Автоматизация тестирования соответствия для телекоммуникационных протоколов.

Пакулин Н.В., Шнитман В.З., Никешин А.В.

Аннотация. В данной статье обобщается опыт разработки тестовых наборов для тестирования соответствия реализаций спецификациям протоколов Интернета. Во всех проектах, представленных в статье, использовалась технология UniTESK в качестве базы для построения тестов. В ходе разработки тестовых наборов были выявлены особенности протоколов, затрудняющие тестирование реализаций с помощью технологии UniTESK, а также особенности инструментов, поддерживающих ее работу, однако все эти особенности удавалось успешно преодолеть, не выходя за рамки ее ограничений.

Ключевые слова: тестирование соответствия, UniTESK, формальные методы, автоматизация тестирования, тестирование, основанное на моделях, тестирование протоколов, формальные спецификации, моделирование протоколов

1. Введение

Создание распределенных систем, в которых различные функции, связанные с хранением и обработкой информации, взаимодействием с пользователем и т. п., распределены по различным компьютерным системам, требует организации обмена информацией между компонентами такой системы. Для успешного взаимодействия между компьютерными системами их коммуникации должны основываться на хорошо определенных правилах. Совокупность таких правил, включая представление данных для передачи, правила отправки данных одной компьютерной системой и правила приема данных другой компьютерной системой, формируют *протокол* передачи данных.

Реализация протокола — это программный (в некоторых случаях программно-аппаратный) компонент компьютерной системы, который реализует конкретные процедуры отправки и приема данных. Для того чтобы реализации протоколов разных производителей компьютерных систем одинаково интерпретировали правила необходимо, чтобы этот набор правил был зафиксирован в виде *спецификации*. Как правило, спецификация представляет собой текстовый документ на естественном языке, хотя предпринимались попытки внедрить подходы к разработке спецификаций на

языках с формализованной семантикой (SDL[1], LOTOS[2], Estelle[3], ASN.1[4], UML[5] и др.) В некоторых областях эти попытки преуспели, и были разработаны спецификации-стандарты протоколов на формальных языках, но для большинства протоколов формальные нотации не использовались. Например, спецификации протоколов Интернета публикуются исключительно в виде текстов на английском языке.

Однако, для того чтобы две системы могли успешно обмениваться информацией по некоторому протоколу, одного наличия спецификации протокола недостаточно. Необходимо, чтобы реализации протоколов, участвующих в обмене информацией, функционировали в соответствии со специфицированными правилами, то есть удовлетворяли спецификации протокола. Разумеется, это требование базируется на предположении о корректности спецификации протокола — отсутствии ошибок формата данных, блокировок, зацикливаний и иных дефектов, которые могут привести к тому, что даже реализации, удовлетворяющие спецификации на 100%, не смогут осуществлять обмен данными. В данной статье мы не будем касаться вопросов проверки собственно спецификаций на корректность. Можно указать книги [6], [7], [8], в которых представлены методы анализа непосредственно протоколов.

Для установления соответствия протокола стандарту одним из наиболее широко используемых подходов является тестирование. В общем случае, тестирование — это процесс экспериментального исследования поведения реализации. Такое исследование, как правило, проводится в специально подготовленном окружении, которое имитирует нормальные или аномальные сценарии взаимодействия с испытываемой реализацией. Тестирование, нацеленное на проверку соответствия реализации требованиям спецификации, называется *тестированием соответствия* (conformance testing). При таком тестировании специально подготовленные данные подаются на вход испытываемой *целевой* системы (system under test, SUT), результаты обработки этих данных в целевой системе собираются и выносится *вердикт* — собранные результаты проверяются на соответствие эталонным (ожидаемым) результатам, или вычисляется степень их соответствия требованиям.

Обычно терминологически верное, но слишком длинное словосочетание «тестирование соответствия реализаций протокола спецификации этого протокола» сокращают до «тестирования соответствия протокола», несмотря на то, что тестируется не протокол (как некоторый концептуальный объект), а его реализации. В данной статье в дальнейшем «*тестирование соответствия протокола*» будет пониматься именно как тестирование реализаций на соответствие спецификации. В тех случаях, когда из контекста ясно, что речь идет о протоколах, будет использоваться минимальная форма «*тестирование соответствия*».

Очевидно, что тестирование может проводиться только в течение ограниченного периода времени. Для систем, используемых на практике, это означает, что невозможно провести испытания реализации во всех мыслимых ситуациях. Поэтому для тестирования очень важной является задача систематического построения тестовых воздействий.

Традиционный подход к тестированию соответствия заключается в разработке небольших специализированных программ, проверяющих отдельные требования — *элементарных тестов* (test cases). Каждый тест выполняется независимо от остальных тестов. Все тесты начинаются в одном и том же начальном состоянии реализации, затем каждый тест, в зависимости от своей цели, оказывает воздействия на целевую систему, чтобы привести её в желаемое состояние. Если такое состояние достигнуто, тест выполняет собственно тестовую процедуру, после чего возвращает целевую систему в начальное состояние. Более подробно этот подход к тестированию соответствия рассматривается в разделе 2.1.

Разработка тестов вручную широко распространена в современной телекоммуникационной индустрии, так как, во-первых, такой подход не требует специальных знаний (формальных нотаций, конечных автоматов, темпоральных логик и т. п.), и, во-вторых, тесты появляются сразу после начала процесса разработки тестового набора, что позволяет немедленно приступить к поиску ошибок.

Однако у ручной разработки тестов есть существенные недостатки. Высокая сложность современных протоколов требует, чтобы тестовый набор состоял из большого числа испытаний. Так, тестовый набор для сетевого протокола IPv6 насчитывает более 6000 тестов. Отсутствие автоматизации делает разработку тестового набора задачей, превосходящей по сложности собственно разработку реализации. Вердикты о корректности наблюдаемого поведения не опираются на строгую модель протокола, что затрудняет проведение инспекций тестового набора. Покрытие требований оценивается эвристически, без привлечения строгих формальных процедур.

Элементарные испытания рассматриваются в академической среде как один из традиционных методов тестирования, в то время как тестирование на основе моделей рассматривается как новый метод, решающий множество неразрешимых с помощью традиционных методов проблем [9-11]. Ряд походов к автоматизации тестирования соответствия кратко рассмотрен в разделе 2.2.

В разделе 3 данной статьи представлен подход к автоматизации тестирования соответствия, развиваемый авторами. Этот подход основан на использовании технологии тестирования UniTESK[12-14] и включает в себя разработку формальной модели протокола и разработку тестов как конечных автоматов, чье состояние строится из состояния модели протокола. В разделе 4 кратко описываются проекты по созданию тестовых наборов для различных протоколов в рамках рассматриваемого подхода. В разделе 5 рассматривается

эволюция методов и средств, использовавшихся авторами при разработке тестовых наборов.

2. Обзор существующих методов тестирования соответствия

Исследования и практические проекты по установлению соответствия для протоколов стали появляться еще в 60-х годах прошлого века. Наиболее значительные результаты были получены прежде всего в теоретических исследованиях — были поставлены и решены задачи установления соответствия между двумя моделями, если модели представимы в виде конечных автоматов, разработан ряд методов построения тестов, гарантирующих обнаружение всех ошибок определенного класса [15-17].

По мере развития программных систем становилось очевидно, что конечные автоматы позволяют описывать только часть функциональности взаимодействующих систем. Были разработаны более выразительные средства для формального представления взаимодействующих программно-аппаратных систем: алгебры взаимодействующих процессов [18-20] и системы размеченных переходов[21-22]. На базе этих средств были получены новые результаты по формализации семантики протоколов, позволяющие более точно моделировать асинхронные взаимодействия.

Всплеск в области тестирования соответствия произошел в конце 80-х — начале 90-х годов XX столетия. Он был вызван разработкой и стандартизацией в середине 80-х годов стека протоколов OSI [23], который был призван объединить существующие на тот момент сетевые технологии в единую систему. В рамках процессов разработки и публикации стандартов были созданы и стандартизованы языки формальных спецификаций, предназначенные для описания протоколов на основе моделей в виде взаимодействующих автоматов или взаимодействующих процессов — SDL [1], Estelle [2] и LOTOS [3], опубликовано большое количество работ, посвященных разнообразным методам тестирования, основанным на конечных автоматах и размеченных системах переходов, например, [24-30]. Кроме того, в указанный период проводится активная систематизация исследований в этой области [31-34], разрабатываются общие схемы тестирования, и на их основе принимаются базовые стандарты, регулирующие проведение тестирования реализаций протоколов — ISO 9646 [35] ITU-T/Z.500 [36].

В начале 90-х годов созданные ранее теоретические конструкции начинают активно применять на практике для разработки и проверки корректности конкретных протоколов. Тогда же появляются регулярные конференции по данной тематике, например, Международный семинар по спецификации, тестированию и верификации протоколов (International Workshop on Protocol Specification, Testing and Verification).

2.1. Общие вопросы тестирования соответствия

Традиционно [37] разделяются два подхода к построению тестов: подход «белого ящика» и подход «черного ящика». Тестирование «белого ящика», иногда называемое *структурным* тестированием [38] ориентируется на внутреннюю структуру тестируемых программ. Цель такого тестирования — исследовать программный код, в частности, вызвать при тестировании все возможные ветвления или (как максимум) пройти все пути исполнения в программе. Тесты могут строиться на основании различных элементов исходных текстов программы (ветвлений, вызовов подпрограмм, строк кода), для построения тестов может использоваться также структура бинарного кода оттранслированной программы.

При тестировании методом «черного ящика» внутренняя структура реализации не известна, либо не используется при построении тестовых данных и вынесении вердикта. Основная цель при тестировании «черного ящика» заключается в том, чтобы установить, что целевая система корректна с точки зрения внешнего наблюдателя, который может судить об этом только на основании сопоставления внешне наблюдаемого поведения реализации и требований спецификации к этому поведению.

В подавляющем большинстве случаев тестирование соответствия протоколов спецификациям реализуется как тестирование черного ящика. Можно указать несколько причин:

- Спецификации многих протоколов явно указывают, что структуры данных для внутренних состояний, алгоритмы обработки данных и другие ненаблюдаемые извне аспекты функциональности, используются в спецификации как *концептуальные* для описания внешне наблюдаемого поведения. От реализаций не требуется использовать именно эти структуры или алгоритмы при условии, что внешне наблюдаемое поведение реализаций будет таким, как если бы реализации их использовали.

- Зачастую спецификации протоколов являются международными или общепринятыми стандартами. В таком случае тестирование соответствия приобретает качество сертификационного тестирования, при котором одни и те же тесты используются для различных реализаций. Это означает, что тесты не могут содержать зависимости от реализации, т. е. тестировать «белый ящик».

- Тесты не имеют доступа к внутреннему устройству целевой реализации, так как выполняются на отдельной *инструментальной* машине, которая физически отделена от узла сети, на котором установлена целевая реализация. Более того, при тестировании на встроенных системах нет практической возможности установить какое-либо дополнительное ПО для инспектирования внутреннего состояния целевой реализации.

Набор тестов для тестирования соответствия должен обладать рядом свойств:

1. Прослеживаемостью требований. Тесты должны соотноситься с требованиями стандарта, должно быть наглядно видно, какие требования какими тестами покрываются.

2. Многообразием настроек на особенности реализаций (MAY, SHOULD, MUST и другие). Должна быть опция для определения множества требований, поддерживаемых тестируемой реализацией. В это множество не должны попадать требования, не поддерживаемые тестируемой реализацией.

3. Полнотой тестового набора в смысле покрытия требований. Полученный тестовый набор должен покрывать как минимум все функциональные требования, помеченные в стандарте как обязательные для каждой реализации.

2.1 Неавтоматизированная разработка тестов для тестирования соответствия

Разработчики стека протоколов OSI [23], в настоящее время известного каждому студенту как «семиуровневая модель», одновременно с разработкой протоколов озаботились вопросом обеспечения совместимости реализаций. Была разработана и закреплена в стандарте ISO 9646[36] универсальная методика тестирования реализаций протоколов. Кратко рассмотрим основные идеи этой методики.

1. Тестовый набор состоит из переносимых, абстрактных, независящих от конкретной реализации протокола, абстрактных тестовых наборов. Абстрактный тестовый набор состоит из отдельных тестов, заданных в однозначной, не зависящей от конкретной реализации форме.

2. Абстрактный тестовый набор состоит из абстрактных элементарных испытаний (abstract test cases), которые записаны в нотации, не зависящей от конкретной реализации, и взаимодействуют с целевой реализацией через набор коммуникационных каналов.

a) В каждом элементарном испытании реализуется последовательность тестовых воздействий на целевую систему и вынесение вердикта о корректности наблюдаемого поведения целевой системы.

b) Для каждого элементарного испытания явно или неявно задаётся цель тестирования (test purpose), которая неформально определяет группу функциональных требований к реализации протокола. Успешное или неуспешное завершение элементарного испытания трактуется следующим

образом: реализация протокола корректно или некорректно реализует требования, соответствующие цели тестирования.

Подход, основанный на абстрактных представлениях тестов, позволил выработать строгие методы тестирования протоколов, четко определяющие различные компоненты тестового набора, их функции и степень зависимости от конкретной тестируемой реализации. Этот подход был закреплён в международном стандарте ISO/IEC 9646 [35,39].

Каждое абстрактное элементарное испытание состоит из:

- преамбулы – последовательности шагов, приводящих реализацию из начального состояния в целевое состояние,
- множества тестовых шагов, которые проверяют поведение реализации в целевом состоянии, и
- постамбулы, переводящей реализацию обратно в начальное состояние.

На каждом шаге испытания осуществляется проверка правильности поведения целевой системы. В частности, расхождение наблюдаемого поведения с ожидаемым в преамбуле делает бессмысленным дальнейшее выполнение теста и, как правило, приводит к остановке выполнения с негативным вердиктом.

Каждое элементарное испытание должно завершиться вынесением вердикта. Все действия элементарного испытания описываются абстрактно, в терминах, не зависящих от реализации. Тем самым абстрактный тестовый набор представляет собой однозначную и не зависящую от конкретной реализации спецификацию тестовой последовательности.

Для представления абстрактных тестовых наборов используются языки TTCN-1 [40], TTCN-2 [41] и TTCN-3 [42], графическая нотация UML Testing Profile [43], разрабатываются представления в виде диалектов XML [44], а также непосредственно языки программирования – Java [45], Perl [46].

В традиционных подходах, где тесты разрабатываются вручную, построение конкретных тестов из множества возможных основывается на *целях тестирования* (test purposes). Цель тестирования — это класс ситуаций, соответствующих конкретному функциональному требованию, в которых реализация должна вести себя сходным образом. Например, ответ на определенное сообщение зависит от конкретных данных в запросе, однако строится по одному алгоритму. Если таких алгоритмов несколько (например, один из них — построение сообщения об ошибке в запросе), то необходимо проверить поведение реализации в каждом из этих сценариев. Таким образом, одному функциональному требованию могут соответствовать несколько тестовых ситуаций, поэтому перед разработкой тестов для требований выписываются перечни ситуаций, которые необходимо проверить. Эти

перечни тестовых ситуаций оформляются в виде целей тестирования и позже программируются на выбранном языке описания тестов. Правда, необходимо заметить, что далеко не во всех проектах по тестированию соответствия выписывают цели тестирования явным образом, а там, где они выписываются, затруднительно проверить полноту выделенного набора тестовых ситуаций.

Достоинством указанной методологии является то, что спецификация тестовых последовательностей не зависит от конкретной реализации. В качестве недостатков можно отметить:

- методология абстрактных тестов, формализованная в серии стандартов ISO 9646, не предполагает автоматизированного построения элементарных испытаний;
- в методологии нет формальной процедуры для оценки полноты тестирования;
- методология не может служить надёжным базисом для анализа протокола, так как не включает разработку формальной модели целевого протокола;
- все элементарные испытания должны разрабатываться экспертами в целевой области, так как в теле каждого элементарного испытания должен выноситься вердикт о корректности поведения целевой системы.

Вообще говоря, подход создания тестового набора, включающего отдельные, разрабатываемые вручную небольшие тесты, позволяет строить тестовые наборы для любых протоколов. Однако, оборотной стороной является значительные затраты ресурсов на разработку такого тестового набора. При разработке абстрактных элементарных тестов для сложных протоколов рано или поздно тестировщики сталкиваются с существенными трудностями:

1. Вердикты о соответствии поведения реализации спецификации протокола выносятся в каждом тесте независимо, поэтому проверки корректности в тестах часто дублируются, что усложняет поддержку тестового набора, его расширение и модификацию.

2. Вердикты выносятся на основе ожидаемого поведения. Обработка «неожиданных сообщений», характерных для недетерминированных систем и параллельно взаимодействующих компонентов, требует дополнительных усилий при разработке тестового набора.

Распространённые индустриальные технологии тестирования, основанные на методологии элементарных тестов, не поддерживают использование формальных моделей при разработке и прогоне тестов. Цикл разработки тестового набора не пересекается с разработкой формальной модели системы. Формальные модели не включаются в состав тестового набора. Проверки корректности поведения реализации в тестах должны основываться на требованиях, изложенных в регламентирующих документах, а формальные спецификации, как правило, не входят в нормативные разделы стандартов, поэтому создатели тестовых наборов в процессе разработки обращаются

непосредственно к тексту стандарта, а не к формальным моделям. Тесты не опираются на строгую модель целевой системы при проведении проверок и определении достигнутого уровня тестирования.

2.2. Методы автоматизации тестирования соответствия

Высокая сложность современных протоколов требует, чтобы тестовый набор состоял из большого числа испытаний. При разработке тестового набора без формальной модели возникают следующие трудности:

1. Отсутствие автоматизации делает разработку тестового набора задачей, превосходящей по сложности собственно разработку реализации. Наличие формальной спецификации позволяет автоматизировать процедуру генерации тестовых воздействий и минимизировать ручной труд при разработке тестов.

2. Вердикты о корректности наблюдаемого поведения не опираются на строгую модель протокола, что затрудняет проведение инспекций тестового набора. Наличие модели, во-первых, дает возможность обеспечить валидацию корректности тестового набора, так как для этого достаточно проверить корректность модели, а во-вторых, позволяет многократно использовать модель для проверки правильности поведения реализаций, уменьшая тем самым размер тестового набора.

3. Из-за отсутствия формальной модели протокола нет строгой процедуры для оценки покрытия функций протокола тестовым набором. Наличие формальной модели протокола позволяет однозначно задавать связи между требованиями и тестами, автоматически отслеживать качество тестирования в терминах покрытия спецификации.

Для формального описания протоколов было разработано множество нотаций. Некоторые из них получили широкое распространение в индустрии, и для них были изданы стандарты международных организаций. Разработка формальных спецификаций телекоммуникационных протоколов позволяет дать ясное, однозначное и полное описание протокола, выявить ошибки и неполноту текстовой спецификации протокола. Формальная спецификация включена как нормативная или информативная часть в ряд стандартов телекоммуникационных протоколов, например GSM Handover procedures [47] или Harmonized Programmable Communication Interface (HPCI) for ISDN [48].

Задачу тестирования соответствия спецификации протокола можно рассматривать как задачу сравнения двух расширенных конечных автоматов. В обзорной работе Д.Ли и М.Янакакиса[32], посвящённой вопросам тестирования конечных автоматов, представлены алгоритмы и подходы к построению тестовых последовательностей для проверки соответствия двух конечных автоматов. Главная проблема проверки соответствия двух автоматов заключается в том, что с ростом числа состояний в автоматах и/или числа переходов длина тестовых последовательностей растёт очень быстро.

Для уменьшения длины тестовых последовательностей используется целенаправленная генерация тестов, при которой обход автомата ограничивается специальными условиями. Примеры реализации целенаправленной генерации есть в инструментах TGV [49], Gotcha-TCBEANS[50,51], SpecExplorer[52]. Но и при целенаправленной генерации тестов остаётся проблема недетерминизма протокола или реализации. Суть проблемы заключается в том, что если тесты генерируются заранее, до исполнения, то для недетерминированного протокола необходимо генерировать все возможные цепочки допустимых переходов, что может привести к значительному увеличению числа тестовых последовательностей.

Для решения этой проблемы был предложен подход к генерации тестов, получивший название «динамической генерации» (on the fly), при котором тестовые воздействия строятся непосредственно во время исполнения тестового набора. Как правило, тестовые воздействия генерируются при обходе исполнимой и, возможно, недетерминированной модели протокола. При этом тестовые воздействия подаются одновременно на целевую реализацию и на модель протокола, а реакции целевой реализации передаются в модель. Это необходимо для того, чтобы состояние модели обновлялось в соответствии с правилами протокола. Передача реакций позволяет обновлять состояние модели в случае недетерминированных протоколов. Например, спецификация SMTP - протокола отправки электронной почты — допускает при обработке запроса клиента вернуть код ответа, обозначающий внутреннюю ошибку сервера или нехватку ресурсов и разорвать соединение. Соответственно, для корректного построения следующего состояния модели необходимо знать код ответа, который вернул сервер.

Основные трудности, которые возникают при практическом использовании динамической генерации, связаны с определением подмножества модели, подлежащего обходу. Для протоколов с большим числом состояний полный обход модели за приемлемое время невозможен, поэтому были разработаны различные методы сокращения обхода. В следующем разделе мы кратко обсудим методы сокращения обхода, применявшиеся в проектах по тестированию соответствия протоколов с использованием технологии UniTESK.

Для тестирования протоколов с большим числом состояний и переходов (исчисляемых тысячами), отдельные авторы предлагают использовать случайный обход явной модели протокола. Этот подход был разработан и реализован под руководством Я.Третманса в инструменте тестирования TorX [53,54] совместно с динамической генерацией тестов. Было показано [55], что если реализация некорректна, то вероятность обнаружить ошибку стремится к 1 при стремлении числа тестов к бесконечности. Оценки числа тестов, которые с заданной вероятностью находят ошибку, не приводятся. Инструмент TorX использовался для тестирования ряда простых протоколов, но к настоящему времени результаты применения к сложным промышленным протоколам или протоколам Интернета не известны.

3. Применение технологии UniTESK к тестированию соответствия

В данной статье обобщается опыт разработки тестовых наборов для тестирования соответствия протоколов Интернета. Все проекты, представленные в статье, использовали технологию UniTESK [12-14] в качестве базы для построения тестов. В этом разделе мы обсудим общие вопросы тестирования соответствия с использованием UniTESK, а сами проекты и их результаты представим в следующем разделе.

Технология UniTESK поддерживает нотацию формальных спецификаций, автоматическую динамическую генерацию тестовых воздействий (on-the-fly) и автоматический анализ результатов. UniTESK предоставляет средства для формальной спецификации протоколов [56] в виде контрактных спецификаций переходов расширенного конечного автомата [57-59].

Как уже отмечалось, спецификации многих современных протоколов публикуются на естественном языке без использования формальных нотаций. Для применения модели в тестировании соответствия необходимо трансформировать *неформальную* спецификацию (на естественном языке) в *формальную*, основанную на машино-читаемой нотации. Однако, одной лишь машино-читаемой формальной нотации недостаточно. Формальная спецификация может использоваться для тестирования соответствия при выполнении ряда требований:

1. однозначность представления спецификации: эксперты в данной предметной области одинаковым образом трактуют спецификацию;
2. проверяемость требований: для каждого требования в спецификации эксперты в предметной области могут предложить процедуру проверки;
3. адекватность исходным текстовым спецификациям: требования в разработанной спецификации соответствуют требованиям, представленным в исходной спецификации;
4. полнота представления исходной спецификации: все требования исходной текстовой спецификации представлены в разработанной спецификации;
5. прослеживаемость исходной спецификации: есть строго определѐнная процедура, при помощи которой можно для каждого требования из исходной спецификации определить, как оно представлено в разработанной спецификации и наоборот, для каждого логического или исполнимого выражения модели найти исходные требования, представленные им;
6. Пригодность для тестирования соответствия: спецификацию можно использовать для автоматизации тестирования соответствия. Это, в свою очередь, подразумевает:

a) возможность генерировать оракулы из спецификации;

b) возможность генерировать тестовые данные из спецификации;

c) наличие инструментальной поддержки генерации оракулов и

тестовых данных из спецификации.

В технологии UniTESK требования к поведению формализуются в виде спецификаций контракта между участниками протокола. Контрактные спецификации развивают идеи Т.Хора[60] о спецификации поведения при помощи логических ограничений (пред- и постусловий). При описании внешне наблюдаемого поведения системы контрактными спецификациями взаимодействие системы с окружением представляется как набор операций в некотором формальном интерфейсе. С каждой операцией формального интерфейса связаны предусловие и постусловие. Предусловие операции накладывает ограничения на ситуации, в которых эта операция может быть вызвана, постусловие операции накладывает ограничение на результаты операции и изменение состояния системы после выполнения операции.

Контрактные спецификации UniTESK удовлетворяют приведенным выше требованиям к формальной спецификации. Действительно, контрактные спецификации являются однозначными, так как записываются средствами строго определённой нотации. Далее, пригодность контрактных спецификаций для тестирования определяется тем, что из контрактных спецификаций можно построить оракул для проверки корректности внешне наблюдаемого поведения, и есть инструментальная поддержка UniTESK для тестирования на основе контрактных спецификаций.

Рассмотрим достоинства контрактных спецификаций в контексте анализа и формализации функциональных требований и тестирования протоколов:

1. Контрактные спецификации позволяют сравнительно легко описывать различные виды недетерминированного поведения, в том числе вызванные неполнотой исходной спецификации или особенностями реализации.

2. Существуют эффективные способы автоматического построения тестовых оракулов из контрактных спецификаций [12]. Это позволяет автоматизировать процесс проверки соответствия реализации требованиям.

3. Создание контрактных спецификаций требует от разработчика мышления в стиле требований к результату, что отличается от характерных стилей мышления разработчиков программных систем. Это позволяет выявить особенности требований, которые остались бы незамеченными при анализе требований средствами прототипирования или разработки экспериментальных реализаций. В частности, мы считаем, что контрактная спецификация позволяет лучше анализировать требования к недетерминированному поведению, чем явные исполнимые спецификации или прототипы.

Условимся называть элементарные акты взаимодействия реализации протокола с окружением *событиями*. Примерами событий, связанных с протоколами, могут служить

- операции, входящие в программный интерфейс реализации (как правило, операции по отправке или получению данных протоколов более высокого уровня, настройка параметров протокола, аудит внутренних событий); возникновение события состоит в приеме сообщения реализацией

протокола;

- обработка входящих сообщений протокола; возникновение события состоит в приеме сообщения реализацией протокола;
- или отправка сообщений; возникновение события – выход сообщения в сеть или передача его протоколу нижнего уровня.

Обращение к реализации протокола со стороны вышележащего уровня или приложения естественно представляется как процедурный вызов в некотором программном интерфейсе. Зачастую именно так события этого вида и реализуются. Но наличие событий, связанных с отправкой или приемом пакетов существенно отличает тестирование соответствия от других приложений технологии UniTESK. Для интеграции событий, связанных с сообщениями, была введения концепция *формального интерфейса*, который объединяет взаимодействия, построенные на вызове процедур, с взаимодействиями, основанными на обмене сообщениями.

Элемент формального интерфейса описывается при помощи сигнатуры, состоящей из его имени и набора типов данных, передаваемых этим событием, и контракта события. Напомним, что в UniTESK контракты состоят из двух компонентов — *предусловия* и *постусловия*. Предусловие является предикатом, зависящим от данных события и состояния протокола, предшествовавшего возникновения события. Постусловие является предикатом, зависящим от предшествовавшего возникновению события состояния реализации протокола, еѐ же состояния после наступления этого события и данных возбужденного события.

Предусловие определяет ответственность другого компонента, возбуждающего это событие. Событие можно возбуждать только тогда, когда его предусловие выполнено, иначе корректная работа реализации протокола не гарантируется (её поведение не определено). Постусловие описывает ответственность реализации протокола — она гарантирует его выполнение после возникновения данного события, если непосредственно перед его возбуждением было выполнено предусловие. Элементы формального интерфейса, представляющие обработку запросов от верхнего уровня или обработку входящих сообщений протокола, оснащаются и пред-, и пост-условием. Практика использования UniTESK, накопленная к моменту первых приложений UniTESK к протоколам, позволила прямо обобщить существовавшую на тот момент семантику контрактных спецификаций для программных интерфейсов на обработку входящих сообщений.

Контракт события, представляющего отправку сообщения, не содержит предусловия — он состоит из одного постусловия, определяющего ответственность реализации протокола. Она должна гарантировать, что после возбуждения данного события постусловие этого события выполнено. Отсутствие предусловия в контракте таких событий обосновывается тем, что выходные события возбуждаются самой реализацией протокола, и она отвечает за их корректность.

В ряде проектов в состав формального интерфейса вводились элементы для описания внутренних событий: тайм-аутов и передачи данных внутри реализации по стеку в случае многоуровневых сетевых систем. В настоящее время нет единой методики, как формировать контракт для таких элементов, так как и вызывающая сторона, и обработчик находятся внутри целевой реализации.

Общие части предусловий и постусловий, определяющие ограничения целостности для элементов состояния протокола, выносятся в инварианты состояния. Реализация протокола должна обеспечивать выполнение инвариантов компонента между возникновением событий, но сохранение инвариантов во время обработки входных событий не определено. Последнее ограничение имеет существенное значение для тестирования протоколов, так как большинство современных протоколов спроектированы для параллельной обработки нескольких сообщений (для ускорения обработки и минимизации задержек реализации могут обрабатывать события в нескольких физически параллельных потоках). Соответственно, для построения модели протокола важно представлять семантику параллельной обработки сообщений.

Параллельная обработка событий, соответствующих элементам формального интерфейса, интерпретируется в рамках семантики чередований, а именно:

- События, входящие в формальный интерфейс, являются атомарным, т.е. их возникновение и обработка являются единым процессом, не прерываемым ничем другим, в том числе никакими другими событиями.
- Входные и выходные события составляют частично упорядоченное множество; порядок на этом множестве означает, что предшествующее событие гарантированно завершилось до начала последующего. События несравнимы, если такой информации нет.
- Выходные события, которые порождает целевая система, и её конечное состояние соответствует некоторому полному упорядочению такого частично упорядоченного множества событий.

3.1. Построение формальной модели из текстовой спецификации

Спецификации современных протоколов, как правило, описывают требуемое поведение реализаций на естественном языке. Для построения тестового набора средствами UniTESK необходимо текстовую спецификацию *формализовать* — то есть преобразовать функциональные требования в текст на языке с более строгой семантикой, чем естественный.

Процедура формализации состоит из нескольких шагов. Разделение на шаги обусловлено природой задач, которые возникают при формализации функциональных требований протоколов:

1. Выделение набора функциональных требований к реализации протокола. В текстовых спецификациях требования, как правило, перемежаются пояснениями и обоснованиями, которые не несут нормативной

нагрузки и призваны облегчить понимание требований читателям спецификации, поэтому перед началом разработки формальной модели протокола необходимо отделить требования от всего остального текста. Результатом этого шага является *каталог требований*.

2. Разработка формального интерфейса протокола. А именно, необходимо определить множества входных, выходных и внутренних событий протокола и разработать их представление в виде сигнатур элементов формального интерфейса. Результатом этого шага является представление операций протокола в виде сигнатур функций.

3. Разработка контракта. Текстовые требования, выделенные на первом шаге, переводятся в логические формулы пред- и постусловий элементов формального интерфейса. Результатом этого шага является контрактная спецификация, в которой сигнатуры функций оснащаются пред- и постусловиями.

Предикаты, составляющие контракт, представляются в виде булевских функций. Так как обработка сообщений в общем случае зависит от текущего состояния протокола, в предикатах используется *абстрактное состояние* — набор переменных, моделирующих состояние протокола. Прилагательное «абстрактный» подчеркивает отличие этого состояния от тех переменных, которые используются внутри конкретной реализации. Как правило, абстрактное состояние повторяет концептуальные структуры данных и переменные, которые используются в текстовой спецификации для описания требуемого поведения реализации.

Определение критериев покрытия. Критерий покрытия задаёт разбиение множества переходов в модели протокола и связывает с каждым элементом разбиения уникальную метку. Назначение критерия покрытия – предоставить классификацию переходов, произошедших в ходе тестирования. Переходы, принадлежащие одному и тому же элементу разбиения, считаются эквивалентными. Так как возможны разные виды эквивалентностей, то с одним элементом формального интерфейса может быть связано несколько критериев покрытия.

Выше мы уже упоминали понятие *цели тестирования*. В UniTESK цель тестирования получает строгое определение — при выполнении тестов должны быть осуществлены переходы, соответствующие всем элементам покрытия. Благодаря тому, что спецификация является формальным объектом, отслеживание полноты тестового набора полностью автоматизируется.

Здесь необходимо сделать одно важное замечание. Формальная спецификация в UniTESK используется для тестирования. Как уже упоминалось выше, модель протокола должна включать в себя модель состояния. Для того чтобы в процессе тестирования модель оставалась адекватной реализации, необходимо после каждого наблюдаемого события обновлять значения переменных состояния модели в соответствии со спецификацией. С одной

стороны, постусловие можно рассматривать как некое уравнение, связывающее состояние до возникновения события, параметры события и состояние после события. Если решить это уравнение, зная начальное состояние и параметры события, то можно вычислить новое состояние. На практике, однако, этот подход реализовать не удалось из-за высокой сложности постусловий, моделирующих реальные протоколы. Поэтому, для успешного использования в тестах контракты необходимо дополнить ещё одним компонентом — *функциями реконструкции состояния.* Эти функции нельзя рассматривать как явную спецификацию протокола, так как они не конструируют результат операции по входным данным, а реконструируют состояние реализации, зная её ответ на входное воздействие. Функции реконструкции состояния всегда детерминированы, в отличие от спецификации протокола.

3.2. Построение тестовых сценариев для тестирования соответствия

В рамках UniTESK тестовый сценарий задает некоторый конечный автомат, обход графа состояний которого порождает тестовую последовательность. Состояние автомата вычисляется из состояния модели, символы на дугах автомата определяют тестовые воздействия, которые необходимо оказать на реализацию. Обходчик автомата стремится подать все тестовые воздействия во всех состояниях автомата, достижимых из начального. Обходчик является библиотечным компонентом и не зависит от конкретного протокола или тестового набора.

Необходимо отметить, что описание автомата в тестовом сценарии отличается от обычного конечного автомата: в описаниях дуг отсутствует конечное состояние. Состояние автомата после перехода вычисляется по состоянию модели, а то, в свою очередь, строится функцией реконструкции состояния из реакций, продемонстрированных целевой системой в ответ на тестовое воздействие. Таким образом, конечное состояние перехода полностью определяется моделью протокола.

При проектировании тестовых сценариев для тестирования соответствия протоколов разработчики тестового набора решают следующие задачи:

1. Создание алфавита входных символов автомата теста: разработчики теста составляют множество ситуаций, которые они считают нужным проверить в данном тестовом сценарии. Каждой тестовой ситуации соответствует отдельный символ алфавита (более строго: автомат теста является расширенным конечным автоматом, поэтому символ алфавита ещё дополнен n-кой параметров, что позволяет варьировать аспекты тестовой ситуации — например, перебирать возможные адреса сообщений или значения каких-то полей, — поэтому помимо символов входного алфавита автомата задаются сигнатуры наборов параметров для каждого символа).

Этим алфавитом пользуется обходчик при построении и хранении графа переходов автоматов во время выполнения теста. О том, как входные символы автомата превращаются в конкретные тестовые воздействия см. ниже.

2. Определение состояния автомата теста: необходимо задать структуры данных для представления состояния теста и разработать функцию, которая строит конкретное состояние автомата теста из модельного состояния.

3. Разработка переходов тестового автомата: необходимо определить функции, которые по абстрактному символу (и набору параметров) построят конкретное тестовое воздействия — вызовут функцию из программного интерфейса реализации или сконструируют и отправят в реализацию тестовый пакет.

Важно, что функции, реализующие переход в автомате теста, не обращаются к реализации напрямую. Они вызывают методы из формального интерфейса: UniTESK предоставляет средства для отображения формального интерфейса на конкретные операции с реализацией, такие как вызов функции из программного интерфейса, отправку сообщения в реализацию или ожидание ответа.

4. Разработка средства настройки теста – параметры теста и механизм передачи параметров в тест при его запуске.

Очевидно, что для протоколов возможно практически бесконечное число тестовых ситуаций. Поэтому тест не может перебрать все возможные сообщения во всех возможных состояниях. Необходимо из каких-то соображений уменьшать размерность задачи, проводить факторизацию пространства входных воздействий. В UniTESK для факторизации используются критерии покрытия, введенные в модели. Тесты разрабатываются таким образом, чтобы тестовые воздействия привели к задействованию того или иного элемента покрытия.

В общем случае даже факторизация по элементам покрытия приводит к слишком большому автомату теста. Поэтому тестовый набор, как правило, состоит из нескольких тестовых сценариев, которые дополняют друг друга и в совокупности обеспечивают полное покрытие функциональных требований.

При тестировании протоколов целевая система может демонстрировать реакции спустя некоторое время после того, как на неё было оказано воздействие. Для облегчения моделирования систем с отложенными реакциями в UniTesK введено представление об *автоматах с отложенными реакциями*. Такие автоматы отличаются от обычных конечных автоматов тем, что переходы между состояниями представляют собой цепочку реакций.

Как правило, реализация протокола генерирует реакции по нескольким каналам. Например, у реализации сетевого протокола есть как минимум два канала реакций – пакеты, которые отправляется в сеть, и данные, которые передаются на верхний уровень. Реакции могут регистрироваться с

запозданием, причем для реакций различных типов величина задержки может быть различной. Тестовая система соотносит полученные реакции с элементами формального интерфейса, которые их моделируют. Данная операция кардинально отличается от ситуации, когда тест оказывает воздействие на реализацию: в случае стимула тест явным образом «говорит», какому элементу формального интерфейса соответствует воздействие. В ситуации с реакциями необходимо найти правильный элемент интерфейса, так как от этого зависит и корректность вердикта, и правильность обновления состояния модели. После того, как нужный элемент формального интерфейса найден, с реакцией связывается его постусловие, которое проверяет, допустима ли зарегистрированная реакция в текущем состоянии.

Напомним, что в UniTESK одновременность событий описывается в семантике чередования. Соответственно, тестовой системе необходимо определить допустимый порядок реакций различных видов. В процессе сериализации тестовая система строит различные цепочки реакций и проверяет их допустимость. Каждая реакция рассматривается как переход между промежуточными состояниями, конечное состояние последней реакции рассматривается как последнее состоянии в цепочке реакций и принадлежит множеству состояний конечного автомата. Если в ходе этой операции не удалось найти ни одной допустимой последовательности реакций, то тестовая система выносит вердикт о нарушении семантика чередования: нет такой линейно упорядоченной последовательности событий, которая была бы эквивалентна частично упорядоченному множеству реакций. Такая ситуация трактуется как рассогласование модели и целевой системы – набор зарегистрированных реакций не соответствуют спецификации.

Важная особенность тестирования протоколов заключается в том, что ошибочное поведение может проявляться не только в виде ошибок в полях сообщений или данных, но и в отсутствии ответа на запрос, или наоборот, генерации сообщения в ситуации, когда передача запрещена спецификацией. Нарушение запрета на передачу выявляется в тестах как ошибка перебора цепочек реакций — какую бы линейную последовательность событий ни построила тестовая система, все они окажутся недопустимыми: постусловие для «лишней» реакции будет нарушено в любом случае.

Для выявления ситуации с «отсутствующими» реакциями на тестовый сценарий накладывается ограничение на *стационарность начальных и конечных состояний*, то есть в начале перехода и в конце перехода целевая система не демонстрирует спонтанных (т.е. без воздействия извне) реакций. В частности, это означает, что существует такой интервал времени $\mathbf{T_0}$, что в течение $\mathbf{T_0}$ с момента оказания воздействия все реакции целевой системы на воздействие будут собраны. Стационарность состояния должна определяется по состоянию модели (что логично — состояние реализации недоступно для наблюдения), поэтому разработчики спецификации должны заранее предусмотреть удобные средства для оценки стационарности тестом.

Результатом работы теста является трасса тестовых событий – выбор абстрактного символа, построение конкретного тестового воздействия, вызов модели, вызов реализации и т. д. По набору трасс, порожденных сценариями из тестового набора, строится отчет, в котором важное место занимает отчет о покрытии.

3.3. Устройство тестового стенда

Стенд для тестирования протоколов включает целевое устройство, инструментальный узел и вспомогательные устройства. На целевом устройстве функционирует тестируемая реализация, а на инструментальном узле — тестовая система. В некоторых случаях в состав тестового стенда добавляются вспомогательные устройства, которые используются в ходе тестирования для оказания воздействий на целевое устройство и регистрации сообщений, высылаемых целевым устройством.

Инструментальный узел и целевое устройство подключены к одному или нескольким общим сегментам локальной сети. Несмотря на то, что тестовый сценарий и модель выполняются на инструментальном узле, часть тестовой системы может быть развернута на целевой системе. Именно, на целевом устройстве, могут быть размещены тестовые агенты для оказания воздействия на реализацию или сбора реакций системы, направленных на приложение.

Такой распределенный тестовый стенд может быть реализован как набор виртуальных машин, подключенных к общему сегменту виртуальной сети. Исполнение тестов на виртуальных машинах имеет ряд преимуществ по сравнению с запуском тестов в реальном физическом окружении:

1. Полный контроль над составом узлов в локальных сетях тестового стенда.

2. Возможность создания идентичных копий тестовых стендов для проведения испытаний тестового набора разными участниками проекта.

3. Возможность гибкой конфигурации состава тестового стенда путем добавления или удаления виртуальных машин.

4. Экономия пространства, в частности, все средства ввода-вывода располагаются на одном физическом устройстве.

Благодаря использованию виртуальных локальных сетей и виртуальных машин в тестовом стенде обеспечивается полный контроль над потоками данных. Все информационные потоки эмулируются тестовой системой.

Разумеется, виртуальные тестовые стенды применимы лишь для протоколов, реализованных в типовых операционных системах, работающих на распространенных архитектурах компьютеров (x86, ARM, PowerPC). Тестирование встроенных систем производится только на реальном оборудовании.

4. Опыт применения UniTESK для тестирования соответствия для различных протоколов

4.1. Тестирование сетевого протокола IPv6

Первым приложением UniTESK к тестированию протоколов был проект по разработке тестового набора для нового на тот момент сетевого протокола IPv6 в операционной системе MS Windows 2000 [61-65]. В качестве объекта тестирования выступала реализация IPv6, созданная в исследовательском подразделении корпорации Microsoft [61].

IPv6 – это сетевой протокол нового поколения, призванный заменить прежний протокол сетевого уровня IPv4. IPv6 содержит ряд усовершенствований по сравнению с IPv4 и разрешает проблемы, которые ограничивают дальнейшее развитие сетей на IPv4.

Протокол IPv6 относится к протоколам сетевого уровня. На этом уровне происходит маршрутизация пакетов на основе преобразования сетевых адресов в адреса канального уровня. Сетевой уровень обеспечивает прозрачную передачу данных между транспортным уровнем и канальным уровнем.

Были разработаны спецификации и тестовые сценарии для следующих базовых протоколов стека IPv6:

- протокол IPv6, базовые функции оконечного узла (IETF RFC 2460, 2461, 2462, 2463, 2464, 2710);
- протокол UDP в сетях IPv6 (IETF RFC 768, 2460)
- программный интерфейс стека протоколов (IETF RFC 2292, 2553).

В ходе тестирования были обнаружены отклонения от стандартов IPv6 и ошибки программирования. Отклонения от стандартов заключаются в том, что в ряде случаев поведение MSR IPv6 отличается от требований, изложенных в стандартах на IPv6. При тестировании были также выявлены дефекты, которые можно охарактеризовать как ошибки, допущенные при программировании. В частности, был выявлен дефект обработки некоторых входящих сообщений, при которых выполнение ядра ОС аварийно завершается («синий экран смерти») [62,63].

Кроме того, в процессе разработки формальных спецификаций были обнаружены пробелы в RFC2460 в описании алгоритма сборки фрагментов.

Помимо базовой функциональности IPv6 тестовый набор включал модель и спецификации тестов для функции мобильности IPv6 (Mobile IPv6) [64,65]. Протокол Mobile IPv6 призван обеспечить надежный и эффективный способ поддержания соединений узла IPv6 при перемещении между различными сегментами локальной сети. Одна из важнейших задач Mobile IPv6 заключается в том, чтобы мобильный узел всегда оставался достижим по

своим «постоянным» адресам, даже если он перешёл в сеть с другими диапазонами адресов.

Тестовый набор для мобильных функций разрабатывался на базе существующего тестового набора для MSR IPv6, который содержит тесты для базовых функций оконечного узла IPv6 (host). К тестовому набору для MSR IPv6 были добавлены спецификации и тесты для функций узла-корреспондента и мобильного узла. Тестирование функций домашнего агента не проводилось, так как для этого требуется большой объем дополнительных работ по добавлению спецификаций и тестов на базовые функции маршрутизаторов.

Тестовый набор для IPv6 разрабатывался средствами реализации технологии UniTESK для языка C — CTESK. В технологии CTESK формальные спецификации записываются на спецификационном расширения языка C – SEC (Specification Extension of the C language) [66].

SEC представляет собой ANSI C, пополненный несколькими ключевыми словами и синтаксическими конструкциями. SEC поддерживает такие элементы академических языков формальных спецификаций, как предусловия, постусловия, инварианты типов, инварианты глобальных переменных и описания доступа (access descriptors).

Помимо разработанного нами тестового набора существуют и другие наборы тестов для проверки соответствия требованиям стандартов IPv6. Сравнение с этими тестовыми наборами показало, что автоматизация средствами UniTESK позволяет сократить объем ручного кода в 1,5-5 раз, при этом обеспечивает более высокое покрытие функциональности тестами. В частности, ни один из тестовых наборов не нашел критическую ошибку в MSR IPv6, приводящую к перезагрузке компьютера. Тестирование соответствия Mobile IPv6 позволило выявить ряд нарушений стандартной спецификации и ошибки целевой реализации, которые не были обнаружены разработчиками традиционными методами [62-65].

4.2. Тестирование протокола безопасности сетевого уровня IPsec

Обеспечение безопасности передачи данных на сетевом уровне осуществляется сервисом IPsec. Средствами IPsec можно обеспечить аутентификацию отправителя, целостность данных, конфиденциальность данных, защиту от повторов.

Спецификация IPsec предусматривает гибкий механизм настройки политик безопасности (security policy) и контекстов безопасности (security association). Политики безопасности определяют, какие виды защиты должны быть применены к данным, а в контекстах безопасности хранятся ключи и другие параметры защиты. Для упрощения администрирования IPsec содержит протокол автоматической настройки контекстов безопасности IKE (Internet Key Exchange).

После публикации в 1998 году первой версии спецификаций IPsec они были подвергнуты всестороннему анализу. Выявленные в процессе критического анализа недостатки способствовали лучшему пониманию их особенностей и послужили причиной продолжения исследований в этой области. В результате в 2005 году была опубликована новая версия спецификаций (RFC 4301-4309). Архитектура IPsec (RFC 4301) сохранила основные черты первой версии, однако был внесен ряд существенных изменений. По умолчанию автоматическим протоколом управления ключами выбран IKEv2. IKEv2 представляет собой новый протокол, созданный на основе IKE первой версии (RFC 2407, 2408, 2409), с целью его упрощения и оптимизации. Спецификации протоколов безопасности AH и ESP (RFC 4302, 4303) незначительно отличаются от предыдущей версии (RFC 2402, 2406):

Далее в текущем разделе мы не будем углубляться в подробности отличий между версиями протокола безопасности, и будем обозначать обе версии как IPsec.

В рамках архитектуры IPsec различаются две роли, которые могут выполнять узлы сети:

1. оконечный узел (host) является источником или узлом назначения данных, защищённых посредством IPsec;

2. защитный шлюз (security gateway) осуществляет защиту передаваемых через него данных. В частности, функции защитного шлюза может выполнять маршуртизатор или сетевой экран.

В соответствии с приведенной выше методикой тестирования формальная спецификация IPsec состоит из нескольких компонентов:

- модельного состояния, которое содержит набор структур данных, моделирующих концептуальные структуры данных из стандартов IPsec , таких как база данных контекстов безопасности и база данных политик безопасности;

- формального интерфейса IPsec, включающего спецификационные стимулы, формализующие требования к изменению состояния реализации IPsec при внешнем воздействии на систему, и спецификационные реакции, которые формализуют требования к реакциям реализации IPsec на внешние воздействия;

- критериев покрытия, идентифицирующих различные ветви функциональности IPsec.

Были разработаны тестовые сценарии для оконечного узла, проверяющие требования к обработке входящих и исходящих пакетов, и протокола управления ключами IKE (обеих версий) [67-71].

Тесты для генерации исходящих сообщений использовали статически настроенные политики и контексты безопасности для транспортного протокола UDP. Тестовое воздействие для создания исходящего сообщения

оказывается через специализированного агента, который по команде тестовой системы отправлял заданное сообщение UDP на заданный адрес. Благодаря тому, что настройки безопасности задаются статически, тестовая система может расшифровать результат IPsec обработки на целевой машине и удостовериться в корректности исходящей IPsec дейтаграммы.

Тесты для обработки входящих сообщений использовали статически настроенные политики и контексты безопасности для транспортного протокола UDP. Для проверки того, что сообщение успешно прошло обработку IPsec и данные были доставлены приложению, на целевой узел устанавливался агент, задача которого была получать данные из UDP сокета и передавать их в тестовую систему на инструментальной машине. Тестовые сценарии для входящих сообщений генерируют различные (в том числе и некорректные) защищенные сообщения, а целевая система должна обработать сообщение и либо доставить получателю, либо обработать ошибку в соответствии с требованиями.

Фактически, в тестовых сценариях для IPsec была разработана частичная реализация IPsec, необходимая для корректной генерации и разбора защищенных сообщений.

Для тестирования протокола управления ключами тестовый сценарий эмулировал вторую сторону обменов сообщениями IKE — инициатора установления соединения или ответчика. Это необходимо для установления корректных параметров алгоритмов защиты и т. п. Правильность работы IKE устанавливается посредством тестирования входящих и исходящих сообщений — если реализация IPsec получается обмениваться UDP сообщениями с тестовым сценарием, значит IKE правильно настроил контексты безопасности.

Для тестирования реализаций IPsec использовались два вида тестовых стендов — стенд на реальном оборудовании и вируальный тестовый стенд. В результате апробации тестовых наборов остановились на виртуальном тестовом стенде.

Тестовый стенд включает инструментальный узел и целевой узел. На инструментальном узле исполняется основной поток управления тестовой системы. На целевом узле функционирует тестируемая реализация. Для целей тестирования на целевой узел устанавливаются тестовые агенты, которые предоставляют средства удалённого доступа к служебным функциям и протоколам верхнего уровня (UDP).

В рамках проектов по тестированию соответствия IPsec были получены следующие результаты[67-71]:

- Был проведен анализ различных видов спецификаций функций безопасности телекоммуникационных протоколов.

- Были разработаны тестовые наборы для верификации реализаций IPsec и IPsec v2. Для этого были извлечены функциональных требований из стандартов IPsec первой и второй версии, составлены каталоги требований.

Разработаны формальные спецификации IPsec первой и второй версий, включая спецификацию IKEv2. Разработан набор тестов, проверяющих соответствие реализаций формальной спецификации IPsec.

- Был разработан тестовый стенд с использованием виртуальных машин и развернут стенд для испытаний реализаций в системе виртуализации VMware.

- Проведена апробация разработанного тестового набора на реализациях IPsec v2.

Тестовые наборы для IPsec разрабатывались средствами CTESK. В ходе тестирования был обнаружен ряд отклонений от требований спецификаций и ошибок реализации [ ].

4.3. Тестирование сервиса безопасности транспортного уровня

Протокол IPsec используется преимущественно для прозрачной защиты трафика на сетевом уровне, прежде всего для создания защищенных туннелей между компьютерными системами. Защита обменов данными по транспортным протоколам между приложениями управляется протоколом Transport Layer Security, TLS.

Среди всех современных протоколов, предназначенных для защиты передачи информации в открытых сетях, именно TLS используется наиболее широко. Протокол TLS обеспечивает криптографическую защиту соединения между двумя участниками прикладного протокола. Протокол TLS применяется для защиты обменов между клиентом и сервером в различных прикладных сценариях:

- для защиты обмена данными между веб-сервером и браузером (протокол HTTPS),

- для защиты передачи почтовых сообщений между почтовыми агентами (протоколы SMTP, IMAP, POP3),

- при организации виртуальных защищенных сетей (OpenVPN),

- с протоколом запуска сессий (SIP: Session Initiation Protocol) и основанными на нем приложениями (например, VoIP),

- а также во многих других прикладных сценариях.

На настоящий момент широко используются более десятка различных реализаций TLS. По этой причине обеспечение совместимости реализаций TLS является актуальной задачей.

В данном проекте для записи моделей и тестовых сценариев использовался формализм JavaTESK [72]. JavaTESK использует язык программирования Java с набором расширений для записи формальных спецификаций и задания

тестов. Спецификация на языке JavaTESK обычно состоит из одного или нескольких спецификационных классов, которые описывают состояния и переходы моделируемого протокола. Переходы протокола представляются как методы класса специального вида (спецификационные методы), кроме того поддерживается возможность задать ограничения на множество допустимых состояний посредством инвариантов типов (ограничений на значения типов) и инвариантов переменных состояния. Автомат теста задается в сценарном классе, который содержит процедуру определения текущего состояния автомата теста и итераторы тестовых воздействий. Инструмент JavaTESK предоставляет набор обходчиков, которые строят цепочки тестовых воздействий из описания автомата теста.

Спецификация протокола TLS написана на естественном языке. Для тестирования реализации на соответствие было необходимо выделить из стандарта отдельные требования и затем их формализовать. В результате анализа текста стандарта был составлен полный список требований (около 300 требований).

Для модельного представления TLS-сообщений разработана библиотека соответствующих спецификационных типов, позволяющая моделировать различные варианты сообщений. В спецификации каждое входящее TLS-сообщение рассматривается как последовательность стимулов. Каждый стимул в этой последовательности соответствует обработке отдельного блока данных в TLS-сообщении (TLS-сообщение может содержать несколько структур данных конкретного типа).

Разработан тестовый набор для тестирования соответствия серверных реализаций TLS [73]. Тестовые воздействия проверяют корректность ответа сервера на запросы клиента. Особое внимание в тестах уделено тестированию аномального поведения клиента: спецификация TLS насчитывает более 40 различных ситуаций, при которых сервер должен прекратить обмен сообщениями и разорвать соединение.

В отличие от протокола IPsec в данном тестовом наборе не используется приложение-агент на целевой машине, который бы отвечал за установление соединения с клиентом. Вместо него используется TLS в реальных серверных приложениях — веб-сервере или почтовом сервере, - с которым тест устанавливает связь по соответствующему порту.

Для тестирования на соответствие стандарту были несколько известных открытых реализаций TLS. В ходе тестирования во всех реализациях были выявлены отклонения от стандартов, причем в одном случае реализация нарушает критическое требование[73].

Тестовый набор для TLS является новым результатом. Мы не обнаружили других открытых тестовых наборов, проверяющих соответствие реализаций TLS стандарту. Существуют отдельные тестовые наборы, созданные разработчиками реализаций TLS (например, openssl, Java SSE), но эти

тестовые наборы сфокусированы на тестировании внутренних аспектов работы реализации и непригодны для тестирования других реализаций.

4.4. Тестирование почтовых протоколов

В данном разделе приводится краткий обзор проектов [74-79] по тестированию реализаций электронной почты.

Электронные письма являются основой современного взаимодействия между людьми. Миллионы писем перемещаются ежедневно в сети Интернет. Надежность и корректность инфраструктуры почтовых сообщений чрезвычайно важна для современного информационного общества. В этой статье мы коснемся двух аспектов этих вопросов: надежности (1) передачи почты в сети Интернет и (2) доставки писем конечным адресатам.

Несмотря на более чем двадцатилетнюю историю почтовых протоколов и существования десятков реализаций протоколов SMTP, POP3 и IMAP4, до сих пор нет открытого и независимого от реализации тестового набора для проверки соответствия стандартам. Несмотря на кажущуюся простоту, почтовые протоколы:

1. недоспецифицированы, существенная часть функциональности оставлена на усмотрение разработчика, в спецификации описаны несколько вариантов возможного дальнейшего поведения системы;

2. недетерминированы: стандарт допускает различные варианты поведения системы, включая отказ в доставке почтовых сообщений или разрыв соединений;

3. расширяемы: реализация протокола может использовать различные расширения, как дополняющие функциональность протокола, так и изменяющие её;

4. функции почтовых протоколов различаются по степени обязательности (MUST, SHOULD, MAY и прочие).

Перечисленные особенности определяют фактическую сложность разработки тестовых наборов для тестирования реализаций протоколов на соответствие спецификациям.

В проекте по тестированию почтовых протоколов был предложен новый метод [78] создания тестов для тестирования соответствия. Кратко опишем ключевые моменты.

Выше были представлена методика разработки тестовых наборов, в которой разработка начинается с выделения требований, затем требования формализуются и только на заключительном этапе появляются тесты. Такой подход к разработке тестовых наборов имеет много общего с известной каскадной моделью разработки ПО, когда разработка начинается с детального анализа требований и ведется путем последовательного проектирования, реализации и тестирования.

Метод разработки модели и тестов, опробованный в проекте по тестированию почтовых протоколов, напротив, больше перекликается с новыми, популярными подходами к разработке ПО из семейства «гибкого программирования» (Agile Programming, AP). Один из ключевых моментов в AP — первая работающая версия продукта появляется достаточно рано в ходе проекта, новые работающие версии продукта появляются достаточно часто. Разумеется, ранние сроки и частота обновления версий объясняются неполнотой реализаций — на ранних этапах клиент видит скорее прототип, чем полную реализацию всех его требований и пожеланий.

В проекте по тестированию почтовых протоколов был опробован аналогичный подход — первые тесты появились достаточно рано по времени проекта, и далее они уточнялись и развивались. Так как разработка модели протокола — это трудоемкий и затратный по времени процесс, который нельзя сделать «наполовину». Модель должна быть полной и точной. Поэтому в описываемом методе модель разрабатывается ближе к концу проекта.

Рассмотрим шаги метода. Разработка тестового набора начинается с изучения и анализа требований. Необходимо выделить набор функций, составляющих протокол, и расставить приоритеты — в какой последовательности тестировать эти функции.

На следующем шаге для самых приоритетных функции разрабатываются элементарные тесты (test cases) без использования модели. Если сравнивать с Agile методами, то этот тестовый набор аналогичен «первому релизу» – он не полон, но функционален и уже работает.

Важно отметить, что в данном методе все разрабатываемые тесты пропускаются на реализации. Они отлажены и в конце каждого этапа гарантированно корректны.

Так как наша цель заключается в том, чтобы в конечном итоге получить тестовый набор, основанный на моделях, последующие шаги направлены на то, чтобы приблизить разработку к этой цели.

Первый тестовый набор, как правило, «бесструктурен», он представляет собой набор независимых тестовых программ. Поэтому на втором шаге проводится структурирование — выделяются интерфейсы, разграничивающие тесты и реализацию. Так, если исходно тест взаимодействовал с реализацией протокола SMTP непосредственно через отправку и получение сообщений, то после второго шага появляется интерфейс, абстрагирующий отдельные команды протокола SMTP: helo, ehlo, maifrom, rcpt и т. д. Это ещё не модель, так как внутри этого интерфейса нет состояний и переходов, это скорее формальный интерфейс протокола (понятие формального интерфейса обсуждается в разделах 3 и 3.1). Подготовка тестовых данных, отслеживание состояния модели и вынесение вердикта на этом шаге метода остаются в тестах. Параллельно с выделением формального интерфейса могут разрабатываться дополнительные тесты, но уже в терминах операций формального интерфейса.

На третьем шаге метода вводится первый элемент автоматизации. Тестовый набор, полученный к настоящему моменту, представляет собой набор линейных программ. Тестирование соответствия требует большого количества тестов, так как полный тестовый набор должен содержать тесты для каждого пути в графе переходов автомата, моделирующего протокол. Разработка такого тестового набора вручную — это монотонный, однообразный труд, причем большинство тестов будут иметь значительные пересечения. Автоматизация заключается в том, что тесты перестают быть линейными, они трансформируются в конечные автоматы. Тесты становятся тестовыми сценариями UniTESK. Количество тестовых программ может остаться тем же самым или даже уменьшиться, но количество тестовых ситуаций, которое они покрываю, значительно возрастет благодаря обходу автомата.

В отличие от канонического подхода, изложенного в разделе 3, в этих тестах структура состояния и обновления состояния реализованы непосредственно в тестовых методах. Благодаря тому, что про эти тесты пропускаются на реализации, мы можем быть уверены, что разработанная модель состояния и структура переходов корректны.

Следующий шаг метода заключается в трансформации формального интерфейса в модель протокола. На одном из предыдущих шагов был выделен компонент, реализующий формальный интерфейс. На этом шаге он разделяется на два: в один компонент отправляется код, реализующий абстрактные операции helo, mailfrom и т. п. через сообщения протокола, и этот компонент становится адаптером (медиатором), а во второй компонент переносятся из тестов состояние и функции обновления состояния. Фактически, появляется модель протокола, включающую модель состояния протокола и функции обновления этого состояния в зависимости от ответов реализации. Соответственно, обновляются тесты: проверки корректности, которые были в них реализованы, переписываются в терминах изменений состояния компонента модели. Параллельно с разработкой модели могут добавляться новые тесты, но эти тесты должны быть устроены как автоматы и выносить вердикты на основе выделенной модели. Следующая трансформация заключается в переносе кода для вынесения вердикта из тестов в модель. В тестах остаются только подготовка данных и вызовы методов формального интерфейса. Так как к текущему моменту все тесты были уже отлажены, то мы можем быть уверены, что проверки, добавленные в модель — корректны.

После того, как модель протокола выделена в виде исполнимого компонента, можно трансформировать её в спецификацию UniTESK. Проверки переносятся в пред- и постусловия, операции с состоянием становятся функциями реконструкции состояния. Все последующие тесты разрабатываются как тестовые автоматы с использованием полученной модели.

Предложенный подход включает несколько этапов разработки тестов, причем этапы сконструированы таким образом, что каждый этап может быть

завершающим. Новизна разработанного метода тестирования соответствия заключается в последовательной трансформации модели тестирования, при которой на каждом шаге получаются отлаженные тесты и/или модель. При выполнении этих шагов получается тот же по функциональности тестовый набор, что был после первого шага. Но на данном этапе уже намного проще расширять тестовый набор путем расширения формальной спецификации тестируемого протокола.

Причины, побудившие разработать изложенный подход:

1. В проекте по созданию тестовых наборов для почтовых протоколов участвовали разработчики, незнакомые с технологией UniTESK. Предложенный метод позволил вводить концепции тестирования, основанного на моделях, постепенно и на практике. Разработка началась с простых и понятных элементарных тестов и постепенно включила в себя автомат теста, модель протокола, спецификацию протокола.

2. Разработчики тестов для почтовых протоколов ранее не были знакомы с этими протоколами. Начав с исполнимых тестов, они на практике познакомились с операциями протокола. Как показывает опыт предыдущих проектов по тестированию соответствия, немногие люди обладают достаточным развитым абстрактным мышлением, чтобы сразу после прочтения стандарта суметь написать модель протокола. Большинству нужно сначала попробовать протокол на практике, например, написав программы, которые используют протокол для решения каких-то несложных задач, вроде отправки письма по SMTP. Первый тестовый набор, состоящий из линейных тестов, как раз и направлен на решение задачи знакомства с протоколом.

3. Опять-таки, как показывает практика, первые прогоны тестов, разработанных с использованием моделей по технологии UniTESK, завершаются с негативными вердиктами из-за ошибок в спецификации. В данном подходе спецификация получается через последовательность трансформаций отлаженных и работающих программ, поэтому степень доверия такой спецификации выше, чем той, которая разработана без запуска тестов.

Разумеется, помимо достоинств у предложенного метода есть недостатки. Во-первых, для его успешного применения требуется наличие корректной реализации. Если реализации нет, или она некорректна, то предложенный метод «не работает»: нет возможности запускать и отлаживать тесты, что является критически важным для успешного применения данного метода. Во-вторых, полученная спецификация, очевидно, неполна, так как она получается из изначально неполного тестового набора. Если бы тестовый набор сразу был полон, то не было бы необходимости разрабатывать модель. Тем не менее, даже такая неполная спецификация обладает определенной ценностью — разработчики спецификации освоили предметную область, и развитие спецификации будет проще, чем разработка «с нуля».

В качестве примера применения предложенного метода с его помощью был разработан открытый тестовый набор для проверки соответствия основной функциональности протоколов SMTP, POP3 и IMAP4 их стандартам. Также был разработан тест, использующий композицию спецификаций протоколов SMTP и POP3. Данный тест используется для проверки корректности взаимодействия реализаций различных почтовых серверов.

Спецификации и тесты разрабатывались средствами технологии JavaTESK.

Тесты применялись для почтовых серверов, разрабатываемых по модели открытого исходного кода – Apache James, hMail Server, Postfix и Dovecot. Сгенерированные тесты обнаружили несколько несоответствий между реализациями протоколов и стандартами, в том числе одну ошибку в реализации, которая приводила к бесконечному зацикливанию письма внутри сервера при определенной конфигурации окружения.

Тестовый набор для почтовых протоколов является новым результатом. Разработчики реализаций почтовых протоколов разрабатывают также и тесты, но, как оказалось, тесты сильно связаны с конкретными реализациями и непереносимы для тестирования серверов от других разработчиков, так как используют особенности реализации: настройки параметров, доступ к внутреннему состоянию, выполнение в одном процессе с реализацией. Использование тестов, разработанных для одной реализации, не предоставляется возможным для проверки других реализаций. Более того, эти тесты не предназначены для тестирования соответствия, они проверяют прежде всего корректность реализации многочисленных настроек почтового сервера, не относящихся напрямую к стандартам SMTP, POP3 и IMAP4. Проблема заключается в том, что такой подход к тестированию не гарантирует обеспечения совместимости сервера со стандартом. В частности, именно тестирование соответствия позволило выявить в одном из серверов серьезную функциональную ошибку.

4.5. Тестирование протоколов авионики

Использование CTESK или JavaTESK для разработки моделей и тестов требует от авторов тестового набора значительных навыков программирования. Мало того, что сами по себе языки C и Java сложны в изучении для неспециалиста, требуется дополнительно освоить спецификационные расширения и научиться пользоваться соответствующими инструментами.

Поэтому была проведена работа по реализации технологии UniTESK в более простом языке и без использования расширений. В качестве базового языка был выбран динамический язык Python. Это простой язык с хорошо определенной семантикой, ясным синтаксисом, мощными и содержательными конструкциями. Язык Python, безусловно, проще в обучении, чем C или Java.

На языке Python были разработаны базовые компоненты UniTESK, прежде всего обходчики автомата теста. Разработаны специальные декораторы, упрощающие спецификацию автомата теста (тестового сценария),

позволившие отказаться от расширения языка дополнительными синтаксическими конструкциями.

Благодаря тому, что Python является динамическим языком, связывание модели и реализации происходит легко и непринужденно. Не требуется сложная трансляция из расширения в базовый язык, как в CTESK или JavaTESK, или динамическая генерация байт-кода Java, как это делается в некоторых фреймворках на Java.

Средствами PyTESK были разработаны тестовые наборы для трех протоколов, использующихся для обмена файлами между бортом самолета и наземными службами.

Тестирование доступной реализации выявило существенные ошибки в реализации, как отклонения от стандарта, так и программные ошибки, которые препятствовали успешному выполнению операций протокола.

Использование динамического языка для автоматизации тестирования является новой тенденцией в Software Engineering. Существуют несколько проектов, нацеленных на автоматизацию тестирования с использованием динамических языков, например PyModel на Python [80] и errfix на Ruby [81].

PyTESK отличается от этих проектов тем, что автомат теста не извлекается из модели, а частично задается разработчиком.

5. Обсуждение применимости инструментов CTESK, JavaTESK и PyTESK для тестирования соответствия

Изначально UniTESK «вырос» из проекта KVEST [82] по верификации ядра операционной системы. В этом проекте модель системы и тесты разрабатывались на академическом языке формальных спецификаций RSL. Проект KVEST завершился успешно — был разработан тестовый набор и выявлен ряд критических ошибок в ядре ОС. Однако, в компании-заказчике не смогли перенять и поддержать разработанный тестовый набор, так как нотация языка и концепции, лежащие в его основе, были трудны для восприятия инженерами-программистами.

В результате, UniTESK стал развиваться в направлении сближения формальных методов и повседневной практики программирования. Ключевую роль в этом сближении играл перенос концепций контрактной спецификации и автомата теста в популярный язык программирования.

5.1. CTESK

Первым языком, для которого была разработана технология автоматизации тестирования с использованием моделей, стал язык C. Это очень популярный и сравнительно простой язык, хорошо знакомый подавляющему большинству программистов. Однако, встроенные средства языка не позволяют добавлять в него новые конструкции, в частности пред- и пост-условия. Поэтому было

разработано *расширение* языка C, которое представляет собой ANSI C с несколькими дополнительными конструкциями [66].

С использованием CTESK были разработаны ряд тестовых наборов для программных интерфейсов ОС Linux, включая программный интерфейс POSIX [83,84]. В этих проектах CTESK хорошо себя зарекомендовал — благодаря близости языка моделирования к языку реализации, профессиональный программист на Си без труда мог понять модели и тесты, даже не зная подробности семантики расширении языка.

Однако применение CTESK для тестирования протоколов оказалось менее успешным. Средствами CTESK были разработаны тестовые наборы для IPv6 и IPsec. В силу того, что язык Си не содержит высокоуровневых конструкций, таких как списки, итераторы, ООП, полиморфизм, и т. п., модели и тесты на Си содержат большое количество «технического» кода, необходимого для компиляции и выполнения тестового набора.

Концепции технологии UniTESK естественным образом ложатся на парадигму объектно-ориентированного программирования. Но в языке Си нет поддержки ООП, поэтому аналоги виртуальных функций приходилось создавать вручную (одна из частей «технического») кода.

Универсальный обходчик, который не зависит от модели и теста, требует нетривиального управления памятью. Аллокация и деаллокация памяти могут отстоять друг от друга на сотни шагов по тесту, поэтому в CTESK был реализован свой сборщик мусора. Что только прибавило сложностей при разработке состояния и тестов. Этот сборщик предоставил специальный API для объектов времени выполнения. API позволяет создавать объекты, модифицировать и удалять, когда они становятся ненужными. Такое ручное управление данными в модели и тестовом сценарии неоднократно приводили к трудновоспроизводимым ошибкам.

Как правило, первые расхождения со спецификацией, которые находит тестовый набор, вызваны ошибками в модели или тесте. Трансляция дополнительных конструкций в базовый язык (Си) порождала сложный код, для отладки которого требовалось хорошее знание особенностей трансляции расширения.

Несмотря на все трудности, обусловленные использованием CTESK, тестовые наборы были разработаны и успешно использованы для тестирования реализации и обнаружения ошибок.

5.2. JavaTESK

Следующий крупный шаг в развитии UniTESK был связан с реализацией технологии на базе объектно-ориентированного языка с автоматической сборкой мусора. Было разработано спецификационное расширение языка Java, получившее название JavaTESK[72].

Переход от Си к Java позволил значительно упростить модели и тестовые сценарии, освободить их от многих вынужденных технических «костылей»,

сделать более понятными. Тем не менее, JavaTESK не сумел разрешить ряд ключевых проблем, связанных с использованием расширения языка программирования:

1 Расширение отличается от базового языка, поэтому среды разработки считают программы, написанные с использованием расширения, некорректными. Требуется разрабатывать специализированные плагины для сред разработки, которые добавляют поддержку расширения в среду разработки. Такие специализированные решения, как правило, отстают по набору функций от поддержки базового языка. Это, безусловно, мешает разработчикам адаптироваться к использованию расширения языка.

- Расширение транслируется в базовый язык нетривиальным образом. Запутанная схема генерации кода порождает трудности с отладкой тестов и поиском ошибок в тестах, если что-то пошло не так.
- Язык Java содержит больше высокоуровневых конструкций, чем Си, но тем не менее, разработка моделей и тестов на JavaTESK требует много технического кода (меньше чем в C, но тем не менее).
- Переход от базовых протоколов к расширяемым показал, что для JavaTESK нетривиально реализуется метод моделирования расширений протоколов. Требуются «фокусы» с динамической генерацией байткода, чтобы собирать модель реализации по набору расширений, которые она поддерживает.

Тем не менее, на JavaTESK были успешно разработаны тесты для протокола TLS и почтовых протоколов.

5.3. PyTESK

Применение современного динамического языка в качестве базы для автоматизации тестирования с использованием моделей открывает новые перспективы. Динамический язык позволяет гибко описывать модели и тесты с минимумом вспомогательного кода, не содержащего логики протокола или теста. Динамический язык позволяет легко строить композиционные модели и тесты. Отсутствие строгой типизации, мощные высокоуровневые конструкции, сравнительно простой синтаксис облегчает освоение этого языка неспециалистам, в частности, занятых тестированием гибридных систем и интеграции программно-аппаратных комплексов.

К настоящему моменту на PyTESK реализованы несколько экспериментальных тестовых наборов, включая протокол безопасности транспортно уровня TLS. Ведутся работы по созданию максимально простого метода разработки тестов.

При всех своих плюсах динамический язык имеет существенный минус — большинство ошибок программирования выявляются на этапе исполнения.

Разработка тестового набора требует наличия реализации, для которой можно запускать тесты и отлаживать их.

6. Заключение

В статье обобщается опыт коллектива в исследовании и разработке формальных методов верификации и тестирования реализаций протоколов Интернета. Необходимая методологическая основа подобных исследований была разработана и внедрена в рамках работ ИСП РАН, поддержанных грантами РФФИ, результатом которых стала широко применяемая в настоящее время оригинальная технология тестирования на основе формальных спецификаций UniTESK.

В результате этих исследований был разработан новый, основанный на контрактных спецификациях, метод верификации, позволяющий эффективно автоматизировать тестирование очень сложных протоколов Интернета. При этом созданные тестовые наборы обладали формально определенным и прослеживаемым покрытием требований спецификаций, что в значительной степени улучшило качество тестирования и позволило выявить целый ряд ошибок в существующих реализациях соответствующих протоколов.

В ходе разработки тестовых наборов были выявлены и некоторые особенности протоколов, затрудняющие тестирование реализаций с помощью технологии UniTESK, а также особенности инструментов, поддерживающих ее работу, однако все эти особенности удавалось успешно преодолеть, не выходя за рамки ее ограничений.

Список литературы

[1] CCITT Recommendation Z.100. Specification and Description Language (SDL). Geneve, Switzerland: ITU, 1993. 245 с.

[2] ISO/IEC 9074. Information Processing Systems — Open Systems Interconnection. Estelle — A Formal Description Technique based on an Extended State Transition Model. Geneve, Switzerland: ISO, 1989 1 я редакция,1997 2-я редакция. Отозван 06.05.1999.

[3] ISO/IEC 8807. Information Processing Systems — Open Systems Interconnection. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. Geneve, Switzerland: ISO, 1989. 142 с.

[4] ITU-T X.680 (11/08) Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation. ITU-T, 2008. 194 с.

[5] OMG Unified Modeling LanguageTM (OMG UML), Infrastructure . OMG, 2011. 230 с.

[6] Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ. // Изд-во Моск. центра непрерыв. мат. образования, 2002. 416 с.

[7] Карпов Ю.Г. MODEL CHECKING. Верификация параллельных и распределенных программных систем. // БХВ-Петербург, 2010г. 560 с.

[8] M. Diaz. Petri Nets: Fundamental Models, Verification and Applications. // Willey, 2013г. 656с.

[9] Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann, San Francisco (2007).
[10] Blackburn, M., Busser, R., Nauman, A.: Why Model-Based Test Automation is Different and What You Should Know to Get Started. Software Productivity Consortium, NFP (2004).
[11] Dalal, S.R., Jain, A., Karunanithi, N., Leaton, J.M., Lott, C.M., Patton, G.C., Horowitz, B.M.: Model-Based Testing in Practice. In: Proceedings of the ICSE 1999 (May 1999).
[12] I. Bourdonov, A. Kossatchev, V. Kuliamin, A. Petrenko. *UniTesK Test Suite Architecture.* Proceedings of FME'2002, Kopenhagen, Denmark, LNCS 2391:77-88, Springer-Verlag, 2002.
[13] В. В. Кулямин, А. К. Петренко, А. С. Косачев, И. Б. Бурдонов. *Подход UniTesK к разработке тестов.* Программирование, 29(6):25-43, 2003.
[14] В. В. Кулямин, А. К. Петренко. Развитие подхода к разработке тестов UniTESK. Труды ИСП РАН, 26(1), 2014. DOI: 10.15514/ISPRAS-2014-26(1)-1.
[15] F. C. Hennie. Fault detecting experiments for sequential circuits. // Proc. 5-th Ann. Symp. Switching Circuit Theory and Logical Design, 1964. С. 95-110.
[16] [Василевский] М. П. Василевский. Диагностика ошибок в автоматах. // Кибернетика и системный анализ, т. 9, № 4, 1973. С. 98-108.
[17] T. S. Chow. Testing software design modeled by finite-state machines. // IEEE Trans. on Software Engineering, vol. 4, no. 3, 1978. С. 178-187.
[18] J. A. Bergstra, J. W. Klop. Algebra of Communicating Processes with Absraction. Theoretical Computer Science, 37(1), 1985. С. 77-121.
[19] C. A. R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985; электронное издание, 2004. 260 с. [PDF] (http://www.usingcsp.com/cspbook.pdf).
[20] R. Milner. Communication and Concurrency. Prentice-Hall, 1989. 260 с.
[21] N.A. Lynch, M.R. Tuttle, "An Introduction to Input/Output Automata" // CWI-Quaterly 3, 1989, P. 219-246. [PDF] http://www.markrtuttle.com/papers/lt89-cwi.pdf.
[22] F. Maraninchi. Operational and compositional semantics of synchronous automaton compositions. // CONCUR'92 Proceedings, LNCS 630. Springer-Verlag, 1992. [PS] http://www-verimag.imag.fr/~maraninx/ArgosCONCUR92.ps.gz
[23] ISO/IEC 7498. Information technology – Open Systems Interconnection – Basic Reference Model. Geneva, Switzerland: ISO, 1994.
[24] Ed. Brinksma. A theory for the derivation of tests. Proc. IFIP WG6.1 8th Intl. Symp. on Protocol Specification, Testing, and Verification, North-Holland, S. Aggarwal and K. Sabnani Ed. pp. 63-74, 1988.
[25] K. K. Sabnani and A. T. Dahbura. A protocol test generation procedure. Computer Networks and ISDN Systems, vol. 15, no. 4, pp. 285-297, 1988.
[26] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. IEEE Trans. on Software Eng., vol. 17, pp. 591-603,1991.
[27] G. Luo, A. Petrenko, and G. v. Bochmann. Selecting test sequences for partially specified nondeterministic finite state machines, Proceedings of the IFIP Seventh International Workshop on Protocol Test Systems, Japan, 1994, pp. 95-110.
[28] Н. В. Евтушенко, А. В. Лебедев, А. Ф. Петренко. Построение проверяющего множества для компоненты последовательной автоматной сети. Автоматика и телемеханика, № 8. стр. 145-153, 1994.
[29] J. C. Fernandez, C. Jard, T. Jeron, C. Viho. Using on-the-Fly Verification Techniques for the Generation of Test Suites // Proceedings of the 8th International Conference on Computer Aided Verification, LNCS 1102, Springer-Verlag, 1996, P. 348-359.

[30] J. Tretmans. Test Generation with Inputs, Outputs, and Repetitive Quiescence. Software — Concepts and Tools, 17(3):103-120, 1996.
[31] D. P. Sidhu and T.-K. Leung. Formal methods for protocol testing: a detailed study. IEEE Trans. Soft. Eng., vol. 15, no. 4, pp. 413-426, 1989.
[32] D. Lee, M. Yannakakis. Principles and methods of testing finite state machines — a survey. Proc. IEEE, 84(8):1090-1123, 1996.
[33] G. v. Bochmann, A. Petrenko. Protocol Testing: Review of Methods and Relevance for Software Testing. Proc. of ACM SIGSOFT ISSTA'1994, Software Engineering Notes, Special Issue, pp. 109—124.
[34] A. Petrenko. Fault Model-Driven Test Derivation from Finite State Models: Annotated Bibliography. In F. Cassez, C. Jard, B. Rozov, M. Dermot, eds. Modeling and Verification of Parallel Processes: 4-th Summer School, Nantes, France, LNCS 2067, pp. 196-200, Springer-Verlag, 2000.
[35] ISO/IEC 9646. Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts. Geneva: ISO, 1994. 46 c.
[36] ITU-T Recommendation Z.500. Framework on formal methods in conformance testing. Geneve, Switzerland: ITU, 1997. 49 c.
[37] Glenford J. Myers, Corey Sandler, Tom Badgett. The Art of Software Testing, 3rd Edition. Wiley, 2011. C. 240.
[38] J. Tretmans, An Overview of OSI Conformance Testing. Translated and adapted from: J. Tretmans and J. van de Lagemaat, Conformance Testen, in Handboek Telematica, Vol. II, pages 1--19. Samson, 1991.
[PDF] http://people.cs.aau.dk/~kgl/TOV03/iso9646.pdf
[39] ISO/IEC 9646. Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 2: Abstract Test Suite specification. Geneva: ISO, 1994. 33 c.
[40] Information technology – Open systems interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation (TTCN). 1-е издание. Geneva, Switzerland: ISO, 1992.
[41] Information technology – Open systems interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation (TTCN). 2-е издание. Geneva, Switzerland: ISO, 1998.
[42] ETSI ES 201 873-1 V3.1.1. Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language. Sophia-Antipolis, France: ETSI, 2005. 210 c.
[43] OMG formal/05-07-07. UML Testing Profile. Version 1.0. Needham, USA: Open Management Group, 2005. [PDF, PostScript] (http://www.omg.org/cgi-bin/doc?formal/05-07-07).
[44] L. Ebrecht, M. Schacher, C. Bühler. Test Specification in XML – the most important Element for Test Automation. // ARTiSAN Benutzerforum D.A.CH, 2005.
[45] Программный комплекс для разработки тестов. [URL] http://www.junit.org.
[46] Проект TAHI по разработке тестового набора для стека протоколов IPv6. [HTML] (http://www.tahi.org/).
[47] Digital cellular telecommunications system (Phase 2+) (GSM); Handover procedures (GSM 03.09 version 5.1.0) // ETSI, Sophia-Antipolis, France, 1997. 81 c.
[48] ETSI ETS 300 838. Integrated Services Digital Network (ISDN); Harmonized Programmable Communication Interface (HPCI) for ISDN. // ETSI, Sophia-Antipolis, France, 1998. 546 c.

[49] C. Jard, T. Jéron. TGV: Theory, principles and algorithms. // International Journal on Software Tools for Technology Transfer (STTT), vol. 7(4). Berlin: Springer, 2005. С. 297 – 315
[50] E. Farchi, A. Hartman, S.S. Pinter. Using a Model-based Test Generator to Test for Standard Conformance. // IBM System Journal - special issue on Software Testing. Volume 41(1), 2002. С. 89 - 110..
[51] G. Friedman, A. Hartman, K. Nagin, T. Shiran. Projected State Machine Coverage for Software Testing. // Proceedings of ISSTA 2002 International Symposium on Software Testing and Analysis. New York, USA: ACM Press, 2002. С. 134 – 143.
[52] Margus Veanes, Colin Campbell, Wolfgang Grieskamp, Wolfram Schulte, Nikolai Tillmann, and Lev Nachmanson, Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer, in Formal Methods and Testing, vol. 4949, pp. 39-76, Springer Verlag, 2008
[53] A. Belinfante, J. Feenstra, R. de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, L. Heerink. Formal test automation: A simple experiment. // G. Csopaki, S. Dibuz, K. Tarnay, editors. 12th Int. Workshop on Testing of Communicating Systems. Budapest, Hungary: Kluwer Academic Publishers, 1999. С. 179-196.
[54] Axel Belinfante. JTorX: a Tool for On-Line Model-Driven Test Derivation and Execution. In: Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2010. LNCS vol. 6015, pp. 266-270. Springer.
[55] N. Goga. A probabilistic coverage for on-the-fly test generation algorithms. // Automated Verification of Critical Systems (AVoCS '03), 2003.
[56] Н.В. Пакулин, А.В. Хорошилов. Разработка формальных моделей и тестирование соответствия для систем с асинхронными интерфейсами и телекоммуникационных протоколов. // Журнал "Программирование" № 5, 2007 г., ISSN 0132-3474, с. 1-29.
[57] V. V. Kuliamin, A. K. Petrenko, N. V. Pakoulin, A. S. Kossatchev, I. B. Bourdonov. Integration of Functional and Timed Testing of Real-time and Concurrent Systems. // Proceedings of the 5-th International Conference on Perspectives of System Informatics, July 9-12, 2003, Novosibirsk, Russia; LNCS 2890, Springer, 2003, pp. 450-461.
[58] V. V. Kuliamin, A. K. Petrenko, N. V. Pakoulin. Practical Approach to Specification and Conformance Testing of Distributed Network Applications. // Proceedings of the 2-nd International Service Availability Symposium, April 25-26, 2005, Berlin, Germany; LNCS 3694, Springer, 2005, pp. 68-83.
[59] V. V. Kuliamin, A. K. Petrenko, N. V. Pakoulin. Extended Design-by-Contract Approach to Specification and Conformance Testing of Distributed Software. // Proceedings of the 9-th World Multiconference on Systemics, Cybernetics, and Informatics, Model Based Development and Testing Workshop, July 10-13, 2005, Orlando, Florida, USA, pp. 65-70
[60] C. A. R. Hoare. An axiomatic basis for computer programming. // Communications of the ACM, volume 12 №10, 1969. С. 576-580.
[61] R. P. Draves, A. Mankin, B. D. Zill. Implementing IPv6 for Windows NT. Proceedings of the 2nd USENIX Windows NT Symposium, Seattle, WA, August 3–4, 1998
[62] Агамирзян И., Грошев С.Г., Хорошилов А.В., Ключников Г.В., Косачев А.С., Омельченко В.А., Пакулин Н.В., Петренко А.К., Шнитман В.З. Применение формальных методов для тестирования MSR IPv6. // Интернет нового поколения. Сборник тезисов международной конференции. Ярославль, 2002. С. 29-33.
[63] Ключников Г.В., Косачев А.К., Пакулин Н.В., Петренко А.К., Шнитман В.З. Применение формальных методов для тестирования реализации IPv6. // Труды ИСП РАН, Том 4. М., 2003. С. 121-140.

[64] Ключников Г.В., Косачев А.С., Пакулин Н.В., Петренко А.К., Шнитман В.З. Применение формальных методов для тестирования Mobile IPv6. // Интернет нового поколения. Сборник тезисов II международной конференции. Ярославль, 2003. С. 20-25.
[65] Зацепин Д.В., Шнитман В.З. Особенности применения технологии UniTESK для тестирования функций мобильности в протоколе IPv6. // Труды ИСП РАН 13 (1). М. , 2007. С. 143-170
[66] CTesK 2.1: SeC Language Reference. М.: ИСП РАН, 2005. 167 с.
[67] Ключников Г.В., Пакулин Н.В., Шнитман В.З. Автоматизированное тестирование сетевых сервисов Интернет-протокола // Труды Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ: технологии распределенных вычислений», г. Новороссийск, 19-24 сентября 2005 г. С. 168-170.
[68] Н.В. Пакулин "Применение формальных методов для тестирования реализаций сложных современных протоколов", Сборник трудов международного семинара "Go4IT - шаг к новым технологиям Интернета", ИСП РАН, М., 2007 г., с. 11-18.
[69] А.В. Никешин, Н.В. Пакулин, В.З. Шнитман. Особенности тестирования сервисов безопасности сетевого уровня IPsec второй версии. // Научный сервис в сети Интернет: решение больших задач: Труды Всероссийской научной конференции (22-27 сентября 2008 г., г. Новороссийск). - М.: Изд-во МГУ, 2008. - 468 с. ISBN 978-5-211-05616-9.
[70] А.В. Никешин, Н.В. Пакулин, В.З. Шнитман. Разработка тестового набора для верификации реализаций протокола безопасности IPsec v2. // Труды Института системного программирования РАН, том 18, 2010 г. Стр. 151-182.
[71] А.В. Никешин, Н.В. Пакулин, В.З. Шнитман. Верификация функций безопасности протокола IPsec v2. // Программирование, том 37 № 1. М., 2011. С. 36-56.
[72] Bourdonov I.B., Demakov A.V., Jarov A.A., Kossatchev A.S., Kuliamin V.V., Petrenko A.K. and Zelenov S.V. Java Specification Extension for Automated Test Development // Proceedings of PSI'2001. Novosibirsk, Russia July 2-6 2001, LNCS 2244:301-307. Springer-Verlag, 2001.
[73] А.В. Никешин, Н.В. Пакулин, В.З. Шнитман. Разработка тестового набора для верификации реализаций протокола безопасности TLS. // Труды ИСП РАН, том 23, 2012 г. Стр. 387-404. DOI: 10.15514/ISPRAS-2012-23-22.
[74] А.Н. Тугаенко. Тестирование соответствия почтовых протоколов сети Интернет. // Материалы Международного молодежного научного форума «ЛОМОНОСОВ-2010» (тезисы), стр.25-27.
[75] Н.В.Пакулин, А.Н.Тугаенко. Разработка тестовых наборов для тестирования соответствия почтовых протоколов. // Конференция АППИ-2009, стр. 154-160.
[76] А.Н. Тугаенко. Метод тестирования соответствия для расширяемых протоколов сети Интернет. // Материалы Международного молодежного научного форума «ЛОМОНОСОВ-2011».
[77] N. Pakulin, A. Tugaenko. Model Based Conformance Testing for Extensible Internet Protocols // Proceedings of SYRCoSE 2011.Н.В. Пакулин, А.Н. Тугаенко.
[78] Пакулин Н.В., Тугаенко А.Н., Шнитман В.З. Тестирование протоколов электронной почты Интернета с использованием моделей. // Труды ИСП РАН, том 20, 2011 г. Стр. 125-141.
[79] Пакулин Н.В., Тугаенко А.Н., Шнитман В.З. Тестирование протоколов электронной почты Интернета с использованием моделей. // Программирование. т. 37 №5 - Москва: МАИК "Наука/Интерпериодика", 2012.
[80] J. Jacky, "PyModel: Model-based testing in Python", Northwest Python Day 2010.

[81] Errfix: библиотека классов для тестирования с использованием моделей на Ruby. https://code.google.com/p/errfix/
[82] I. Bourdonov, A. Kossatchev, A. Petrenko, and D. Galter. KVEST: Automated Generation of Test Suites from Formal Specifications. FM'99: Formal Methods. LNCS, volume 1708, Springer-Verlag, 1999, pp. 608–621.
[83] А.И.Гриневич, В.В.Кулямин, Д.А.Марковцев, А.К.Петренко, В.В.Рубанов, А.В.Хорошилов. Использование формальных методов для обеспечения соблюдения программных стандартов.// Труды Института системного программирования РАН, №10, 2006.
[84] В.В.Кулямин, А.К.Петренко, В.В.Рубанов, А.В.Хорошилов. Формализация интерфейсных стандартов и автоматическое построение тестов соответствия. Информационные технологии, 8:2-7, М. Новые технологии, 2007.

Automation of conformance testing for communication protocols

Nikeshin A.V., Pakulin N.V., Shnitman V.Z.
alexn@ispras.ru, npak@ispras.ru, vzs@ispras.ru
ISP RAS, Moscow, Russia

Abstract. This article summarizes the experience gained by the UniTESK team in development of model-based test suites for conformance testing of Internet protocols. Here we call “Internet protocols” the protocols of TCP/IP stack starting network level. We performed testing of network-level protocols IPv6 and Mobile IPv6, security protocols , IPsec v2 and TLS/SSL, mail protocols SMTP, POP3 and IMAP. The projects described in this article used the UniTESK technology as a base for test construction: we modeling protocol under test as a state machine with implicitly defined transitions, and the tests are constructed as traversal of test state machine. During the development of test suites we identified certain features of the protocols that make it difficult to apply UniTESK unmodified to protocol conformance testing, as well as some limitations of UniTESK toolkits. Nevertheless all these obstacles were successfully overcome without going beyond UniTESK concepts.

Keywords: conformance testing, UniTESK, formal methods, test automation, Model-based testing, protocol testing, formal specification, protocol models.

References

[1]. CCITT Recommendation Z.100. Specification and Description Language (SDL). Geneve, Switzerland: ITU, 1993. 245 p.
[2]. ISO/IEC 9074. Information Processing Systems — Open Systems Interconnection. Estelle — A Formal Description Technique based on an Extended State Transition Model. Geneve, Switzerland: ISO, 1989 1 я редакция,1997 2-я редакция. Отозван 06.05.1999.
[3]. ISO/IEC 8807. Information Processing Systems — Open Systems Interconnection. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. Geneve, Switzerland: ISO, 1989. 142 p.
[4]. ITU-T X.680 (11/08) Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation. ITU-T, 2008. 194 p.
[5]. OMG Unified Modeling LanguageTM (OMG UML), Infrastructure . OMG, 2011. 230 p.
[6]. Edmund M. Clarke, Orna Grumberg and Doron Peled. Model checking. MIT Press, 1999. 325 pp.

[7]. Karpov Yu. MODEL CHECKING. BHV-Peterburg, 2010, 560 pp.
[8]. M. Diaz. Petri Nets: Fundamental Models, Verification and Applications. Willey, 2013г. 656с.
[9]. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann, San Francisco (2007).
[10]. Blackburn, M., Busser, R., Nauman, A.: Why Model-Based Test Automation is Different and What You Should Know to Get Started. Software Productivity Consortium, NFP (2004).
[11]. Dalal, S.R., Jain, A., Karunanithi, N., Leaton, J.M., Lott, p.M., Patton, G.C., Horowitz, B.M.: Model-Based Testing in Practice. In: Proceedings of the ICSE 1999 (May 1999).
[12]. I. Bourdonov, A. Kossatchev, V. Kuliamin, A. Petrenko. UniTesK Test Suite Architecture. Proceedings of FME'2002, Kopenhagen, Denmark, LNCS 2391:77-88, Springer-Verlag, 2002.
[13]. V. V. Kuliamin, A. K. Petrenko, A. S. Kossatchev, and I. B. Burdonov. 2003. The UniTesK Approach to Designing Test Suites. Program. Comput. Softw. 29, 6 (November 2003), 310-322.
[14]. V. Kuliamin, A. Petrenko. Evolution of UniTESK Test Development Technology. ISP RAS Proceedings, vol. 26, issue 1, 2014. pp. 9-26. DOI: 10.15514/ISPRAS-2014-26(1)-1. (in Russian).
[15]. F. p. Hennie. Fault detecting experiments for sequential circuits. Proc. 5-th Ann. Symp. Switching Circuit Theory and Logical Design, 1964. p. 95-110.
[16]. M. Vasilevsky. Error diagnostics in state machines. Cybernetics and system analysis, vol. 9, № 4, 1973. pp. 98-108 (in Russian).
[17]. T. S. Chow. Testing software design modeled by finite-state machines. IEEE Trans. on Software Engineering, vol. 4, no. 3, 1978. p. 178-187.
[18]. J. A. Bergstra, J. W. Klop. Algebra of Communicating Processes with Absraction. Theoretical Computer Science, 37(1), 1985. pp. 77-121.
[19]. p. A. R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985; e-book http://www.usingcsp.com/cspbook.pdf, 2004. 260 p.
[20]. R. Milner. Communication and Concurrency. Prentice-Hall, 1989. 260 p.
[21]. N.A. Lynch, M.R. Tuttle, "An Introduction to Input/Output Automata" CWI-Quaterly 3, 1989, P. 219-246. [PDF] http://www.markrtuttle.com/papers/lt89-cwi.pdf.
[22]. F. Maraninchi. Operational and compositional semantics of synchronous automaton compositions. CONCUR'92 Proceedings, LNCS 630. Springer-Verlag, 1992.
[23]. ISO/IEC 7498. Information technology – Open Systems Interconnection – Basic Reference Model. Geneva, Switzerland: ISO, 1994.
[24]. Ed. Brinksma. A theory for the derivation of tests. Proc. IFIP WG6.1 8th Intl. Symp. on Protocol Specification, Testing, and Verification, North-Holland, S. Aggarwal and K. Sabnani Ed. pp. 63-74, 1988.
[25]. K. K. Sabnani and A. T. Dahbura. A protocol test generation procedure. Computer Networks and ISDN Systems, vol. 15, no. 4, pp. 285-297, 1988.
[26]. S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. IEEE Trans. on Software Eng., vol. 17, pp. 591-603,1991.
[27]. G. Luo, A. Petrenko, and G. v. Bochmann. Selecting test sequences for partially specified nondeterministic finite state machines, Proceedings of the IFIP Seventh International Workshop on Protocol Test Systems, Japan, 1994, pp. 95-110.
[28]. N. V. Evtushenko, A. V. Lebedev, A. F. Petrento, "Development of the checking set for a component of a sequential automaton network". Automation and Remote Control, 1994, 55:8, 1203–1210

[29]. J. C. Fernandez, C. Jard, T. Jeron, C. Viho. Using on-the-Fly Verification Techniques for the Generation of Test Suites Proceedings of the 8th International Conference on Computer Aided Verification, LNCS 1102, Springer-Verlag, 1996, P. 348-359.
[30]. J. Tretmans. Test Generation with Inputs, Outputs, and Repetitive Quiescence. Software — Concepts and Tools, 17(3):103-120, 1996.
[31]. D. P. Sidhu and T.-K. Leung. Formal methods for protocol testing: a detailed study. IEEE Trans. Soft. Eng., vol. 15, no. 4, pp. 413-426, 1989.
[32]. D. Lee, M. Yannakakis. Principles and methods of testing finite state machines — a survey. Proc. IEEE, 84(8):1090-1123, 1996.
[33]. G. v. Bochmann, A. Petrenko. Protocol Testing: Review of Methods and Relevance for Software Testing. Proc. of ACM SIGSOFT ISSTA'1994, Software Engineering Notes, Special Issue, pp. 109—124.
[34]. A. Petrenko. Fault Model-Driven Test Derivation from Finite State Models: Annotated Bibliography. In F. Cassez, p. Jard, B. Rozov, M. Dermot, eds. Modeling and Verification of Parallel Processes: 4-th Summer School, Nantes, France, LNCS 2067, pp. 196-200, Springer-Verlag, 2000.
[35]. ISO/IEC 9646. Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts. Geneva: ISO, 1994. 46 p.
[36]. ITU-T Recommendation Z.500. Framework on formal methods in conformance testing. Geneve, Switzerland: ITU, 1997. 49 p.
[37]. Glenford J. Myers, Corey Sandler, Tom Badgett. The Art of Software Testing, 3rd Edition. Wiley, 2011. p. 240.
[38]. J. Tretmans, An Overview of OSI Conformance Testing. Translated and adapted from: J. Tretmans and J. van de Lagemaat, Conformance Testen, in Handboek Telematica, Vol. II, pages 1--19. Samson, 1991.
[39]. ISO/IEC 9646. Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 2: Abstract Test Suite specification. Geneva: ISO, 1994. 33 p.
[40]. Information technology – Open systems interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation (TTCN). 1-е издание. Geneva, Switzerland: ISO, 1992.
[41]. Information technology – Open systems interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation (TTCN). 2-е издание. Geneva, Switzerland: ISO, 1998.
[42]. ETSI ES 201 873-1 V3.1.1. Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language. Sophia-Antipolis, France: ETSI, 2005. 210 p.
[43]. OMG formal/05-07-07. UML Testing Profile. Version 1.0. Needham, USA: Open Management Group, 2005. [PDF, PostScript] (http://www.omg.org/cgi-bin/doc?formal/05-07-07).
[44]. L. Ebrecht, M. Schacher, p. Bühler. Test Specification in XML – the most important Element for Test Automation. ARTiSAN Benutzerforum D.A.CH, 2005.
[45]. JUnit testing framework. http://www.junit.org.
[46]. IPv6 Test Suite TAHI. http://www.tahi.org/.
[47]. Digital cellular telecommunications system (Phase 2+) (GSM); Handover procedures (GSM 03.09 version 5.1.0) ETSI, Sophia-Antipolis, France, 1997. 81 p.

[48]. ETSI ETS 300 838. Integrated Services Digital Network (ISDN); Harmonized Programmable Communication Interface (HPCI) for ISDN. ETSI, Sophia-Antipolis, France, 1998. 546 p.
[49]. C. Jard, T. Jéron. TGV: Theory, principles and algorithms. International Journal on Software Tools for Technology Transfer (STTT), vol. 7(4). Berlin: Springer, 2005. p. 297 – 315
[50]. E. Farchi, A. Hartman, S.S. Pinter. Using a Model-based Test Generator to Test for Standard Conformance. IBM System Journal - special issue on Software Testing. Volume 41(1), 2002. p. 89 - 110.
[51]. G. Friedman, A. Hartman, K. Nagin, T. Shiran. Projected State Machine Coverage for Software Testing. Proceedings of ISSTA 2002 International Symposium on Software Testing and Analysis. New York, USA: ACM Press, 2002. p. 134 – 143.
[52]. Margus Veanes, Colin Campbell, Wolfgang Grieskamp, Wolfram Schulte, Nikolai Tillmann, and Lev Nachmanson, Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer, in Formal Methods and Testing, vol. 4949, pp. 39-76, Springer Verlag, 2008
[53]. A. Belinfante, J. Feenstra, R. de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, L. Heerink. Formal test automation: A simple experiment. G. Csopaki, S. Dibuz, K. Tarnay, editors. 12th Int. Workshop on Testing of Communicating Systems. Budapest, Hungary: Kluwer Academic Publishers, 1999. p. 179-196.
[54]. Axel Belinfante. JTorX: a Tool for On-Line Model-Driven Test Derivation and Execution. In: Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2010. LNCS vol. 6015, pp. 266-270. Springer.
[55]. N. Goga. A probabilistic coverage for on-the-fly test generation algorithms. Automated Verification of Critical Systems (AVoCS '03), 2003.
[56]. N. V. Pakulin and A. V. Khoroshilov. 2007. Development of formal models and conformance testing for systems with asynchronous interfaces and telecommunications protocols. Program. Comput. Softw. 33, 6 (November 2007), 316-335.
[57]. V. V. Kuliamin, A. K. Petrenko, N. V. Pakoulin, A. S. Kossatchev, I. B. Bourdonov. Integration of Functional and Timed Testing of Real-time and Concurrent Systems. Proceedings of the 5-th International Conference on Perspectives of System Informatics, July 9-12, 2003, Novosibirsk, Russia; LNCS 2890, Springer, 2003, pp. 450-461.
[58]. V. V. Kuliamin, A. K. Petrenko, N. V. Pakoulin. Practical Approach to Specification and Conformance Testing of Distributed Network Applications. Proceedings of the 2-nd International Service Availability Symposium, April 25-26, 2005, Berlin, Germany; LNCS 3694, Springer, 2005, pp. 68-83.
[59]. V. V. Kuliamin, A. K. Petrenko, N. V. Pakoulin. Extended Design-by-Contract Approach to Specification and Conformance Testing of Distributed Software. Proceedings of the 9-th World Multiconference on Systemics, Cybernetics, and Informatics, Model Based Development and Testing Workshop, July 10-13, 2005, Orlando, Florida, USA, pp. 65-70
[60]. C. A. R. Hoare. An axiomatic basis for computer programming. Communications of the ACM, volume 12 №10, 1969. p. 576-580.
[61]. R. P. Draves, A. Mankin, B. D. Zill. Implementing IPv6 for Windows NT. Proceedings of the 2nd USENIX Windows NT Symposium, Seattle, WA, August 3–4, 1998
[62]. I. Agamirzian, S. Groshev, A. Khoroshilov, G. Kluchnikov, A. Kossatchev, V. Omeltchenko, N. Pakulin, A. Petrenko, V. Shnitman. Using formal methods for MSR IPv6 conformance testing. Proc. of Next Generation Internet, Yaroslavl, 2002. Pp. 29-33 (in Russian)

[63]. G. Kluchnikov, A. Kossatchev, A. Petrenko, N. Pakulin, V. Shnitman. Using formal methods in testing of an IPv6 implmentation. ISPRAS Proceedings, vol. 4, pp. 121-140, 2003 (in Russian).
[64]. G. Kluchnikov, A. Kossatchev, A. Petrenko, N. Pakulin, V. Shnitman. Using formal methods in testing of Mobile IPv6 . Proc. of Next Generation Internet-2, Yaroslavl, pp. 20-25, 2003 (in Russian).
[65]. D. Zatsepin, V. Shnitman. Aspects of applications of UniTESK to testing of mobility functions of IPv6 protocol. ISPRAS Proceedings, vol. 13(1), pp. 143-170, 2007 (in Russian)
[66]. CTesK 2.1: SeC Language Reference. ISPRAS, 2005. 167 p.
[67]. G. Kluchnikov, N. Pakulin, V. Shnitman. Automated testing of network services of Internet protocols. Proc. of Scientific services in Internet — 2005, pp. 168-170 (in Russian).
[68]. N. Pakulin. Applying formal methods to testing of implementations of complex modern protocols. Proc. of int. workshop «Go4IT — towards new Internet technologies» 2007, pp. 11-18 (in Russian).
[69]. A. Nikeshin, N. Pakulin, V. Shnitman. Aspects of testing of network-level security services of IPsec version 2. Proc. of Scientific services in Internet — 2008 (in Russian).
[70]. A. Nikeshin, N. Pakulin, V. Shnitman. Development of a test suite for verification of IPsec v2 implementations. ISPRAS Proceedings, vol. 18, 2010, pp. 151-182 (in Russian).
[71]. A. Nikeshin, N. Pakulin, V. Shnitman. Verification of security functions of IPsec v2. Programmirovanie, vol. 37 № 1. 2011. pp. 36-56 (in Russian)
[72]. Bourdonov I.B., Demakov A.V., Jarov A.A., Kossatchev A.S., Kuliamin V.V., Petrenko A.K. and Zelenov S.V. Java Specification Extension for Automated Test Development Proceedings of PSI'2001. Novosibirsk, Russia July 2-6 2001, LNCS 2244:301-307. Springer-Verlag, 2001.
[73]. A. Nikeshin, N. Pakulin, V. Shnitman. Test Suite development for verification of TLS security protocol. IPSRAS Proceedings, vol. 23, pp. 387-404. 2012. DOI: 10.15514/ISPRAS-2012-23-22. (in Russian).
[74]. A. Tugaenko. Conformance testing of extensible mail Internet protocols. Proceedings of Young Researchers Forum «Lomonosov 2010». (in Russian)
[75]. N. Pakulin, A. Tugaenko. Development of conformance test suites for email protocols. Proceedings of APPI-2209, pp 154-160 (in Russian)
[76]. A. Tugaenko. Method for conformance testing of extensible Internet protocols. Proceedings of Young Researchers Forum «Lomonosov 2011». (in Russian)
[77]. N. Pakulin, A. Tugaenko. Model Based Conformance Testing for Extensible Internet Protocols Proceedings of SYRCoSE 2011.
[78]. N. Pakulin, A. Tugaenko, V. Shnitman. Model-based testing of internet e-mail protocols. Proceedings of ISPRAS, vol 20, 2011. p. 125-141. (in Russian)
[79]. N. Pakulin, A. Tugaenko, V. Shnitman. Model-based testing of internet e-mail protocols. Programming and Computer Software, vol. 38, №5, 268-275. 2012
[80]. J. Jacky, "PyModel: Model-based testing in Python", Northwest Python Day 2010.
[81]. Errfix: model-based testing in Ruby. https://code.google.com/p/errfix/
[82]. I. Bourdonov, A. Kossatchev, A. Petrenko, and D. Galter. KVEST: Automated Generation of Test Suites from Formal Specifications. FM'99: Formal Methods. LNCS, volume 1708, Springer-Verlag, 1999, pp. 608–621.

[83]. A.Grinevich, V.Kuliamin, D.Markovcev, A.Petrenko, V.Rubanov, A.Khoroshilov. Using formal methods to ensure conformance for programming standards. // Proceedings of ISPRAS, №10, 2006.
[84]. V.Kuliamin, A.Petrenko, V.Rubanov, A.Khoroshilov. Formalization of interface standards and automated construction of conformance tests. Information technologies. 8:2-7, Moscow, New Technologies, 2007.