DOI: 10.15514/ISPRAS-2023-35(4)-2

Вызовы в реализации систем глубокого анализа сетевого трафика методом полного протокольного декодирования

[1] *Р.Е. Пономаренко, ORCID: 0009-0009-5741-3627 <rerandom@ispras.ru>*
[1] *В.И. Егоров, ORCID: 0009-0001-5168-6474 <unclehook@ispras.ru>*
[1,2,3,4] *А.И. Гетьман, ORCID: 0000-0002-6562-9008 <ever@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*
[2] *Московский физико-технический институт, 141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*
[3] *Национальный исследовательский университет «Высшая школа экономики», 101978, Россия, г. Москва, ул. Мясницкая, д. 20*
[4] *Московский государственный университет имени М.В. Ломоносова 119991, Россия, г. Москва, Ленинские горы, д. 1*

Аннотация. В данной статье описываются проблемы, возникающие при реализации инструментов глубокого анализа сетевого трафика методом полного протокольного декодирования. Описываемые проблемы условно делятся на две группы. Первая группа проблем связана с основополагающими задачами, которые необходимо решить при реализации систем полного протокольного декодирования. В частности, важно обеспечить корректный разбор протоколов, что включает в себя правильное определение и интерпретацию заголовков и полей протоколов. Также требуется обеспечить обработку фрагментированных пакетов и сборку фрагментов в исходное сообщение. Важной задачей является также обработка и анализ зашифрованного трафика, что может потребовать использования специальных алгоритмов и инструментов. Вторая группа проблем связана с оптимизацией процесса полного протокольного декодирования для обеспечения высокой скорости обработки трафика, а также с поддержкой новых протоколов и возможностью добавления пользовательских расширений. Существуют системы с открытым исходным кодом, которые в некоторой мере решают базовые проблемы, связанные с полным протокольным декодированием. Однако, для эффективной работы и расширения функционала таких систем могут потребоваться дополнительные усилия и разработка специализированных решений.

Ключевые слова: глубокий анализ сетевого трафика; декодирование протоколов; параллельная обработка; управление памятью.

Для цитирования: Пономаренко Р.Е., Егоров В.И., Гетьман А.И. Вызовы в реализации систем глубокого анализа сетевого трафика методом полного протокольного декодирования. Труды ИСП РАН, том 35, вып. 4, 2023 г., стр. 45–64. DOI: 10.15514/ISPRAS–2023–35(4)–2.

Challenges in the implementation of systems for deep packet inspection by the method of full protocol decoding

[1] *R.E. Ponomarenko ORCID: 0009-0009-5741-3627 <rerandom@ispras.ru>*
[1] *V.I. Egorov ORCID: 0009-0001-5168-6474 <unclehook@ispras.ru>*
[1,2,3,4] *A.I. Getman ORCID: 0000-0002-6562-9008 <ever@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*
[2] *Moscow Institute of Physics and Technology (National Research University) 9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia.*
[3] *National Research University «Higher School of Economics» 20, Myasnitskaya ulitsa, Moscow 101978, Russia*
[4] *Lomonosov Moscow State University 1, Leninskie Gory, Moscow, 119991, Russia*

Abstract. This paper presents a summary of experience in developing the deep packet inspection system using full protocol decoding. The paper reviews the challenges encountered during implementation and provides a high-level overview of the solutions to these issues. The challenges can be grouped into two groups. The first group is related to the fundamental tasks which must be addressed when implementing full protocol decoding systems. This includes ensuring correct protocol parsing, which involves identifying and interpreting protocol headers and fields correctly. Moreover, it is necessary to ensure the processing of fragmented packets and the assembly of fragments into the original message. Additionally, the processing and analysis of encrypted traffic is a crucial task that may require the use of specialized algorithms and tools. The second group of problems is related to optimizing the process of full protocol decoding to ensure high-speed traffic processing, as well as supporting new protocols and the ability to add user-defined extensions. While there are open-source systems that address some of the primary issues associated with full protocol decoding, there may be a need for additional effort and specialized solutions to efficiently operate and expand the functionality of such systems. Although implementing deep network traffic analysis tools using full protocol decoding requires the use of advanced hardware and software technologies, the benefits of such analysis are significant. This approach provides a more complete understanding of network traffic patterns and enables more effective detection and prevention of cyber-attacks. It also allows for more accurate monitoring of network performance and the identification of potential bottlenecks or other issues that may impact network efficiency. In this article, we also emphasize the importance of system architecture development and implementation to ensure the successful deployment of deep network traffic analysis tools using full protocol decoding. At last, we conducted an experiment where several advanced optimizations were implemented in the system that had already solved primary issues. These optimizations related to working with memory, based on the features of the traffic processing scheme. By results, we evaluated significant performance improvement in solving secondary tasks, described in this work.

Keywords: deep packet inspection; decoding protocols; parallel processing; memory management.

For citation: Ponomarenko R.E., Egorov V.I., Getman A.I. Challenges in the implementation of systems for deep packet inspection by the method of full protocol decoding. Trudy ISP RAN/Proc. ISP RAS, vol. 35, issue 4, 2023. pp. 45-64 (in Russian). DOI: 10.15514/ISPRAS-2023-35(4)-2.

1. Введение

В сфере информационных технологий постоянно возрастает потребность в создании высокопроизводительных систем для решения сложных задач. Не всегда это напрямую связано с увеличением скорости работы. Под таким углом можно взглянуть на развитие систем передачи данных в сети интернет. Физических параметров у таких систем не много – скорость передачи и задержки при передаче. Улучшение этих параметров влечёт за собой большие расходы на обновление оборудования, замену линий передачи и тому подобное. Но влиять на субъективное восприятие эффективности работы таких систем (QoE) [1] можно при помощи других параметров, в частности, за счёт приоритизации потоков данных. При помощи

этого механизма трафик, чувствительный к скорости или задержкам работы систем передачи, обрабатывается в первую очередь, и у пользователя возникает ощущение работы с более производительными системами, хотя выигрыш был достигнут за счёт ухудшения параметров для трафика, который оказался менее приоритетным, исходя из предположения, что пользователь этого не заметит.

Другой важной и актуальной темой в информационных технологиях остаются атаки на компьютерные сети. Целей для атак (устройств и систем) становится больше, число угроз растёт, вредоносное программное обеспечение также не стоит на месте, однако успешные атаки не становятся обыденностью для большинства пользователей. Для того, чтобы оградить пользователей от деструктивного воздействия со стороны сети без вмешательства в его работу, необходимо распознавать такие атаки до того, как они достигнут пользователя, то есть на стороне поставщика услуг доступа в интернет.

В обоих частных случаях, описанных выше может применяется технология глубокого анализа сетевого трафика (DPI).

Многие из всемирно известных разработчиков систем глубокого анализа сетевого трафика, такие как Enea, Allot и Sandvine [2], прекратили продажи своих разработок в России. Благодаря накопленной экспертизе в разработке таких систем, программным обеспечением упомянутых компаний пользуются по всему миру и альтернативные решения возникают достаточно редко. В качестве российских компаний среди таковых можно упомянуть VAS Experts и РДП.РУ.

Глубокий анализ сетевого трафика является базовым механизмом, необходимым для решения множества задач, в частности, задач обеспечения информационной безопасности, что уже упоминалось выше. Поэтому в этом году российские компании, такие как Positive Technologies, Ростелеком-Солар и ВТБ, заявили о разработке систем глубокого анализа сетевого трафика в составе комплексных средств обеспечения информационной безопасности, преимущественно межсетевых экранов следующего поколения (NGFW) [3-5].

Приведённые выше факты показывают актуальность задач проработки архитектуры систем глубокого анализа сетевого трафика.

Разработка таких систем – нетривиальный процесс, сопряженный с большим количеством трудностей. Не разработана единая архитектура программного обеспечения, позволяющая проектировать одинаково эффективные системы глубокого анализа трафика для различных целей. В последующих разделах этой работы произведён обзор трудностей, связанных с реализацией системы глубокого анализа трафика методом полного протокольного декодирования, который позволяет получить максимальное количество информации.

1.1 Применение анализа сетевого трафика

Анализ сетевого трафика можно поделить по степени “глубины” анализа: поверхностный, средний и глубокий (SPI, MPI и DPI) [6]. К глубокому относится анализ полезной нагрузки уровня приложений сетевой модели OSI. Такой тип анализа даёт самые широкие возможности, однако и является самым трудоёмким. Цели применения этой технологии могут быть самыми разными.

Можно выделить две схемы работы в зависимости от входных данных: *онлайн-анализ*, то есть анализ трафика, поступающего на вход сетевого интерфейса и *офлайн-анализ*, то есть анализ предварительно записанного трафика [7]. В зависимости от цели применения DPI можно использовать ту или иную схему.

Онлайн-анализ трафика:

- Классификация пользователей и приложений:
  - Приоритизация или тарификация

Для справедливого распределения канала между абонентами при максимальной утилизации канала, провайдерами может использоваться DPI. Применяя эту технологию, можно выделять чувствительные к задержкам приложения, и отдавать приоритет в обработке трафику таких приложений.

Также, при выделении категорий приложений, возможна отдельная тарификация таких категорий. Например, безлимитный трафик для мессенджеров.

- Управление доступом групп пользователей к группам приложений (и функциям в этих приложениях)

  Основной функционал, который предоставляют межсетевые экраны актуального поколения (NGFW). При помощи DPI решаются такие задачи как: выделение групп пользователей по трафику, выделение приложений, а также отдельных функций приложений.

- Выделение полезной нагрузки седьмого уровня модели OSI:
  - Предотвращение утечек информации (DLP)

    Чувствительные к утечкам данные располагаются в полезной нагрузке протоколов прикладного уровня, для их выделения так же используются DPI.
- Выявление особых признаков:
  - Анализ и выявление сетевых атак

    Для выделения атак на приложения, а также целевых, ранее неизвестных атак (APT) системами обнаружения и предотвращения (NTA/NDR) [8] также применяется DPI, для выделения признаков.
  - Балансировка нагрузки

    Не всегда балансировки потоков данных (L4) приложения достаточно для распределения нагрузки по узлам. В случаях, когда отдельный клиент может генерировать большие объёмы информации для обработки, балансировка по потокам направит все запросы такого клиента на один узел. Для более эффективного решения такой задачи применяются L7 балансировщики. В них уровень приложений дополнительно разбивается на подуровни и по ним также возможна балансировка [9].

    В качестве другого примера, масштабируемое приложение, для которого нужна балансировка потока данных, чувствительно к балансировке, иными словами, промах при балансировке приводит к серьёзным последствиям, то для балансировки имеет смысл использовать глубокий анализ, что позволит точнее выделять сессии. Кроме того, это позволяет корректно обрабатывать трафик большого количества протоколов, например, это актуально для балансировки трафика агрегированных каналов.

Анализ записанного трафика:

- Отладка

  Отладка программ, генерирующих и обрабатывающих сетевой трафик, отладка средств обеспечения информационной безопасности, исследование произошедших инцидентов.

- Сетевая криминалистика

  Применяется для сбора доказательств при расследовании инцидентов. Инциденты не обязательно связаны с информационной безопасностью. Но по механизму работы ближе всего к анализу сетевых атак, за тем исключением, что время на анализ не ограничено в общем случае.

1.2 Классификация систем глубокого анализа сетевого трафика

В [10] выделяют 4 метода реализации функций глубокого анализа трафика:

- Классификация пакетов по портам L4.
- Поиск совпадения шаблонов (в основном, на основе регулярных выражений).
- Техники, основанные на машинном обучении.
- Декодирование протокола.

Наиболее полную информацию даёт анализ результатов полного протокольного декодирования. Полное декодирование каждого пакета, в общем случае, осуществляется получателем. Максимальное количество данных, которые получатель может принять определяется шириной канала и вычислительными возможностями входного тракта получателя. Именно получателя, а не отправителя, потому что обработка при отправке и получении может отличаться.

Для полного протокольного декодирования в сетевом исполнении и последующего анализа на скорости канала связи необходимы вычислительные мощности, превосходящие суммарную мощность входных трактов всех получателей, обрабатывающих исследуемый трафик, что в общем случае недостижимо.

Поэтому при решении различных задач происходит уменьшение числа анализируемых данных, вкупе с разными методами ускорения, до достижения приемлемой скорости обработки. С ростом возможностей вычислительной техники усложнялись и методы ускорения обработки. От чтения данных по фиксированному смещению (вычислительно самый дешёвый вариант разбора, к нему относится классификация пакетов по портам L4) к обработке потока байт регулярными выражениями (например, в случае поиска признаков атак в текстовых протоколах, таких как HTTP). Но всегда оставался класс задач, которые решались только протокольным декодированием. Это задачи, в которых важна высокая точность распознавания протоколов или их признаков, с небольшим количеством ложных срабатываний [10].

Кроме того, с ростом возможностей вычислительной техники проводить более “глубокий” анализ стало проще, что позволило как решать задачи в большем числе случаев, так и решать всё более сложные задачи. Примерами первого может быть избавление от ограничений на протоколы низкого уровня при анализе высокого, то есть анализ любого стека протоколов или любой вложенности. Если средство анализа не поддерживает какой-то из протоколов низкого уровня, использующийся для передачи или инкапсуляции (GRE, например), то анализ высокого уровня становится невозможен. Примерами второго – классификация трафика внутри отдельного приложения, например, выделение отдельных сервисов предоставляемыми социальными сетями (чаты, видео, музыка и тому подобное).

Все это приводит к тому, что задача анализа трафика методом полного протокольного декодирования, а также ускорение этого анализа, остаётся актуальной.

2. Базовые проблемы реализации протокольного декодирования и подходы к их решению

В данном разделе описываются проблемы, являющиеся основополагающими при реализации систем глубокого анализа трафика методом полного протокольного декодирования. Обобщённое представление процесса разбора сетевого трафика представлено на рис. 1.

2.1 Захват трафика с интерфейса

Прежде чем анализировать трафик, его необходимо получить с сетевого интерфейса. С ростом скоростей передачи данных делать это средствами, предоставляемыми ОС становится всё сложнее.

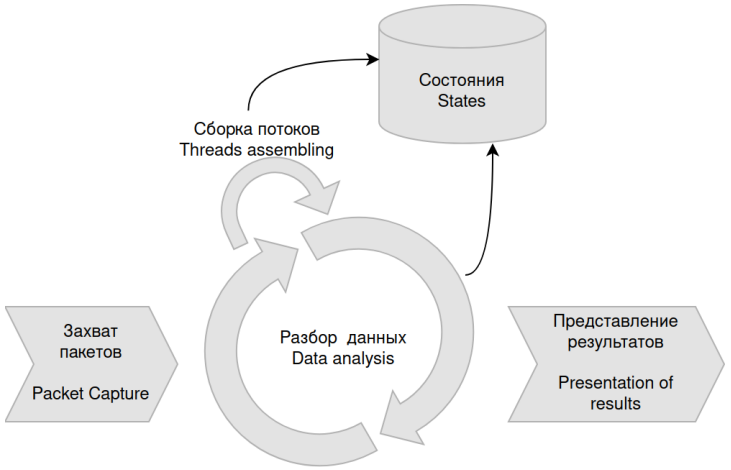

Рис. 1. Обобщённое представление процесса разбора сетевого трафика
Fig. 1. Generalized representation of the process of network traffic analysis

Можно выделить несколько проблем, которые влияют на скорость захвата трафика:

- Переключение контекста

  Взаимодействие с сетевым интерфейсом производится драйвером ОС, который, как правило, работает в привилегированном режиме. В то время как логика приложения, анализирующего трафик, выполняется в непривилегированном режиме. Переключение между этими режимами – очень затратная операция. Для сокращения количества таких переключений применяется отказ от попакетной обработки, тогда переключение будет происходить не на каждый пакет. Полный перенос обработки трафика в ядро, как и вынесение функций драйвера за пределы ядра, позволяет полностью отказаться от таких переключений [11].

- Копирование данных

  Аналогично переключению контекста, уменьшить число копирований позволяет отказ от попакетной обработки. Лучшее решение предоставляют библиотеки, позволяющие задействовать функции прямого обращения к памяти сетевой карты (DMA), а также копирование напрямую из буфера сетевой карты в память процесса, осуществляющего обработку. Такие библиотеки содержат в своём названии “zero copy” [11].

- Задействование возможностей аппаратного уровня

  Ускорить копирование из буфера сетевой карты позволяет уже упомянутый механизм DMA. Но современные сетевые карты и сопроцессоры могут брать на себя гораздо больше функций, причём выполнять их аппаратно, что снимает нагрузку с программной составляющей: ядра или библиотеки, осуществляющие обработку. К таковым функциям можно отнести как простую проверку контрольной суммы, так и полное управление сервисными сообщениями (TCP offloading) [12].

2.2 Выделение и сборка потоков

Данную проблему можно рассмотреть с двух сторон. На низком уровне – это задача обработки переупорядоченных, повторяющихся и повреждённых данных. Базовым примером тут является обработка протокола TCP, которым предусмотрены такие механизмы [7]. Дополнительно можно упомянуть QUIC (на базе UDP), в котором реализовано большое количество подобных механизмов, с которыми должен работать DPI. На более высоком уровне – это задача о реализации единого интерфейса обработки для потоков данных.

“Сырым” потоком данных будем называть тот, который можно получить с сетевого интерфейса. “Логическим” потоком можно назвать поток, передающийся в качестве полезной нагрузки какого-либо протокола, если протоколом предусмотрена передача потока, а не отдельных сообщений. Хорошим примером тут также может послужить TCP, так как в качестве полезной нагрузки у него выступает именно поток данных. Другими примерами могут быть QUIC, HTTP, IPv4. У последних двух полезная нагрузка не является потоком, но несмотря на это, они поддерживают фрагментацию полезной нагрузки.

2.3 Организация конвейера обработки потоков данных

Процесс обработки сетевого трафика методом протокольного декодирования должен быть устроен итеративно, из-за инкапсуляции потоков данных друг в друга. Организовать процесс таким образом позволит единый интерфейс взаимодействия отдельных этапов обработки. Единый интерфейс обработки потоков данных позволит выделять обработчики в изолированные и взаимозаменяемые модули [7].

Для создания такого интерфейса можно создать новый непрерывный буфер, в который скопировать полезную нагрузку отдельных пакетов. Это позволит реализовать простой интерфейс анализа вложенных протоколов, так как на вход им будет подаваться непрерывный буфер. Недостатком такого решения являются затраты времени на копирование данных. Также нужно иметь в виду, что данные полезной нагрузки могут быть закодированы. К ним может применяться сжатие, шифрование и тому подобные процедуры.

Примером тут может стать протокол HTTP. В этом протоколе длинная последовательность байт полезной нагрузки (например, при передаче большого файла) разделяется на отдельные пакеты – чанки (куски, chunks) с добавлением небольших заголовков. Но, помимо этого, полезная нагрузка может быть сжата. Для последующего анализа нужно не только скопировать данные в новый буфер, но и предварительно распаковать их.

Другим подходом в реализации может стать отказ от копирований. В этом случае придётся реализовывать структуру для хранения адресов отрезков, имеющих отношение к разбираемому потоку данных. В таком случае вместо затрат времени на копирование данных, появятся затраты времени на обращение к данным, так как нужно корректно высчитывать смещение. Работать в такой схеме, с данными, подверженными кодированию, всё сложнее, так как для каждого отрезка нужно хранить параметры для декодирования, либо высчитывать их в момент обращения.

Какой бы подход не был выбран, система анализа должна в своём внутреннем представлении хранить информацию о выделенных потоках, и осуществлять разбор исходя из этой информации. Это обязательное требование к DPI [7].

2.4 Отслеживание и хранение состояния

Для каждого анализируемого потока необходимо сохранить информацию для его идентификации, так называемый “ключ”. При поступлении нового пакета на обработку, при помощи этой информации, необходимо однозначно отнести пакет к какому-либо потоку или создать новый.

По нашему опыту, идентификация потока сводится к двум случаям. Первый случай – когда протокол предусматривает передачу идентификатора потока. Зачастую, он представлен в виде пары идентификаторов в заголовках каждого пакета, для каждой стороны обмена. Например, это можно видеть в QUIC, TCP, UDP. Второй случай – когда поток определяется идентификатором потока нижестоящего протокола, то есть тем, в который он инкапсулирован. Например, “поток” HTTP (до HTTP/2) в рамках TCP-сессии может быть только один.

Однако при определении потока только по таким идентификаторам возможны коллизии.

Первый вид коллизий связан с возможностью инкапсуляции разных потоков одинаковых протоколов с одинаковыми идентификаторами друг в друга. Это может происходить при использовании туннелирования (GRE, протоколы, реализующие VPN и так далее). Пример подхода к разрешению такой коллизии будет подробнее описан в следующем подразделе, о представлении результатов.

Другим видом коллизии можно назвать коллизии во времени, происходящие с протоколами без явного признака начала и конца сессии, например, с протоколом UDP. При очередном соединении порт клиента может быть выбран случайно. Спустя время, после освобождения этого порта он может быть использован снова. Аналогичная ситуация может возникнуть при анализе протокола с явным признаком конца сессии, если этот признак по какой-либо причине не был захвачен системой анализа трафика. Для разрешения такого типа коллизии чаще всего используются метки времени, то есть вводят промежуток времени. Если новый пакет в рамках существующих идентификаторов был создан через время, большее заданного промежутка, то данный пакет относится к новому потоку данных [13].

Разрешение подобных коллизий кажется чем-то самим собой разумеющимся, однако достаточно часто в исследованиях можно встретить ошибки, связанные именно с такими коллизиями [14-15].

Помимо идентификаторов, каждый поток, в зависимости от его “типа”, может сохранять дополнительную информацию. Как минимум, к этой информации относится состояние, в котором пребывает обработчик потока в данный момент. При использовании кодирования – информация для дальнейшего декодирования. При клиент-серверном взаимодействии с различным форматом сообщений для каждого из них – информацию о том, какая сторона является сервером, а какая клиентом.

2.5 Представление результатов

При инкапсуляции потоки данных (сессии) вложены друг в друга. Например, в исходный поток данных (файл pcap или последовательность пакетов Ethernet), поданный на вход вложен поток пакетов IPv4 (возможно фрагментированный). В поток IPv4 вложен поток TCP, а в него прикладной протокол. Это базовый пример, который может быть усложнён при использовании туннелирования (например GRE, VPN и так далее).

При клиент-серверном взаимодействии есть два потока данных: от клиента к серверу и обратный. При одноранговом (P2P) число потоков возрастает вместе с числом участников обмена. Тем не менее, все эти потоки связаны между собой и результат работы полного протокольного декодирования должен это отражать [16].

Так как разбор происходит в рамках потоков данных (сессий), то на высоком уровне результат разбора представляет собой зависимость этих потоков данных (вложенность, логическая связь, и пр.).

В предыдущем подразделе описывалась коллизия при инкапсуляции. Для решения этой проблемы, “ключ” потока должен представлять из себя не только идентификатор данного потока, но также и структуру вложенности протокола, то есть быть составным и включать в себя идентификаторы всех протоколов, в которые он вложен.

При полном протокольном декодировании должен осуществляться разбор каждого поля каждого отдельного пакета. Это означает, что для каждого такого поля должно быть обозначены его начало и конец, должно быть извлечено значение, в соответствии с форматом, значение должно быть интерпретировано и влияние значения на состояние потока должно быть отображено в дополнительной информации потока. Так как поля чаще всего имеют вложенную структуру, то и представляется результат в виде дерева. Подробнее это описано в [17].

2.6 Расширяемость (модульность)

Затруднительно организовать монолитную структуру сетевых анализаторов в связи с тем, что постоянно появляются новые протоколы, а уже существующие всё время дополняются. Поэтому зачастую прибегают к подходу, когда реализация разбора конкретного протокола представляется в виде модуля (библиотеки) [18].

В таком случае для центрального компонента системы анализа возникает необходимость подсистемы для управления отдельными модулями разбора. В задачи такой подсистемы входит: обнаруживать такие модули, последовательно передавать им входные данные и получать результат. При этом нужно выдерживать слабую связность компонентов между собой, чтобы обновление одного не приводило к повторному проектированию других.

В качестве альтернативного подхода тут можно рассматривать некое “уведомление” всех модулей о появлении нового. В таком случае упрощается реализация центрального компонента и усложняется реализация модулей разбора отдельного протокола.

2.7 Неприменимость стандартных алгоритмов и библиотек для обработки трафика

Отдельного упоминания в данной работе заслуживает тот факт, что в системах глубокого анализа, которые захватывают сетевой трафик по пути его прохождения (а не на конечных узлах), почти не встречаются готовые библиотеки, в отличии от клиентских реализаций протоколов. Вместо этого, разбор отдельных полей протоколов, отслеживание состояния, и прочий функционал реализуется разработчиками систем анализа. Это связанно с вероятными отличиями трафика, который будет получен принимающей стороной, от трафика, который будет захвачен в одной отдельной точке пути его прохождения. Например, какие-то пакеты могут быть модифицированы или вообще отброшены в процессе передачи. На этом основаны многие атаки на сетевые системы обнаружения и предотвращения вторжений, в составе которых имеются системы глубокого анализа сетевого трафика [19].

3. Обзор архитектур открытых инструментов

В этом разделе представлен обзор существующих архитектурных решений и свойств открытых инструментов для глубокого анализа сетевого трафика. Инструменты были выбраны из-за использования метода протокольного декодирования или наличия релевантных механизмов.

3.1 nDPI

На данный момент, nDPI самый популярный инструмент для создания прототипов инструментов, содержащих глубокий анализ сетевого трафика. У него относительно простой для внедрения API, и он работает с приемлемой скоростью [20-21].

В связи с этим, многие разработчики внедрили этот инструмент в свою инфраструктуру, однако результаты, которые могут быть получены с помощью этого инструмента, не всегда хорошо соответствуют требованиям бизнеса. Запрос таких разработчиков – получить более глубокий разбор или большее количество метаинформации по большему числу протоколов, не изменяя механизмов взаимодействия, а также повысить скорость и реализовать распараллеливание обработки.

У nDPI очень гибкий прикладной интерфейс, используемый для получения трафика и его последующей обработки. Есть примеры использования с различными библиотеками, позволяющими производить высокоскоростной захват трафика с сетевого интерфейса. К таковым относятся PF_RING, DPDK, AF_XDP и прочие.

nDPI предполагает как расширение числа модулей, так и интерфейс для получения результатов обработки сетевого трафика.

Для добавление нового протокола или метрики (если речь только о сборе статистики) нужно изменять исходный код nDPI, после чего заново компилировать. Это обусловлено попыткой достичь максимальной скорости работы инструмента и монолитной архитектурой.

Функционал по сборке потоков транспортных протоколов TCP и UDP реализован как частный случай. Информация о собранных потоках располагается непосредственно в ядре системы разбора и не подлежит расширению или модификации без перекомпиляции всего проекта.

Несмотря на то, что nDPI выполняет лишь частичное протокольное декодирование, в данной работе он упомянут из-за схожего с полным протокольным декодированием механизма работы. Каждый протокол имеет свои, независимые функции распознавания и разбора. Хотя для разработчиков оставлена возможность объединить некоторые распознаватели, из-за организации кода в виде монолита.

Однако nDPI часто используется для категорирования потока, а не каждого отдельного пакета в нём. Поэтому в коде предусмотрены оптимизации, позволяющие не обрабатывать потоки, которые уже были категоризированы.

Есть несколько примеров использования nDPI, предоставляемых разработчиками библиотеки. Существуют модули ядра Linux, называемые ndpi-netfilter или xt_ndpi [22]. Эти модули позволяют классифицировать трафик по протоколам прикладного уровня и ограничивать прохождение трафика, исходя их этой информации.

В библиотеке nDPI существует программное API, которое помимо классификации трафика по прикладным протоколам и категориям, позволяет выявлять признаки некоторых сетевых атак (например, возможные XSS, SQL injection, RCE, слабую криптографию и так далее).

В библиотеку также включен демон (nDPId), пропускающий через себя трафик и генерирующий поток событий [23]. К событиям могут относиться создание и завершение потока, обработка очередного пакета, успешное или неуспешное детектирование протокола транспортного или прикладного уровня, а также сообщения об ошибках в процессе разбора.

Есть инструменты, например, proof of concepts (ntop) [24], реализующие полный функционал системы фильтрации контента, и системы предотвращения угроз, на сколько это позволяет библиотека nDPI.

3.2 Wireshark

Другим примером может служить Wireshark. Тем более, что в научных статьях можно встретить множество упоминаний сравнения результатов работы именно с этим инструментом [25-26].

Для захвата трафика может использоваться библиотека libpcap, что ставит под вопрос производительность при работе с трафиком, получаемым из сетевого интерфейса (*онлайн-анализ*). Хотя из-за модульности архитектуры возможны исключения, со своими ограничениями [27].

С точки зрения организации кода в Wireshark, центральным функциональным блоком, интересным для изучения в рамках данной работы, является Epan (Enhanced Packet ANalyzer) [28]. Он включает в себя функционал для хранения состояния разбора (дерева разбора), интерфейс для обращения к модулям, реализующим разбор отдельных протоколов и фильтрацию результатов исходя из запросов пользователя.

Разбор отдельных протоколов делится на два этапа, предварительный и этап запросов пользователей информации о конкретных пакетах. В некоторых модулях за оба прохода отвечает один и тот же код. Интерфейс ядра позволяет выделить поток из буфера пакета и произвести его сборку в отдельный буфер [29].

Хранение состояния модуля разбора происходит на стороне ядра, состояние передаётся каждый раз при разборе очередного пакета. Хранение дополнительных параметров также возможно.

Предполагается, что результаты обработки будут показаны человеку посредством графического интерфейса. Для автоматической же обработки предполагается либо использование плагинов, схожее с добавлением поддержки нового протокола, либо использование скриптов на языке Lua. Однако стоит упомянуть также про tshark – клиент командной строки Wireshark. После обработки трафика, tshark преобразует результаты в строковый формат и может передать на вход другому приложению или в файл. Для вывода доступно большое количество форматов [30].

Wireshark, помимо возможности добавления поддержки новых протоколов через изменение исходного кода, как в nDPI, также поддерживает добавление новых протоколов через разделяемые библиотеки [31].

Другими инструментами, позволяющими расширять базовый функционал Wireshark, можно назвать написание скриптов на языке Lua и декларативное описание разборщиков при помощи языка ASN.1 [32-33].

4. Актуальные проблемы в реализации полного протокольного декодирования и подходы к их решению

После реализации полного протокольного декодирования разработчики приходят к схожим проблемам, которые можно разбить на 3 группы:

- Скорость обработки
- Хранение внутреннего состояния
- Расширение функционала

В ИСП РАН ведётся разработка системы глубокого анализа пакетов “Протосфера” [34]. В процессе разработки были выявлены проблемы из этих групп, подробнее описанные далее.

Эти же проблемы наблюдаются и у ближайшей по возможностям открытой системы – Wireshark [35-36].

4.1 Подходы к общему ускорению работы системы анализа

Для этой группы проблем можно предложить два направления развития решений, потенциально приносящие ускорение:

- Сужение области применимости решаемой задачи, или, другими словами, отбрасывание всего “ненужного” из результатов.
- Распараллеливание алгоритмов работы.

Для первого направления показателен пример операции поиска по метаданным. Во время разбора пакетов часто выполняются операция поиска: поиск нужного потока, поиск встречного потока, поиск подходящего ключа и так далее. В зависимости от контекста можно применять тот или иной алгоритм, чтобы приблизиться к теоретическому минимуму времени работы. Но время всё равно будет зависеть от количества метаданных, по которым осуществляется поиск. Поэтому для ещё большего ускорения в абсолютных значениях необходимо снижать количество метаданных. Этого, в свою очередь, можно достичь, если сокращать время жизни метаданных. Ниже приведены такие эксперименты вкупе с другими методами оптимизации производительности, этот эксперимент будет описан в отдельном разделе.

Также к этому направлению решения можно отнести “насыщение” метаинформацией поверхностный разбор пакетов. Когда полное протокольное декодирование запускается на отдельном участке процесса разбора, в то время как уровни ниже и даже уровни выше могут быть уже распознаны и размечены. Такая схема показывает свою эффективность в системах, где стек протоколов, как правило низкого уровня, зафиксирован, в виду доменной специфики. В этом случае, система, реализующая полное протокольное декодирование, должна иметь интерфейс, для описания протоколов ниже, и смещения, с которого следует начать разбор.

В направлении распараллеливания алгоритмов работы можно предложить несколько подходов.

Исторически первый и самый простой – деление программы на части. Одни, “критические” – те, которые не поддаются распараллеливанию. Остальные же запускать параллельно допустимо. Данный подход позволяет ускорить обработку в рамках одного вычислительного узла. Ускорение в данном случае будет изменяться по закону Амдала, то есть если предположить, что для распараллеливания подходит только половина алгоритма, то ускорение не может быть более 2x раз. В качестве примера можно представить использование библиотеки OpenMP [37].

Другой подход заключается в распределённых вычислениях в рамках кластера:

- Использовать MapReduce для разделения DPI на модули и запускать модули распределённо, например, в таких системах, как Apache Hadoop [38].
- Использовать оркестраторы, например, Kubernetes. Что наиболее предпочтительно ввиду сложности модулей, особенно тех, которые требуют для эффективной работы большой объём контекстной информации.

Популярным подходом является пул “воркеров”, то есть специально заготовленные потоки и/или процессы, которые работают по принципу producer-consumer [37]. Наиболее эффективной реализацией видится деление по потокам данных.

Эффективность работы такой схемы зависит от эффективности балансирования потоков по обработчикам. Но исходя из специфики обрабатываемых данных, в начале работы всегда будет занят только один обработчик, так как входной поток может быть только один для одного источника данных. А этот входной поток позже породит новые потоки. Решением может стать предварительная балансировка, в частности, с привлечением аппаратных средств.

В попытках повысить утилизацию ресурсов можно прийти к концепции асинхронного программирования, при которой возможность распараллеливания сильно увеличивается. За счёт использования “обещаний” в асинхронном программировании можно параллельно обрабатывать даже данные внутри одного пакета.

У этого метода есть много оговорок, также он существенно усложняет поддержку проекта. Например, появляется необходимость синхронизации. Поэтому конкретное описание применения этого метода к задаче полного протокольного декодирования выходит за рамки этой работы и будет рассматриваться отдельно.

4.2 Подходы к управлению памятью

С точки зрения выделения памяти можно выделить два подхода: статическое распределение памяти и использование динамического выделения памяти.

Второй подход проще в реализации, однако имеет несколько значительных недостатков. Главный из которых в том, что нельзя заранее предсказать, в какой момент библиотека или ОС, на которую переложили ответственность за выделение памяти откажет в этой операции. Также критичной является скорость работы, при обращении к ОС.

Статическое распределение памяти решает главный недостаток динамического, количество доступной памяти в используемых структурах заранее известно. Однако с эффективной утилизацией памяти могут быть проблемы, когда память заполняется метаданными неравномерно. А при попытке удалить данные из заполненного буфера приходится также удалять и связанные с ними данные других, не заполненных буферов. Эта ситуация тесно связана со способом хранения связанной информации, что будет описано ниже.

Компромиссным решением в случае реализации системы полного протокольного декодирования сетевого трафика может быть комбинация этих подходов. Например, динамическое выделение больших регионов памяти и статическое распределение этих регионов между структурами разных видов.

Размер данных (и метаданных) рано или поздно превысит доступный объём памяти и встанет вопрос о том, какую информацию в данный момент можно больше не хранить. Существует большое количество подходов к реализации вытеснения информации, среди которых можно упомянуть кольцевой буфер и LRU кэш [39]. Анализ того, какие подходы в конкретных случаях показывают наилучшую эффективность выходят за рамки данной работы и будет рассмотрен отдельно.

Особняком стоит вопрос организации связности данных. Можно хранить связанные данные в виде одной (большой) структуры, а все данные представить в виде массива таких структур, как это показано на рис. 2. Другой же подход состоит в распределении данных по небольшим структурам, массивов в этом случае будет несколько, см. рис. 3.

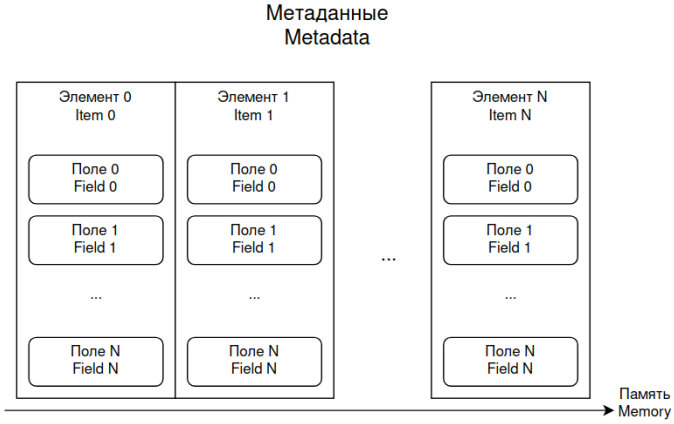

Рис. 2. Представление метаданных в виде массива больших структур
Fig. 2. Representation of metadata as an array of large structures

Показателен пример игровой индустрии, где повсеместно наивный подход хранения коллекции объектов в парадигме ООП заменяется на Data-oriented design. В последнем данные группируются в коллекции по своему типу, что позволяет производить их последовательную обработку, снижая количество “промахов” кэш памяти [40-42].

Выбор того или иного подхода обусловлен спецификой данных, целей анализа, реализацией конструкций языка программирования, размерами и иерархией кэш памяти процессора и прочими факторами, поэтому подробный разбор также выходит за рамки этой работы.

Среди базовых проблем была упомянута проблема сборки потоков. Влияние выбранных механизмов управления памятью на этот процесс очевидно. Однако после оптимизации этого процесса, заметнее проявляет себя проблема обращения к отдельным полям пакета, которые не расположены последовательно. Так как это большое количество обращений к небольшим участкам памяти, копирование на этом этапе оптимизаций является очень затратной операцией. Особенно актуальна эта проблема в Wireshark, где происходит двухэтапный разбор, а также повторный разбор каждый раз, когда пользователь обращается к конкретному пакету или потоку.

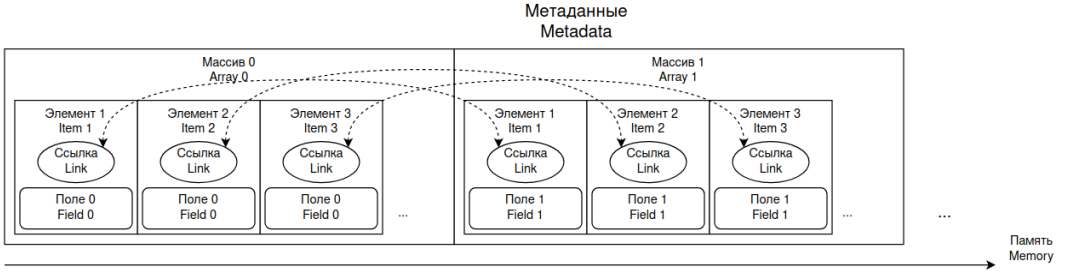

Рис. 3. Представление метаданных в виде массивов небольших связанных структур
Fig. 3. Representation of metadata in the form of arrays of small related structures

4.3 Проблемы с расширением функционала

Различные разработчики могут по-разному реализовывать и использовать различные протоколы. Эта проблема особенно актуальна для разработчиков сетевого оборудования, где можно видеть много расширений открытых и проприетарных протоколов, которые затрудняют согласованную работу оборудования различных производителей. Помимо намеренного внесения усложнений, также свои проблемы вносят допускаемые разработчиками ошибки.

Данная проблема частично была описана среди базовых и часть решения такого рода проблем существует у открытых инструментов, таких как Zeek и Wireshark. Оно состоит в упрощении разработки модулей разбора протоколов. В Wireshark для расширения функционала может использоваться скриптовый язык Lua, так как это императивный язык, то скрипты позволяют модифицировать процесс анализа [32]. В Zeek используется DSL язык генерации парсеров – Spicy [43]. Данный язык основной своей целью ставит именно разработку разборщиков, поэтому на первый план выходят декларативные свойства.

При анализе подобного недетерминированного сетевого трафика, система разбора должна либо по каким-либо косвенным признакам догадываться об используемых расширениях и протоколах (fingerprinting). Либо хранить состояние и производить разбор всех протоколов и расширений, которые возможны в конкретном контексте. На практике для получения наиболее полных результатов необходимо совместное использование этих подходов.

В различных случаях использования DPI требуется различная информация из внутреннего представления, а также различная дополнительная обработка этой информации (собственно, анализ). Например, в решении задач обеспечения информационной безопасности чаще используются анализ потока событий (например, в Zeek). В то же время в задачах отладки и криминалистики – задач, где чаще анализируются записанные данные, анализируется каждое отдельное поле в пакете.

Достаточно сложно организовать универсальный доступ ко всем этим данным, так как итеративно кодовая база системы анализа трафика превращается в набор “заплаток”, которые помогают получать информацию из различных мест в разных контекстах. Именно такую “архитектуру” можно наблюдать в nDPI. С течением времени расширение функционала и исправление ошибок в такой системе даётся всё труднее. Поэтому имеет смысл организовать структуру модулей (менеджеров, компонентов) таким образом, чтобы было возможно получить *всю* информацию. При этом, для того чтобы избежать наложения заплаток должен быть реализован интерфейс, который бы позволял универсально получать всю информацию, и в то же время позволял бы на раннем этапе обработки производить настройку разбора и фильтрацию результата. Последнее особенно важно в контексте полного протокольного декодирования для того, чтобы “иметь возможность не платить за то, что не используется”.

Дополнительно стоит упоминания подход, предложенный при создании eBPF в ядре Linux [44], и который используется в инструменте bpftrace [45]. В этом случае можно небольшие программы оформлять как обратные вызовы при обращениях к API. За счёт этого, например,

иметь возможность собирать (или не собирать) статистику во время выполнения без перезапуска программы.

4.4 Проблемы обработки «одностороннего» трафика

При прохождении трафика в глобальной сети интернет возможны ситуации, когда разные пакеты в рамках одной сессии могут передаваться по различным маршрутам, иногда даже по различным каналам. В качестве примера здесь можно привести спутниковый интернет, где зачастую непосредственно через спутник проходит только входящий для абонента трафик.

Для анализа ряда протоколов очень важно располагать доступом к обоим потокам данных: входящему и исходящему. В качестве примера можно привести протокол ISAKMP из IPSec [46], в котором после инициализации защищённого канала шифрование пакетов происходит последовательно по обоим направлениям. Без доступа к пакетам из одного из них невозможно получить вектор инициализации, необходимый для расшифровки пакета встречного направления. Приведённый пример показывает скорее невозможность полного анализа в такой схеме работы. Однако можно привести и примеры, когда анализ может быть затруднён, например, HTTP 1.1. При анализе только ответов нужно будет определять, должны ли после заголовка идти данные или нет (как в случае ответа на запрос HEAD) [47].

Для полного решения данной проблемы могут применяться системы, предполагающие распределённый захват трафика в различных точках и последующую их синхронизацию.

5. Эксперимент

В качестве оценки прироста производительности при решении актуальных задач, описываемых в данной работе, был проведён эксперимент, в ходе которого в систему, в которой были решены базовые проблемы, было внедрено несколько оптимизаций второго порядка. Оптимизации касались работы с памятью, исходя из особенностей схемы обработки трафика.

Метаинформация была поделена на два класса: “необходимую для дальнейшего анализа” и прочую. Примером первого класса можно привести информацию о “ключе” потока, описанную в предыдущих разделах. Примером второго может послужить информация о поле контрольной суммы отдельного пакета. Такая информация может быть полезна в момент анализа отдельного пакета, а в рамках анализа всего потока, данная информация не несёт ценности.

Время жизни метаинформации первого класса совпадало с временем жизни потока, то есть этот функционал не изменялся. Время жизни метаинформации второго класса зависело от скорости поступления (или скорости вытеснения) такого рода метаинформации, так как она хранилась в ограниченном по размеру буфере, то есть то, что такая информация будет удалена *после* окончания обработки не гарантируется. Но количество одновременно хранимой метаинформации было существенно сокращено, что в свою очередь, привело к росту скорости её обработки, в частности, при поиске связанной информации.

Для измерения производительности был подготовлен набор данных объёмом 5,8 Гб (6076000 пакетов), содержащий трафик реальной сети. Отдельным запуском системы анализа было посчитано количество генерируемой метаинформации (блоков из [7]): всего 164049139 структур, из них 138905530 помечены как метаинформация второго класса, то есть распределение по классам было 15% и 85%.

Система разбора запускалась с включением порядка 30 модулей разбора протоколов, которые детектировали и разбирали порядка 50 протоколов.

При этом стоит пояснить, что время жизни объекта было ограничено только в ядре разбора. Если в это время произошло копирование данных за пределы ядра разбора, то там объект принудительно не удалялся.

Для получения “лучшего времени” исходные данные для анализа считывались с диска, то есть захват трафика с сетевого интерфейса был исключён, но тестирование с захватом трафика с сетевого интерфейса также проводились.

Одновременно с этим проводилась оптимизация использования ресурсов памяти системы в целом. Размеры очередей разных типов данных, выравнивания данных, а также механизмы выделения и высвобождения данных (на куче) эмпирически подбирались таким образом, чтобы оптимизировать обращения к этим данным, в частности, сокращая количество “промахов” при обращении к кэшу памяти. Конкретные результаты в этой работе не приводятся ввиду большого объёма проделанной работы и сложности сравнения различных этапов оптимизаций, в частности, связанных с изменением структуры метаинформации.

Совокупно, данные изменения ограничили возможность обработки метаинформации второго класса, в обмен на ускорение обработки на общих примерах в 6 раз в среднем и 6000 раз в лучшем случае.

6. Заключение

В данной работе был собран и обобщён опыт в разработке системы глубокого анализа трафика (DPI) методом полного протокольного декодирования, который был представлен в виде обзора проблем, возникающих при реализации. Описанные проблемы были разделены на две группы: базовые и актуальные. Рассмотрены решения, представленные в системах с открытым исходным кодом nDPI и Wireshark (преимущественно базовых проблем). Описаны направления решения рассматриваемых проблем, подробное рассмотрение решений каждой будет представлено дополнительно. Приведены результаты эксперимента по применению описываемых оптимизаций в существующей системе глубокого анализа трафика, которые показывают большой потенциал в дальнейшем исследовании возможности применения оптимизаций в описываемой области.

Список литературы / References

[1]. Brunnström K., Beker S., De Moor K., Dooms A., Egger S., Garcia M., Hoßfeld T., Jumisko-Pyykkö S., Keimel C., Larabi C., Lawlor B., Le Callet P., Möller S., Pereira F., Pereira M., Perkis A., Pibernik J., Pinheiro A., Pibernik J., Raake A., Reichl P., Reiter U., Schatz R., Schelkens P., Skorin-Kapov L., Strohmeier D., Timmerer C., Varela M., Wechsung I., You J., Zgank A. Qualinet white paper on definitions of quality of experience. – 2013.

[2]. Gallagher, R. Sandvine Pulls Back From Russia as US, EU Tighten Control on Technology It Sells / R. Gallagher. – Текст: электронный // Bloomberg: [сайт]. – URL: https://www.bloomberg.com/news/articles/2022-06-03/sandvine-pulls-back-from-russia-as-us-eu-tighten-control-on-technology-it-sells 03.06.2022 (дата обращения: 29.06.2023).

[3]. Афанасьева, О. Positive Technologies разрабатывает собственный NGFW / О. Афанасьева. – Текст: электронный // Anti-Malware.ru: [сайт]. – URL: https://www.anti-malware.ru/news/2023-01-13-118537/40305 13.01.2023 (дата обращения: 28.06.2023).

[4]. Афанасьева, О. РТК-Солар показала импортонезависимый NGFW / О. Афанасьева. – Текст: электронный // Anti-Malware.ru: [сайт]. – URL: https://www.anti-malware.ru/news/2023-04-13-118537/40945 13.04.2023 (дата обращения: 28.06.2023).

[5]. Королёв, И. В России вложат более 3 миллиардов в разработку межсетевых экранов нового поколения / И. Королёв. – Текст: электронный // CNews: [сайт]. – URL: https://www.cnews.ru/news/top/2023-04-14_v_rossii_vlozhat_bolee_3_mlrd 14.04.2023 (дата обращения: 28.06.2023).

[6]. Christopher Parsons. Deep Packet Inspection in Perspective: Tracing its lineage and surveillance potentials // Working Paper, January 2009

[7]. Маркин Ю. В. Методы и средства углубленного анализа сетевого трафика //автореферат дис. кандидата технических наук/Ин-т систем. программирования. Москва. – 2017.

[8]. Ким Д., Рыжков В. NTA, IDS, UTM, NGFW – в чем разница? / Д. Ким, В. Рыжков – Текст: электронный // SecurityLab.ru: [сайт]. – URL: https://www.securitylab.ru/analytics/517592.php 19.03.2021 (дата обращения: 05.07.2023).

[9]. Klein, M. Introduction to modern network load balancing and proxying / M. Klein. – Текст: электронный // The official Envoy Proxy blog: [сайт]. – URL: https://blog.envoyproxy.io/introduction-to-modern-network-load-balancing-and-proxying-a57f6ff80236 28.12.2017 (дата обращения: 29.05.2023).
[10]. Çelebi M., Özbilen A., Yavanoğlu U. A comprehensive survey on deep packet inspection for advanced network traffic analysis: issues and challenges //Niğde Ömer Halisdemir Üniversitesi Mühendislik Bilimleri Dergisi. – vol. 12. – №. 1. – pp. 1-29.
[11]. Ларин Д. В., Гетьман А. И. Средства захвата и обработки высокоскоростного сетевого трафика //Труды Института системного программирования РАН. – 2021. – т. 33. – №. 4. – с. 49-68.
[12]. Pismenny, B., Eran, H., Yehezkel, A., Liss, L., Morrison, A., Tsafrir, D. Autoomous NIC noffloads //Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. – 2021. – С. 18-35.
[13]. Borisov, N., Brumley, D., Wang, H. J., Dunagan, J., Joshi, P., Guo, C. Generic Application-Level Protocol Analyzer and its Language //NDSS. – 2007.
[14]. Engelen G., Rimmer V., Joosen W. Troubleshooting an intrusion detection dataset: the CICIDS2017 case study //2021 IEEE Security and Privacy Workshops (SPW). – IEEE, 2021. – с. 7-12.
[15]. Гетьман, А. И., Горюнов, М. Н., Мацкевич, А. Г., Рыболовлев, Д. А. Методика сбора обучающего набора данных для модели обнаружения компьютерных атак //Труды Института системного программирования РАН. – 2021. – т. 33. – №. 5. – с. 83-104.
[16]. Рекомендация МСЭ-Т Y.2770 - Требования к углубленной проверке пакетов в сетях последующих поколений.
[17]. Гетьман, А. И., Иванников, В. П., Маркин, Ю. В., Падарян, В. А., Тихонов, А. Ю. Модель представления данных при проведении глубокого анализа сетевого трафика //Труды Института системного программирования РАН. – 2015. – т. 27. – №. 4. – с. 5-22.
[18]. Andrew Moore, James Hall, Christian Kreibich, Euan Harris, and Ian Pratt. Architecture of a Network Monitor // International Workshop on Passive and Active Network Measurement, PAM 2003
[19]. Bukac V. IDS system evasion techniques //Master. Masarykova Univerzita. – 2010.
[20]. Bujlow T., Carela-Espanol V. Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification. – 2013.
[21]. Satrya G. B., Nugroho F. E., Brotoharsono T. Improving network security-a comparison between ndpi and l7-filter //International Journal on Information and Communication Technology (IJoICT). – 2016. – vol. 2. – №. 2. – pp. 11-11.
[22]. ndpi-netfilter – Текст: электронный // github.com: [сайт]. – URL: https://github.com/betolj/ndpi-netfilter (дата обращения: 10.07.2023).
[23]. nDPId: Tiny nDPI based deep packet inspection daemons / toolkit – Текст: электронный // github.com: [сайт]. – URL: https://github.com/utoni/nDPId (дата обращения: 10.07.2023).
[24]. ntopng – Текст: электронный // ntop: [сайт]. – URL: https://www.ntop.org/products/traffic-analysis/ntop/ (дата обращения: 10.07.2023).
[25]. C. Shen, L. Huang, On detection accuracy of L7-filter and OpenDPI, in: 2012 Third International Conference on Networking and Distributed Computing (ICNDC), IEEE, Hangzhou, China, 2012, pp. 119–123, dOI: 10.1109/ICNDC.2012.36.
[26]. R. Goss, R. Botha, Deep Packet Inspection – Fear of the Unknown, in: Information Security for South Africa (ISSA), 2010, IEEE, Sandton, Johannesburg, South Africa, 2010, pp. 1–5, dOI: 10.1109/ISSA.2010.5588278
[27]. Capture, Filter, Extract Traffic using Wireshark and PF_RING. – Текст: электронный // ntop: [сайт]. – URL: https://www.ntop.org/pf_ring/capture-filter-extract-traffic-using-wireshark-and-pf_ring/ 04.04.2017 (дата обращения: 10.07.2023).
[28]. Chapter 7. How Wireshark Works – Текст: электронный // ntWiresharkop: [сайт]. – URL: https://www.wireshark.org/docs/wsdg_html_chunked/ChWorksOverview.html (дата обращения: 24.07.2023).
[29]. 9.5. How to reassemble split packets. – Текст: электронный // wireshark: [сайт]. – URL: https://www.wireshark.org/docs/wsdg_html_chunked/ChDissectReassemble.html (дата обращения: 28.06.2023).
[30]. tshark(1) Manual Page. – Текст: электронный // Gitlab: [сайт]. – URL: https://gitlab.com/wireshark/wireshark/-/blob/master/doc/tshark.adoc (дата обращения: 28.06.2023).
[31]. Chapter 9. Packet Dissection. – Текст: электронный // Wireshark: [сайт]. – URL: https://www.wireshark.org/docs/wsdg_html_chunked/ChapterDissection.html (дата обращения: 28.06.2023).

[32]. Chapter 11. Wireshark's Lua API Reference Manual. – Текст: электронный // Wireshark: [сайт]. – URL: https://www.wireshark.org/docs/wsdg_html_chunked/wsluarm_modules.html (дата обращения: 28.06.2023).
[33]. Chapter 14. Creating ASN.1 Dissectors. – Текст: электронный // Wireshark: [сайт]. – URL: https://www.wireshark.org/docs/wsdg_html_chunked/CreatingAsn1Dissectors.html (дата обращения: 28.06.2023).
[34]. Свид. 2019614453 Российская Федерация. Свидетельство об официальной регистрации программы для ЭВМ. Ядро системы глубокого разбора пакетов «ПРОТОСФЕРА» / А. И. Аветисян, С. С. Гайсарян, А. И. Гетьман, Ю. В. Маркин, Д. О. Обыденков, В. А. Падарян, А. Ю. Тихонов; Правообладатель: Федеральное государственное бюджетное учреждение науки Институт системного программирования им. В.П. Иванникова Российской академии наук (RU). – №2019613262; заявл. 28.03.2019; опубл. 04.04.2019, Реестр программ для ЭВМ. – 1 с.
[35]. KnownBugs - OutOfMemory. – Текст: электронный // Wireshark Wiki: [сайт]. – URL: https://wiki.wireshark.org/KnownBugs/OutOfMemory.md (дата обращения: 28.06.2023).
[36]. Multithreading. – Текст: электронный // Wireshark Wiki: [сайт]. – URL: https://wiki.wireshark.org/Development/multithreading (дата обращения: 28.06.2023).
[37]. Garg R. P., Sharapov I. A. Techniques for optimizing applications: high performance computing. – Palo Alto: Sun Microsystems Press, 2002. – С. 394.
[38]. MapReduce Tutorial. – Текст: электронный // Apache Hadoop: [сайт]. – URL: http://apache.github.io/hadoop/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html (дата обращения: 29.06.2023).
[39]. Larin D. V., Get'man A. I. Tools for Capturing and Processing High-Speed Network Traffic //Programming and Computer Software. – 2022. – vol. 48. – №. 8. – pp. 756-769.
[40]. Llopis N. Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP) / Llopis N. – Текст: электронный // Games from Within: [сайт]. – URL: https://gamesfromwithin.com/data-oriented-design 04.12.2009 (дата обращения: 29.05.2023).
[41]. Llopis N., Touch S. High-performance programming with data-oriented design //Game Engine Gems. – 2011. – vol. 2. – pp. 251-261.
[42]. Fabian R. Data-oriented design //framework. – 2018. – vol. 21. – pp. 1.7.
[43]. Spicy – Generating Robust Parsers for Protocols & File Formats. – Текст: электронный // Spicy: [сайт]. – URL: https://docs.zeek.org/projects/spicy/en/latest/index.html (дата обращения: 29.06.2023).
[44]. BPF: the universal in-kernel virtual machine. – Текст: электронный // Linux Weekly News: [сайт]. – URL: https://lwn.net/Articles/599755/ (дата обращения: 28.06.2023).
[45]. bpftrace. – Текст: электронный // github.com: [сайт]. – URL: https://github.com/iovisor/bpftrace (дата обращения: 28.06.2023).
[46]. Maughan D., Schertler M., Schneider M., Turner J. Internet Security Association and Key Management Protocol (ISAKMP) IETF RFC № 2408 // IETF. - 1998 г. - № 2408
[47]. Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., Berners-Lee T. Hypertext Transfer Protocol - HTTP/1.1 IETF RFC № 2616 // IETF. - 1999 г. - № 2616

Информация об авторах / Information about authors

Роман Евгеньевич ПОНОМАРЕНКО – аспирант, стажёр-исследователь ИСП РАН. Научные интересы: архитектура программного обеспечения, оптимизация программ, глубокий анализ сетевого трафика.

Roman Evgenevich PONOMARENKO – PhD student, intern researcher at ISP RAS. Research interests: software architecture, program optimization, deep packet inspection.

Владислав Игоревич ЕГОРОВ – аспирант, стажёр-исследователь ИСП РАН. Научные интересы: обработка, анализ и хранение результатов анализа сетевого трафика.

Vladislav Igorevich EGOROV – PhD student, intern researcher at ISP RAS. Research interests: processing, analysis and storage of network traffic analysis results.

Александр Игоревич ГЕТЬМАН – кандидат физико-математических наук, старший научный сотрудник ИСП РАН, ассистент ВМК МГУ и МФТИ, доцент ВШЭ. Сфера научных интересов: анализ бинарного кода, восстановление форматов данных, анализ и классификация сетевого трафика.

Aleksandr Igorevich GETMAN – Ph.D in physical and mathematical sciences, senior researcher at ISP RAS, assistant at CMC MSU and MIPT, associate professor at HSE. Research interests: binary code analysis, data format recovery, network traffic analysis and classification.