E. Eglit. Investigation of the solutions to snow avalanche movement equations. Glaciological data. GD-16, 1984, pp. 117-128.

[3]. Eloise Bovet, Bernardino Chiaia, Luigi Preziosi. A new model for snow avalanche dynamics based on bingham fluids. Meccanica, vol. 45, 2010, pp. 753-765.

[4]. W. Brandstatter, F. Hagen et al. Dreidimensionalle simulation von staublawinen unter berucksichtigung realer gelandeformen. Zeitschrift der Wildbach- und Lawinebnverbauung Osterreichs, 120, 1992, pp. 107-137 (in German).

[5]. A. Briukhanov, S. Grigorian et al. On some new approaches to the dynamics of snow avalanches. In Proc. of the Conference on Physics of Snow and Ice, 1966, pp, 1223-1241.

[6]. Yves Bühler, Marc Christen et al. Sensitivity of snow avalanche simulations to digital elevation model quality and resolution. Annals of Glaciology, vol. 52, 2011, pp. 72-80.

[7]. M. Christen, J. Kowalski et al. Numerical simulation of dense snow avalanches in three-dimensional terrain. Cold Regions Science and Technology, vol. 63, issue 1, 2010, pp. 1-14.

[8]. Marc Christen, Perry Bartelt et al. Calculation of Dense Snow Avalanches in Three-Dimensional Terrain with the Numerical Simulation Program Ramm. In Proc. of the Whistler 2008 International Snow Science Workshop, 2008, pp. 709-716.

[9]. J. Dent, T. Lang. Modeling of snow flow. Glaciology, vol. 26, 1980, pp. 131-140.

[10]. M.D Dokukin, A Khatkutov. Lakes near the glacier maliy azau on the elbrus (central caucasus): dynamics and outbursts. Ice and Snow, vol. 56, 2016, pp. 472-479.

[11]. М.Э. Эглит. Теоретические подходы к расчету движения снежных лавин. В сборнике Итоги науки. Гидрология суши, гляциология. М., ВИНИТИ, 1968, стр. 60-98 / M. Eglit. Theoretical approaches to the calculation of the motion of snow avalanches. Glaciological data. GD-16, 1984, pp. 63-118.

[12]. М.Э. Эглит. Вычисление параметров лавин в зонах торможения и остановки. Материалы гляциологических исследований, вып. 43, 1982 г., стр. 35-39 / M. Eglit. Calculation of the parameters of avalanches in the runout zone. Materialy Glyatsiologicheskikh Issledovaniy, issue 53, 1982, pp. 35-39 (in Russian).

[13]. М.Э. Эглит. Неустановившиеся движения в руслах и на склонах. Изд-во МГУ, 1986 г., 96 стр. / M. Eglit. Unsteady motions in channels and on slopes. Moscow State University, 1986, 96 p. (in Russian).

[14]. Margarita Eglit. Some mathematical models of snow avalanches. In Advances in the Mechanics and the Flow of Granular Materials, vol. 2, 1983, pp. 557-588.

[15]. M.E. Egli, A.E. Yakubenko. Numerical modeling of slope flows entraining bottom material. Cold Regions Science and Technology, vol. 108, 2014, pp. 139-148.

[16]. Joel Ferzige , Milovan Peric. Computational Methods for Fluid Dynamics, 3rd ed. Springer, 2001, 426 p.

[17]. Jan-Thomas Fischer, Julia Kowalski et al. Dynamic avalanche modeling in natural terrain. In Proc. of the International Snow Science Workshop, 2009, pp. 448-453.

[18]. С.С. Григорян, М.Э. Эглит, Ю.Л. Якимов. Новая математическая постановка задачи о движении лавины и решение этой задачи. Труды Высокогорного геофизического института, no. 12, 1967 г., стр. 104-113 / S.S. Grigorian, M.E. Eglit, Y.L. Iakimov. A new formulation and solution of the problem of snow avalanche motion. Trudy Vycokogornogo Geofizicheskogo Instituta, no. 12, 1967, pp. 104-113 (in Russian).

[19]. Serge Guillas, Nina Glover, Liora Malki-Epshtein. Bayesian calibration of the constants of the k − ε turbulence model for a cfd model of street canyon flow. Computer Methods in Applied Mechanics and Engineering, vol. 279, 2014, pp. 536-553.

[20]. C.W. Hirt, B.D. Nichols. Volume of fluid (vof) method for the dynamics of free boundaries. Journal of Computational Physics, vol. 39, issue 1, 1981, pp. 201–225.

[21]. Dieter Issler. Modelling of snow entrainment and deposition in powder-snow avalanches. Annals of Glaciology, vol. 26, 1998, pp. 253-258.

[22]. Dieter Issler, Manuel Pastor Pérez. Interplay of entrainment and rheology in snow avalanches: a numerical study. Annals of Glaciology, vol. 52, issue 58, 2011, pp. 143-147.

[23]. Rebecca Anne Jones. The Design of Slushflow Barriers: CFD Simulations. PhD thesis, Faculty of Industrial Engineering, Mechanical Engineering, and Computer Science, University of Iceland, 2019.

[24]. M.A. Kern, F. Tiefenbacher, J.N. McElwaine. The rheology of snow in large chute flows. Cold Regions Science and Technology, vol. 39, issue 2-3, 2004, pp. 181-192.

[25]. A.G. Kulikovskii, M.E. Eglit. Two-dimensional problem of the motion of a snow avalanche along a slope with smoothly changing properties. Journal of Applied Mathematics and Mechanics, vol. 37, issue 5, 1973, pp. 792-803.

[26]. A. Morse, B.E. Launder et al. Prediction of free shearflows - a comparison of the performance of six turbulence models. In Proc. of the of NASA Conference on Free Shear Flows, 1972, pp. 361-426.

[27]. Brian Launder, D.B. Spalding. The numerical computation of turbulent flows. Computer Methods in Applied Mechanics and Engineering, vol. 3, issue 2, 1974, pp. 269-289.

[28]. Julia Ling, Andrew Kurzawski, Jeremy Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. Journal of Fluid Mechanics, vol. 807, 2016, pp. 155-166.

[29]. Е.М. Миронова. Математическое моделирование движения водных потоков, снежных лавин и селей. Диссертация на соискание ученой степени кандидата физико-математических наук. МГУ, 1987/ E. Mironova. Mathematical Modelling of the Motion of Water Flows, Snow Avalanches, and Floods. PhD thesis, Lomonosov Moscow State University, 1987 (in Russian).

[30]. Kenichi Oda, Shuji Moriguchi et al. Simulation of a snow avalanche model test using computational fluid dynamics. Annals of Glaciology, vol. 52, issue 58, 2011, pp. 57-64.

[31]. Д.И. Романова Трёхмерное моделирование потоков жидкости Хершеля-Балкли на склоне в OpenFOAM. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 85-100 / Romanova D.I. 3D flow modeling of Herschel-Bulkley fluid on the slope in OpenFOAM. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 1, 2017, pp. 85-100 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-6.

[32]. Daria Romanova. Comparison of single-velocity and multi-velocity multiphase models for slope flow simulations. In Proc. of the 2020 Ivannikov Ispras Open Conference (ISPRAS), 2020, pp. 170–174.

[33]. Д.И. Романова. Архитектура программного средства с открытым исходным кодом для численного моделирования потоков на горных склонах. Труды ИСП РАН, том 32, вып. 6, 2020 г., стр. 183-200

/ D.I. Romanova. Design of open source software architecture for numerical modeling of flows on mountain slopes. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 6, 2020, pp. 183-200 (in Russian). DOI: 10.15514/ISPRAS-2020-32(6)-14.

[34]. P. Sampl. Current status of the avl avalanche simulation model—numerical simulation of dry snow avalanches. In Proc. of the "Pierre Beghin" International Workshop on Rapid Gravitational Mass Movements, 1993, pp. 269-296.

[35]. S.B. Savage, K. Hutter. The motion of a finite mass of granular material down a rough incline. Journal of Fluid Mechanics, vol. 199, 1989, pp. 177-215.

[36]. S. B. Savage, K. Hutter. The dynamics of avalanches of granular materials from initiation to runout. part i: Analysis. Acta Mechanica, vol. 86, issue 1, 1991, pp. 201-223.

[37]. Thomas Scheiwiller. Dynamics of powder-snow avalanches. PhD thesis, ETH Zurich, 1986.

[38]. Sherif H. El Tahry. k-epsilon equation for compressible reciprocating engine flows. Journal of Energy, vol. 7, issue 4, 1983, pp. 345-353.

[39]. A. Voellmy, A. Roch. Über die Zerstörungskraft von Lawinen. Schweizerische Bauzeitung – Wochenschrift für Architektur, Ingenieurwesen und Maschinentechnik, 73, 1955 (in German).

[40]. Н.А. Володичева, Е.М. Миронова др. Использование математического моделирования для определения границ распространения лавин. Материалы гляциологических исследований, вып. 56, 1986 г., стр. 78-81 / N. Volodicheva, E. Mironova et al. The use of mathematical modelling to determine the boundaries of propagation of avalanches. Materialy Glyatsiologicheskikh Issledovaniy, issue 56, 1986, pp. 78-81 (in Russian).

[41]. Yuya Yamaguchi, Shinsuke Takase et al. Three-dimensional nonstructural finite element analysis of snow avalanche using non-newtonian fluid model. Transactions of the Japan Society for Computational Engineering and Science, 2017, Paper No. 20170011 (in Japanese).

Информация об авторах / Information about authors

Дарья Игоревна РОМАНОВА получила степень магистра на кафедре гидромеханики механико-математического факультета МГУ в 2017 году, в настоящее является младшим научным сотрудником лаборатории вычислительных методов механико-математического факультета МГУ и стажёром-исследователем ИСП РАН. Она является разработчиком программного обеспечения с открытым исходным кодом для численного моделирования задач механики сплошных сред, включая турбулентные течения, течения со свободной поверхностью, многофазные течения с фазовым переходом и включениями твердых частиц.

Daria Igorevna ROMANOVA received her master's degree at the Department of Hydromechanics, Faculty of Mechanics and Mathematics, Moscow State University in 2017, and is currently a junior researcher at the Laboratory of Computational Methods at the Faculty of Mechanics and Mathematics of Moscow State University and an intern-researcher at ISP RAS. She is an open source software developer for the numerical modeling of continuum mechanics problems, including turbulent flows, free surface flows, multiphase flows with a phase transition, and solid inclusions.