МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ РАДИОЭЛЕКТРОНИКИ

На правах рукописи

Иссам Саад

УДК. 621.391

МОДЕЛИ И МЕТОДЫ АНАЛИЗА И ВЕРИФИКАЦИИ ПРОТОКОЛОВ УПРАВЛЕНИЯ В ПРОГРАММНО-КОНФИГУРИРОВАННЫХ СЕТЯХ БАЗИРУЮЩИЕСЯ НА АЛГЕБРЕ КОММУНИКАЦИОННЫХ РАСПРЕДЕЛЕННЫХ РЕСУРСОВ И ГРАФАХ ДОСТИЖИМОСТИ

05.12.02 – телекоммуникационные системы и сети

ДИССЕРТАЦИЯ НА СОИСКАНИЕ УЧЕНОЙ СТЕПЕНИ
КАНДИДАТА ТЕХНИЧЕСКИХ НАУК

Научный руководитель
доктор технических наук, доцент
Дуравкин Евгений Владимирович

.

Харьков -2016

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ACSR - Algebra Of Communicating Shared Resources

API - Application Programming Interface

ETSI - European Telecommunications Standards Institute

IEEE - Institute of Electrical and Electronics Engineers

IETF - Internet Engineering Task Force

IP - Internet Protocol

IRTF - Internet Research Task Force

ITU-T - International Telecommunication Union-Telecommunication

LLDP - Logical Link Discovery Protocol

MIB - Management Information Base

NDB - Non-Directional Beacon

OFDP - OpenFlow Discovery Protocol

ONF - Open Network Foundation

QoS - Qulaty of Service

RTT - Round-Trip Time

SDL - Specification and Description Language

SDN - Software-Defined Networking

SDNRG - Software-Defined Networking Research Group

TCP - Transmission Control Protocol

TLV - Tag Length Value

ToS - Type of Service

TTL - Time To Live

UDP - User Datagram Protocol

UML - Unified Modeling Language

ВВЕДЕНИЕ

Развитие современных мультисервисных сетей сопряжено с рядом трудностей: возрастание объемов и расширение функциональных возможностей предоставляемых сервисов влечет за собой усложнение механизмов управления и повышение требований к оборудованию сетевой инфраструктуры. Так, на сетевое оборудование транспортного и канального уровня на сегодняшний день накладывается ряд задач сеансового и прикладного уровней [20, 143, 144], а также необходимость поддержки множества дополнительных протоколов, отвечающих за управление передаваемой информацией. Сложившаяся ситуация приводит либо к повышению себестоимости сетевой инфраструктуры, либо к возникновению перегрузок и снижению качества предоставляемых сервисов.

Необходимость решения задачи отделения функций управления и функций передачи данных между разными типами оборудования обусловила развитие концепции программно-конфигурируемых сетей (Software-Defined Network, SDN) [1, 5, 25]. В основу концепции SDN положено физическое отделение уровня управления от уровня передачи данных, а также формирование открытого интерфейса взаимодействия, гибкая адаптация сетевой среды в соответствии с требованиями QoS [16, 17], автоматизированное администрирование и обновление компонент. Так, применение концепции SDN позволяет значительно разгрузить физическое оборудование сети и избавится от необходимости поддержки множества служебных протоколов, закрепив за ним лишь функции передачи данных [29, 33].

Функции управления и мониторинга выносятся на отдельное сетевое устройство – контроллер [25, 101, 104]. Контроллер выполняет мониторинг и генерирует управляющие команды, основываясь на требования, предъявляемых к качеству сервисов, и общих характеристиках сети. Основным протоколом, обеспечивающим обмен управляющими сообщениями и являющимся связующим звеном между уровнем управления, и уровнем передачи данных,

является OpenFlow. От корректности функционирования протокола OpenFlow зависит корректность функционирования всей сети в целом [30, 53].

На сегодняшний день протокол OpenFlow не имеет конечной реализации. Множество компаний, разрабатывающих сетевое оборудование (Cisco [21], HP [73], Juniper [49], Dell [78], IBM [77] и ряд других), используют собственные модификации протокола, что зачастую приводит к несовместимости и отсутствию поддержки необходимых функциональных требований.

Отсутствие единой унифицированной спецификации также является причиной ряда ошибок, возникающих в процессе проектирования SDN решения. Например, функциональные особенности версии v.1.1.0 [69, 72] и v.1.3.0 [70] значительно разнятся: в версии v.1.1.0 не включена поддержка типов туннелирования, IPv6 и широковещательных запросов.

Одним из способов устранения возможных расхождений и ошибок является формирование универсального подхода к этапам разработки и последующей поддержки OpenFlow протокола [69-73]. В процессе разработки обязательным является выполнение следующих этапов: анализ предметной области и формирование требований, разработка спецификации, реализация протокола, тестирование и верификация. Стоит отметить, что при реализации протокола OpenFlow следует учитывать возможность параллельного формирования несколько прототипов реализации. Применение спиральной модели жизненного цикла позволяет в полной мере охватить все приведенные этапы [124, 134].

Ввод ряда математических методов и моделей на каждом этапе процесса разработки позволяет обеспечить раннее обнаружение ошибок, что приводит к сокращению временных затрат и снижению стоимости как нового сетевого решения, так и последующих его модификаций. Однако существующие на сегодняшний день математические методы и модели разработки не всегда эффективны при разработке сложных протоколов межуровневого взаимодействия, таких как OpenFlow [143, 144].

Разработка новых или модификация существующих методов и инструментов, применяемых в процессе проектирования OpenFlow, а также ввод подэтапа анализа результатов и обнаружения возможных ошибок на каждом этапе разработки позволят усовершенствовать процесс разработки и повысить эффективность функционирования готового решения.

Актуальность темы диссертации.

SDN представляет собой новую концепцию построения сетевой инфраструктуры и методики предоставления сервисов оконечным пользователям. Основными преимуществами SDN являются гибкая адаптация, открытость интерфейсов и упрощение процесса передачи данных. Однако наряду с преимуществами возникает множество неоднозначностей, которые затрудняют развитие и внедрение концепции в существующую сетевую инфраструктуру. Анализ результатов внедрения архитектуры SDN [24] показал, что большинство ошибок в процессе функционирования готового SDN решений возникает из-за различий в программных реализациях протокола OpenFlow [72], несовместимости требований спецификации [18, 47, 69], отсутствия поддержки управляющих команд OpenFlow коммутаторами [27].

Современные подходы (модели и методы), применяемые при разработке протоколов управления сетевыми ресурсами, не позволяют полноценно реализовать процесс разработки и последующей проверки сетей, основанных на концепции SDN. Так, например, используемые на сегодняшний день, методы формализации носят описательный характер или базируются на логиках низшего порядка [126, 132]. Вычислительная мощность таких методов недостаточна для описания сложных процессов взаимодействия типа «коммутатор-контроллер-коммутатор»: не всегда удается точно описать последовательность смены процессов в сети, правила управления и распределения сетевых ресурсов.

Без применения строгих математических методов достаточно затруднительным является процесс анализа корректности поведения протокола и проверки непротиворечивости требований в рамках спецификации [83, 85,

128]. Также, при решении задачи проверки соответствия реализации протокола его спецификации, возникает ряд трудностей, связанных с несовершенством современных методов тестирования и верификации: зависимости от выбранного метода (тестирование или формальная верификация), возможно выявление ограниченного количества ошибок [134] или, наоборот, возникновение эффекта «комбинаторного взрыва» пространства исследуемых состояний [121].

Задачами развития и совершенствования концепции SDN, а именно усовершенствование методов разработки и внедрения протокола OpenFlow, на сегодняшний день занимается множество передовых компаний, таких как Cisco, HP, IBM, Juniper, и академических кругов - SDNRG, ONF, IRTF, IETF, ETSI. Методам разработки протокола OpenFlow, в частности, анализу корректности поведения и проверке соответствия спецификации, посвящен ряд работ иностранных исследователей Макломса Б., Канини М., Менезеса П., Томаса А., Шервуда Р. Вопросам анализа, разработки и адаптации OpenFlow решений также посвящены работы российских и украинских ученых Захарова В. А., Смелянского Р. Л., Кучерявого А.Е., Чемерецкого Е.В. и др.

Основываясь на результатах анализа существующих подходов к разработке и внедрению протоколов SDN, в частности протокола OpenFlow, в работе предложен ввод математических моделей и методов как на этапах анализа предметной области и разработки спецификации, так и этапах реализации протокола и верификации.

Предлагаемые методы позволяют учитывать отличительные особенности функционирования протокола OpenFlow. Их применение на ранних этапах жизненного цикла позволяет своевременно выявить и устранить множество критических ошибок. Таким образом, предлагаемый подход позволяет решить ряд актуальных задач, влияющих прогрессивное развитие концепции SDN.

Связь работы с научными программами, планами, темами.

Задачи диссертационного исследования тесно связаны с положениями «Концепции Национальной программы информатизации», «Концепции

развития электронного управления в Украине», «Концепции создания и функционирования информационной системы электронного взаимодействия государственных электронных информационных ресурсов». Материалы диссертационной работы реализованы в ходе выполнения научно-исследовательской работы № 299-1 «Методы проектирования телекоммуникационных сетей NGN и управления их сетевыми ресурсами» (№ ДР 0115U002432).

Цель и задачи исследования.

Целью работы повышение качества управления в программно-конфигурируемых сетях за счет совершенствования процессов разработки и внедрения протоколов управления.

Объект исследования: процесс разработки, анализа и верификации протокола OpenFlow.

Предмет исследования: математические модели и методы анализа и верификации протокола OpenFlow, основанные на алгебре коммутационных распределенных ресурсов и графах достижимости.

Методы исследования основаны на применении алгебры коммуникационных распределенных ресурсов, в частности, использовании ее семантических и синтаксических составляющих при разработке метода формализации спецификации протоколов SDN; математическом аппарате E-сетей и методах анализа функциональных и нефункциональных свойств (дерево достижимости) при решении задачи проверки корректности протокола; формальных методах верификации (символьный подход, проверка на моделях), применяемых в процессе разработки метода проверки эквивалентности модели реализации протокола требованиям спецификации.

Для достижения поставленной цели исследования необходимо решение следующих задач.

Основная задача: исследование особенностей протокола OpenFlow и разработка методов функционального и структурного анализа, которые позволят обеспечить совершенствование концепции SDN.

Частные задачи исследования:

- разработка метода формализации требований спецификации протокола OpenFlow, представленных на подмножестве естественного языка, с помощью алгебры коммуникационных распределенных ресурсов;

- разработка методов поиска и устранения противоречий между требованиями в рамках спецификации протокола OpenFlow;

- разработка метода синтеза модели E-сети реализации протокола на основе формализмов алгебры коммуникационных распределенных ресурсов;

- разработка метода построения дерева достижимости для модели реализации протокола, представленной на базе аппарата E-сети;

- разработка алгоритма последовательного разбора и проверки соответствия множества ветвей дерева достижимости моделей реализации и спецификации протокола;

- разработка метода построения контрпримера, позволяющего выявить причины возникновения ошибок и несоответствий в реализации протокола, а также указать пути их устранения.

Научная новизна результатов исследования.

1. Получил дальнейшее развитие математический аппарат алгебры коммуникационных распределенных ресурсов в качестве инструмента формализации требований спецификации протокола OpenFlow. Новизна предлагаемого подхода заключается в том, что операторы алгебры коммуникационных распределенных ресурсов применяются с целью формализации процессов и событий и определения порогового значения длительности процессов. Впервые предложен метод проверки поиска противоречий в рамках спецификации путем сопоставления формализмов алгебры коммуникационных распределенных ресурсов.

2. Получил дальнейшее развитие аппарат E-сетей как средство моделирование поведения протокола OpenFlow. Впервые предложено применение управляющих переходов для моделирования свойств протокола и ввод в атрибуты переходов численных значений.

3. Получили дальнейшее развитие алгебраические методы анализа моделей протокола, новизна заключается в разработке алгоритма построения дерева достижимости модели E-сети с возможностью выявления количества возможных циклов и подциклов. Применение данных инструментов позволило решить задачу проверки корректности поведения и оценки соблюдения функциональных требований.

4. Впервые разработан метод верификации протокола OpenFlow, который позволяет выполнить как частичную проверку на основе шаблонов, так и полную проверку, избегая при этом эффекта «комбинаторного взрыва» пространства исследуемых состояний. Также предложен алгоритм построения контпримера на основе анализа дерева достижимости. Новизна данного метода является вывод цепочек, указывающих локацию расхождения реализации со спецификацией протокола OpenFlow и возможные пути устранения расхождений. На основе полученного контрпримера предложены способы совершенствования реализации протокола.

Научное и практическое значение полученных результатов.

Предлагаемые в диссертационном исследовании математические модели и методы позволяют усовершенствовать процессы разработки и последующую эксплуатацию как оборудования, поддерживающего протокол OpenFlow, та и всей сети в целом. Они могут применяться в качестве научно-методической базы для последующих исследований функционирования протокола OpenFlow и создания новых средств поддержки процесса разработки.

Применение разработанных моделей и методов позволит существенно сократить эксплуатационные затраты и повысить качество предоставления сервисов. Благодаря применению предлагаемых моделей и методов возможно раннее обнаружение несоответствия как на этапах анализа требований спецификации и функциональных характеристик реализации протокола, так и при решении проблем, возникающих при совместной работе уже существующих реализаций одного и того же протокола.

Результаты диссертационной работы могут быть рекомендованы при проектировании и совершенствовании телекоммуникационных систем в целом.

Личный вклад соискателя.

Все основные научные положения, результаты, выводы и рекомендации диссертационной работы получены автором самостоятельно. В публикациях, написанных в соавторстве, соискателю принадлежат следующие результаты:

- в статье [111] автором разработаны формализмы, определяющие правила композиции и согласования комплексных Web-сервисов, которые позволяют объединить элементы распределенной мультисервисной системы в единое целое.

- в статье [45] автором предложен метод верификации распределенных систем, базирующийся на модельном подходе, позволяющий учитывать асинхронную природу комплексных сервисов, а также выполнять динамическую проверку.

- в статье [114] автором проведен анализ развития концепции SDN, сформирован ряд основных задач исследования SDN. На основе полученных результатов предложены методы совершенствования процессов разработки и внедрения SDN, а также приведен метод проверки соответствия компонентов сети, функционирующей на основе протокола OpenFlow, требованиям спецификации.

- в статье [120] автором в рамках решения задачи анализа и верификации предложен метод, в основе которого лежит модельный подход и применение аппарата E-сетей, также приведен метод анализа таких свойств модели реализации протокола, как достижимость, ограниченность, безопасность и живость.

- в статье [113] автором предложено применение математического аппарата алгебры коммуникационных распределенных ресурсов в качестве средства формализации спецификации, аппарат алгебры коммуникационных распределенных ресурсов позволяет полноценно формализовать причинно-следственные связи между событиями и процессами, формирующими

поведение протокола. Также в рамках решения задачи проверки на непротиворечивость предложено применение двух взаимодополняющих методов.

- в статье [47] предложена методика анализа и верификации основанная на модельном подходе и проверке соответствия последовательности смены состояний протокола OpenFlow. Предлагаемый автором подход позволяет учитывать асинхронный характер сложных процессов и выполнения динамическую проверку.

- в статье [115] автором предложено введение метрики оценки масштабируемости и оценки эффективности функционирования контроллеров, а также приведен анализ процесса изменения коэффициента масштабируемости для трех основных структур уровня управления Software-Defined Networking: централизованной, децентрализованной, иерархической. По предложенным формализмам произведена оценка времени отклика для разных типов контроллеров.

- в статье [44] автором в рамках задачи проверки на непротиворечивость предложено применение двух взаимодополняющих методов: метод последовательного нахождения и сверки всех формализмов спецификации, содержащих утверждение, непротиворечивость которого необходимо проверить. Данный метод позволяет эффективно решить ряд узконаправленных задач; метод проверки достижимости графа состояний, соответствующих требованиям спецификации, который позволяет выявить все множество противоречий в рамках всей спецификации или ее фрагмента. Приведен пример применения каждого из методов.

Апробация результатов диссертации.

Апробация основных положений диссертационной работы была проведена в ходе четырех конференций и одного форума, а именно в трудах Второй Международной научно-технической конференции «Проблемы инфокоммуникаций. Наука и технологии (PICS & T)» (2014, г. Харьков, ХНУРЭ), во второй Всеукраинской научно-технической конференции

«Проблемы развития глобальной системы связи, навигации, наблюдения и организации воздушного движения CNS/ATM» (2014., г. Киев, Институт аэронавигации НАУ), в Первой Международной научно-практической конференции «Проблемы телекоммуникаций» (2014, г. Харьков, ХНУРЭ), в Первой Международной научно-технической конференции «Проблемы электромагнитной совместимости перспективных беспроводных сетей связи ЭМС -2015» (2014, г. Харьков, ХНУРЭ) и в 19-м Международном молодежном форуме «Радиоэлектроника и молодежь в XXI веке» (2015, г.. Харьков, ХНУРЭ).

Публикации.

Основные результаты диссертационной работы опубликованы в восьми научных статьях. Из них шесть научных статей опубликованы в научных профессиональных изданиях Украины [111, 113-116, 119] и две статьи в зарубежных профессиональных изданиях [44, 47]. Материалы диссертационных исследований тезисно опубликованы в пяти сборниках научно-технических конференций, в том числе и публикации, проиндексированы и научных базах IEEExplore.

РАЗДЕЛ 1
АНАЛИЗ ФУНКЦИОНАЛЬНЫХ ОСОБЕННОСТЕЙ ПРОТОКОЛА OPENFLOW, ПЕРСПЕКТИВЫ И ПРОБЛЕМЫ НА ПУТИ РАЗВИТИЯ

1.1 Анализ развития концепции Software-Defined Networking

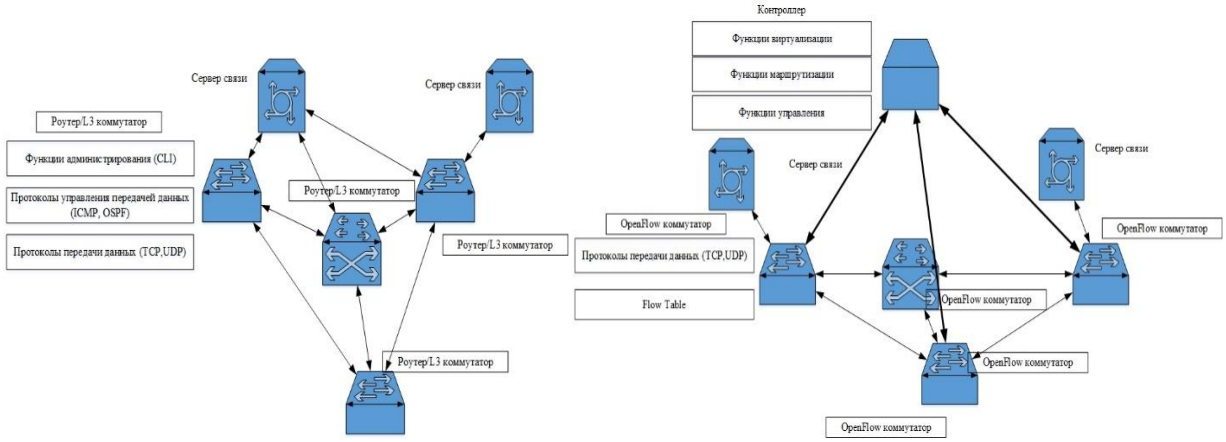
С возрастанием бизнес-потребностей пользователей и организаций связано возрастание популярности и стремительное развитие современных мультисервисных сетей и облачных сетевых решений. Доступность и гарантированное качество предоставляемых сервисов в режиме 24/7 влечет за собой потребности совершенствования механизмов управления и передачи, а, следовательно, возрастание сложности протоколов, которые отвечают за передачу и предоставление сервисов [29, 65, 103, 141, 143, 155].

На сегодняшний день число только стандартизированных протоколов всемирно известными комитетами IEEE [40], IETF [26], ETSI [28], ITU-T [58-60] превышает 900. В сложившейся ситуации на сетевое оборудование, в частности, маршрутизаторы и L3 коммутаторы, наряду с расширенными требованиями к передаче данных, накладывается ряд дополнительных требований по управлению и мониторингу состояния сети. Зачастую реализация протоколов передачи и протоколов управления осуществляется на одном и том же сетевом оборудовании, что приводит к возрастанию стоимости оборудования, усложнению процесса и увеличению времени предоставления сервисов [143].

В качестве решения современные провайдеры и крупные IT-организации при построении мультисервисных сетей и предоставлении сервисов с целью снижения стоимости оконечных услуг и уменьшению времени их предоставления все чаще прибегают к использованию концепцию SDN [2, 5, 21, 24].

Основной идеей SDN является разделение уровня управления сетью (control plane) и уровня передачи трафика (forwarding plane) [5, 25, 84]. В соответствии с

концепцией SDN вся логика и функции управления переносится на отдельное централизованное устройство – контроллер [102, 103, 113]. Именно на контроллере реализуется функции управления и мониторинга, необходимые для полноценной работы сети. В отличие от SDN, в традиционных сетях данные функции реализованы в одном устройстве, на базе общего (единого) набора системной логики, поэтому их отделение невозможно [27, 65, 70]. Отличительные особенности традиционной сети от мультисервисной сети, построенной на основе концепции SDN, приведены на рис. 1.1.



а) б)

Рис.1.1. Структурная модель традиционной сети передачи данных (а), модель SDN сети (б)

Основными принципами, заложенными в основу эффективного функционирования SDN, являются [25, 87]:

- централизация уровня управления;
- поддержка функций виртуализации сетевой инфраструктуры, что в значительной мере расширяет возможности центров обработки данных;
- унифицированный интерфейс между уровнем управления и уровнем передачи данных;
- использование открытых протоколов взаимодействия, что позволяет добиться максимальной совместимости сетевых реализаций.

К основным преимуществам SDN относятся быстрое развертывание, гибкое комбинирование множества сетевых функций на одной серверной

платформе, простота и систематизированный характер процедуры администрирования. По статистическим данным отчета Infonetics Research «SDN and SDN Service Provider Survey» [41], применение концепции SDN позволяет значительно снизить материальные затраты на такие виды услуг, как модернизация сервисов, управление, администрирование, техподдержка и (рис.1.2).

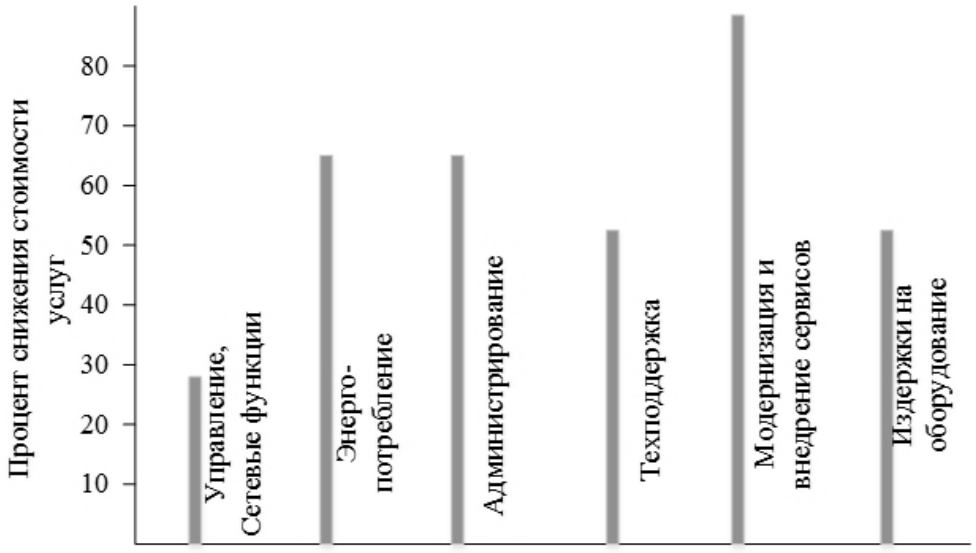


Рис. 1.2. Процент снижения стоимости услуг при внедрении концепции SDN

К 2020 году по данным опроса Infonetics Research планируется масштабный запуск SDN в эксплуатацию, как для центров обработки данных, так и крупных IT-организаций [49, 52, 78]. На сегодняшний день свои решения имеют такие корпорации как Google, Amazon, Netfix, Juniper.

1.1.1 Анализ архитектуры Software-Defined Networking

Формально архитектура SDN содержит три уровня [1, 5, 24]:

- уровень приложений. Данный уровень включает множество прикладных программ, формирующих ряд условий и требований, предъявляемых к сетевой среде, выполнение которых необходимо для предоставления сервиса в соответствии с заданными показателями QoS;

- уровень управления. На уровне управления реализованы алгоритмы, обеспечивающие организацию сетевой среды в соответствии с сформированными требованиями;

- уровень передачи данных. Данный уровень состоит из множества физических сетевых элементов (маршрутизаторов, коммутаторов, серверов и других) и связей между ними.

Элементами архитектуры SDN в соответствии с [5, 24] являются: приложения SDN, SDN контроллер, управляющие агенты, функции которых заложены в OpenFlow коммутаторы, VlowVisor или интерфейс, отвечающий за передачу управляющей информации, компоненты управления и администрирования. Логическое представлениеархитектурыSDN приведено на рис. 1.3.

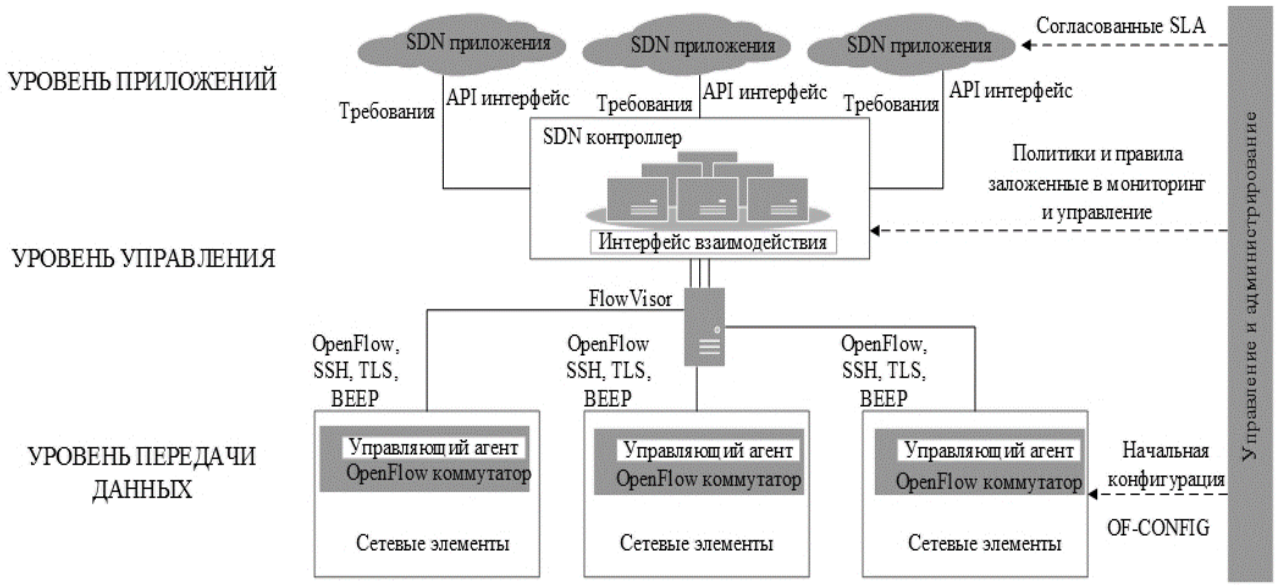


Рис.1.3. Архитектура SDN, основные компоненты и их взаимодействие

Приложения SDN. Приложения, которые предоставляют конечным пользователям желаемые сервисы. Приложения SDN содержат ряд требований к состоянию и поведению сетевой инфраструктуры.

Контроллер SDN. Контроллер выступает единой централизованной точкой управления, который взаимодействует с уровнем приложений посредством открытого интерфейса API, а также выполняет мониторинг и управления физическими устройствами сети посредством открытого интерфейса - протокола OpenFlow [68-71].

Применение контроллера как единой интеллектуальной точки управления позволяет значительно упростить логику работы и стоимость оборудования

SDN архитектуры, так как исчезает необходимость в поддержке и обработкемножества стандартов и протоколов управления на транспортном уровне [72, 73].

Архитектура контроллера состоит из нескольких уровней, каждый из которых отвечает за ряд необходимых функциональных возможностей. Основными уровнями SDN контроллера являются: уровень взаимодействия с сетью; уровень обработки OpenFlow сообщений; уровень обработки событий; уровень сетевых сервисов и внутренних приложений; интерфейс для сетевых приложений контроллера; уровень сетевых приложений [27, 30, 101].

OpenFlow коммутатор. Коммутаторы OpenFlow обеспечивают непосредственное взаимодействие сетевой инфраструктуры с уровнем управления [68-70]. Коммутатор обеспечивает сбор статистических данных о структуре, состоянии и характеристиках уровня передачи данных, создает метки в таблицах переадресации и перенаправляет их контроллеру для принятия дальнейших решений.

FlowVisor. FlowVisorявляетсяответственным за распределение управляющей информации между потоками данных [86, 143]. FlowVisor представляет собой прокси-сервер между множеством коммутаторов и контроллером. FlowVisor обеспечивает виртуализацию потоков управляющих пакетов в отдельные срезы сети (slices) [141, 143], каждый из таких потоков имеет свою логику управления и передачи.

Компоненты управления и администрирования. Набор статических данных, которые включают внешние задачи: координацию политик и правил, установленных при проектировании бизнес-модели архитектуры SDN, начальная конфигурация оборудования и правила распределение сетевых ресурсов.

1.1.2 Анализ протоколов взаимодействия Software-Defined Networking

В соответствии с [5, 102, 104, 144] требования, предъявляемые к среде передачи данных, поступают от уровня приложений к контроллеру посредством Northbound API. При этом OpenFlow коммутаторы через

Southbound API передают контролеру информацию о текущих характеристиках сети. На основании анализа данных, полученных от OpenFlow коммутаторов, и множество требований, полученных от уровня приложений, SDN контролер принимает управляющие решения и формирует соответствующие команды управления. Сформированные команды отправляются OpenFlow коммутаторам. На основе полученных команд OpenFlow коммутаторы осуществляют передачу данных оконечным пользователям.

Основными протоколами SDN являются OF-CONFIG [66], OpenFlow Discovery Protocol [81, 104] и непосредственно OpenFlow протокол, который обеспечивает передачу управляющей информации между контролером и OpenFlow коммутаторами.

В процессе установления и управления соединением между OpenFlow коммутаторами и контроллером SDN значительную роль также играют протокол OF-CONFIG и OpenFlow Discovery Protocol (OFDP).

OF-CONFIG предоставляет возможность удаленного конфигурирования параметров SDN оборудования. Так, OF-CONFIG принимает участие в процессе формирования таблиц переадресации и принятии решений относительно действий протокола OpenFlow.

OpenFlow Discovery Protocol дает возможность исследовать топологию сети, осуществить сбор статистических данных и уведомлять об изменениях в топологии сети.

Основные этапы конфигурирования и обмена управляющей информацией, приведенные на рис.1.4, включают следующие шаги:

1. Начальная конфигурация оборудования. OF-CONFIG является первичным протоколом, который необходим для полноценного функционирования SDN. Протокол OF-CONFIG позволяет конфигурировать статические параметры OpenFlow коммутатора, выделяет диапазоны виртуальных и физических портов, устанавливает приоритетность передаваемых данных, тип QoS и другие. В качестве точки удаленного конфигурирования может выступать как сетевой сервис (приложение), так и

администратор сети. Однако спецификация протокола OF-CONFIG не дает рекомендаций относительно процесса конфигурирования и последующей передачи данных OpenFlow коммутатору.

2. Контроллер SDN осуществляет сбор информации о начальной топологии сети. Сбор сведений о сетевой топологии осуществляется при помощи OpenFlow Discovery Protocol (рассылки пакетов протокола LLDP). В этом случае контроллер отправляет на все коммутаторы своей зоны обслуживания LLDP пакет.

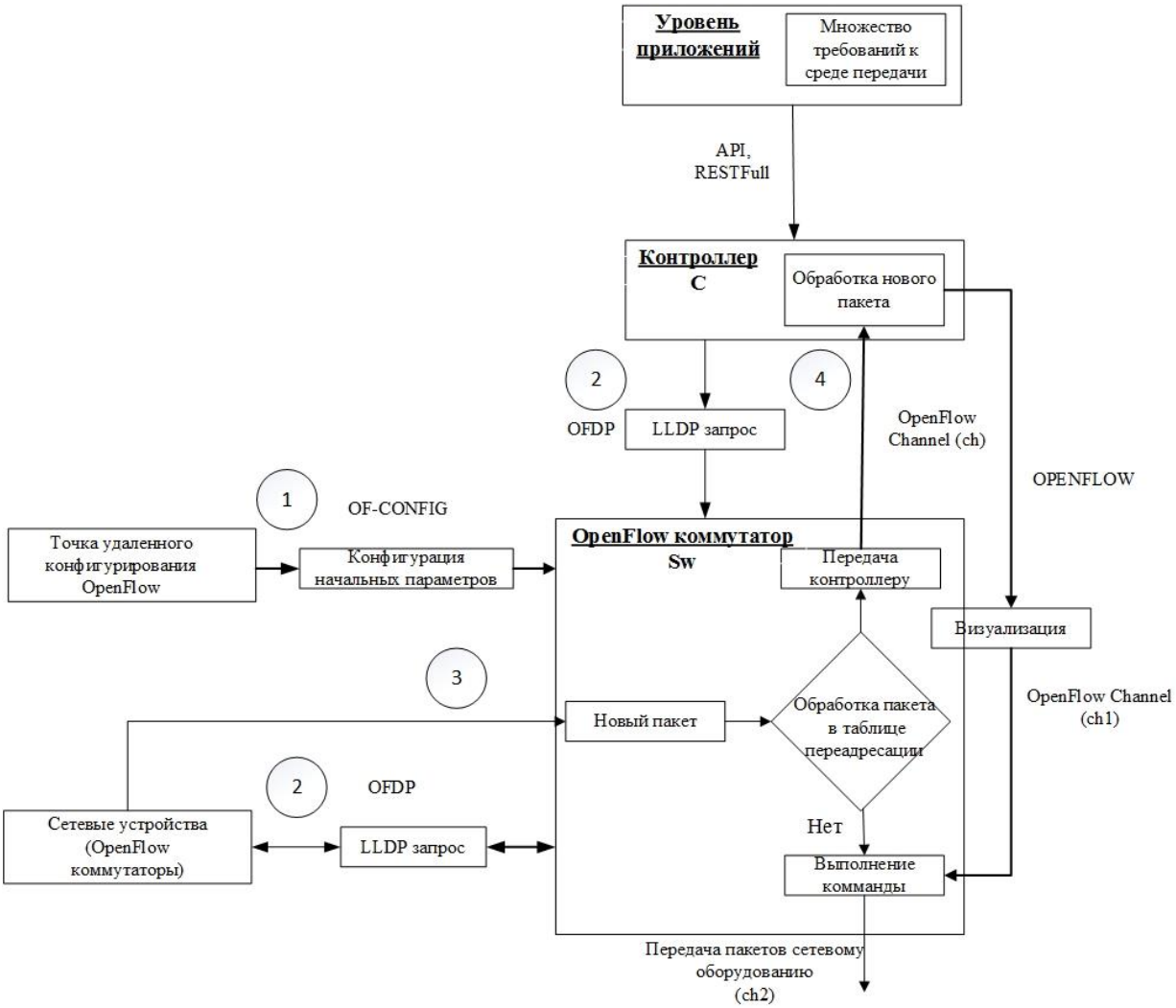


Рис. 1.4. Взаимодействие между элементами SDN архитектуры

1. OpenFlow коммутатор на один из портов получает сообщение, содержащее служебную информацию о типе пересылаемых данных. Взаимосвязь между OpenFlow коммутатором и другим сетевым оборудованием может осуществляться при помощи любого протокола передачи данных.

2. Перенаправление пакета контроллеру осуществляется в том случае, если в таблице переадресации коммутатора не найдено ни одного совпадения для поля *Match* [5, 68-71]. В поле *Match* пересылаемого пакета содержится следующая информация: номер входящего порта, Ethernet адреса источника и получателя, тип Ethernet пакета, идентификатор и приоритет VLAN, IP адреса источника и получателя, тип протокола IP, Type of Service (ToS) биты протокола IP и TCP/UDP порты источника и получателя временные метки.

OpenFlowобеспечивает доступ, обмен информацией и доставку управляющих команд элементам сетевой инфраструктуры [69].

Функционирование протокола OpenFlow основывается на трех ключевых понятиях [68, 143, 144]:

- все коммутаторы физической сети передачи данных являются OpenFlow-совместимыми;
- контроллеры, находящиеся в плоскости управления, должны поддерживать все типы команд, описанные в спецификациях протокола OpenFlow;
- между уровнем управления и уровнем передачи должен быть обеспечен стабильный и безопасный канал передачи управляющей информации.

Протокол OpenFlow поддерживает три типа сообщений: контроллер-коммутатор, асинхронные и симметричные. При этом, каждый тип сообщений имеет несколько подтипов.

Выпуском OpenFlow совместимого оборудования на сегодняшний день занимается множество компаний: Cisco, IBM, Juniper, HP, Ericsson, Arista и другие. При этом, программные реализации протокола на сетевом оборудовании различных разработчиков могут существенно различаться. Спецификации протокола OpenFlow также претерпевают постоянных изменений. В таком случае применение сетевого оборудования, которое поддерживает различные версии протокола, не гарантирует качественного предоставления сервисов.

К основным проблемам, возникающим в процессе функционирования сетей, построенных на основе концепции SDN, относятся отсутствие стандартизированных протоколов взаимодействия и наличие неопределенностей и расхождений в существующих версиях спецификаций протоколов [68-71].

Неполнота и наличие неопределенностей при формировании требований, а также ошибки, допущенные при анализе предметной области, могут привести к некорректному функционированию протоколов, следствием чего может быть выход из строя, как ряда сетевых элементов, так и всей сети в целом [35, 57, 85]. Отсутствие единых стандартов также существенно влияет на логику функционирования реализуемого оборудования: производителям необходимо учитывать, нормализировать и внедрять в свои решения все требования, заложенные в различных версиях OpenFlow протоколов.

1.2 Методы формального описания спецификации протокола OpenFlow

Методы формального описания требований, предъявляемым к протоколам информационного обмена, в частности протокола OpenFlow, могут быть разделены на методы графической и математической нотации.

Методы графической нотации приобрели довольно широкое применение в процессе формирования и структуризации множества инвариантов поведения протокола. Методы базируется на применении разновидностей графического языка, состоящего из текстовых описаний, диаграмм, блок-схем и алгоритмов функционирования [127, 132, 150, 151]. Основными графическими языками формализации спецификаций являются SDL [88, 100] и UML [152, 153].

На сегодняшний день особое внимание, как разработчиков, так и ученых направленно на апробацию и практическое применение математических методов формализации требований, предъявляемых к функциональным особенностям протоколов. Отличительной особенностью математических средств формализации является однозначность понятий (при условии их

первичного определения), строгий учет и описание причинно-следственных связей, различная степень детализации объектов протокола, наглядность.

При этом выразительная мощность математических методов формализации зависит от используемого типа логики. Основываясь на логическом походе, формализмы спецификации протоколов определяются составляющими трех типов: синтаксическая, семантическая составляющие, а также доказательство истинности утверждений спецификации протокола [99, 126, 136, 154].

Синтаксическая составляющая включает в себя все компоненты, которые участвуют в построении спецификации (атомарные утверждения спецификации). Семантическая составляющая определяет смысл выражения и указывает на причинно-следственные связи между элементами спецификации. Эти составляющие в последующем образуют формулы истинности [136].

Формальное описание спецификации протоколов управления, в частности протокола OpenFlow, имеет конструкцию следующего вида [126]:

$$\begin{aligned} &True: A(x_i : y_i) \models B(x_j : y_j), i \to j \\ &False: A(x_i : y_i) \models B(x_j : y_j), i \to j \end{aligned}, \qquad (1.1)$$

где A и B - логические утверждения; x и y являются наборами переменных (элементов спецификации): $x = x_1, x_2, \ldots x_m$, $y = y_1, y_2, \ldots y_m$, причем $n \geq 0, m > 0$. Переменные попарно различны.

Набор $A(x : y)$ определяет аргументы первичного утверждения A, а набор $B(x : y)$ - результаты выполнения определенного действия или условия, *True* - формула, а, следовательно, и утверждение спецификации, истинно. *False* - утверждение ложно.

Следует учитывать, что формальные выражения, построенные на одном из типов логики, корректны (выполнимы) лишь в случае совместимости со всеми предыдущими интерпретациями первоначального набора выражений.

Доказательство истинности состоит из манипуляции над формулами по заданным правилам [92, 93]. Основными математическими инструментами методов формализации требований являются: логика Хоара [36], темпоральные логики [21, 95, 96], алгебра распределенных коммуникационных ресурсов [8, 12, 58].

В качестве математического аппарата формализации, а также и последующей верификации протоколов уровня управления SDN, в частности протокола OpenFlow предлагается использовать алгебру распределенных коммуникационных ресурсов [8, 12, 13, 58].

Впервые аппарат алгебры распределенных коммуникационных ресурсов (algebra of communicating shared resources, ACSR) был предложен в 1994 г. Патриком Бремонд-Грегри и Инсуп Ли, Университет Пенсильвании. Аппарат ACSR был разработан для формальзиции и верификации сложных интегрированных систем, которые функционируют в режиме реального времени, чувствительны к временным задержкам ивремени выполнения процессов (доступность сетевых ресурсов, время обработки пакета). Математический аппарат алгебры распределенных коммуникационных ресурсов позволяет формализовать приоритет выполнения процессов, параллельное выполнение нескольких процессов и хронологическую последовательность их смены с учетом временных характеристик.

Каждый процесс в соответствии с ACSR может быть определен следующим множеством операторов.

Аппарат ACSR также позволяет формализовать [12, 13]:

- конфликты, возникающие между сетевыми ресурсами;
- первоочередность выполнения одного из нескольких процессов;
- приоритетность выполнения процессов.

Отличительной особенностью аппарата ACSR является то, что он позволяет формализовать ряд требований, анализ которых крайне важен для оценки корректного функционирования реализации протоколов управления SDN. Такими требованиями являются [8, 58]:

– отсутствие петель и зацикливаний в процессе функционирования протокола;

– живость сети;

– достижение заключительного состояния;

– ограниченный доступ к сетевым ресурсам;

– доступность сетевых ресурсов;

– непротиворечивость существующих версий протоколов в процессе функционирования.

Таким образом, математический аппарат алгебры распределенных коммуникационных ресурсов обеспечивает детальное представление процессов, имеющих место на уровне управления SDN, в частности позволяет формализовать процессы передачи и обработки управляющих пакетов OpenFlow, процесс формирования и поддержки виртуальных каналов связи с разным уровнем приоритета (логических срезов), обработку полей таблиц переадресации.

1.3 Обзор методов верификации протоколов Software-Defined Networking

Основываясь на статистических данных [66] можно отметить, что стоимость исправления ошибок, обнаруженных непосредственно на стадии внедрения протоколов или в процессе эксплуатации, является наивысшей проверка соответствия готового продукта (реализации протокола) всем требованиям, описанным в его спецификации. Одним из широко распространённых методов подобной проверки является верификация [6, 7, 46].

Основной целью верификации является доказательство того, что результат работы конечного решения соответствует перечню поставленных начальных целей, задач и требований, определенных спецификацией.

Задача верификации может быть выражена следующим формализмом:

$$V : R_{S_i} \equiv S_{S_i}, (\{r_i\} \subseteq \{s_i\}, r_i \in R_s, s_i \in S_s), \tag{1.2}$$

где R_S - реализация протокола в соответствии с требованиями спецификации (конечное множество функциональных элементов и отношений между ними) i - атомарное требование спецификации $i \in \{N\}$, N – множество требований спецификации, r_i - i-е состояния реализации протокола, S_S - эталонная система или спецификация (набор требований к системе и правил их взаимодействия), s_i - i-е требование спецификации.

Задачи верификации изначально направлены на проверку программных составляющих реализаций. В настоящее время программная составляющая основной для протоколов уровня управления SDN архитектуры.

На сегодняшний день можно выделить несколько методов применимых при верификации управляющих протоколов SDN [13], условно они могут быть разделены на методы, применяющие динамический подход, и формальные методы верификации.

Методы, применяющие динамический подход, базируются на принципах имитационного моделирования и тестовой проверки моделей протоколов и элементов SDN. Например, аппарат NICE разработан для проверки корректности функционирования SDN контроллера. NICE позволяет проверить такие свойства контроллера, как зацикливание и образование тупиковых ситуации при обработке запросов. Однако недостатком NICE является отсутствие возможности проверки других процессов (взаимодействие с коммутатором). Данная особенность вносит существенные ограничения и не позволяет выполнить верификацию всей сети.

NDB отладчик [14] позволяет выполнять как автоматическую верификацию, так и анализ фиксированных состояний протоколов управления. Недостаткам NDB является одновременная проверка лишь небольшого сегмента сети, как правило, одной зоны SDN контроллера.

Преимуществом программ-эмуляторов, как Mininet, GENY [15] являетсяанализ поведения протоколов управления, который позволяет учитывать реальные временные характеристики и параметры сети. Однако

применение Mininet, GENY, как и любого инструмента имитационного моделирования не позволяет определить все граничные состояния протоколов.

Второй подход базируется на формальных методах верификации, к нему относятся такие верификаторы, как SPIN[6], PRISM [7], SAL [8] и Alloy [9]. В основе данных верификаторов лежит систематизированный поиск и анализ множества пространства состояний, достижимых в процессе функционирования модели сетевых протоколов. Например, метод SPIN в настоящее время используется для различных приложений от аппаратной проверки распределенной программного обеспечения управления, используемого в атомной энергетике растения и космических аппаратов. Преимуществом подобных формальных методов является их выразительная мощность: с их помощью может быть сформирована любая система взаимодействующих процессов. В основе логики функционирования данных методов заложен метод верификации «проверка на моделях».

Метод проверки на моделях (Model Checking) [50, 61] позволяет осуществить проверку свойств на конечных моделях программ. При данном подходе свойства, которыми должен обладать протокол формируются в терминах значения предикатов для различных состояний протокола. При использовании этого подхода пользователь вводит описание модели системы (возможное поведение) и описание спецификации требований (желаемое поведение).

Метод проверки на моделях (Model Checking) разработан Кларком и Эмерсонов в 1980-х годах [50, 61]. На сегодняшний день Model Checking представляет собой комбинированный поход к верификации, который основан на проверке поведенческих свойств протоколов, заданных на моделях с конечным множеством состояний. Так, требования спецификации и проверяемые свойства, которыми должен обладать протокол, формируются в терминах определенной логики (предикатов, пред- и пост- условий), далее они накладываются на структурную модель на основании которой и осуществляется проверка.

Основной задачей проверки на моделях доказательство того, что $G \models \varphi$, где G - набор формул, которые определяют поведение протокола, выраженные одним из видов логик, φ - набор формул, определяющих требования спецификации.

Таким образом, при применении данного метода происходит попарное сравнение модели протокола (возможное или реальное поведение) и модели спецификации требований (желаемое поведение). Изначально, в качестве аппарата моделирования было предложено использовать модель Крипке, которая является разновидностью вероятностно-временных графов.

Процесс верификации, при использовании метода проверки на моделях, можно разделить на следующие части [11, 124, 126]:

1. Построение математической модели анализируемой системы.
2. Формализацию поведения системы на базе построенной модели.
3. Анализ поведения системы.
4. Представление формального доказательства наличия или отсутствия у системы заданного свойства.

Существует несколько подходов метода «проверки на моделях»:

- ограниченная проверка на моделях;
- символьная проверка на моделях;
- статистическая проверка на моделях;
- вероятностная проверка на моделях.

Каждый из методов имеет свою направленность, которая зависит от заложенной в него логики функционирования (LTL, CTL, μ-calculus, PCTL). Таким образом, применение одной из разновидностей данного метода позволяет провести точную проверку необходимых свойств протоколов.

Идея Model Checking базируются на исчерпывающем переборе множества всех состояний модели протокола: для каждого состояния протокола проверяется, удовлетворяет ли оно требованиям и целям, заложенным при разработке. На основе анализа последовательного изменения состояний делается вывод о корректности поведения протокола в целом.

Первоначальный акцент метода верификации «проверка на моделях» был сделан на проверку телекоммуникационные систем и протоколов. В связи с активным развитием усовершенствованием систем передачи информации метод «проверки на моделях» также претерпевает ряд существенных изменений. Например, изначальные виды логики, заложенные в процесс верификации, уже не позволяют полноценно формализовать, а, следовательно, и верифицировать современные протоколы.

1.4 Постановка научной задачи и формулировка частных задач исследования

Стремительное внедрение новых сетевых сервисов, увеличение их функциональности, а, следовательно, и сложности, влечет за собой возрастание требований к процессу их предоставления. Нагрузка на сетевое оборудование (маршрутизаторы, коммутаторы, серверы) существенно возрастает, так как наряду с традиционными протоколами передачи и управления образуется множество надстроек. Это приводит к снижению эффективности процессов обработки и влечет за собой ухудшение качества предоставляемых сервисов.

Применение концепции SDN при построении сервис-ориентированных сетей позволяет отделить функции управления сетью от функций передачи данных. В основе подхода SDN лежит возможность динамического управления предоставляемыми сервисами в сети с помощью открытого протокола OpenFlow. Все активные сетевые устройства объединяются под управлением единой сетевой операционной системы (контроллера), которая обеспечивает приложениям доступ к управлению сетью. Ввод центрального управляющего элемента позволяет существенно сократить нагрузку на сетевое оборудование, уменьшить объемы служебных данных, гибко распределять поступающую нагрузку, что приводит к повышению эффективности предоставления сервисов.

Наряду с перечисленными преимуществами SDN сформирован ряд недостатков, которые затрудняют ее полноценное внедрение.Отсутствие

единых стандартов, неполнота и наличие неопределенностей при формировании требований, предъявляемых к функционированию протоколов SDN, в частности протоколу OpenFlow, существенно влияют на эффективность взаимодействия оборудования. Проверка непротиворечивости требований, предъявляемых к протоколу OpenFlow, анализ корректности поведения и верификация являются важными задачами, решение которых позволит ускорить процесс внедрения концепции SDN.

Решение следующих частных задач позволит сформировать общую методику анализа и верификации протоколов SDN

Задача 1. Разработка метода формализации требований, предъявляемых к архитектуре Software-Defined Networking и протоколу OpenFlow, выразительная мощность которого позволит выявлять и устранять ошибки на начальных этапах жизненного цикла.

Задача 2. Разработка метода поиска противоречий в формализмах требований, изложенных в спецификации.

Задача 3. Анализ выполнения функциональных и нефункциональных требований в реализации протокола OpenFlow. Выполнение анализа таких свойств модели протокола, как достижимость, активность, ограниченность, безопасность.

Задача 4. Разработка формальных методов проверки соответствия готового SDN решения его спецификации.

Задача 5. Разработка прототипов и шаблонов фрагментов спецификации. Их формализация посредством алгебры распределенных коммуникационных ресурсов.

Задача 6. Разработка методов трансляции формализмов спецификации, заданных с помощью алгебры распределенных коммуникационных ресурсов, в модель E-сети.

Задача 7. Разработка метод формирования контрпримера, позволяющего выявить последовательность действий в процессе функционирования протокола, которая приводит к возникновению ошибок.

1.5 Выводы по первому разделу

1. Проведенный анализ архитектуры SDN и протоколов, обеспечивающих взаимодействие ее элементов, показал, что существующие проблемы функционирования, в первую очередь, связанны с наличием нескольких версий спецификации, которые содержат внутренние отличия; отсутствием строгой формализации требований, а также разрозненностью используемых сетевых операционных систем контроллеров.

2. Раннее обнаружение и устранение противоречий требований, функциональных расхождений и ошибок, связанных с программной реализацией протокола OpenFlow позволит гармонизировать процесс функционирования сети и повысить показатели QoS.

3. В качестве модели жизненного цикла предложено использовать спиральную модель. Выбор обусловлен тем, что спиральная модель позволяет учитывать возможную динамическую модернизацию и обновление требований: на каждом этапе жизненного цикла существует подэтап проверки и возможность возврата на предыдущий этап, при обнаружении ошибок. Таким образом, применение спиральной модели жизненного цикла позволяет реактивно реагировать на изменения требований к протоколу OpenFlow и своевременно обнаруживать возникающие ошибки.

4. В рамках совершенствования этапа формирования требований предложено применять алгебру распределенных коммуникационных ресурсов (ACSR). Анализ показал, что аппарат ACSR позволяет наиболее полно описать хронологию возникновения событий с учетом временных меток. Анализ реализации протокола предложено выполнять на моделях Е-сетей. Е-сети обладают высокой степенью универсальности и позволяют моделировать протокол с различной степенью детализации. Наличие управляющих переходов позволяет также учитывать множество особенностей OpenFlow протокола.

5. На этапе тестирования предложено проводить формальную проверку соответствия реализации всем требованиям спецификации.

Применение такого метода верификации, как Model Checking позволяет охватить все возможное множество инвариантов поведения протокола и выполнить проверку их соответствия требованиям спецификации. Однако, применение классического подхода «проверки на моделях» затруднительно при верификации протокола OpenFlow из-за возникновения эффекта «комбинаторного взрыва» пространства исследуемых состояний.

РАЗДЕЛ 2
РАЗРАБОТКА МЕТОДОВ ФОРМАЛИЗАЦИИ СПЕЦИФИКАЦИИ И АНАЛИЗА ПРОТОКОЛОВ SOFTWARE-DEFINED NETWORKING

2.1 Формализация требований спецификации протокола OpenFlow посредством алгебры коммуникационных распределенных ресурсов

Увеличение спектра предоставляемых сервисов, а также расширение их функциональных возможностей и повышение уровня требований к характеристикам QoS накладывает дополнительные требования на функционирование компонент архитектуры SDN [5, 16, 25]. Сложившаяся ситуация приводит к необходимости постоянной модификации протокола OpenFlow, а, следовательно, корректировке, расширению и изменению его спецификации. Добавление или обновление требований спецификации в ряде случаев вызывает возникновение противоречий между ними, например, применение различных команд для выполнения одного и того же действия, различная последовательность обработки сообщений [8, 12, 58].

Строгая формализации требований при формировании спецификации протокола, напротив, позволяет однозначно отображать множество возможных событий и процессов, формирующих утверждения спецификации с учетом причинно-следственные связи между ними.

Анализ, проведенный в п.1.3 показал, что аппарат алгебры коммуникационных распределенных ресурсов позволяет полноценно формализовать причинно-следственные связи между событиями и процессами, описывающими поведение протокола OpenFlow [117, 120]. ACSR имеет как формульную, так и графическую нотацию, что позволяет расширить описательные и аналитические возможности в процессе формализации спецификации.

При формировании требований спецификации протокола OpenFlow с помощью алгебры распределенных коммуникационных ресурсов множество возможных событий разделяется на два типа [12, 13]:

1.Длительныесобытия, которые позволяют учитывать длительность выполнения и изменение состояний протокола на протяжении определенного времени(например, обработка запроса контроллером, формирование или модификация таблиц переадресации, «прослушивание» канала передачи);

2.Мгновенныесобытия задают детерминированную смену состояний протокола (например,синхронизацию между процессами, установление виртуального соединения, получение управляющего сообщения).

Процессы, имеющие место при функционировании протокола, также могут быть разделены на несколько типов:

- мгновенные процессы;
- длительные процессы;
- синхронные процессы;
- асинхронные процессы;
- процессы типа «точка-точка», такой тип процессов позволяет описать однонаправленные действия, например, запрос на обработку данных;
- приоритетный доступк каналам, данный тип процесса характеризуется наличием атрибутов, которые указывают приоритет каждого процесса.

Синтаксически верные высказывания спецификаций формируются с помощью следующего набора операторов [12, 13]:

$$P \mid= \Theta \mid Q^u : P \mid e.P \mid P + Q \mid P \mid\mid Q \mid P \Delta_u Q \mid P \perp Q \mid [P\_or\_Q]_U \mid P \setminus F \mid recX.P \mid X . \quad (2.1)$$

Оператор Θ указывает на отсутствие активных событий или процессов в текущий момент времени (например, определяет начальную конфигурацию сети или возникновение тупика в процессе функционирования);

оператор $Q^u : P$ соответствует выполнению процесса Q на протяжении времени u (учитывается временной интервал), по истечению времени u наступает процесс P (например, освобождение сетевого ресурса: «порт переходит в спящий режим по истечению времени ожидания» - $Act^{250} : Sleep$);

оператор $e.P||e.s$ соответствует выполнению события e (без учета времени его выполнения – событие считается мгновенным), после которого обязательно выполняется процесс P или состояние s (например, процесс синхронизации: после нахождения совпадение поля *Match* счетчик таблицы переадресации обнуляется – $(Match_i(N) \equiv Match_j(N)) : \Theta^C$);

оператор выбора или приоритета $(P+Q) \parallel (s+\beta)$ описывает недетерминированный выбор одного из возможных последующих событии или процессов, в этом случае процесс P или Q может произойти с равной долей вероятности («при проверке нового пакета совпадение может быть обнаружено или не обнаружено»: $F + \neg F$);

оператор $P \parallel Q$ соответствует параллельному выполнению двух процессов P и Q («при получении нового пакета на определенный порт, коммутатор направляет его в таблицу переадресации и продолжает прослушивание порта»: $FlowTable||Listen$);

оператор $P\Delta_u Q$ определяет условие, при котором процесс Pможет быть выполнен в течении времени u включительно, по истечению этого времени u процесс P останавливается и начинается выполнение процесса Q или осуществляется переход к событию s. («приход эхо-сообщения ожидается на протяжении 100мс, по истечению этого времени осуществляется процесс повторной пересылки пакета»: $Wait\Delta_{100}\,Resent$);

оператор $P \perp Q$ определяет прерывание процесса P может быть прекращено в любое время в пользу выполнения процесса Q (получение и обработка пакета с более высоким приоритетом: $P_0 \perp P_1$);

оператор $[P\_or\_Q]_U$ указывает на то, что в течении времени u будет выполнимо множество процессов, результатом которых является процесс P или событие Q (в результате проверки пакета коммутатором OpenFlow в течении времени u, пакет будет либо отброшен, либо передан на дальнейшую обработку: $[drop\_or\_in\_procces]_U$;

оператор ограничения $P \setminus A$ формирует условие прекращения выполнения либо завершения события или процесса. Любое событие A, возникшее в процессе функционирования *P*, приводит к его завершению или ограничению (закрытие существующего соединения при приходе сообщения «fail security mode»: $Channel\_active \perp Fail\_Sec\_Mode$);

оператор $recX.P$ описывает стандартную рекурсию.

Логические связки между элементарными формализмами ACSR определяются следующим набором правил [12, 13, 58]:

1. P, A являются формулами для всех $P \in Proc, A \in Act$;

2. Если A является формулой, то $\neg A$ тоже формула;

3. Если P и A формулы, то $P \vee A$ и $P \wedge A$ тоже формулы;

4. Если P и A являются формулами, то все формализмы, соответствующие выражению (2.3) являются формулами;

5. Если P и A являются формулами, то $P \prec A$ тоже формула. Символ $\prec$ определяет отношение преимущества;

6. Если P и A являются формулами, то $P \xrightarrow{y,x} A$ тоже формула, y процесс, x- приоритет выполнения процесса.

Выразительная мощность ACSR позволяет формализовать большое количество сложных процессов, как синхронной, так и асинхронной природы.

Так параллельное выполнение нескольких процессов может быть представлено следующим формализмом:

$$Paral: \frac{P \xrightarrow{e} P'}{Q \xrightarrow{c} Q'}. \qquad (2.2)$$

Например, проверка таблицы переадресации на наличие информации о входящем *i*-том пакете и модификация учетной записи *j* (смена порта назначения), информация которой относится к данному пакету, может быть задана с помощью (2.4) следующим образом:

$$Sw: \frac{P_{input(i)} \xrightarrow{Match} P_{add}}{E_j(port = 80) \xrightarrow{Modify} E_j(port = 81)}.$$

Синхронное выполнение двух процессов может быть описано следующим формализмом:

$$Sinch: \frac{(P \xrightarrow{(a)} P')^t, (Q \xrightarrow{(b)} Q')^t}{(P \| Q \xrightarrow{(a.b)} P' \| Q')^t}, \quad (2.3)$$

В формализме (2.3) приведено условие обязательного выполнения процесса Q при запуске P и их одновременное выполнение на протяжении времени t.

Примером может служить формальное представление процесса синхронизации таблиц переадресации коммутатора OpenFlow:

$$\sum_{n=1}^{i} T: \frac{((P_{-in})_1 \xrightarrow{(a)} (P\_add)_1)^t \cup ((P_{-in})_2 \xrightarrow{(b)} (P\_add)_2)^t \cup \ldots \cup ((P_{-in})_i \xrightarrow{(a)} (P\_add)_i)^t}{((P_{-in})_1 \| (P_{-in})_2 \| (P_{-in})_i \xrightarrow{(a.b,\ldots i)} (P\_add)_1 \| (P\_add)_2)^t \| (P\_add)_i)^t},$$

где n - количество таблиц переадресации на OpenFlow коммутаторе, P_{-in} - входящий пакет, P_add - модифицированный исходящий пакет.

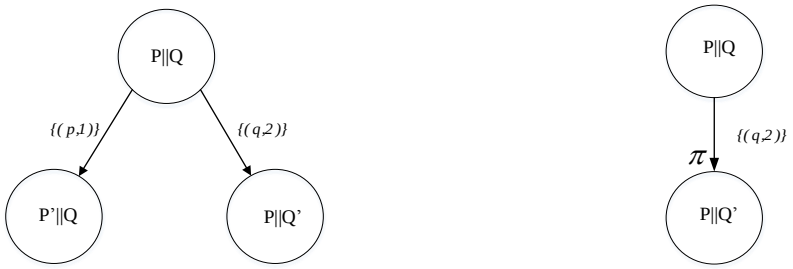
Каждый формализм ACSR может быть представлен графом состояний. Так, визуализация приоритета выполнения того или иного процесса может быть представлена следующим образом (рис. 2.1):

Наоснове правилACSR могут быть построены следующие базовые формализмы, которые соответствуют требованиям спецификации протокола [66-69]:

- конфликты, возникающие при распределении сетевых ресурсов: если $P = \{(p,1)\}: P'$ и $Q = \{(p,1)\}: Q'$, то $P \| Q \sim \mathrm{NIL}$;

- очередность выполнения одного из нескольких процессов: $P = \{(p,1)\}: P' + \Theta: P$ - первоочередным является выполнение процесса, соответствующего первому слагаемому;

- приоритетность выполнения процессов: может быть выражена как $P \| Q \rightarrow_{\pi} P \| Q' : \{(q,2)\}$.



а б

Рис. 2.1. Графическая интерпретация ACSR формализмов $P \| Q \Rightarrow P' \| Q : \{(p,1)\} \prec P \| Q' : \{(q,2)\}$ и $P \| Q \rightarrow_{\pi} P \| Q' : \{(q,2)\}$: отсутствие приоритета выполнения между процессами (а), приоритетное выполнение процесса (б)

2.2 Правила построения утверждений спецификации протокола OpenFlow посредством алгебры коммутационных распределенных ресурсов

Используя набор приведенных в предыдущем пункте правил можно сформировать следующую последовательность шагов построения ACSR утверждений, которые однозначно определяют требования, предъявляемые к протоколу:

1. В рамках спецификации формируется множество объектов (символов алфавита), принимающих участие в процессе функционирования протокола.

Такими объектами являются: канал связи - *Ch*, коммутатор - *Sw*, контроллер - *C*, сообщение - *M*. При этом, выразительная мощность ACSR позволяет задавать атрибуты каждого объекта, например, тип сообщения (*M(OF_MODIFY)*), характеристики коммутатора *Sw(virt_port*) или контроллера *C(Beacon).*

2. На основе полученных объектов формируется множество атомарных состояний протокола (например, сообщение отправлено -

M(OF_MODIFY)_send, канал связи сформирован - *Ch(i)_ready*, порт коммутатора активен - *Sw(pi)_active* и т.д.).

При этом в соответствии с (2.1, 2.2) состояние может быть начальным, промежуточным или заключительным.

3. Формируется множество процессов, задающих условие переходов из одного состояния другое (обработка сообщения, проверка значения полей, формирование записи в таблице переадресации и т.д.);

4. Формируются тройки типа «состояние протокола$\times$процесс$\times$ состояние протокола»:

4.1 Устанавливается хронологическая последовательность построения;

4.2 Устанавливается приоритет выполнения каждого процесса;

4.3 Определяются логические связи между каждой тройкой *«состояние протокола$\times$процесс$\times$состояние протокола»*;

4.4 Определяется наличие параллельного выполнения нескольких событий;

4.5 Определяется природа выполняемых событий и процессов, которые могут иметь как синхронный, так и асинхронный характер.

В рамках примера построения ACSR приведены требования спецификации OpenFlow, относящиеся к коммутатору. Коммутатор OpenFlow является связующим звеном между уровнем управления и уровнем передачи данных. Сообщения на стороне коммутатора обрабатываются и передаются с помощью таблиц переадресации или FlowTable [68-70]. При этом коммутатор может иметь несколько таблиц переадресации, которые обязательно должны быть синхронизированы друг с другом. Каждая таблица переадресации содержит определенное количество полей, основные назначения которых приведены на рис. 2.2.

Рис. 2.2. Поля таблицы переадресации OpenFlow коммутатора

Поле Match fields (Поле соответствия) определяет соответствие входящего пакета имеющимся данным таблицы. В первую очередь выполняется проверка заголовка пакета и номер порта, с которого поступил пакет.

Поле priority определяет приоритет входящего пакета.

Поле counters количество полученных байт либо пакетов, пришедших на определенный порт коммутатора. Также счетчики учитывают значения потерянных пакетов или длительность получение информации. Спецификация OpenFlow определяет различные значения поля counters.

Поле instructions определяет действие, которое необходимо произвести с полученным пакетом.

Поле timeouts определяет максимальное время ожидания либо максимальное время обработки нового пакета коммутатором.

На рис. 2.3 приведена блок-схема процесса обработки входящих пакетов коммутатором OpenFlow [68, 69].

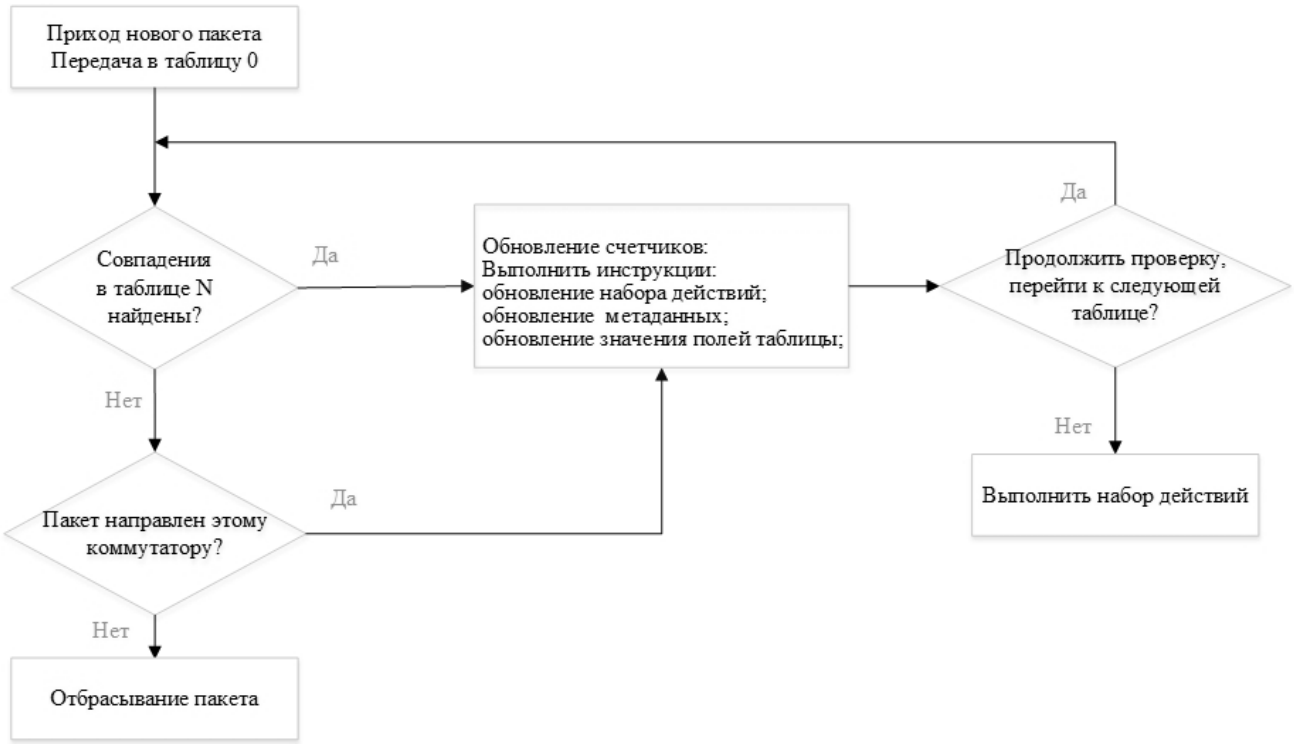


Рис.2.3. Блок-схема процесса обработки входящего пакета коммутатором OpenFlow

В соответствии с приведенными выше этапами процесс обработки входящего пакета OpenFlow коммутатором может быть формализован следующим образом:

1. Формируется множество объектов, принимающих участие в построении утверждений данного фрагмента спецификации: пакет – *P*; таблица переадресации – *TN*; счетчик таблицы – *Co*; набор инструкций или действий, содержащий набор атрибутов - $S(Set, field, metadata)$.

2. Формируется множество состояний, содержащих данные атрибуты: приход нового пакета $P(in)$; обновления счетчика таблицы переадресации Co_update ; отбрасывание пакета - P_drop; набор действий $S\_proceed$, $(Set, field, metadata)$;

3. Формируется множество процессов: обработка полей пакета в таблице переадресации - $T_N\_proceed$, пересылка пакета - P_forw, отбрасывание пакета - P_drop, обработка пакета следующей таблицей переадресации - $T_{N+1}\_proceed$.

4. Формируются возможные тройки:

4.1 Определяется хронологическая последовательность действий. В данном случае длительность процессов и вероятность возникновения событий зависит от значений поля timeouts и поля counters. Для алгоритма обработки входящего пакета может быть определена следующая зависимость:

$$(T_N\_procced \Delta_u P\_drop) || (T_N\_procced \Delta_u P\_drop) || (T_{N+1}\_procced \Delta_u P\_drop).$$

4.2 Формирование приоритетов выполнения процессов. В приведенном случае выполнимые процессы приоритетов не имеют.

4.3 Формирование логических связок. В соответствии с блок-схемой, приведенной на рисунке 2.2 могут быть сформированы следующие логические связки:

$$(P\_in)\Delta_{t<u}(T_N\_procced) \xrightarrow{Yes} (C\_update(Set, field, metadata)).(T_{N+1}\_pocced)$$
$$\xrightarrow{Yes} rec((P\_in).(T_N\_pocced)) \vee$$
$$(P\_in)\Delta_{t<u}(T_N\_procced) \xrightarrow{Yes} (C\_update(Set, field, metadata)).(T_{N+1}\_pocced)$$
$$(P\_in)\Delta_{t<u}(T_N\_procced) \xrightarrow{No} P\_in \in Sw \xrightarrow{No} P\_drop$$
$$(P\_in)\Delta_{t<u}(T_N\_procced) \xrightarrow{No} P\_in \in Sw \xrightarrow{Yes}$$
$$(C\_update(Set, field, metadata)).(T_{N+1}\_pocced) \xrightarrow{No} P\_forw;$$

Событие P_forw может иметь несколько атрибутов, так как в процессе функционирования протокола OpenFlow возможно появление трех типов сообщений [66, 69, 70]: сообщения типа «контролер-коммутатор»; синхронные сообщения; асинхронные сообщения.

Выполнения процесса P_forw возможно лишь в случае установления соединения между контроллером и коммутатором. На рис. 2.4 приведена схема инициализации соединения или управляющего потока OpenFlow.

Рис. 2.4. Формирование управляющего потока в SDN

В рамках данной схемы предполагается, что оконечное устройство пользователя A и оконечное устройство B находится в одной и той же зоне обслуживания: OpenFlow коммутаторы, к которым подключены оконечные пользователи, управляются с помощью одного общего SDN контроллера [14, 104, 107].

Начальным условием формирования нового потока или установлением соединения между контроллером и коммутатором является поступление нового пакета на один из портов коммутатора. Рассмотрим каждый этап формирования управляющего потока:

1. Регистрация нового пакета в таблице переадресации. При получении информации о новом пользователе коммутатор OpenFlow сверяет его данные с данными своей таблицы переадресации (FlowTable). Детальная схема обработки входящего пакета коммутатором OpenFlow приведена на рис. 2.3. В соответствии со спецификацией коммутатор может иметь несколько таблиц переадресации, которые содержат информацию о пересылаемых пакетах и набор инструкций. Записи таблицы переадресации состоят из полей сверки, счетчика, набора правил, которые необходимо выполнить для данного пакета. Запись таблицы OpenFlow коммутатора посредством ACSR может быть задана следующим образом:

$$R_{sw}(J) = Tf(Match_j, Count_j, Instruction_j, Timer), \quad (2.4)$$

где $Match_j$ - поле проверки служебных заголовков для j-того пакета. Поле $Match_j$ позволяет анализировать заголовки пакетов на уровнях L2–L4, что позволяет реализовать коммутацию и маршрутизацию на основе протокола OpenFlow., $Count_j$ - номер счетчика (может быть временная метка) j-того пакета;

$Action_j$ - набор правил, которые можно применить к j-му пакету. Поле $Action_j$ в соответствии с рассмотренными спецификациями [66-69] содержит следующие действия:

Отправить пакет на определенный порт коммутатора - Forward;

Отправить пакет на все порты – «Forward all»;

Отправить пакет контроллеру – «Forward controller»;

Отправить пакет назад во входящий порт - «Forward in port» и т.д.

2. Перенаправление пакета контроллеру осуществляется в том случае, если в таблице переадресации коммутатора не найдено ни одного совпадения для поля $Match_j$:

$$Send(t): Out_{pac}(j) \to Tf(Match_j \notin Match_{Tf}) \Rightarrow Out_{sw}(J) \xrightarrow{FV} Inp_c(J). \qquad (2.5)$$

В поле $Match_j$ пересылаемого пакета $Out_{sw}(J)$ содержится следующая информация: номер входящего порта, Ethernet адреса источника и получателя, тип Ethernet пакета, идентификатор и приоритет VLAN, IP адреса источника и получателя, тип протокола IP, Type of Service (ToS) биты протокола IP и TCP/UDP порты источника и получателя и т.д.

Обязательным требованием при этом является поддержка одной и той же версии протокола OpenFlow. Только в этом случае пакет будет корректно обработан контроллером. Возникновение ошибки возможно, если коммутатор поддерживает только протокол версии 1.0 - недостатком OpenFlow 1.0 является отсутствие поддержки протокола IPv6, способов обнаружения топологии зоны сети, отсутствие возможности многоадресной рассылки [33, 34].

3. Обработка пакета контроллером может происходить различными способами. Это зависит от типа операционной системы контроллера, заложенным алгоритмам обработки, объема буферной памяти. Контроллер SDN является критической точкой отказа в работе всей сети, в процессе

функционирования необходимо учитывать его надежность и производительность.

Пакет P_in, который передается от коммутатора к контроллеру по каналу (*ch1*) имеет определенный набор атрибутов, которые задают правила последующей его обработки ($T\_pr \equiv MatchP(T)$). Контроллер проводит обработку пришедшего пакета P_forw и формирует ответное сообщение-команду $C\_message$. При этом может быть сформирован логический срез или виртуальный канал $Slice$.

Процесс формирования управляющего потока посредством ACSR может быть представлен графически. Формализмы ACSR могут быть представлены в виде двудольно-ориентированного графа [12]:

$$G = < P, T(\alpha) >, \tag{2.6}$$

где P - множества вершин, задающее состояния протокола, $T(\delta,\alpha)$ - множество переходов, задающее процессы, имеющих место при функционировании протокола, δ - атрибуты процесса, α - приоритет выполнения процесса

Граф сообщения, отображающий обмен сообщениями между OpenFlow коммутатором и контроллером приведен на рис. 2.5.

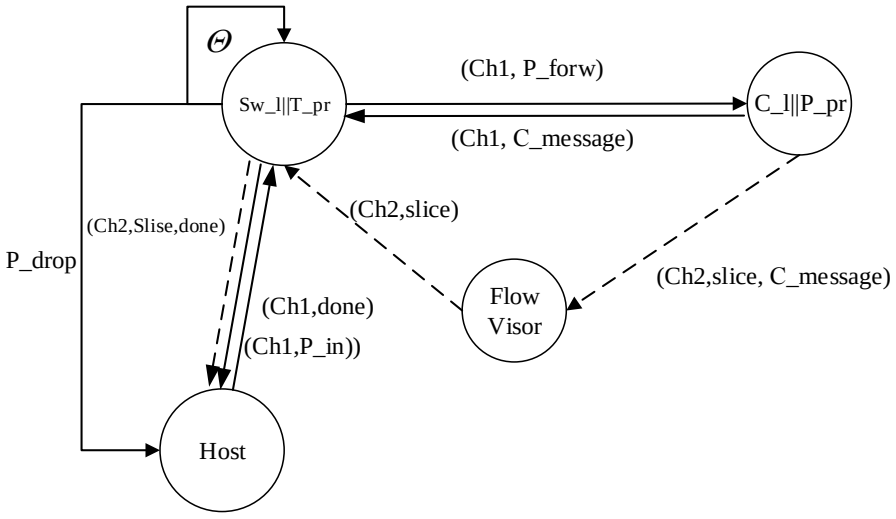


Рис. 2.5. Обмен сообщениями между OpenFlow коммутатором и контроллером

В соответствии с (2.4) процесс передачи сообщения, приведенный на рис. 2.6, может быть описан следующим образом:

$$Sw \xrightarrow{\Theta} Sw \xrightarrow{ch2,P(T)} C \xrightarrow{ch1,P(T')} Sw \xrightarrow{ch2,done} H,$$

$$Sw \xrightarrow{\Theta} Sw \xrightarrow{ch2,P(T)} C \xrightarrow{ch1,slice} FL \xrightarrow{ch1,slice} Sw \xrightarrow{ch2,done} H. \quad (2.7)$$

Входные события определяются следующим формализмом: $(e?,P_i)$, выходные $(e!,P_i)$.

Наоснове правилACSR могут быть построены следующие базовые формализмы, которые соответствуют требованиям спецификации протокола OpenFlow [17]:

- коммутатор (*Sw*) находится в неактивном состоянии – новые пакеты не поступают:

$$Sw := \{\Theta\}:Sw; \quad (2.8)$$

- коммутатор (*Sw*) перенаправляет новый пакет $P(T)$, контроллеру по каналу *ch1*:

$$Sw := ch1!C; \quad (2.9)$$

- коммутатор (*Sw*) получает новый пакет $P(T)$ от любого физического устройства по каналу *ch2*:

$$Sw := ch2?Host; \quad (2.10)$$

- коммутатор (*Sw*) одновременно может получать пакеты от контроллера или других физических устройств сети по каналу *ch1* и *ch2* соответственно:

$$Sw := ch1?P(T).C(T) + ch2?P(T).Host(T); \quad (2.11)$$

- перенаправление пакета контроллеру $P(T)$ осуществляется в том случае, если в таблице переадресации коммутатора не найдено ни одного совпадения для поля $Match_j$:

$$Sw(T) := match_j SrcIP(T,P_0) \to ch2!Host(T) + \sim match_j SrcIP(T,P_0) \to ch1!C(T) + e.R(T); \quad (2.12)$$

- коммутатор (*Sw*) выполняет проверку IP-адреса пересылаемого пакета $P(T)$, в случае, если IP-адрес не принадлежит диапазону сети, отбрасывает пакет:

$$Sw(x) := matchSrcIP(T,P_i) \to \{\}:Drop(T). \quad (2.13)$$

Кроме того, для корректной оценки функционирования реализации протоколов управления SDN крайне важен анализ таких свойств, как:

- отсутствие петель и зацикливаний в процессе функционирования протокола: любой пакет должен быть доставлен непосредственно получателю, который идентифицирован в поле destination source таблицы переадресации [86]. С помощью ACSR данное свойство может быть представлено следующим образом:

$$Swi := matchIP(T(dest), P_0) \to chi!(T) \setminus Swi := matchIP(T(dest), P_0) \to chi?(T); \quad (2.14)$$

- живость сети. Под живостью подразумевается состояние, когда все элементы сети, участвующие в передачи данных должны быть доступны и активны:

$$SDN := P_0 \xrightarrow{ch_i} P', [P']_U, P' \notin \Theta; \quad (2.15)$$

- достижение заключительного состояния: все состояния, гарантирующие корректную работу протокола, должны быть достигнуты:

$$SDN := (Sw1 || Sw2 || Swi || Host)/\{ch1,ch2,ch3,chi\}; \quad (2.16)$$

- согласованное функционирование оборудования OpenFlow, произведенного различными разработчиками и их эффективное совместное функционирование [1, 2, 10]:

$$SDN := \{R,Conf\} : Sw1 \setminus \{R,Conf\} : Sw2 \setminus \{R,Conf\} : Swi . \quad (2.17)$$

Формализмы (2.10 – 2.19) описывают процесс установления соединения (управляющего потока) в рамках соединения типа «точка-точка», однако применение аппарата ACSR возможно для разных зон обслуживания. Таким образом, математический аппарат алгебры распределенных коммуникационных ресурсов обеспечивает детальное представление процессов, имеющих место на уровне управления SDN, в частности позволяет формализовать процессы передачи и обработки пакетов OpenFlow коммутаторами.

2.3 Проверка непротиворечивости требований спецификации протокола OpenFlow

Спецификация протокола OpenFlow формируется подмножеством естественного языка, зачастую носит описательный характер и не имеет четкой систематизации и формализации требований, предъявляемых к протоколу [68]. Выразительная мощность такого похода не позволяет доказать истинность утверждений спецификации и их непротиворечивость друг другу.

В предыдущем пункте приведено однозначное преобразование текстового содержания спецификаций протокола OpenFlow в формализмы алгебры распределенных коммуникационных ресурсов. Применение ACSR дает

возможность проверить истинность утверждений, а также установить наличие возможных противоречий в требованиях спецификации [24].

Проверка непротиворечивости требований в рамках спецификации протокола OpenFlow может базироваться как на анализе формализмов, полученных в соответствии с этапами 1-5 п.2.2, так и на основе анализа графа состояний протокола.

2.3.1 Алгоритм поэлементной проверки формализмов спецификации

Первый метод основан на нахождении всех формализмов спецификации, содержащих утверждение, непротиворечивость которого необходимо проверить.

Предлагаемый метод включает следующие этапы:

1. Определение множества элементов, которые формируют утверждение спецификации – символов конечного алфавита ACSR.

Пусть $M(S)$ – множество элементов формулы $F_r(t_0)$ ACSR, содержащей данное утверждение спецификации (t_0). При этом элементы $Proc$ и Act, $(Proc \in S, Act \in S)$, формирующие данное утверждение, могут иметь разный уровень вложенности, который характеризуется глубиной возникновения событий и процессов, входящих в данное утверждение, r. Последовательность $(\{Proc\}\&\{Act\})_r$ определяет глубину возникновения искомого утверждения.

2. Поиск требований спецификации, содержащих проверяемое утверждение. Пусть $M(S,n)$ – множество всех найденных вхождений искомого утверждения в формализмы требований спецификации $F(t)$, $M(S,n) \equiv \Sigma F_r(t)$. Множество $\Sigma F_r(t)$, содержит все возможные высказывания спецификации протокола, содержащие данное утверждение. Каждое найденное утверждение может иметь различную глубину вложенности r в пределах содержащего его формализма $F(t)$.

3. Построение связки предусловие-утверждение-постусловие для всех найденных утверждений. Для всех возможных последовательностей $\Sigma F_r(t)$ из множества $M(S,n)$ задается отношение A или отношение непосредственного

следования, указывающее, что за каждым состоянием $P: p_0, p_1, p_2, ... p_r$, $P \in Proc \in S$, непосредственно может следовать $2^{|S|}$ состояний вида $Q: q_1, q_2, q_3, ... p_r$, $A \in Act \in S$, где $\{q\} \in S, \{p\} \in S$. Логические связки определены в соответствии с [12]. Формируется отношение B или отношение непосредственного предшествования, которое является обратным отношению A. Очевидно, что состояния $Q(B): p_1, p_2, p_3, ... p_r$ непосредственно предшествуют состояниям вида $Q(A): q_1, q_2, q_3, ... q_r$.

4. Проверка на непротиворечивость найденных высказываний. Предполагается, что проверяемое утверждение спецификации считается истинным: все последовательности действий, являющиеся его составляющими, истинны.

Пусть найдено первое утверждение $F(t_1)$, такое что $S_1 \subseteq \sum(S, n)$. Определяются границы проверяемого утверждения: $B(F_r(t_1))$, множество всех тех состояний, которые непосредственно предшествуют состояниям из множества $M(S, n)$ и $A(F_r(t_1))$ множество всех последующих состояний. При этом $B(F_r(t)) \xrightarrow{log.link} A(F_r(t))$. Таким образом, формирование суммарной цепочки, $\Sigma F_r(t) | t_1 = t_0$, возможно тогда и только тогда, когда для всех пост- и пред- состояний входящих в $M(S, n)$ и образующих логическую последовательность, соответствующую определенному утверждению спецификации, каждая элементарная конъюнкция для одинаковой глубины r не дает результат ноль. Проверка последовательно осуществляется для всех формализмов, которые содержат искомое утверждение $\sum_{i=0}^{n} F_r(i) \neq 0$.

Если результат, имеющий значение ноль, найден на шаге r, то p_{r-1} - содер

Поиск и выявление противоречий может быть также выполнен с помощью метода, который заключается в проверке дерева достижимости графа состояний протокола. Его особенностью является нахождение полной последовательности действий (цепочка процессов и событий) приводящих к

возникновению противоречий. Преимуществом данного метода является одновременный поиск и обнаружение множества противоречий в рамках спецификации.

Однако для реализации данного метода необходимо задействовать большое количество вычислительных ресурсов, что приводит к увеличению сроков и стоимости этапа формирования спецификации. Таким образом, целесообразно применять данный метод при проверке непротиворечивости требований небольшого фрагмента протокола.

2.3.2 Алгоритм поиска противоречий в требованиях спецификации на основе оценки достижимости графа состояний спецификации протокола

Граф состояний протокола может быть задан следующим образом [44, 128]:

$$G(S_{r_{max}}) = \langle C, T \rangle, \tag{2.18}$$

где C — множество вершин графа, $C \in \{Act\}$, а T — множество дуг графа, $T \in \{Proc\}$.

Предлагаемый метод проверки состоит из следующих этапов:

1. Формирование множества вершин графа.

Пусть $M(C_0)$ - множество вершин графа, которое содержит все возможные состояния проверяемого утверждения. При этом находятся все возможные требования спецификации, содержащие данное утверждение и сформируется множество $M(C_n)$, где n количество найденных утверждений.

2. Формирование множества переходов.

3. Определение последовательности переходов, приводящих к заключительному состоянию.

В качестве заключительного состояния выбирается последнее состояние проверяемого утверждения спецификации $G_0(S_{r_{max}})$. При этом множество активных вершин графа, приводящих к заключительному состоянию $G'(S_{r_{max}})$, формирует ядро графа, которое включает все найденные формализмы

спецификации $\sum G(S_{r_{max}})$. При этом выполнимо следующее утверждение: если $G'(S_{r_{max}})$ составляет ядро множества состояний графа $G(S_{r\,max}) \equiv \sum F_r(t))$, то для каждой цепочки состояний $q, q_1, q_2, \ldots q'_r$ из $G'(S_{r_{max}})$ существует подграф $G(S_\mu)$, на котором выполнима $F_r(\mu)$, содержащая отрезок, совпадающий с цепочкой состояний $q, q_1, q_2, \ldots q'_r$

4. Поиск и проверка на непротиворечивость. Пусть существуют два подмножества $F'(t)$ и $F''(t)$ множества $M(F(t))$ такие, что $B(F(t))=F'(t)$, а $A(F(t))=F''(t)$, где $A(F(t))$ и $B(F(t))$ определены ранее.

Поиск и проверка на непротиворечивость выполняется посредством построения дерева достижимости графа состояний протокола [124] для множеств состояний графа, принадлежащих $F'(t)$ и $F''(t)$ соответственно.

Построение $F'(t)$: $Q_0 = B(F_0(t))$, $Q_1 = Q_0 \cap B(F_0(t))$, $Q_{i+1} = Q_i \cap B(F_i(t))$. Построение осуществляется до тех пор, пока для некоторого k не выполнится $Q_{k-1} = Q_k$.

Построение $F''(t)$: $Q_1 = Q_0 \cap A(F_0(t))$, …, $Q_{i+1} = Q_i \cap A(F_i(t))$. Построение осуществляется до тех пор, пока для некоторого r не выполнится соотношение $Q_{r-1} = Q_r$, где $r = k$.

Необходимым условием проверки является наличие хотя бы одного активного состояния графа. Необходимой является проверка выполнения следующих утверждений.

Утверждение 2.1. Каждое состояние множества $F_r'(t)$ достижимо из некоторого состояния этого множества, т.е. из $q \in F_r'(t)$ следует, что $F_r'(q) \neq 0$.

Утверждение 2.2. Из каждого состояния множества $F_r'(t)$ достижимо некоторое состояние $F_r'(t)$ этого множества, то есть, из каждого состояния множества $F_r'(t)$ достижимо его ядро.

Таким образом, формализмы спецификации протокола OpenFlow не содержат противоречий тогда и только тогда, когда имеется конечное

множество переходов $F(t) \in \Sigma F_r(t)$, приходящих к заключительному состоянию. Глубина вхождения каждой позиции соответствует r. Если граф является циклическим, то ни один из циклов не должен содержать пустого подцикла.

Анализ различных версий спецификаций протокола OpenFlow показал, что противоречия в рамках одной спецификации, чаще всего возникают в процессе модификации или удаления таблиц переадресации. Так противоречие требований к автоматическому удалению записей таблицы приводит к потере маршрута и дополнительным временным затратам на его восстановление.

В соответствии со спецификацией протокола к сообщениям, указывающим на модификацию или удаление записей из таблиц переадресации относятся следующие типы сообщений [69]:

```
enum ofp_flow_mod_command {
OFPFC_ADD, /* Новый поток. */
OFPFC_MODIFY, /* Модифицировать все подходящие потоки. */
OFPFC_MODIFY_STRICT,  /* Модифицировать записи, строго
соответствующие эталону и приоритету. */
OFPFC_DELETE, /* Удалить все подходящие потоки. */
OFPFC_DELETE_STRICT /* Удалить записи, строго соответствующие
эталону и приоритету. */};
```

Существенным недостатком при этом является отсутствие строгих определений функций сообщений-команд (OFPFC_MODIFY и OFPFC_DELETE) и (OFPFC_MODIFY_STRICT или OFPFC_DELETE_STRICT) в различных версиях протокола OpenFlow. При этом для поиска строки таблицы переадресации с определенными параметрами используются эталонные шаблоны (wildcards) и все записи переадресации, которые соответствуют этим эталонам модифицируются или удаляются.

Если команда OFPFC_DELETE содержит утверждение «удалить все записи переадресации с портом назначения 80», то должны быть удалены только те строки таблицы переадресации, которые содержат порт 80 с меткой «порт назначения» [70].

Правило удаления с помощью ACSR может быть представлено следующим формализмом:

$$OFPFC\_DELETE : Receive(OFPFC\_DELETE,1).Find(T_i,entry_i,eng\_port\_80) \xrightarrow{v.entry_i=v.eng\_port\_80} Delete(entry_i,eng\_port\_80) \xrightarrow{v.entry_j=v.eng\_port\_80} Delete(entry_j,eng\_port\_80)... \xrightarrow{v.entryn\neq v.eng\_port\_80} Wait(OFPFC\_DELETE,2) . \quad (218)$$

Данное утверждение может иметь вложенность r, которая соответствует $m*n$, где – m количество таблиц, n - количество записей в таблице.

Однако, команда OFPFC_DELETE, которая определяет необходимые поля таблицы переадресации посредством wildcard, удалит все записи, которые содержат метку «порт 80» [68]. Правило удаления в данном случае может быть представлено следующим образом:

$$OFPFC\_DELETE : Receive(OFPFC\_DELETE,1).Find(T_i,entry_i,eng\_port\_80) \xrightarrow{v.entry_i=v.eng\_port\_80} Delete(entry_i,eng\_port\_80) \xrightarrow{v.entry_j=v.eng\_port\_80} Delete(entry_j,eng\_port\_80)... \xrightarrow{v.entryn\neq v.eng\_port\_80} Wait(OFPFC\_DELETE,2) . \quad (2.19)$$

Для проверки на непротиворечивость за эталон истинности принимается одно из приведенных выше утверждений. Пусть утверждение (2.19) истинно, тогда все его составляющие принимают логическое значение «1».

Формируется множество $A(F(t))$ и множество $B(F(t))$ и устанавливаются процессы, приводящие к их смене:

$$M'(t): Receive(OFPFC\_DELETE,1) \xrightarrow{v.entry_i=v.eng\_port\_80} Delete(entry_j,eng\_port\_80),$$
$$Find(T_i,entry_i,eng\_port\_80) \xrightarrow{v.entry_i=v.eng\_port\_80} Delete(entry_j,eng\_port\_80).$$

$$M''(t)): Receive(OFPFC\_DELETE,1) \xrightarrow{v.entry_i=v.wildcard} Delete(entry_j,*80),$$
$$Find(T_i,entry_i,wildcard,*80) \xrightarrow{v.entry_i=v.wildcard} Delete(entry_j,*80).$$

При проверке на непротиворечивость мощность множества $A(F(t))$ равна нулю. Это указывает на факт возникновения противоречий.

Для построения графа состояний протокола, из которых состоит определенное утверждение, и последующей проверки на непротиворечивость методом построения графа состояний необходимо сформировать два множества $F'(t)$ и $F''(t)$, которые содержат все возможные события, первого и второго высказывания спецификации соответственно:

$$F'(t): Receive(OFPFC\_DELETE,1)\cup$$
$$Find(T_i,entry_i,eng\_port\_80)(Receive(OFPFC\_DELETE,1))\cup$$
$$\cup(delete(entry_i,eng\_port\_80))_r(Find(T_i,entry_i,eng\_port\_80)_r(Receive(OFPFC\_DELETE,1)))\cup,$$
$$\cup Wait(OFPFC\_DELETE,2)((delete(entry_i,eng\_port\_80))_r(Find(T_i,entry_i,eng\_port\_80)$$
$$(Receive(OFPFC\_DELETE,1))))$$

$$F''(t): Receive(OFPFC\_DELETE,1)\cup$$
$$Find(T_i,entry_i,wildcard,*80)(Receive(OFPFC\_DELETE,1))\cup$$
$$\cup(delete(entry_i,*80))_r(Find(T_i,entry_i,wildcard,*80)_r(Receive(OFPFC\_DELETE,1)))\cup$$
$$\cup Wait(OFPFC\_DELETE,2)((delete(entry_i,*80))_r$$
$$(Find(T_i,entry_i,wildcard,*80)(Receive(OFPFC\_DELETE,1)))).$$

В этом случае, граф $G'(S_{r_{max}})$ включает объединение множеств $F'(t)$ и $F''(t)$, которые и являются ядром множества рассматриваемых утверждений спецификации или задействованных состояний протокола (рис. 2.6).

Анализ дерева достижимости построенного графа показал, что для данного графа $G'(S_{r_{max}})$ конечное множество переходов $F(t)\in\Sigma F_r(t)$, приходящих к заключительному состоянию, не существует. Таким образом, утверждение спецификации содержит противоречие.

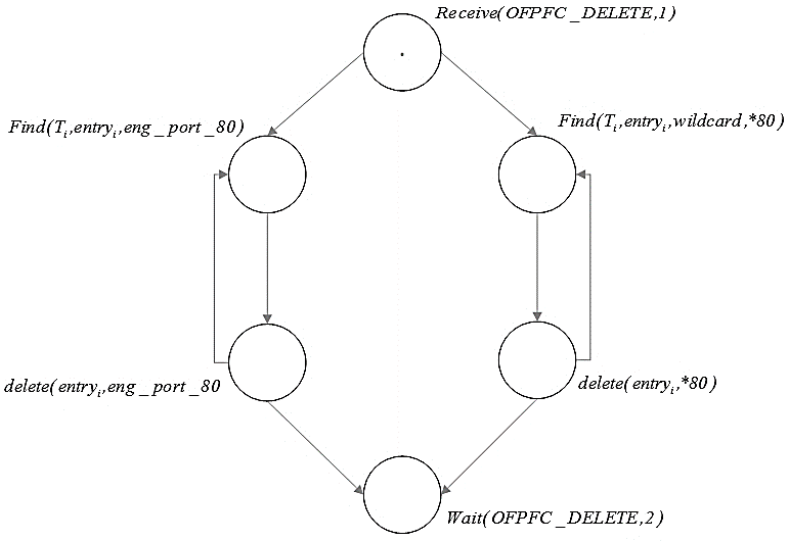


Рис. 2.6. Граф исследуемых состояний протокола

Одним из фундаментальных принципов функционирования протокола OpenFlow является возможность самостоятельной обработки и пересылки управляющих сообщений OpenFlow контроллером: «switch control traffic without involving the OpenFlow controller» [69]:

$$Switch: Receive(\ packet_i,1) \xrightarrow{find,i} Modify(\ packet_i,1) || Delete(\ packet_i,1) || Forward(\ packet_i,1) \xrightarrow{forward,i} Set(\ flow,1)$$

Однако, одним из первоначальных требований к коммутаторам является следующее «If no match is found, the switch is forwarded to the controller over the secure channel» [70]:

$$Switch: (Receive(\ packet_i,1) \xrightarrow{find,i} Forward(\ packet_i,1)) || (Receive(\ packet_i,1) \xrightarrow{don't\_find,i}$$
$$(Send(\ packet_i,1) + Receve(contol,i)) \xrightarrow{add,i} Modify(\ packet_i,1) || Delete(\ packet_i,1) || Forward(\ packet_i,1)$$

При проверке требований спецификации на непротиворечивость формируется два множества состояний $F'(t)$ и $F''(t)$ для первого и второго утверждения соответственно.

$$F'(t): Receive(\ packet_i,1) \cup Modify(\ packet_i,1) Receive(\ packet_i,1) \cup$$
$$Delete(\ packet_i,1) Receive(\ packet_i,1) \cup Forward(\ packet_i,1) Receive(\ packet_i,1),$$

$$F''(t): Receive(\ packet_i,1) \cup Forward(\ packet_i,1) Receive(\ packet_i,1) \cup$$
$$\cup (Send(\ packet_i,1) + Receve(contol,i)) Receive(\ packet_i,1) \cup$$
$$\cup (Modify(\ packet_i,1) || Delete(\ packet_i,1) || Forward(\ packet_i,1)) (Send(\ packet_i,1) + Receve(contol,i)) Receive(\ packet_i,1)$$

Ядром множества всех возможных состояний является $F^*(t)$. Граф состояний, содержащий множество $F^*(t)$ приведен на рис. 2.7

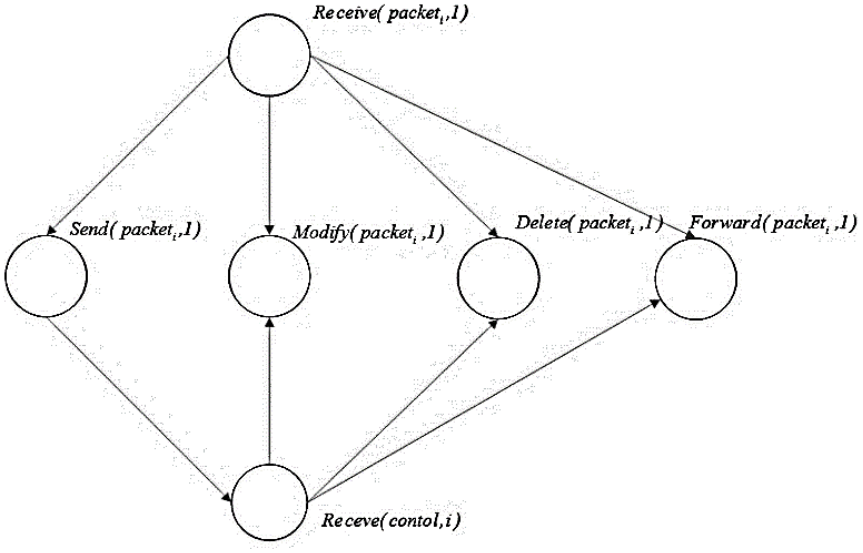


Рис. 2.7. Граф исследуемых состояний протокола

Для данного графа $G'(S_{r_{max}})$ и множества $F^*(t)$ выполнимы лишь следующие последовательности переходов, которые приводят к заключительному состоянию:

$Receive(\ packet_i,1),Forward(\ packet_i\ ,1)$

$Receive(\ packet_i,1)(\ Send(\ packet_i,1)(\ Modify(\ packet_i\ ,1)Forward(\ packet_i\ ,1)$

При этом достижения всех элементов из множества $F_r''(t)$ невозможно, что указывает на факт обнаружения противоречия. Последовательностями, которые содержат противоречия, являются $Receive(\ packet_i,1)\cup(\ Modify(\ packet_i\ ,1),\ Receive(\ packet_i,1)\cup Delete(\ packet_i\ ,1)$.

2.4 Анализ корректности функционирования протокола OpenFlow

Эффективность функционирования протокола OpenFlow в значительной мере зависит от программных реализаций контроллера и коммутаторов. В соответствии со спиральной моделью жизненного цикла протокола ключевое значение имеет этап реализации протокола [10, 136, 145]. Требования, предъявляемые к реализации протокола, могут быть разделены на функциональные и нефункциональные. Функциональные требования определяют поведение протокола и набор обязательно выполнимых им функций. Каждое функциональное требование, предъявляемое к работе протокола, подразумевает достижение желаемого конечного результата (например, сообщение обработано, соединение установлено, запрашиваемые данные доставлены, виртуальный канал сформирован). Например, исходящий пакет должен быть обязательно доставлен получателю. Нефункциональные требования определяют частные свойства протокола либо накладываемые ограничения: время выполнения процесса, доступность ресурсов [10].

Завершающей стадией этапа реализации является анализ соответствия реализации системы предъявляемым к ней требованиям. В п.1.3 отмечено, что построение графа модели протокола с помощью аппарата Е-сетей и его последующий анализ дает возможность проверить соответствие выполнимости требований, предъявляемых к протоколу.

Проверка соблюдения функциональных требований при реализации протокола может быть реализована путем анализа таких свойств Е-сети, как

активность, безопасность и достижимость. При проверке нефункциональных требований необходимо выполнить анализ таких свойств модели протокола, как ограниченность, сохраняемость, активность [136].

Для анализа свойства достижимости необходимо определить начальное состояние модели Е-сети (начальную маркировку) и множество заключительных состояний.

Пусть M_0 – начальная маркировка графа Е-сети $E=(P,H,L,D,A,M_0)$, а M_r - множество заключительных разметок. Тогда задача анализа достижимости заключительных разметок Е-сети может быть представлена следующим образом: для Е-сети модели протокола с начальной разметкой M_0 определить достижимость маркировки M' такой, что $M' \in f(E,M_r)$ [71].

Потенциальная активность сети предполагает достижимость всего множества возможных маркировок, полученных из начальной разметки сети. Все переходы Е-сеть модели протокола по умолчанию активны, так как каждый переход соответствует определенной функции, обязательно выполняемой на одном из этапов функционирования протокола. Переход называется активным, если существуют метки, которая соответствует входному состоянию или функции прямой инцидентности и отсутствует в исходящей позиции, иначе переход является заблокированным.

Задача проверки активности может быть представлена следующим образом: каждое следующее состояние $p_i, p_i \in P$ Е-сети должно быть достижимо для маркировки M_i, таким образом, что: $f_D(M_{i-1},h_j,p_{i-1}) \equiv f_L(M_i,h_{j+1},p_i)$, где f_D - функция обратной инцидентности, f_L - функция прямой инцидентности, h_j и h_{j+1} - последовательности переходов для которых справедливо $f(h_j) \to p_j$. Каждый переход может быть охарактеризован различным уровнем активности: от 0 до *n*:

- переход h_j обладает активностью 0, если его запуск невозможен: $f(h_j)=0$;

- переход h_j обладает активностью 1, если он потенциально запустим: $f(h_j)=p_i$;

- переход h_j обладает активностью n, если для него существует последовательность запусков, приводящих к определенной маркировке M', такая что $f_D(M,(h_j)^n,p_{i-1}) \equiv f_L((M,h_{j+1},p_i)\cup\ldots\cup(M,h_{j+n},p_n))$, т.е. для перехода существует количество потенциальных запусков равное n.

Нахождение переходов с активностью 0 позволяет выявить факт возникновения заблокированных разметок. Что говорит о существовании в Е-сети таких последовательностей событий, при которых некоторая функция никогда не выполнится, например, в случае некорректного распределения доступа к общим сетевым ресурсам.

Корректность поведения протокола и эффективность распределения ресурсов сети в значительной мере зависит от нефункциональных требований.

Проверка выполнимости нефункциональных требований в первую очередь связанна с задачей анализа сохраняемости Е-сети.

Е-сеть E с начальной маркировкой M называется сохраняющей, если для всех M_i, $M_i \in (E,M)$ выполняется следующее равенство [34, 56]:

$$\sum_{p_i \in P} M_i(p_i) = \sum_{p_i \in P} M(p_i) \qquad (2.21)$$

Такой вид сохраняемости является строгим ограничением, из которого следует, что количество входов обязательно должно соответствовать количеству выходов из позиции.

В процессе функционирования протокола OpenFlow при объединении или разветвлении потоков число меток может меняться (например, при формировании виртуального канала или формирование multicast запроса) [117]. Таким образом, при анализе основных алгоритмических свойств модели Е-сети следует определять взвешенную сумму меток, которая должна быть постоянной.

Е-сеть считается сохраняющей, если существует вектор $m=(m_1,m_2...m_i)$, m - количество меток, такой, что для всех маркировок M_i, достижимых из начальной маркировки M_0, справедливо равенство [34, 56]:

$$\sum m_i M_i(p_i)=\sum m_i M_0(p). \quad (2.22)$$

Анализ свойства ограниченности Е-сети позволяет оценить степень выполнимости таких нефункциональных свойств протокола, как надежность и производительность. Задача анализа ограниченности Е-сети может быть сформулирована следующим образом: при заданной начальной маркировке M_0 и допустимом числе меток n для любой достижимой в сети разметки M_i справедливо следующее неравенство $M_i(p)\le n$ [34, 56].

Частным случаем ограниченности является свойство безопасности. Свойство безопасности однозначно характеризует условие $\forall M(p)\in E\{\forall p_i \mid M(p_i)\le 1\}, i=1...n, n=|P|$. То есть в такой сети ни при каких условиях в любой позиции не может появиться более одной метки.

При некорректном выполнении определенных функций протокола возможно возникновения петель, которые приводят к неэффективному функционированию протоколов в целом.

Путь между двумя вершинами p_i, $p_i\in P$ и p_j, $p_j\in P$ является петлей $\pi(p_i,p_j)$, если из маркировки M_i достижима лишь маркировка M_j, при этом $\sum m_i M_i(p_i)=\sum m_j M_j(p_j)\ne\sum m\, M_0(p)$. Наличие петель указывает на цикличность процессов, протекающих в моделируемом протоколе, которые могут привести к блокировке переходов. Таким образом, обнаружение петель является важной задачей при построении полной модели взаимодействия элементов сети.

Таким образом, задачи проверки выполнения функциональных и нефункциональных требований в реализации протокола OpenFlow могут быть

решены путем анализа таких свойств модели Е-сети, как активность, достижимость, сохраняемость (в частности, безопасность), ограниченность и цикличность.

На сегодняшний день основными методами анализа алгоритмических свойств модели протокола являются метод построения матричных уравнений и дерево достижимости [34, 56, 72].

Матричный подход основывается на описании Е-сети модели протокола двумя матрицами D^- и D^+, представляющими функции прямой и обратной инцидентности: $D^-(j,i)=K(p_{i-1},D(h_j))$, $D^+(j,i)=K(p_i,L(h_j))$, где h_j - активный переход модели Е-сети, p_i - состояние, формируемое в процессе запуска перехода h_j, K – кратность дуги, ведущей из позиции p_i в переход h_j и дуги, ведущей из перехода h_j в позицию p_{i+1}.

Таким образом, каждая строка матрицы характеризует определенный переход h, каждый столбец – позицию p. Вводится вектор m, который содержит нули во всех строках кроме j-той компоненты: $e(j)$. Если переход h_j в процессе маркировки разрешен, результат запуска этого перехода при маркировке М определяется функцией $\varphi(M,h_j)$ задающей новую маркировку M':

$$\varphi(M,h_j)=M-e(j)D^-+e(j)D^+=M+e(j)(D^+-D^-)=M+e(j)D, \qquad (2.23)$$

где $D=D^+-D^-$- составная матрица изменений состояний модели протокола.

Последовательность запусков переходов $H:(h_{j1},h_{j2},...h_{jk})$ задается следующим образом:

$$\varphi(M,H)=\varphi(M,h_{j1},h_{j2},...h_{jk})=M+e(j_1)D+e(j_2)D+...+e(j_k)D=M+f(H)D. \quad (2.24)$$

Вектор $f(H)=e(j_1)+e(j_2)+...+e(j_k)$ называется вектором запуска последовательности $H:(h_{j1},h_{j2},...h_{jk})$, при этом $f(H)_i$ определяет число запусков перехода в последовательности $(h_{j1},h_{j2},...h_{jk})$.

Матричная теория является хорошим инструментом анализа таких свойств модели Е-сети как достижимость, активность, безопасность.

Однако применение матричного метода имеет ряд существенных ограничений:

- матрица D не позволяет выполнить полноценный анализ в случае возникновения петель;
- функция запуска дает представление об общем количестве переходов, но информация о последовательности их срабатывания отсуствует.

Дерево достижимости представляет собой метод поиска множества достижимых разметок модели Е-сети. Результат анализа модели протокола посредством дерева достижимости может быть представлен в виде следующего графа состояний:

$$G(D)=<P,H,\gamma>, \qquad (2.25)$$

где $P=\{p_1,p_2,...,p_n\}$ – множество вершин, которые соответствуют множеству достижимых разметок $D(M_f)$; $H=\{e_{ij}\}(\forall p_j,p_j\in V)$ множество переходов, активность которых приводит к достижению определенной разметки M', $M'\in M_f$; $\gamma: f(H\to P)$ – функция срабатывания переходов, которая обеспечивает выполнение соответствующего отношения достижимости для рассматриваемой пары маркировок.

Построение дерева достижимости позволяет найти все множество достижимых разметок модели протокола и проверить активность каждого перехода. Исследование смены достижимых разметок дает возможность проанализировать последовательности запусков переходов, что, в свою очередь, позволяет выявить условия функционирования протокола, приводящие к возникновению тупиков и циклов.

В работах [55, 70, 73, 126] рассмотрено применение метода дерева достижимости для анализа основных алгоритмических свойств сетей Петри. При этом отмечено, что существенным недостатком при анализе основных свойств модели сети является появление символа ω при построении дерева достижимости [63, 129]. Однако, введение конечного множества решающих позиций, которые характеризуются набором атрибутов, например, счетчиков срабатывания, и наличие одной входящей и исходящей дуги для каждой вершины-позиции позволяют устранить данный недостаток.

Таким образом, модификация метода построения дерева достижимости для решения задачи анализа основных алгоритмических свойств Е-сети, а, следовательно, и проверки соблюдения требований, предъявляемых к протоколу OpenFlow, является актуальной.

Модификация метода построения дерева достижимости

Построение модифицированного дерева достижимости модели Е-сети начинается с определения начальной маркировки.

Пусть M_0 - начальная маркировка или корень дерева достижимости. Далее устанавливается функция срабатывания каждого перехода. Если переход h_{i-1} сети активен и функция $\gamma(M_f(i),h_{i-1})$ определена, то формируется дерево достижимых маркировок $\gamma(M(j),h_j)=M'(x)$, $M'(x)$ – маркировка, полученная после срабатывания перехода h_j. При этом определяется тип вершины x, если она является внутренней, то вершина i достижимая при выполнении $\gamma(M(i),h_i)$ становится новой граничной вершиной с маркировкой M_i.

При этом для каждой позиции p_i или вершины i маркировка сети M_i определяется следующим образом:

- если на пути от корневой вершины к заключительному состоянию существует вершина i, такая что $M_i \to M_f$, $M_i < \gamma(M_f,h_{f-1})(M_i)_\omega < \gamma(M_f,h_{f-1})_\omega$, то $(M_i)_\omega = \omega$ или $(M_i)_\omega = (0,0,...,\underset{i}{\omega},...,0)$.

- если $(M_f)_\omega = \omega$, то и $(M_z)_\omega = \omega$, где ω - любое целое натуральное число отличное от нуля.

Наличие решающих мест $r(x)$ в модели Е-сети позволяет заменить символ ω на определенное численное значение. Замена возможна при введении счетчиков меток в решающие позиции. При этом такое решение позволяет проверять соблюдение нефункциональных требований, которые содержат численные характеристики. Наличие только одной входящей и исходящей дуги для каждой позиции Е-сети позволяет ограничить размножение меток в сети, а, следовательно, и избежать несанкционированного изменений значений счетчиков на управляемых переходах при увеличении количества меток.

Таким образом, алгоритм построения дерева достижимости включает несколько дополнительных этапов и может быть задан следующей последовательностью действий:

1. Определяется начальная маркировка M_0 Е-сети и формируется множество потенциально активных переходов.

2. Определяется множество потенциально достижимых вершин.

Пока существуют потенциально достижимые вершины выполняется набор следующих действий:

2.1 Последовательно определяется множество активных переходов и множество вершин, достижимых при их запуске. Каждая достижимая вершина может быть терминальной, дублирующей или граничной.

Пусть существуют достижимые вершины x и y:

- если маркировки x, M_x и вершины y, M_y совпадают, то вершина y является дублирующей вершиной.

- если маркировка $M_x \neq M_y$ и не существует активных переходов из вершины y, то вершина является терминальной, а ее маркировка M_y - заключительной.

- если маркировка $M_x \neq M_y$ и существуют активные переходы из вершины y, в любую другую вершину модели протокола, то вершина y считается граничной, а x - внутренней вершиной.

2.2. Определяется множество проходов, приводящих к смене маркирования сети.

Устанавливается соответствие вершины x определенной позиции Е-сети p_x и вершины y - позиции p_y, устанавливается значение перехода, переводящего маркировку M_x в маркировку M_y.

3. Шаги 1-2 осуществляются до тех пор, пока существуют достижимые маркировки. После того, как все терминальные, дублирующие или граничные вершины получены – дерево достижимости считается сформированным.

4. Формируются множества $\{M_f'(p_i)\}$, $\{M_f'(h_{i-1})\}$, которые определяют множество маркировок задействованных в процессе перемещении метки от корня дерева к определенной терминальной вершине из множества M_f.

4.1 Устанавливается количество проходов метки через каждую достижимую вершину (внутреннюю, дублирующую или граничную) при построении полного дерева достижимости.

а) если на пути от корневой вершины к заключительному состоянию существует вершина i, такая что $M_i \to M_f$, $M_i < \gamma(M_f, h_{f-1})\,(M_i)_\omega < \gamma(M_f, h_{f-1})_\omega$, то $(M_i)_\omega = \omega$ или $(M_i)_\omega = (0,0,...,\underset{i}{\omega},...,0)$

б) если $(M_f)_\omega = \omega$, то и $(M_z)_\omega = \omega$, где ω - значение счетчика управляющего перехода.

4.2. Определяется первый момент появления символа ω. При этом определяется тип перехода, если переход содержит счетчик, то каждый раз при запуске перехода значение счетчика увеличивается на единицу.

5. Определить «вектор разметок»

Определение 2.1 Вектор разметок является упорядоченным множеством всех достижимых позиций, которые задействованы в формировании достижимой разметки.

Пусть начальная маркировка $M_0 = (\underbrace{1,0,0,0,...,0}_{k})$, где k - количество потенциально достижимых вершин. Наличие «1» в позиции маркировки определяет номер позиции Е-сети. При этом справедливы следующие утверждения $M_0 = (\underbrace{p_1,0,0,0,...,0}_{k})$, $M_1 = (\underbrace{0,p_1,0,0,...,0}_{k}) \dots M_k = (\underbrace{0,0,0,0,...,p_k}_{k})$. В таком случае вектор разметок может быть задан как $\overrightarrow{M_{0k}} = (\underbrace{p_1, p_2, p_i, ..., p_k}_{k})$

6. Определяется результирующий вектор разметок для заданной конечной маркировки M_k. В этом случае должно быть сформировано однозначное отношение переходов и позиций, а также полученное численное значение счетчиков.

При этом результирующий вектор разметки может быть задан следующим образом:

$$\overrightarrow{\sum M_{0k}} = (\underbrace{\sum_1^n p_1, \sum_1^m p_2, \sum_1^l p_i, ,..., \sum_1^j p_k,)}_{k} = (\underbrace{p^n{}_1, p^m{}_2, p^l{}_i, ..., p^j{}_k}_{k}). \quad (2.26)$$

7. При наличии множества альтернативных последовательностей срабатывания последовательно выполняются шаги 1-5, формируется множество ветвей дерева:

$$T(M_f) = \sum \overrightarrow{M_{0k}} \cup \sum \overrightarrow{M_{0n}} \cup \sum \overrightarrow{M_{0p}} =$$
$$= (\underbrace{\sum_1^n p_1, \sum_1^m p_2, \sum_1^l p_i, ,..., \sum_1^j p_k}_{k}) \cup (\underbrace{\sum_1^n p_1, \sum_1^m p_2, \sum_1^l p_i, ,..., \sum_1^j p_n}_{n}) \cup (\underbrace{\sum_1^n p_1, \sum_1^m p_2, \sum_1^l p_i, ,..., \sum_1^j p_p}_{p}). \quad (2.27)$$

Процесс формирования дерева достижимости считается завершенным, если все терминальные вершины найдены.

Необходимым условием применения модифицированного метода построения дерева достижимости модели протокола является наличие конечного множества вершин.

Предложенный метод может быть использован как для проверки выполнения отдельных требований спецификации, так и при комплексной проверке реализации протокола.

Требования спецификации, включающие поведение коммутатора при формировании нового сообщения представлены следующим фрагментом Е-сети (рис. 2.8).

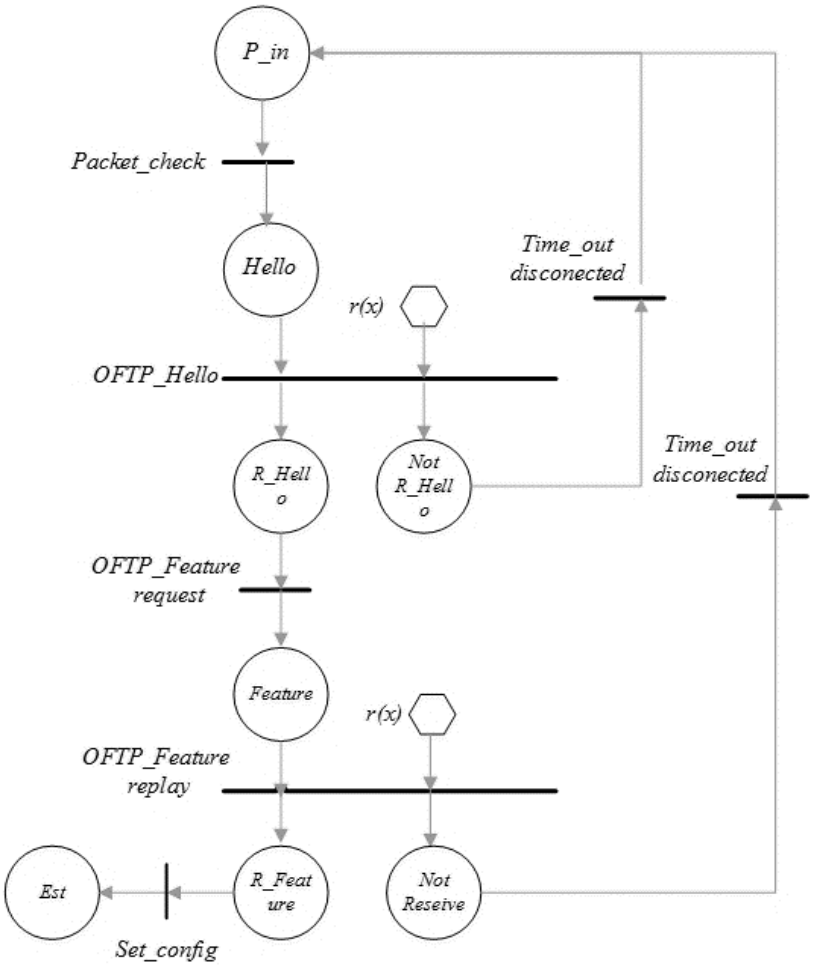


Рис.2.8. Последовательность действии на OpenFlow коммутаторе при формировании сообщений

Состояние P_in – соответствует получению нового пакета одним из портов коммутатора, *Hello* - формирования приветственного сообщения коммутатором, R_Hello - получение ответа от контроллера, *Feature*-

формирование информационного сообщения о свойствах коммутатора, $R\_Feature$ - получение сообщения о согласовании свойств, Est - установление соединения; Not_R_Hello - приветственное сообщение не подтверждено; $Not\_R\_Feature$ - набор предложенных свойств не подтвержден.

Дерево достижимости модели Е-сети, отображающей последовательность действий на OpenFlow коммутаторе при формировании сообщений, имеет следующий вид:

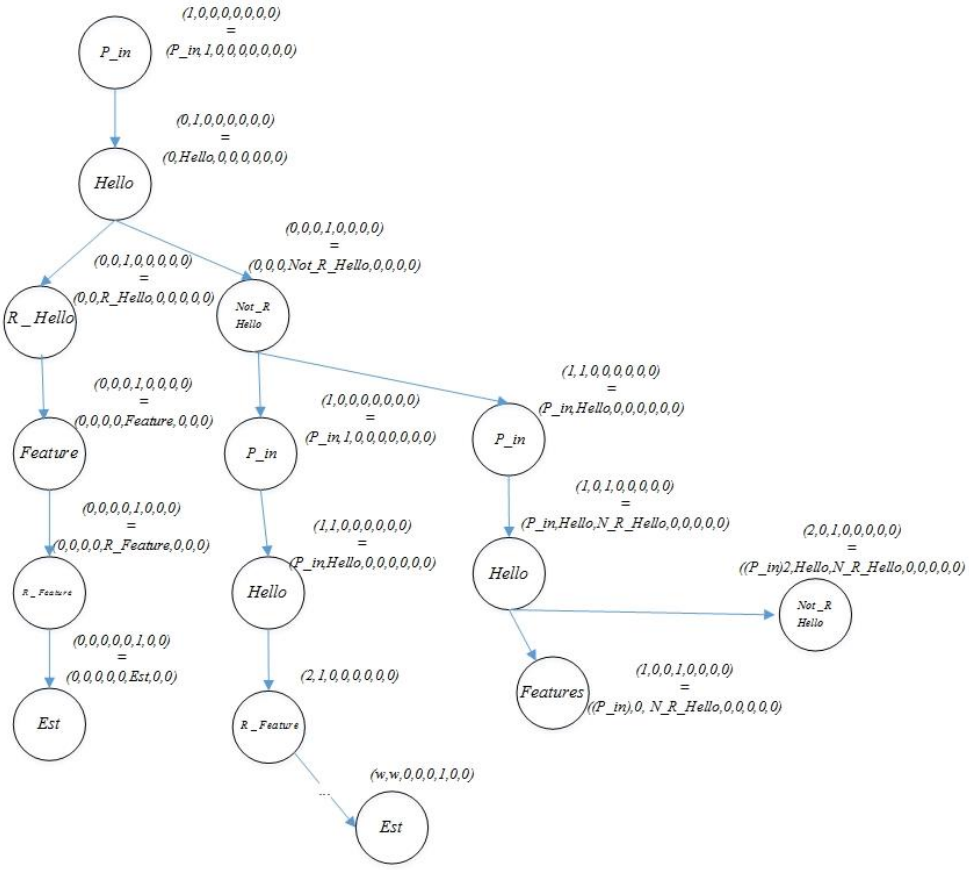


Рис.2.9. Дерево достижимости для модели Е-сети отображающей последовательность действий на OpenFlow коммутаторе

Результатом выполнения пунктов 1-7 могут быть следующие последовательности состояний, формирующие множество ветвей дерева достижимости Е-сети:

В утверждении (2.27) приведены ветви дерева достижимости, которые включают последовательности, приводящие к терминальным состояниям.

$$T(M_f)=\sum\overrightarrow{M_{0Est}}\cup\sum\overrightarrow{M_{0(N\_R\_Hello)}}\cup\sum\overrightarrow{M_{0(N\_R\_Fearute)}}=$$
$$=(P\_in,Hello,R\_Hello,Feature,R\_Feature,Est)\cup$$
$$\cup((P\_in,Hello)^2,R\_Hello,Feature,R\_Feature,Est)\cup \quad . \quad (2.28)$$
$$\cup((P\_in,Hello)^3,(R\_Hello,Feature,N\_R\_Feature)^2,Est)\cup$$
$$\cup(P\_in,Hello,N\_R\_Hello)^3\cup(P\_in,Hello,R\_Hello,Feature,N\_R\_Feature)^3$$

При анализе достижимости маркировки $M(0,0,0,0,0,1,0,0)$, приводящей к успешному завершению процесса, сформированы четыре последовательности:

$(P\_in,Hello,R\_Hello,Feature,R\_Feature,Est)$,

$((P\_in,Hello)^2,R\_Hello,Feature,R\_Feature,Est)$,

$((P\_in,Hello)^3,(R\_Hello,Feature,N\_R\_Feature)^2,Est)$.

Появление показателя степени у нескольких последовательностей указывает на несоблюдение свойства сохраняемости.

Последовательности $(P\_in,Hello,N\_R\_Hello)^3$ и $(P\_in,Hello,R\_Hello,Feature,N\_R\_Feature)^3$ имеют терминальные вершины, которые не соответствуют достижению желаемого результата.

В качестве примера приведем модель Е-сети, которая соответствует первичной стадии обмена управляющими сообщениями между контроллером и коммутатором (рисунок 2.10). Состояние *Switchlearning(porti)* соответствует приходу нового сообщения на один из портов коммутатора *(porti)* об изменениях в топологии сети или добавлении нового пользователя. При этом перенаправление нового управляющего пакета контроллеру осуществляется в том случае, если в таблице переадресации коммутатора не найдено ни одного совпадения для поля $Match_j$ (состояние *Don't find*). Если совпадения обнаружены (состояние *Find*), то коммутатор пересылает сообщение адресату, который указан в поле destination source таблицы переадресации (состояние *Forwardingdata*) или осуществляет обработку сообщения (смена счетчиков, приоритета, адресата) – состояние *ChangeTable*. Состояние *ChangeCommand* соответствует передаче обработанного пакета коммутатору. Передача может осуществляться напрямую или с помощью FlowVisor (состояние FlowVisor)

посредством инкапсуляции различных пакетов управляющей информации в единый канал.

Получение пакета OpenFlow коммутатором осуществляется тем портом, который указан в заголовке $Match_j$ (состояние $Receive\,porti$). Стоит отметить, что в соответствии со спецификацией OpenFlow пакеты могут быть доставлены как коммутатору, который инициировал установления соединения, так и всем коммутаторам сети, такую возможность в модели Е-сети определяет состояние $Receive all\ ports$.

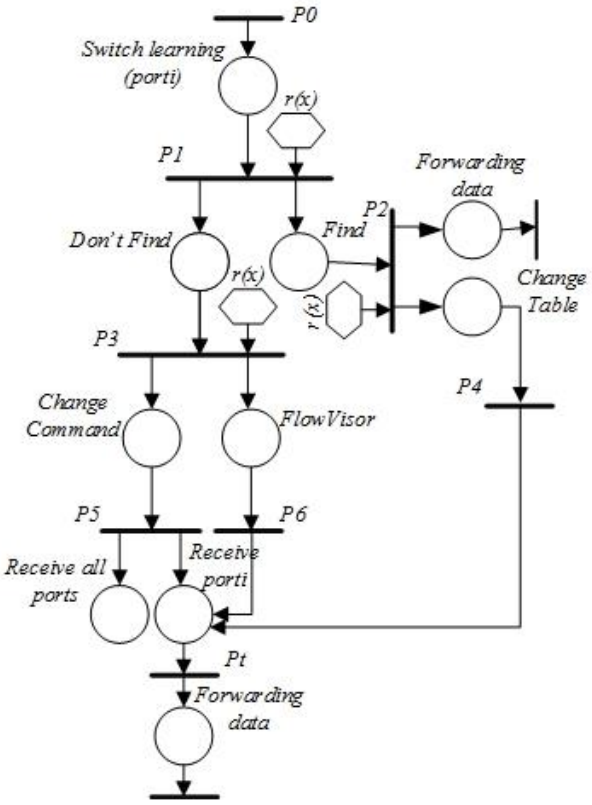


Рис.2.10. Модель процесса инициализации управляющего потока в сети по протоколу OpenFlow

Переход модели Е-сети P_0 соответствует процессу поступления нового сообщения (пакета) от устройств, находящихся на уровне передачи данных. Переход P_1 является управляемым, он соответствует процессу "обучения" коммутатора. Управляющий предикат $r(x)$ содержит информацию о данных, занесенных в таблицу переадресации. Здесь работает следующее требование: пакет пересылается контроллеру только в том случае, если в таблице

переадресации коммутатора не найдено ни одного совпадения в уже имеющемся поле $Match_j$: $P_1 \to Tf(Match_j \notin Match_{Tf})$. Процесс P_2 также содержит предусловия выполнения, в соответствии с которыми либо осуществляется его передача адресату либо последующая обработка полей и передача адресату – этому процессу соответствует переход P_4. Переход P_3 соответствует процессу обработки пакета контроллером с дальнейшей его передачей по тому же маршруту (переход P_5) либо инкапсулируя его посредством FlowVisor (переход P_6). Переходы P_t и $P_t^{'}$ соответствует процессу обработки полученного от контроллера пакета с набором команд, которые должны быть к нему применены: занесение в таблицу переадресации, модификация $Tf(f_{i_{din}}(Modif,t))$ или отбрасывание пакета $Tf(Drop)$.

Для модели, приведенной на рис. 2.10, формируется следующее дерево достижимости (рис. 2.11).

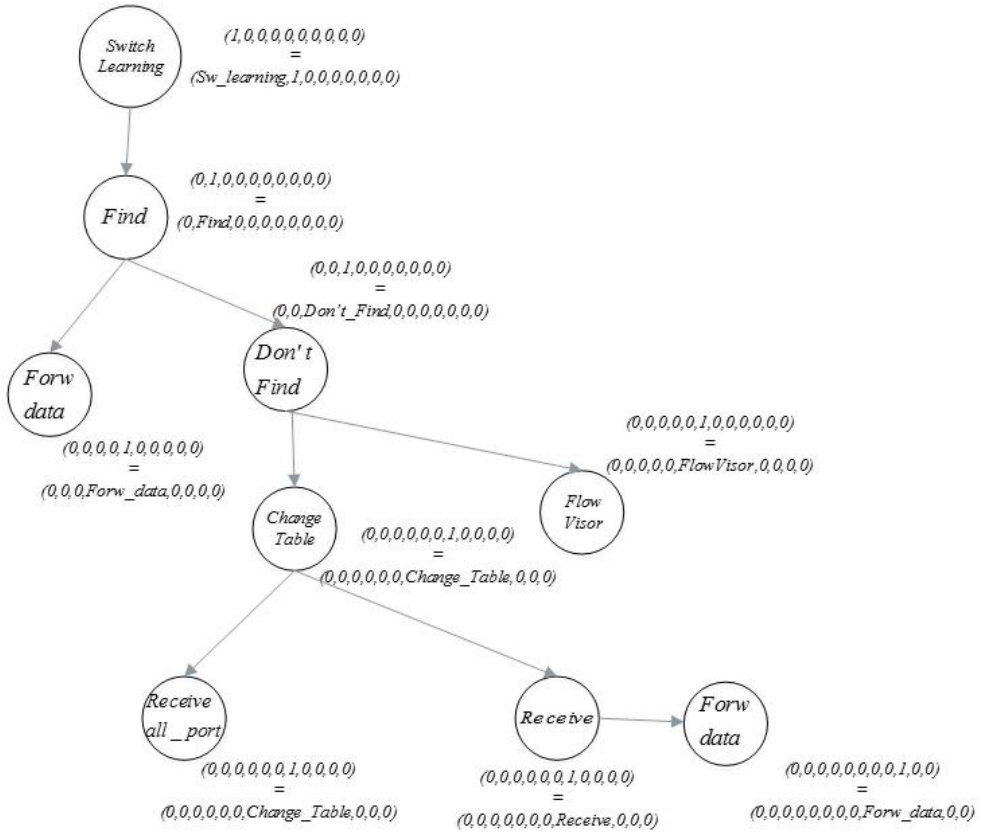


Рис.2.11. Дерево достижимости модели процесса инициализации управляющего потока

Таким образом, модифицированный метод построения дерева достижимости Е-сети позволяет формировать хронологические последовательности смены состояний и устанавливать численное значение возникновение метки в каждой вершине, что позволяет выполнить анализ таких алгоритмических свойств, как достижимость, ограниченность, сохраняемость, достижимость.

2.5 Метод синтеза Е-сети по формализмам алгебры коммутационных распределенных ресурсов

В основе предлагаемого метода лежит метод трансляции алгебры процессов в конечные автоматы и табличный метод преобразования формул, содержащих темпоральные операторы, в граф состояний [28, 62, 70]. В отличии от конечных автоматов Е-сети представляют собой двудольный ориентированный граф и имеют два типа вершин: вершины-позиции (P), и вершины-переходы (H), а также множество атрибутов $r(x)$, влияющих на срабатывание переходов.

Моделирование протокола посредством аппарата Е-сетей основано на определении множества состояний и процессов протокола, а также взаимосвязей между ними. Вершины-позиции соответствуют состояниям протокола, а вершины-переходы модели Е-сети определяют условия (процессы), результат выполнения которых соответствует переходу протокола в состояние [68-70].

Связь между вершинами-позициями и вершинами-переходами определяется функциями прямой $L(P,T)$ и обратной $D(T,P)$ инцидентности. Каждая вершина-позиция может содержать одну входящую $in: L(P,T)$ и одну исходящую $out: D(T,P)$ дугу.

В соответствии с (2.1-2.3) формализмы требований спецификации протокола содержат множество процессов $M(X,p)$, с указанием их приоритета p; множество событий Act и множество логических операторов.

Синтез Е-сети по формализмам алгебры коммуникационных распределенных ресурсов содержит следующую последовательность действий:

1. Формирования множества вершин-позиций, которые соответствуют множеству Act требований спецификации. При этом каждая позиция Е-сети описывается множеством State, содержащим следующие параметры:

State= [ID: StateID,
CURRENT: current state definition;
INCOMING: previous transmission,
OUTCOMING: next transmission],

где *ID* – уникальный идентификатор позиции; *CURRENT* – текущий элемент алфавита, соответствующий данной вершине (характеризующий определенное состояние – порт активен, сообщение сформировано); *INCOMING* – множество входящих переходов; *GOING* – множество исходящих переходов.

2. Формирование множества вершин-переходов, которые соответствуют множеству $M(X,p)$ требований спецификации. При этом каждый переход Е-сети описывается множеством *Transition*, содержащим следующие параметры:

Transition = [ID: Transition ID,
CURRENT: current Transition definition;
AT: atribute,
INCOMING: previous state,
OUTCOMING: next state],

где *ID* – уникальный идентификатор перехода; *CURRENT* – текущий элемент алфавита, соответствующий данному переходу, *AT* – предикат перехода, включает время срабатывания, приоритет, количество возможных запусков; *INCOMING* – множество входящих позиций; *GOING* – множество исходящих позиций.

3. Установление отношений инцидентности

При установлении отношений инцидентности между вершинами-позициями и вершинами-переходами могут возникнуть следующие варианты взаимосвязей:

а) текущее состояние предшествует только одному состоянию. Связь между элементами алфавита определена следующими логическими святками: следования ($\psi \to \gamma$), завершение одного процесса в пользу другого $P \perp Q$ и $P \setminus F$, выполнению мгновенного события или процесса $e.P$.

В данном случае позиции связываются через Т-переход.

б) текущее состояние предшествует нескольким состояниям. Связь между элементами алфавита определяется логическими операторами конъюнкции, дизъюнкции ($\varphi \to \gamma \vee \psi$ или $\varphi \to \gamma \wedge \psi$), а также логическими связками $P+Q$, $P \| Q$.

В данном случае позиции связываются через F-переход.

В случае если данному набору формул состояния соответствует формула пути с логическими связками $Q^u : P$, то рассматриваемые позиции связываются через *M Y*, значение предиката перехода зависит от требований спецификации.

в) несколько состояний предшествуют текущему состоянию. Связь между элементами алфавита определяется логическими операторами конъюнкции или дизъюнкции ($\gamma \vee \psi \to \varphi$ или $\gamma \wedge \psi \to \varphi$), а также темпоральным оператором *U* ($\gamma U \psi \to \varphi$), а также логическими связками $(P+Q) \xrightarrow{p} P \| Q$ с указанием приоритета.

В данном случае позиции связываются через J-переход.

В случае если данному набору формул состояния соответствует формула пути с логическими связками $P \Delta_u Q$ и $[P\_or\_Q]_U$, то рассматриваемые позиции связываются через *MX*, значение предиката перехода зависит от требований спецификации.

Правила формирования множества переходов представлены в таблице 2.1.

Таблица 2.1

Правила формирования множества переходов

Элемент логики	Семантика элемента логики	Тип вершины-перехода	Вариант взаимосвязи состояний
$\vee$ *(or)*	Логическая связка	F	б
$\wedge$ *(and)*	Логическая связка	F	б
$\vee$ *(or)*	Логическая связка	J	в
$\wedge$ *(and)*	Логическая связка	J	в
$\rightarrow$	Безусловная импликация	T	а
$\xrightarrow{x,p}$	Условная импликация, приоритет выбора	J	в
$\rightarrow^{r}$	Условная импликация	MY	б
$\leftarrow^{r}$	Обратная условная импликация	MX	в
$P \perp Q$	Логический оператор	T	а
$P + Q$	Логический оператор	T	а
$P + Q$	Логический оператор	F	б
$P \parallel Q$	Логический оператор	F	б
$Q^{u} : P$	Логический оператор	MY	б
$(P+Q) \xrightarrow{p} P \mid\mid Q$	Логический оператор	MX	в
$P \Delta_{u} Q$	Логический оператор	MX	в
$[P\_or\_Q]_{U}$	Логический оператор	MX	в
$\cup$	Темпоральный оператор Until	F	б
$\cup$	Темпоральный оператор Until	J	в

Либо следующее утверждение, задающее последовательную смену событий: «коммутатор (*Sw*) выполняет проверку IP-адреса пересылаемого пакета $P(T)$, в случае, если IP-адрес не принадлежит заданному диапазону, то осуществляется отбрасывание пакета»:

$$Sw(x) := matchSrcIP(T) \rightarrow Drop(T), \qquad (2.29)$$

ID: i

INCOMING: $\emptyset$;

CURRENT: $matchSrcIP(T)$;

GOING: $Drop(T)$;

ID: $i+1$;

INCOMING: $matchSrcIP(T)$;

AT: -

CURRENT: $Drop(T)$;

GOING: $\emptyset$;

На рисунок 2.12 представлен фрагмент Е-сети, соответствующей формуле (2.29).

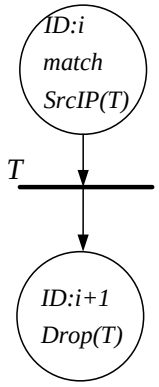


Рис. 2.12. Граф Е-сети соответствующий формуле (2.29)

Примером второго варианта взаимосвязи служит утверждение, соответствующее требованию спецификации протокола OpenFlow: «после получения нового пакета коммутатор может его модифицировать (Mes_MOD), передать получателю (Mes_FORW) либо удалить (Mes_DEL)» [68]:

$$Sw:receive(Mes) \rightarrow (Sw:Mes\_MOD)+(Sw:Mes\_FORW)+(Sw:Mes\_DEL) \quad (2.30)$$

В данном случае после процесса получения сообщения $receive(Mes)$ с равной долей вероятности могут возникнуть такие новые состояния, как, $(Sw:Mes\_FORW)$ или $(Sw:Mes\_DEL)$.

В этом случае формируется следующее множество вершин:

ID: i;

INCOMING: $\emptyset$;

ID: $i+1$;

INCOING: $Sw:receive(Mes)$;

CURRENT: $Sw : receive(Mes)$;

GOING: $(Sw : Mes\_MOD)$;

$(Sw : Mes\_FORW)$;

$(Sw : Mes\_DEL)$

CURRENT: $Sw : Mes\_MOD$;

GOING: $\varnothing$.

ID: $i+2$;

INCOMING: $Sw : receive(Mes)$;

CURRENT: $Sw : Mes\_FORW$;

GOING: $\varnothing$.

ID: $i+3$;

INCOMING: $Sw : receive(Mes)$;

CURRENT: $Sw : Mes\_DEL$;

GOING: $\varnothing$.

На рис. 2.13 представлен фрагмент Е-сети, соответствующей формуле (2.30).

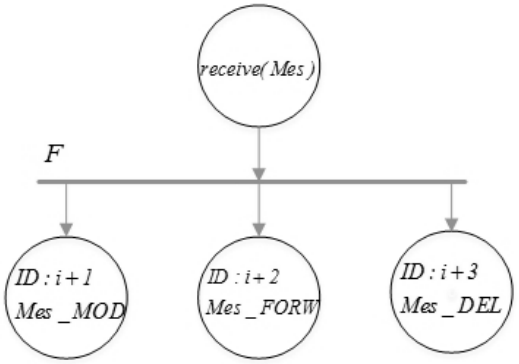


Рис. 2.13. Граф Е-сети соответствующий формуле (2.30)

Примером третьего варианта взаимосвязи является утверждение: «при переполнении буфера входящие пакеты отбрасываются». В данном случае состояние $M : cut$ достижимо, только в том случае если $M : send$ и $H.congestion$ истины [155]:

$$(M : send) \vee (H.congestion) \rightarrow G(M : cut), \qquad (2.31)$$

ID: i

INCOMING: $\varnothing$;

CURRENT: $M : send$;

GOING: $M : cut$;

ID: $i+1$;

INCOMING: $(M : send),(H.congestion)$

;

CURRENT: $M : cut$;

GOING: Ø;

ID: $i + j$ *;*

INCOMING: Ш;

CURRENT:; $H.congestion$

GOING: M **:** cut *;*

На рис.2.14 представлен фрагмент Е-сети, соответствующей формуле (2.31).

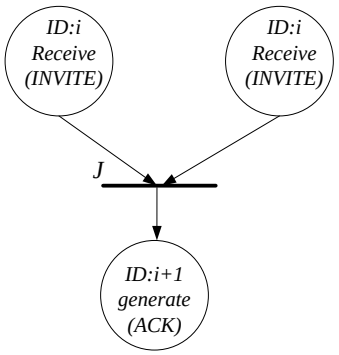


Рис.2.14. Граф Е-сети соответствующий формуле (2.31)

Примером синтеза MX перехода может служить следующее утверждение «по истечению RTT ($RTT>x$) сообщение Hello, переданное коммутатором, отбрасывается, но контролер находится в состоянии ожидания» и соответствующий формализм алгебры распределенных коммуникационных ресурсов [69]:

$$(Sw: send(Hello) \rightarrow (C^{RTT<x}: receive(Hello) + (C^{RTT>x}: not\_receive(Hello)), \qquad (2.32)$$

Для формулы (2.32) возможно выделить следующие состояния:

ID: i *;*

INCOMING: Ø;

CURRENT: $Sw: send(Hello)$ *;*

GOING: $receive(Hello)$ *;*

$not\_receive(Hello)$

ID: $i+1$*;*

INCOING: $Sw: send(Hello)$ *;*

AT:RTT, x;

CURRENT: $receive(Hello)$ *;*

GOING: Ø.

$ID:\ i+2;$

$INCOMING:\ Sw:send(Hello);$

$AT:RTT, x;$

$CURRENT:\ not\_receive(Hello);$

$GOING:\ \emptyset.$

На рис. 2.15 представлен фрагмент Е-сети (MX-переход), соответствующей формуле (2.32).

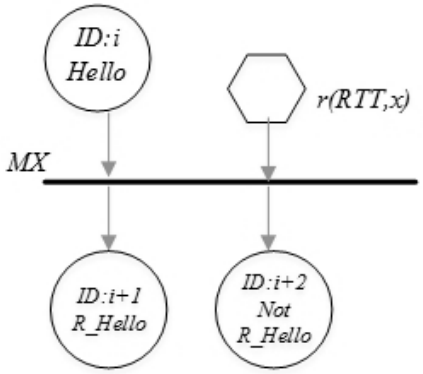


Рис.2.15. Граф Е-сети соответствующий формуле (2.32)

Примером синтеза MY перехода может служить следующее требование, относящееся к обработке полей таблицы переадресации: «при получении нового пакета производится сверка полей, содержащих IP-адрес и порт получателя сообщения, если совпадение найдено и значение поля «счетчик» в этот момент не равно нулю, то пакет передается получателю» и соответствующий формализм алгебры распределенных коммуникационных ресурсов [27]:

$$((DistIP:10.13.6.7)\&\&(port=80))^{C\neq 0}:Mes\_FORW. \qquad (2.33)$$

На рис. 2.16 представлен фрагмент Е-сети (MY - переход), соответствующей формуле (2.33).

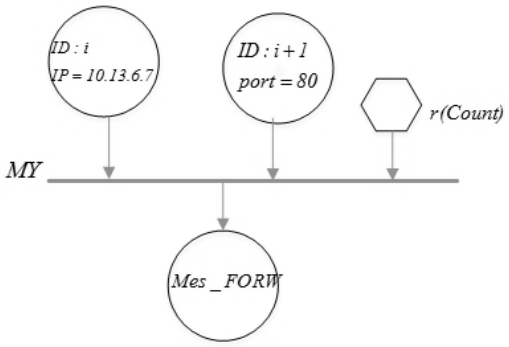


Рис.2.16. Граф Е-сети соответствующий формуле (2.33)

Таким образом может быть представлено однозначное преобразование формализмов спецификации протокола OpenFlow в фрагменты Е-сети. После того, как все фрагменты Е-сети сформированы, происходит их последовательное объединение.

2.4 Выводы по второму разделу

1. В рамках решения задачи формализации требований спецификации протокола OpenFlow предложено применять аппарат ACSR, который в полной мере позволяет формализовать условия выполнения процессов и сценарии наступления событий, позволяя при этом учитывать их приоритет и временные зависимости. На основе базовых правил ACSR составлены шаблоны поведения протокола в соответсвии с требованиями спецификации. Данные шаблоны поведения являются истинными в процессе всего жизненного цикла протокола.

2. Задача анализа и оценки корректности функционирования протокола OpenFlow сведена к решению задачи проверки выполнимости функциональных и нефункциональных требований, предъявляемых к реализации протокола. С целью сокращения материальных затрат в процессе проверки выполнимости как функциональных, так и нефункциональных требований предложено использовать модельный подход. В качестве аппарата моделирования использован апарат Е-сетей, в этом случае анализ и оценка корректности функционирования протокола OpenFlow выполняется путем

анализа таких алгоритмических свойств Е-сетей, как ограниченность, достижимость, активность, сохраняемость и безопасность.

3. С целью автоматизации процесса построения разработан метод синтеза модели Е-сети по ACSR формализмам требований спецификации, который базируется на однозначном преобразовании атомарных утверждений и в вершины-позиции, а логических связок - в вершины-переходы Е-сети.

4. В качестве метода анализа основных алгоритмических свойств Е-сетей рассмотрены матричные уравнения, формальные грамматики и дерево достижимости. Дерево достижимости предложено в качестве основного инструмента анализа. С его помощью предложено решать задачу проверки основных свойств, устанавливать факт возникновения зацикливаний в процессе функционирования протокола, а также выявлять избыточность.

РАЗДЕЛ 3
РАЗРАБОТКА МЕТОДА ВЕРИФИКАЦИИ ПРОТОКОЛА OPENFLOW

3.1 Анализ особенностей процесса верификации протокола OpenFlow

Отделение уровня управления от уровня передачи данных в OpenFlow-сетях наряду с множеством преимуществ имеет ряд недостатков, которые способны повлиять на эффективность функционирования всей сети. К таким особенностям отнесены следующие:

- двухуровневая структура сетевой архитектуры: уровень управления, представлен функциональными характеристиками контроллера, а уровень передачи данных - функциональными характеристиками коммутатора. При этом связь контроллер-коммутатор является чувствительной к изменению конфигурации и характеристик одного из элементов [88, 99, 101]. Так, например, задержка пакетов при обработке OpenFlow коммутатором может привести к смене их хронологического порядка и повлиять на дальнейшую корректную обработку контроллером;
- недетерминированный порядок следования пакетов: OpenFlow коммутаторы используют ряд алгоритмов, позволяющих максимизировать их производительность. Однако изменение порядка получения пакетов контроллером влечет за собой возрастание количества процессов обработки, что влияет на эффективность контроллера [53, 54];
- отсутствие стандартных механизмов обработки и передачи сообщений: спецификация протокола OpenFlow не определяет стандартных механизмов обработки и передачи сообщений OpenFlow на коммутаторе [32, 33];
- различия в реализациях сетевой операционной системы контроллеров: контроллеры имеют разную сетевую операционную систему, а, следовательно, функциональные особенности контроллеров могут значительно отличатся.

Исходя из вышесказанного, особое внимание при верификации OpenFlow следует уделить процессу распределения ресурсов и установлению верного хронологического порядка взаимодействий между сетевыми элементами, в частности, между контролером и коммутатором. Модель Крипке [11, 14, 37], в данном случае, не позволяет полноценно описать распределение ресурсов сети и условия смены состояний протокола. Выразительная мощность темпоральных логик, в свою очередь, не позволяет учитывать сложные функциональные зависимости изменения состояний ресурсов сети.

Множество возможных параллельных процессов, возникающих при функционировании протокола OpenFlow, значительно увеличивает размер пространства исследуемых состояний. Поиск соответствия между множественными взаимодействиями состояний вызывает эффект «комбинаторного взрыва» при дальнейшей верификации посредством классического подхода.

Таким образом, необходима разработка модифицированного метода верификации на основе классического подхода Model Checking, который позволит полноценно выполнять проверку соответствия требованиям и избежать эффекта «комбинаторного взрыва» [3, 14, 125].

3.2 Разработка метода верификации протокола OpenFlow

С целью повышения эффективности процесса верификации и устранения эффекта «комбинаторного взрыва» пространства исследуемых предложена следующая модификация основных этапов классического подхода Model Checking:

1. Этап моделирования.

1.1 Построение модели реализации протокола.

1.2 Формализация требований спецификации.

2 Этап верификации.

2.1 Определение области верификации.

2.2 Процесс верификации.

3. Выдача результатов верификации.

4. Формирование контрпримера.

В качестве исходных данных наряду с требованиями спецификации предлагается ввести дополнительную базу знаний: множество общих шаблонов поведения, которые являются истинными для всех версий протокола OpenFlow. Шаблон поведения – требование или совокупность требований спецификации, верные на протяжении всего времени функционирования протокола.

В зависимости от начальных условий может быть проведена полная либо частичная верификация, которая включает проверку соответствия только определенному набору требований спецификации. Схема предлагаемого модифицированного метода верификации на основе Model Checking приведена на рис. 3.1.

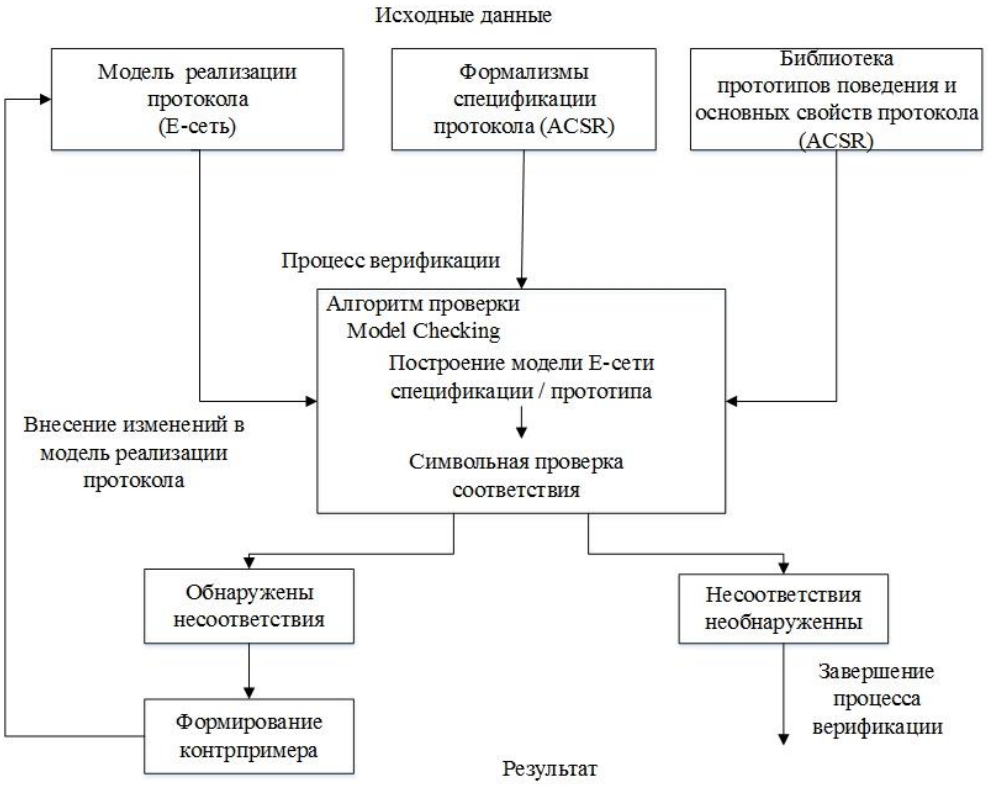


Рис.3.1. Структурная схема модифицированного метода Model Checking

3.2.1 Этап моделирования

На данном этапе предложено использовать модели реализации протокола с разной степенью детализации. Таким образом, модель реализации протокола

OpenFlow может включать набор характеристик, имеющих различный уровень детализации или вложенности: канал передачи данных в зависимости от поставленных задач, может быть представлен как целостный объект или совокупность нескольких объектов (детально могут быть представлены различные типы срезов), при построении таблица переадресации в качестве подуровней могут быть представлены команды контроллера, сообщения коммутатора, правила переадресации.

В качестве математического аппарата моделирования предложено применять Е-сети, так как аппарат Е-сетей позволяет учитывать все особенности протокола OpenFlow (п. 2.3). В зависимости от основных целей верификации такой поход позволяет охватывать конечное пространство состояний.

Формализацию требований спецификации предложено осуществлять посредством математического аппарата алгебры коммуникационных распределенных ресурсов (ACSR). При этом для каждого процесса или состояния могут быть заданы дополнительные характеристики и уточнения, позволяющие определять диапазон возможных дальнейших действий. Такое представление ведет к значительному сокращению пространства исследуемых состояний. Структурная схема этапа моделирования приведена на рис. 3.2.

Рис. 3.2. Этап моделирования

Дополнительным процессом на данном этапе является построение шаблонов поведения протокола. Шаблон представляет собой формализованный фрагмент спецификации протокола, содержащий набор правил, истинных в течении всего жизненного цикла протокола. Так, шаблонами основных свойств протокола OpenFlow могут выступать:

1. Корректный порядок следования OpenFlow пакетов.

В процессе передачи данных от коммутатора к контролеру возможно нарушение порядка следования пакетов (при применении различных алгоритмов сжатия на коммутаторе, при обработке таблиц переадресации, при возникновении задержек в каналах связи), что приводит к возникновению ложной информации о среде передачи данных и, как следствие, некорректной работе протокола.

Формализация требования учета последовательности обмена сообщениями между коммутатором и контролером посредством аппарата алгебры коммуникационных распределенных ресурсов представлена следующим образом:

$$S\_order \overset{def}{=} rec((\sum_{p=1}^{m} S\_order(sw),1)^{t} : (\sum_{p=1}^{k} S\_order(c),2)^{t+\tau}) + \\ +...+(\sum_{p=1}^{l} S\_order(sw),i)^{t+\tau+i} : (\sum_{p=1}^{n} S\_order(c),(i+1))^{t+\tau+(i+1)}))$$

где *k, l, n, m* - произвольные целые положительные числа, определяющие количество переданных пакетов, t,τ - временные характеристики, $p = 1...m,1...k,1...n,1...l$ - задает порядковый номер пакета.

Данная формализация позволяет учитывать несколько сценариев посылки и получения сообщений между контролером и коммутатором. В таком случае необходимо выполнять проверку по каждому отдельному сценарию. Шаблон, содержащий несколько сценариев обмена сообщениями, может быть представлен как:

$$S\_order = \begin{cases} br1 : rec((S\_order(sw),1)^{t} : (S\_order(c),2)^{t+\tau}) + ... + (S\_order(sw),i)^{t+\tau+i} : \\ (S\_order(c),(i+1))^{t+\tau+(i+1)})) \\ br2 : rec((S\_order(sw),1)^{t} : (S\_order(c),2)^{t+\tau}) + ... + (S\_order(sw),i)^{t+\tau+i} : \\ (S\_order(c),(i+1))^{t+\tau+(i+1)})) \\ ... \\ bri : rec((S\_order(sw),1)^{t} : (S\_order(c),2)^{t+\tau}) + ... + (S\_order(sw),i)^{t+\tau+i} : \\ (S\_order(c),(i+1))^{t+\tau+(i+1)})) \end{cases},$$

где S_order - порядковый номер пакета, сгенерированного либо коммутатором *(sw)* или контроллером (c), $br1$, $br2$, bri - возможный сценарий, sw - состояние коммутатора при отправке либо получении пакета, c - состояние контролера при отправке либо получении пакета.

2. Отсутствие взаимозависимости между потоками данных.

В этом случае, предлагаемый подход позволяет уменьшить пространство исследуемых состояний из-за систематической поочередной проверки смены состояний в одном канале.

Множество процессов, принимающих участие во взаимодействии контроллера и коммутатора, вне зависимости от версии протокола OpenFlow, является одинаковым. Так, алгоритм установления соединения, формирование виртуальных каналов связи, управление логическими срезами, обработка пакетов на коммутаторе всегда должны содержать определенный набор значений и атрибутов, поддерживающихся каждой версией протокола.

3.2.2 Этап верификации

Определение области проведения верификации

В зависимости от решаемых задач верификация протокола OpenFlow может быть полной (в случае построения нового сетевого решения на основе концепции SDN) или частичной (в случае модернизации сети или обновления версии протокола OpenFlow). В этом случае возможны следующие варианты верификации протокола [98, 105]:

1. Частичная проверка соответствия реализации протокола требованиям спецификации или проверка выполнимости основных свойств. При этом процесс верификации может быть описан следующим образом:

$$\begin{array}{l}((f_i, S_{iOF}, P_i)_{мод} \equiv (f_i, S_{iOF}, P_i)_{требование}) \\ \{(f, S_{OF}, P)_{свойства}\} \in (f, S_{OF}, P)_{мод}\end{array}, \quad (3.1)$$

где P - множество процессов (включая терминальные P_t), которые определены в спецификации OpenFlow и соответствует взаимодействию возможных элементов OpenFlow S_{OF} (смене их состояний), $f : P \to S_{OF}$ - зависимость, отображающая причинно-следственные связи между процессами и событиями, обобщенная система $(f, S_{OF}, P)_{мод}$ - модель реализации OpenFlow протокола, $(f, S_{OF}, P)_{свойства}$ - модель отдельного требования или свойства OpenFlow протокола.

2. Частичная проверка соответствия реализации протокола на основе прототипов (шаблонов поведения), которые являются общими для всех версий протокола. Процесс верификации в этом случае может быть представлен следующим образом:

$$\begin{array}{l}((f_i, S_{iOF}, P_i)_{мод} \equiv (f_i, S_{iOF}, P_i)_{прототип}) \\ \{(f, S_{SDN}, P)_{прототип}\} \in (f, S_{SDN}, P)_{мод}\end{array}, \quad (3.2)$$

где $(f_i, S_{iOF}, P_i)_{прототип}$ - прототип, содержащий набор требований, которые должны быть реализованы в протоколе OpenFlow.

3. Полная проверка соответствия реализации протокола требованиям спецификации, которая может быть задана следующим образом:

$$\begin{array}{l}((f, S_{OF}, P)_{мод} \equiv (f, S_{OF}, P_i)_{спец}) \\ \{(f, S_{OF}, P)_{мод}\} \in (f, S_{OF}, P)_{спец}\end{array}, \quad (3.3)$$

где $(f_i, S_{iOF}, P_i)_{спец}$ - модель спецификации OpenFlow протокола, зависит от текущей версии протокола.

Проверка однозначного соответствия модели реализации протокола требованиям спецификации.

Современные верификаторы протоколов информационного обмена, в частности протокола OpenFlow, выполняется с помощью метода поиска символьного соответствия между реализацией и требованиями спецификации. В основу данного метода заложен поиск соответствия между поведенческими цепочками спецификации и модели реализации.

Применение формальных грамматик [110, 111, 131] позволяет выполнять как частичную проверку, путем построения небольших поведенческих шаблонов в соответствии с требованиями спецификации и поиск их вхождения в модели реализации, так и полную проверку, путем поиска вхождения в модель реализации каждого символа поведенческой цепочки, соответствующей требованиям спецификации.

Поведенческая цепочка, соответствующая требованиям спецификации, строится по правилам порождающей грамматики. Формальная порождающая грамматика G формируется с помощью четырех основных объектов и может быть задана с помощью следующего выражения [137]:

$$G = (V_t, V_n, Pr, S), \tag{3.4}$$

где V_t – непустое конечное множество терминальных символов; V_n – непустое конечное множество нетерминальных (вспомогательных) символов. Объединение множеств V_n и V_t, образует словарь формальной грамматики L, $L = V_n \cup V_t$; Pr – непустое конечное множество правил продукций вывода символьной цепочки; S – начальный символ грамматики.

Продукцией вывода является цепочка правил вида [131, 137]: $\delta(\alpha, \beta)$, где $\delta()$ – функция перехода, α – начальное значение вывода или входной символ, β – выходной символ, при этом $\alpha \to \beta$.

Определение ограничений, накладываемых на правила вывода продукций Е–сети является ключевым звеном построения формальной грамматики. Согласно классификации по Хомскому [126] грамматики Е-сетей относятся к классу укорачиваемых контекстно-свободных (УКС) либо

регулярных грамматик (УКС без учета пустой цепочки). Цепочка, которая не содержит ни одного символа, называется пустой цепочкой (ε) [137].

В общем случае процесс верификации посредством символьной проверки на основе формальных грамматик может быть задан следующим способом:

1. Выбирается тип порождающей грамматики.
2. Определяется множество состояний, которые формируют алфавит формальной грамматики (Σ).
3. Определяется начальное состояние s_0, и искомое заключительное состояние (F). F является ключевым состоянием, которое должно быть достигнуто в процессе функционирования протокола.
4. Определяется последовательное соответствие смены символов формальной грамматике и смены состояний модели протокола. Проверка соответствия основывается на эквивалентности функций срабатывания переходов между состояниями модели реализации и требованиями спецификации $\delta(h_{S_i}, S_i, h_{S_{i+1}}) \equiv \delta(h_{M_i}, M_i, h_{M_{i+1}})$ [126]:

$$v_i(h_{C_i}, h_{M_i}) = \begin{cases} s_{C_i}, True \mid (s_{C_i} \equiv s_{M_i}) \wedge (s_{C_{i+1}} \equiv s_{M_i+1}), \\ \delta(s_{C_i}, h_{Ci}, s_{C_i+1}) \equiv \delta(s_{M_i}, h_{Mi}, s_{M_i+1}) \\ and \\ 0, False \mid (s_{C_i} \neq s_{M_i}) \vee (s_{C_i+1} \neq s_{M_i+1}), \end{cases}, \quad (3.5)$$

где s_{C_i} - текущее состояние модели спецификации; s_{M_i} - текущее состояние модели реализации, s_{C_i+1} - состояния предшествующие и следующие состоянию s_{C_i}, s_{M_i+1} - состояния предшествующие и следующие за состоянию s_{M_i}, h_{Ci} - переход, соответствующий смене состояний s_{C_i} и s_{C_i+1}, h_{Mi} - переход, соответствующий смене состояний s_{M_i} и s_{M_i+1}.

Применение формальных грамматик позволяет существенно сократить пространство исследуемых состояний, что эффективно при проверке корректности поведения разрабатываемых протоколов (логики работы заложенных в них алгоритмов). Однако эффективность данного метода снижается при верификации сложных сетевых решений. Так, не всегда

возможно описать множество функциональных особенностей OpenFlow протокола посредством формальных грамматик, например, динамическую смену таблиц переадресации коммутатора при изменении топологии сети или реактивную модель обработки запросов.

Верификация выполняется путем установления однозначного соответствия между состояниями ветвей дерева достижимости модели реализации и модели спецификации протокола. При этом совокупность ветвей дерева достижимости является возможными инвариантами поведения реализации протокола.

В соответствии с п.2.3 на этапе анализа, предшествующему этапу верификации, осуществляется построение дерева достижимости модели Е-сети реализации протокола. Таким образом, модель Е-сети реализации и соответствующее ей дерево достижимости являются исходными данными предлагаемого процесса верификации. Структурная схема процесса верификации приведена на рис. 3.3.

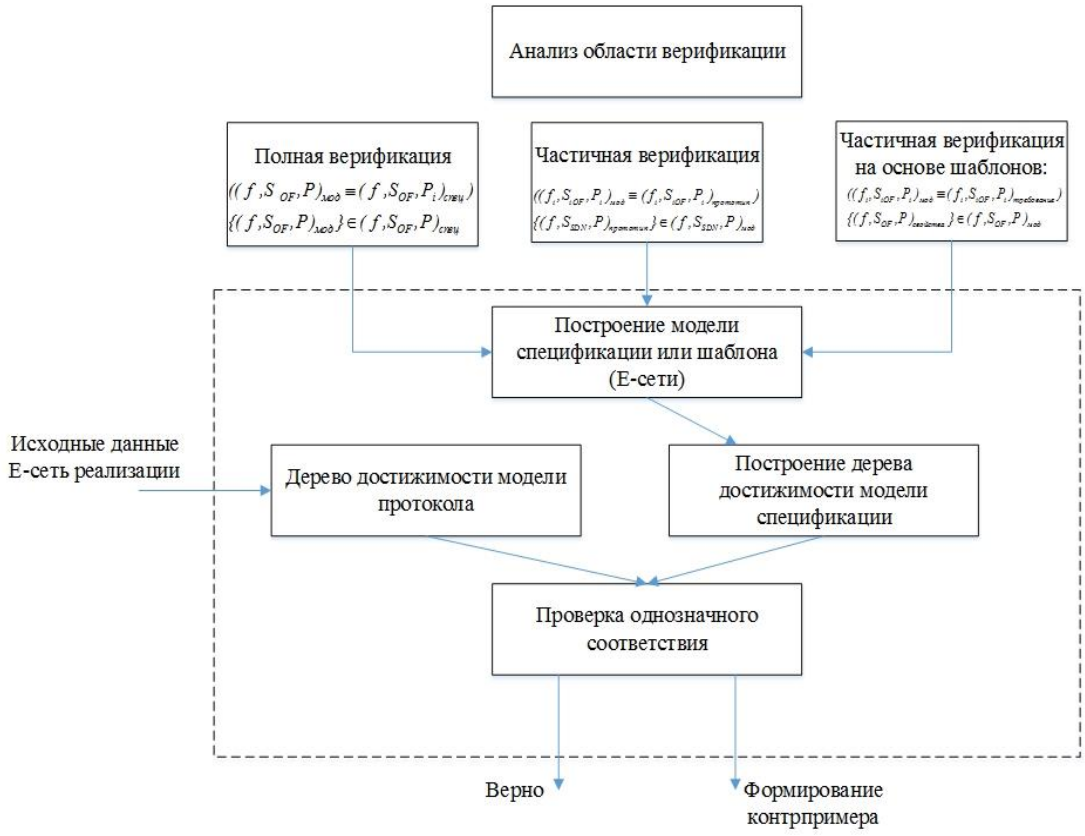


Рис. 3.3. Процесс проверки соответствия модели реализации протокола требованиям спецификации

Нахождение соответствия между состояниями дерева достижимости модели реализации и модели спецификации протокола может быть задано в виде следующей системы уравнений:

$$R_i(s_{C_i}, s_{M_i}) = \begin{cases} M(S_M) \in M(S_C), \\ s_{c_i}, True \mid s_{c_i} \equiv s_{M_i}, \\ \delta(s_{c_i-1}, s_{c_i}) \equiv \delta(s_{M_i-1}, s_{M_i}) \\ and \\ 0, False \mid s_{C_i} \neq s_{M_i}, \\ \delta(s_{C_i-1}, s_{C_i}) \neq \delta(s_{M_i-1}, s_{M_i}), \end{cases}, \quad (3.6)$$

где S_M - множество всех возможных вершин модели реализации протокола, S_C - множество всех возможных вершин модели спецификации, s_{C_i} - текущее состояние модели спецификации; s_{M_i} - текущее состояние модели реализации; s_{C_i-1} и S_{C_i+1} - состояния предшествующие и следующие состоянию s_{C_i}, s_{M_i-1} и s_{M_i+1} - состояния предшествующие и следующие за состоянию s_{M_i}.

Предлагаемый метод проверки базируется на символьном подходе, логика процесса нахождения соответствия (3.5) и (3.6) схожа, однако в (3.6) анализируется лишь текущее состояние модели реализации протокола.

3.2.3 Построение контрпримера

При верификации, завершившейся с отрицательным результатом (return false – ошибка найдена) осуществляется построение контрпримера.

Контрпример представляет собой последовательность состояний реализации протокола (одну из ветвей дерева достижимости), приходящих к возникновению ложного состояния и, как следствие, некорректному поведению протокола.

При построении контрпримера формируется множество состояний протокола $M(S')$, где $S' \in F \cup \neg F$.

Основным требованием, при формировании контрпримера является достижение конечного состояния.

Формирование контрпримера осуществляется по следующему принципу:

1. Построение последовательности, содержащей контрпример, начинается из начального состояния модели реализации протокола;

2. На каждом шаге следующее состояние пути выбирается из потомков текущего состояния:

- если до текущего момента все состояния последовательности принадлежали $M(S)$, $S \in F$, то при наличии среди потомков состояний, принадлежащих также $M(S)$, следующее состояние выбирается из них: $\{ s_i \to s_{i+1} \mid s_i, s_{i+1} \in F \}$;

- если текущее состояние последовательности не принадлежит $M(S)$, то оно считается состоянием ошибки; при наличии среди потомков состояний, принадлежащих $M(S')$, следующее состояние выбирается из них: $\{ s_i \to s'_{i+1} \mid s_i \in F, s'_{i+1} \in \neg F \}$;

- если к текущему моменту в последовательности уже присутствуют состояния, не принадлежащие множеству $M(S)$, то следующее состояние пути выбирается из всех состояний-последователей, принадлежащих множеству $M(S')$, $\{ s'_i \to s'_{i+1} \mid s'_i, s'_{i+1} \in \neg F \}$;

- если текущее состояние не имеет потомков, построение контрпримера окончено. В таком случае контрпример может быть задан следующим образом:

$$P_{контр} = P(s_0 \ldots s_{i-1} \mid s_0, s_1, \ldots s_{i-1} \in F) + s_i + P(s_{i+1} \ldots s_{закл} \mid s_{i+1}, s_{i+2}, \ldots s_{закл} \in \neg F \cup F); \quad (3.7)$$

- если последующее состояние уже встречалось ранее, то создается пометка образования цикла и осуществляется поиск в глубину. Чаще всего он реализуется посредством алгоритма двойного обхода в глубину, который представлен следующим псевдокодом [98, 126]:

```
bool emptiness()
{ для всех s_i ∈ Σ
 dfs1(s_i);
terminate false;  }
```

```
void dfs1(s_j)
{ пометить s_j флагом flag1;
для всех последователей s_{j+1} перехода s_j
if (s_{j+1} не помечена флагом flag1)
     dfs1(s_{j+1});
          if (accept(s_j))
     dfs2(s_j);}
      void dfs2(s_j) { пометить s_j флагом flag2;
 для всех последователей s_{j+1} вершины s_j
{ if (s_{j+1} помечена флагом flag1)
     terminate true;
     else
          if (s_{j+1} не помечена флагом flag2)
          dfs2(s_{j+1}); }
}
```

Структурная схема построения контрпримера приведена на рисунке 3.4.

Рис. 3.4. Процесс построения контрпримера

Алгоритм двойного обхода выдает результат *true*, если был найден допускающий путь, и *false* – в противном случае. При этом необходим возврат на этап анализа основных алгоритмических свойств протокола. При получении состояния *true* осуществляется формирование допускающего пути $post(s_{i\to j})$.

При этом в стеке второго DFS хранится путь из состояния s_i или начального состояния модели в некоторое состояние s_j, содержащееся в стеке первого DFS. Тогда, достроив этот путь состояниями, находящимися в стеке первого DFS выше состояния s_j, получим допустимый цикл $s_i \to ...s_j... \to s_i$.

Таким образом, допускающий путь будет иметь вид:

$$P_{доп} = s_i(s_i \to ...s_j... \to s_i)^n \equiv s_i post(s_{i \to j}). \quad (3.8)$$

Контрпример может быть задан следующим образом:

$$\begin{aligned} &P_{контр} = P(s_0...s_{i-1} \mid s_0, s_1, ...s_{i-1} \in F) + s_i(post(s_{i \to j}))^n + \\ &+P(s_{j+1}...s_{закл} \mid s_{j+1}, s_{j+2}, ...s_{закл} \in \neg F \cup F) \,||\, +(\neg s_i(post(s_{i \to j}))^n) + \\ &+(\neg P(s_{j+1}...s_{закл} \mid s_{j+1}, s_{j+2}, ...s_{закл} \in \neg F \cup F)). \end{aligned} \quad (3.9)$$

Обобщённый алгоритм предлагаемого метода верификации Model Checking, может быть представлен следующим псевдо кодом:

```
verified states = [];
explored states = [];
errors = [];
initial state_pr = create initial state();
initial state_sp = create initial state();
for initial state(s0_pr) do {
     s0_pr.atributes = {}
     s0_pr. enable transition()={}
     P0_pr.atribute = {}
}
     for s0 in initial state & p0 is active transitions:
               create discover pattern ((M(s_pr) ∩ M(s_sp)
          ≠∅) & (pi_pr=pi_sp)):
               do state pr stack.push([si_pr, pi_pr])
                state sp stack.push([si_sp, pi_sp])
     while si_pr!=terminal & pi_pr is active do
               discover pattern;
               si_pr, pi_pr = choose(next pr state stack);
               (next state pr = run(state (i++), transaction
          (i++)))
               for si in state.si and Pi in state.Pi do
                         (enable       transaction([ Pi → Si_OF ],
                    state, id)
                         discover pattern;
                         explore state ++;
                         Sequence:
```

```
                    (next state stack)0=[];
                    (next state stack)1=[];
                    …
                    (next state stack)i=[];
                    Return true
                if def discover pattern(error):
                errors.append([e, trace])
            if si!=terminal & pi not active
            error =[si];
        form contrexampe;
        if si=terminal & pi active
        error =[si];
    form contrexampe; verification state++;
    feturn false
    stop procedure && create report
        End if;
    End while;
    End for;
    End for all.
```

В соответствии с разработанным методом верификации исходными данными для проверки соответствия является Е-сеть реализации протокола и ACSR формализмы спецификации, которые включают множество требований спецификации и библиотеку шаблонов. В зависимости от исходных данных в ходе проверки эквивалентности модели спецификации и реализации протокола возможны следующие ситуации.

1. Для модели спецификации протокола построена лишь одна ветвь дерева достижимости. Такая ситуация возможна при проверке определенного свойства или требования:

$$\overrightarrow{\sum M_{0f}} = (S_0 p_1 .... p_f), \tag{3.10}$$

где S_0 - начальное состояние модели, определяемое маркировкой Е-сети, $p_1 .... p_f$ - множество достижимых состояний модели.

Такая ситуация возникает при частичной проверке соответствия. На рисунке 3.5 приведен фрагмент Е-сети, определяющий одну поведенческую цепочку (последовательная смена состояний).

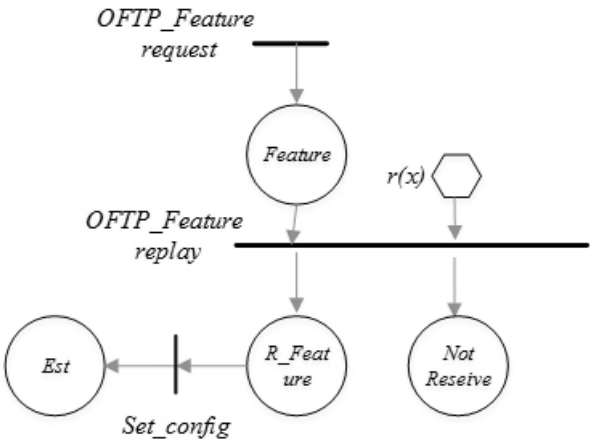


Рис. 3.5. Фрагмент Е-сети, соответствующий одному варианту смены состояний (определение характеристик канала связи)

Начальным состоянием является состояние *Feature*. Заключительным состоянием является состояние *Est*. Дерево фрагмента Е-сети содержит конечное множество состояний и не содержит циклов.

2. Дерево достижимость модели спецификации протокола содержит несколько независимых ветвей. Такая ситуация возможна при наличии нескольких параллельных процессов и полной проверке соответствия.

Данному сценарию соответствует объединение нескольких последовательностей состояний:

$$T(M_f) = \sum \overrightarrow{M_{0f}} \cup \sum \overrightarrow{M_{0f}} \cup \sum \overrightarrow{M_{0f}} = (S_0 p_1 \ldots p_f) \cup (S_0 p'_1 \ldots p_f) \cup (S_0 p''_1 \ldots p_f). \quad (3.11)$$

На рис. 3.6 приведен фрагмент Е-сети, для которого возможно возникновение нескольких цепочек. Ниже приведены все возможные сценарии поведения модели в зависимости от условия срабатывания перехода $ForwTable\_proceed$.

Начальным состоянием является состояние P_in, заключительными состояниями - $Receive$, $Modify$, $Drop$.

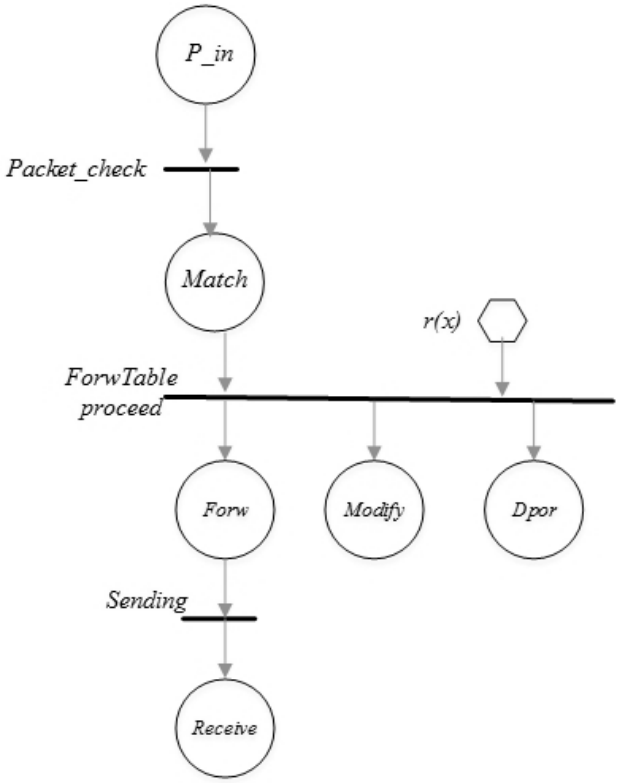


Рис. 3.6. Фрагмент Е-сети, соответствующий нескольким независимым последовательностям смены состояний

Упорядоченные множества последовательности смены состояний модели Е-сети, могут быть представлены следующим образом:

$$T(M_f) = (P\_in, Match, Forw, Receive) \cup (P\_in, Match, Modify) \cup \cup (P\_in, Match, Drop), \quad (3.12)$$

3. Существуют взаимодействующие параллельные процессы (соответствует образованию циклов):

$$M(T_f) = (S_0 p_1 \ldots (h_j \ldots)^n p_f). \quad (3.13)$$

Фрагмент модели Е-сети с возможным возникновением циклов приведен на рис. 3.7.

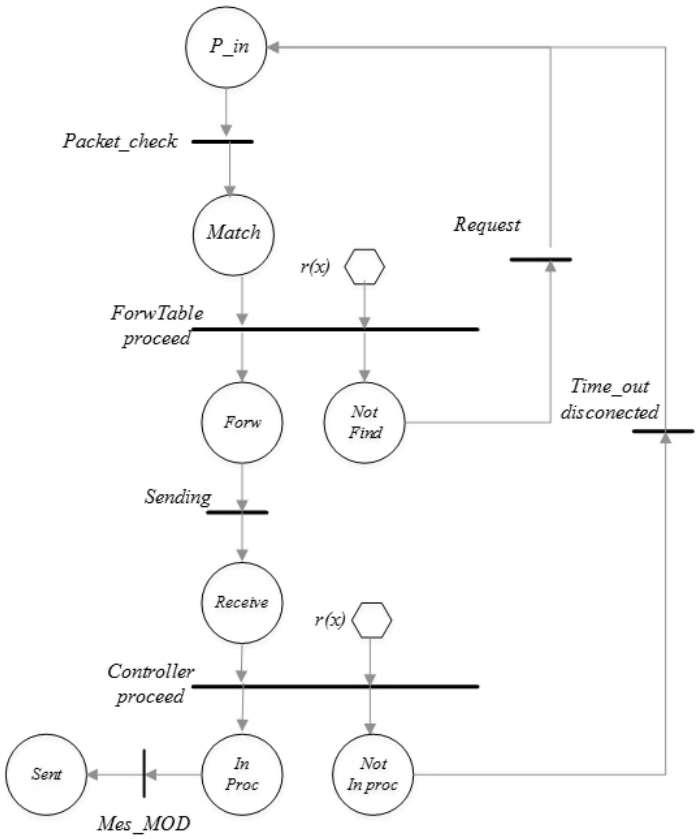


Рис. 3.7. Фрагмент E-сети, соответствующий образованию циклов

Начальным состоянием является состояние P_in. Заключительным состоянием является состояние $Sent$. Множество последовательностей, образующие ветви дерева достижимости данного фрагмента E-сети может быть представлено следующим образом:

$$T(M_f) = (P\_in, Match, Forw, Receive, In\_proc, Sent \cup$$
$$\cup((P\_in, Match, Not\_Find)^n Forw, Receive, In\_proc, Sent \cup \quad . \quad (3.14)$$
$$\cup((P\_in, Match, Not\_Find, Forw, Receive, Not\_in\_proc)^j, In\_proc, Sent)$$

Наличие неоднократно повторяющихся состояний может свидетельствовать об образовании циклов. Верификация моделей, в которых возможно появление петель является наиболее сложной задачей. Для ее решения дополнительным методом проверки является алгоритм двойного поиска в глубину (*Depth-First Search*, DFS), который приведен в п.3.2.3 [97].

Если при построении конечного дерева достижимости моделей протокола и спецификации возникают перемежающиеся последовательности состояний $M(T_f)=(S_0 p_1 ...(h_j ...)^n p_f)$, то необходима количественная оценка соответствия между запусками переходов модели спецификации и модели реализации. Количественное соотношение можно установить с помощью ввода дополнительного счетчика n, он соответствует количеству срабатываний перехода модели спецификации. Фрагмент Е-сети, приведенный на рис.3.9, имеет два состояния (*Not Find, Not In_proc)*, приходящих к образованию цикла. Таким образом, целесообразно ввести атрибуты счетчиков в позиции *ForwTable proceed* и *Controller proceed.* Процесс построения контрпримера при применении формальных грамматик приведен в [126].

3.3 Верификация протокола балансировки нагрузки в сетях с поддержкой концепции Software-Defined Networking

В качестве примера приведен процесс верификации протокола выбора каналов связи с наименьшей загруженностью сетевых ресурсов в OpenFlow-сетях [105, 109, 116] посредством формальных грамматик [128] и с помощью построения дерева достижимости.

Распределение сетевой нагрузки между множеством коммутаторов, серверов и вычислительных узлов OpenFlow-сети позволяет достигать высоких показателей производительности, масштабирования и отказоустойчивости. Важной задачей при этом становятся внедрение эффективных алгоритмов распределения нагрузки между сетевыми элементами, а также возможности, динамического перераспределения нагрузки между коммутаторами, и закрепленными за ними серверами и вычислительными узлами. OpenFlow-сети, как правило, проектируются и функционируют как географически распределенные системы. Загруженность каналов связи между контролером и коммутатором имеет динамический характер.

Основной задачей протокола балансировки загрузки является анализ эффективности каналов связи, которая зависит от производительности сетевых компонент, времени отклика, пропускной способности и динамики ее изменения. Пороговые значения для данных показатели могут быть получены на основе численных характеристик параметров качества. На основании полученных значений формируется правило выбора наилучшего (менее загруженного) канала связи.

В результате работы протокола определяются каналы связи, которые имеют постоянную пропускную способность или значение пропускной способности которых варьируется незначительно; а также каналов связи с прогрессивно уменьшающейся или не соответствующей требуемым показателям качества QoS пропускной способностью [1, 4].

Для эффективного решения поставленной задачи на этапе построения Е-сети модели алгоритма в вектор атрибутов меток управляющих переходов вводятся следующие параметры: время жизни запроса (метки) TTL, интервал времени проведения повторной проверки работоспособности канала связи - T_{to} пороговые значения пропускной способности канала связи $Th_i^{ch}(min)$ - минимальная пропускная способность согласно параметрам QoS для k-ого канала, величина изменения пропускной способности $\Delta\ Th_i^{ch}$ ($\Delta\ Th_i^{ch} = Th_i^{ch} \mp Th_{i-1}^{ch}$), где Th_i^{ch}, Th_{i-1}^{ch} – значение пропускной способности на i-том и $i-1$ интервале для каждого канала связи. Модель Е-сети алгоритма определения ненагруженных каналов связи с допустимой пропускной способностью представлена на рис. 3.8.

Задача установления каналов с ухудшающейся или несоответствующей требованиям QoS пропускной способностью решается с помощью анализа изменения значений пропускной способности $\Delta\ Th_i^{ch}$ для каждого временного интервала, определяемого T_{to}. В случае, если ΔTh_i^{ch} наблюдается незначительное снижение ΔTh_i^{ch}, то передача потока данных по каналу связи не прекращается. Если значение ΔTh_i^{ch} снижается скачкообразно, то загрузка на канал связи

уменьшается (в зависимости от других параметров сети) и производится дальнейший мониторинг значений Th_i^{ch}.

В случае, если значение Th_i^{ch} оказывается меньше значения нижнего допустимого порога $Th_i^{ch}(min)$, то канал связи не используется в течении интервала T_{to}; если значение Th_i^{ch} больше нижнего допустимого порога $Th_i^{ch}(min)$, то проверка продолжается, при этом значение Th_i^{ch} сохраняется как наименьшее достигнутое - $Mem(Th_i^{ch})$. В дальнейшем отсчет изменений идет относительно сохраненного хранения $Mem(Th_i^{ch})$. Модель соответствует реальному поведению алгоритма и детально приведена в [126].

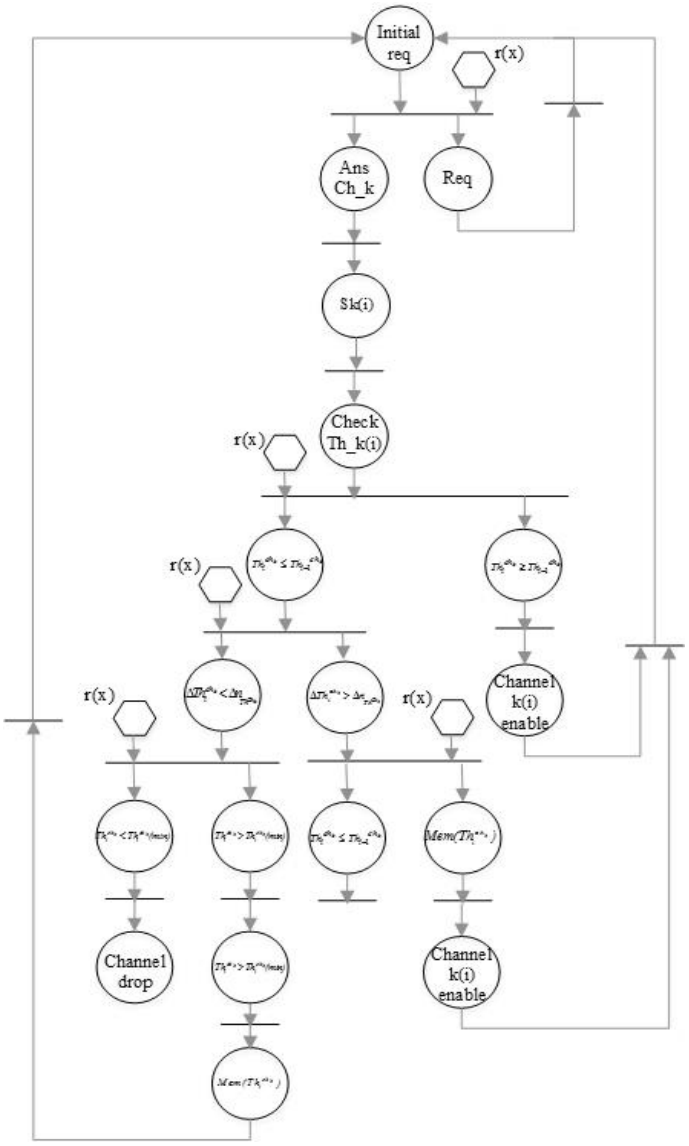


Рис. 3.8. Граф Е-сети процедуры определения каналов связи с допустимой пропускной способностью

Позиции модели Е-сети соответствуют следующим состояниям: *Initial request* - начальное состояние алгоритма (определение каналов связи – посылка эхо-запроса); *Ans Ch_k* – получение ответа от *k*-го канала связи; Reg – ответ от канала связи в течении времени T_{to} не получен, генерирование повторного запроса; s_k - предоставлении информации о k-м канале связи; $Check\_Th\_k(i)$ - начало процедуры определения каналов связи с допустимой пропускной способностью; $Th_i^{ch_k} < Th_{i-1}^{ch_k}, Th_i^{ch_k} > Th_{i-1}^{ch_k}$ - состояния, свидетельствующее об изменении (снижении/увеличении) пропускной способности канала связи $Th_i^{ch_k}$ на i-ом шаге проверки; $Mem(Th_i^{ch})$ - фиксирование значения изменения пропускной способности на i-том и $i-1$ интервале времени; $Th_i^{ch_k} < \Delta n$ - определение скачка изменения пропускной способности; $Th_i^{ch_k} > Th_i^{ch_k}(min)$, $Th_i^{ch_k} < Th_i^{ch_k}(min)$- сравнение текущего значения с минимальным значением; $Channel\_k(i)\_enable$ - занесение канала связи в список каналов с допустимой пропускной способностью; $Channel\_drop$- занесение канала связи в список игнорируемых; *Drop Tto* - сброс счетчика интервала проверки.

Для определения языка модели необходимо четко указать множество заключительных состояний, достижимых в ходе функционирования процедуры определения пропускной способности. Заключительным этапом процедуры является состояние $Channel\_k(i)\_enable$, где k - номер канала связи.

В соответствии с требованиями, предъявляемые к функционированию алгоритма поиска каналов связи с недостаточной пропускной способностью, необходимыми условиями являются:

- установка временного интервала проверки на соответствие QoS пропускной способности каналов связи: $(M, s_0) = T_{to}$;
- определение эталонной пропускной способности каждого канала связи Th^{ch_k}: $(M, s_k) = Th^{ch_k}$;
- определение минимального порогового значения пропускной способности для каждого возможного канала связи: $(M, s_k) = Th_i^{ch}(min)$;

- определение допустимого интервала уменьшения пропускной способности: $(M, s_k) = \Delta n_{Th^k}$;
- установление дополнительного динамического бинарного атрибута каналов связи ($r_i(A)$который равен *0* – на начальном этапе сравнения и *1* – обнаружении первого уменьшения $Th_i^{ch_k}$)

Приведенные выше начальные состояния являются атомарными утверждениями и могут быть представлены с помощью аппарата ACSR следующим образом:

$$(M, s_0) \models T_{to} \vee k \vee Th^{ch_k} \vee Th^{ch_k}(min) \vee \Delta n_{Th^{ch_k}} \wedge r_i(A); \qquad (3.15)$$

Проверка изменения пропускной способности на каждом шаге работы алгоритма для определенного *k*–го канала связи может быть представлена следующим образом:

$$\begin{aligned} Check\_Th^{chk} \models (Check\_Th^{chk})^{t<T_{to}} : ((Th_i^{ch_k} \le Th_{i-1}^{ch_k}) \cup \\ \cup (Th_i^{ch_k} > Th_i^{ch_k}(min) \rightarrow Check\_Th^{ch+k}; \end{aligned} \qquad (3.16)$$

Если уменьшение пропускной способности зафиксировано впервые, то справедлива следующая формулировка:

$$\begin{aligned} &((Th_i^{ch_k} \le Th_{i-1}^{ch_k} \mid r_i(0))^{t<T_{to}} : ((|Th_i^{ch_k} - Th_{i-1}^{ch_k}| < \Delta n_{Th^{Ch_k}}) \rightarrow (Th_i^{ch_k} > Th_i^{ch_k}(min)) \rightarrow Mem(Th_i^{ch_k}) \rightarrow \\ &\rightarrow rec((Channel\_enable) \vee Check(Th_i^{ch_k}) \vee Check((|Th_i^{ch_k} > Th_i^{ch_k}(min)| > \Delta n_{c^k})^{t<T_{to}} : Enable(Ch_k) \end{aligned} \qquad (3.17)$$

При этом состоянию присваивается значение равное $M, (Th_i^{ch_k} \le Th_{i-1}^{ch_k}) \models true;$

В случае постоянного уменьшения пропускной способности *k* –го канала связи в течении всего времени проверки, канала связи признается неактивным (отбрасывается - *Channel _ drop*) и поток данных перераспределяется по другим каналам связи:

$$
\begin{aligned}
&A((Th_i^{ch_k} \le Th_{i-1}^{ch_k} \mid r_i(0)) : Check((Th_i^{ch_k} > Th_i^{ch_k}(min)) \vee (Th_i^{ch_k} \le Th_i^{ch_k}(min)));\\
&(Th_i^{ch_k} \le Th_i^{ch_k}(min)^{t<T_{to}}) : (Channel\_drop(k));\\
&(Th_i^{ch_k} > Th_i^{ch_k}(min)^{t<T_{to}} \mid r_i(1)) : (Channel\_drop(k));\\
&(Th_i^{ch_k} > Th_i^{ch_k}(min)^{t<T_{to}} \mid r_i(0)) : (Mem(Th_i^{ch_k}) \to Enable(Ch_k);
\end{aligned}
\tag{3.18}
$$

Если пропускная способность проверяемого канала связи не изменилась, то истинно следующее утверждение:

$$
(\neg(Th_i^{ch_k} \le Th_{i-1}^{ch_k}))^{t<T_{to}} : (X(Channel\_enable) \to (Check\_Th^{ch_{k+1}})). \tag{3.19}
$$

Проверка таймера проверки и мониторинга каналов связи посредством формулируется следующим образом:

$$
\begin{aligned}
&G((Channel\_enable) \wedge r(t \mid t < T_{to})) \to\\
&\to X(counter\_K{+}{+}) \to X(s_0\_send\_req));\\
&G((Channel\_enable) \wedge r(t \mid t > T_{to})) \to X((Drop\_t) \to (s_0\_send\_req)).
\end{aligned}
\tag{3.20}
$$

Дальнейшие шаги функционирования алгоритма проверки загруженности каналов связи в OpenFlow-сети выполняются рекурсивно

$$
\begin{aligned}
&rec((Check\_Th^{chk})^{t<T_{to},k+1} : ((Th_i^{ch_k} \le Th_{i-1}^{ch_k}) \cup (Th_i^{ch_k} > Th_i^{ch_k}(min) \to Check\_Th^{ch+k});\\
&rec((Th_i^{ch_k} \le Th_{i-1}^{ch_k} \mid r_i(0))^{t<T_{to},k+1} : ((|Th_i^{ch_k} - Th_{i-1}^{ch_k}| < \Delta n_{Th^{Ch_k}}) \to (Th_i^{ch_k} > Th_i^{ch_k}(min)) \to Mem(Th_i^{ch_k}) \to\\
&\to rec((Channel\_enable) \vee Check(Th_i^{ch_k}) \vee Check((|Th_i^{ch_k} > Th_i^{ch_k}(min)| > \Delta n_{c^k})^{t<T_{to}} : Enable(Ch_k)\\
&rec((Th_i^{ch_k} \le Th_{i-1}^{ch_k} \mid r_i(0)) : Check((Th_i^{ch_k} > Th_i^{ch_k}(min)) \vee (Th_i^{ch_k} \le Th_i^{ch_k}(min)));\\
&(Th_i^{ch_k} \le Th_i^{ch_k}(min)^{t<T_{to}}) : (Channel\_drop(k));\\
&(Th_i^{ch_k} > Th_i^{ch_k}(min)^{t<T_{to}} \mid r_i(1)) : (Channel\_drop(k));\\
&(Th_i^{ch_k} > Th_i^{ch_k}(min)^{t<T_{to}} \mid r_i(0)) : (Mem(Th_i^{ch_k}) \to Enable(Ch_k);
\end{aligned}
$$

Модель Е-сети, отображающая функционирование алгоритма поиска каналов связи с недостаточной пропускной способностью в соответствии с с предъявляемыми требованиями приведена на рис. 3.9.

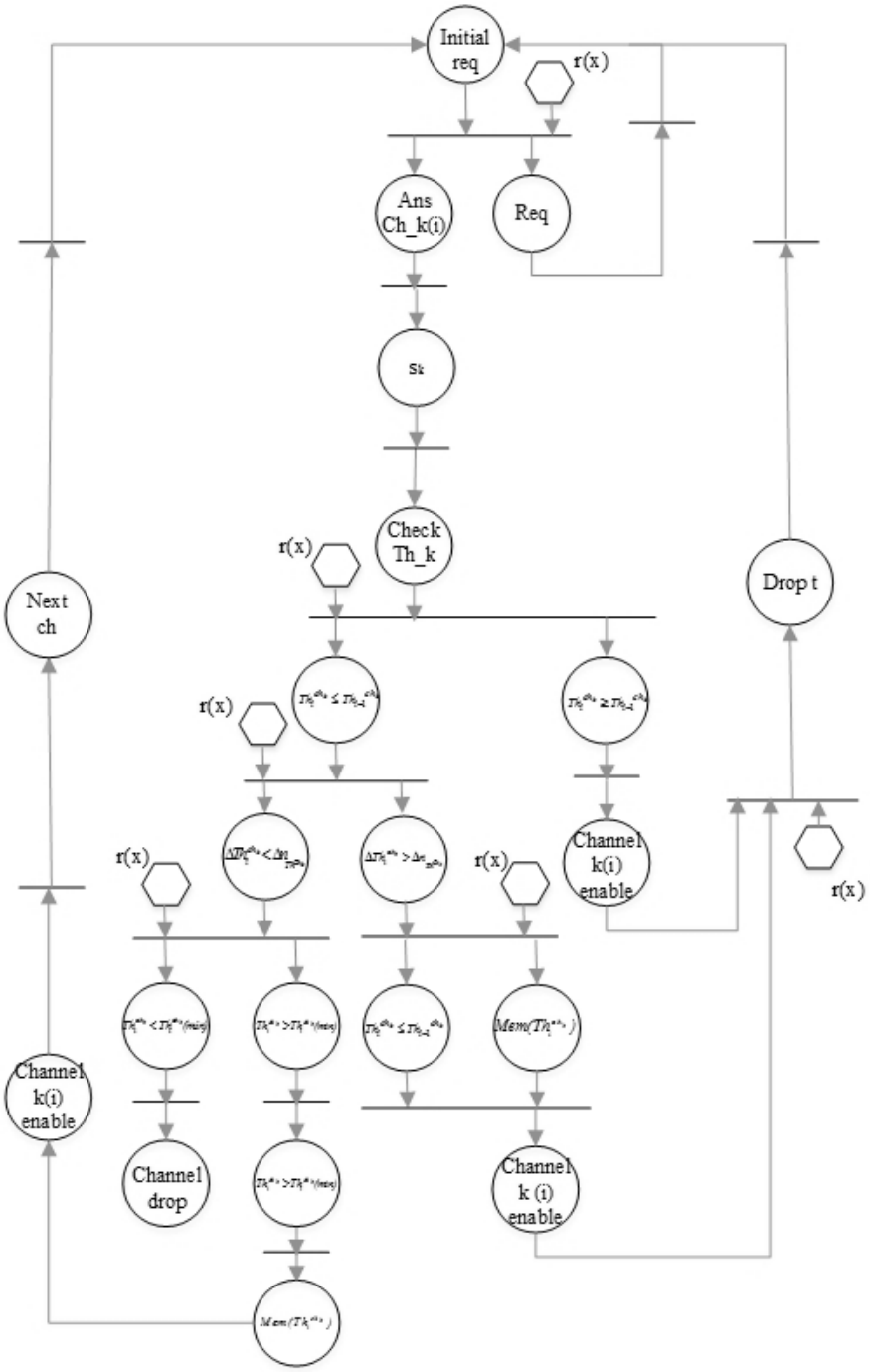


Рис. 3.9. Е-сеть спецификации протокола, определяющего канал связи с достаточной пропускной способностью

Позиции соответствуют следующим состояниям: *Initial request* - начальное состояние алгоритма (определение каналов связи – посылка эхо-запроса); *Ans Ch_k* – получение ответа от *k*-го канала связи; Reg – ответ от канала связи в течении времени T_{to} не получен, генерирование повторного запроса; s_k - предоставлении информации о k-м канале связи; *Check _Th _k(i)* -

начало процедуры определения каналов связи с допустимой пропускной способностью; $Th_i^{ch_k} < Th_{i-1}^{ch_k}, Th_i^{ch_k} > Th_{i-1}^{ch_k}$ - состояния, свидетельствующее об изменении (снижении/увеличении) пропускной способности канала связи $Th_i^{ch_k}$ на i-ом шаге проверки; $Mem(Th_i^{ch})$ - фиксирование значения изменения пропускной способности на i-том и $i-1$ интервале времени; $Th_i^{ch_k} < \Delta n$ - определение скачка изменения пропускной способности; $Th_i^{ch_k} > Th_i^{ch_k}(min)$, $Th_i^{ch_k} < Th_i^{ch_k}(min)$- сравнение текущего значения с минимальным значением; $Channel\_k(i)\_enable$ - занесение канала связи в список каналов с допустимой пропускной способностью; $Channel\_drop$- занесение канала связи в список игнорируемых; $Drop\ Tto$ - сброс счетчика интервала проверки; $Next\_channel$ - проверка следующего канала связи.

В приведенном примере проверка соответствия реализации требованиям спецификации может быть выполнена двумя способами: посредством формальных грамматик и построения дерева достижимости.

Для построения поведенческих цепочек модели спецификации и модели реализации используется язык P-типа [138].

Язык P-типа, определяющий поведение модели спецификации протокола, может быть задан следующим образом:

$$
\begin{aligned}
&P_1 = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}) \mid r(A \mid A=0)), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} < Th_{i-1}^{chk}(min)),\\
&Mem(Th_i^{chk}), Channel\_drop, Initial\_req;\\
&P_2 = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}) \mid r(A \mid A=0)), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)),\\
&Mem(Th_i^{chk}), Channel\_k(i)\_enable, Drop\_t, Initial\_req;\\
&P_3 = Initial\_req, Req(t \mid t > T_{to}), Initial\_req;\\
&P_4 = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}) \mid r(A \mid A=1)), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)),\\
&Mem(Th_i^{chk}) \vee (Th_i^{chk} \le Th_{i-1}^{chk}), Channel\_k(i)\_enable, Drop\_t, Initial\_req;\\
&P_5 = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} > Th_{i-1}^{chk}), Channel\_k(i)\_enable, Drop\_t, Initial\_req;
\end{aligned}
\qquad (3.21)
$$

Анализ языка, определяющего поведение модели спецификации, показал, что в рамках спецификации существует два варианта поведения протокола, соответствующие решению поставленной задачи.

$$Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}) \mid r(A \mid A=0)), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)),$$
$$Mem(Th_i^{chk}), Channel\_k(i)\_enable, Drop\_t, Initial\_req$$
$$\cup$$
$$Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}) \mid r(A \mid A=1)), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)),$$
$$Mem(Th_i^{chk}) \vee (Th_i^{chk} \le Th_{i-1}^{chk}), Channel\_k(i)\_enable, Drop\_t, Initial\_req$$

Язык *P*-типа для приведенной на рис.3.10 модели реализации протокола может быть представлен следующим образом:

$$\begin{aligned}
&P_1 = Initial_req, Ans_Ch_k(i), sk, Check_Th^k(Th_i^{chk} < Th_{i-1}^{chk}) \mid r(A \mid A=0)), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} < Th_{i-1}^{chk}(min)),\\
&Mem(Th_i^{chk}), Channel_drop, Initial_req;\\
&P_2 = Initial_req, Ans_Ch_k(i), sk, Check_Th^k(Th_i^{chk} < Th_{i-1}^{chk}) \mid r(A \mid A=0)), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)),\\
&Mem(Th_i^{chk}), Channel_k(i)_enable, Initial_req;\\
&P_3 = Initial_req, Req(t \mid t > T_{to}), Initial_req;\\
&P_4 = Initial_req, Ans_Ch_k(i), sk, Check_Th^k(Th_i^{chk} < Th_{i-1}^{chk}) \mid r(A \mid A=1)), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)),\\
&Mem(Th_i^{chk}), Channel_k(i)_enable, Initial_req;\\
&P_5 = Initial_req, Ans_Ch_k(i), sk, Check_Th^k(Th_i^{chk} < Th_{i-1}^{chk}) \mid r(A \mid A=1)), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} \le Th_{i-1}^{chk}),\\
&Channel_k(i)_enable, Initial_req;\\
&P_6 = Initial_req, Ans_Ch_k(i), sk, Check_Th^k(Th_i^{chk} > Th_{i-1}^{chk}), Channel_k(i)_enable, Initial_req.
\end{aligned} \tag{3.22}$$

Для установления соответствия между языком модели спецификации и модели реализации предлагается использовать формулу 3.5. При последовательном сравнении цепочек были следующие фрагменты, которые содержат расхождения:

$$(\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} \le Th_{i-1}^{chk}), Channel\_k(i)\_enable;$$

$$(Th_i^{chk} > Th_{i-1}^{chk}(min)), Mem(Th_i^{chk}) \vee (Th_i^{chk} \le Th_{i-1}^{chk}), Channel\_k(i)\_enable, Drop\_t;$$

Обнаружение неэквивалентности реализации протокола и его спецификации говорит о том, что реализация протокола не соответствует техническому заданию, однако не показывает способы решения данной проблемы.

Построение контрпримеров позволяет решить данную задачу. В соответствии с правилами, указанными в [126, 138], могут быть сформированы следующие контрпримеры:

$$P_{контр1} = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}) | r(A | A = 0)), (\Delta Th_i^{Chk} < \Delta n_{c^k}),$$
$$(Th_i^{chk} > Th_{i-1}^{chk}(min)), Mem(Th_i^{chk}) + \varnothing;$$
$$P_{контр2} = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}) | r(A | A = 1)), (\Delta Th_i^{Chk} < \Delta n_{c^k}),$$
$$(Th_i^{chk} > Th_{i-1}^{chk}(min)) + (Th_i^{chk} \le Th_{i-1}^{chk}), Channel\_k(i)\_enable.$$

Данные контрпримеры позволяют установить, что в модели реализации алгоритма отсутствуют состояния $Th_i^{chk} \le Th_{i-1}^{chk}$, *Next Ch* и *Drop_t*.

При применении предлагаемого метода верификации построены следующие деревья достижимости для модели реализации (рис. 3.8) и модели спецификации (рис. 3.9) алгоритма. Дерево достижимости процедуры определения каналов связи с допустимой пропускной способностью приведено на рис.3.10.

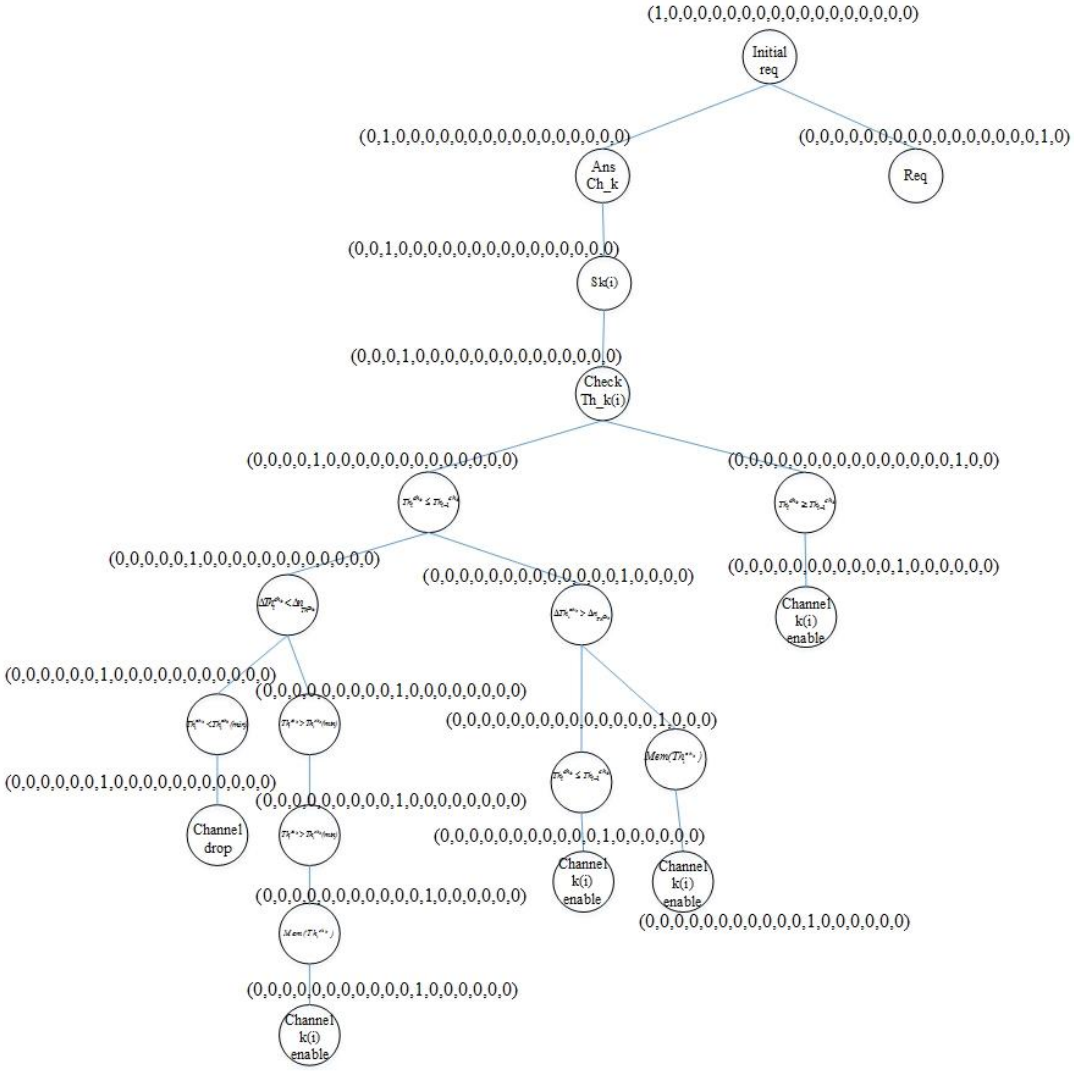


Рис. 3.10. Дерево достижимости процедуры определения каналов связи с допустимой пропускной способностью

Для дерева достижимости, приведенного на рис.3.12 начальным состоянием является *Initial req*, начальной маркировкой - *M(1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)*. Терминальными состояниями являются состояния*Channel drop, Channel k(i) enable, Req.*

Ветви дерева достижимости, включающие множество состояний, приводящих из начальногосостояния*Initial req* к заключительным, может быть представлено следующим образом*:*

$$
\begin{aligned}
&\overrightarrow{M(s_0,T)_1} = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} < Th_{i-1}^{chk}(min)),\\
&Mem(Th_i^{chk}), Channel\_drop;\\
&\cup\\
&\overrightarrow{M(s_0,T)_2} = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k, (Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)),\\
&Mem(Th_i^{chk}), Channel\_k(i)\_enable\\
&\cup\\
&\overrightarrow{M(s_0,T)_3} = Initial\_req, Req\\
&\cup\\
&\overrightarrow{M(s_0,T)_4} = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k, (Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)),\\
&Mem(Th_i^{chk}), Channel\_k(i)\_enable\\
&\cup\\
&\overrightarrow{M(s_0,T)_5} = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} \le Th_{i-1}^{chk}), Channel\_k(i)\_enable
\end{aligned}
\qquad (3.23)
$$

Дерево достижимости процедуры определения каналов связи с допустимой пропускной способностью в соответствии с требованиями спецификации приведено на рис. 3.10.

Для дерева достижимости модели спецификации, приведенного на рис.3.11 начальным состоянием является *Initial req*, начальная маркировка - *M(1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)*. Терминальными состояниями при этом являются состояния*Channel drop, Channel k(i) enable, Next Ch, Drop t, Req.* В процессе достижения приведенных терминальных состояний из начального состояния *Initial req* сформированы следующие ветви дерева*:*

$$
\begin{aligned}
&\overrightarrow{M(s_0,T)_1} = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} < Th_{i-1}^{chk}(min)),\\
&Mem(Th_i^{chk}), Channel\_drop, Next\_channel\\
&\cup\\
&\overrightarrow{M(s_0,T)_2} = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)),\\
&Mem(Th_i^{chk}), Channel\_k(i)\_enable, Next\_channel\\
&\cup\\
&\overrightarrow{M(s_0,T)_3} = Initial\_req, Req
\end{aligned}
$$

$$\cup$$

$$\overrightarrow{M(s_0,T)_4} = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)),$$

$$Mem(Th_i^{chk}), Channel\_k(i)\_enable, Next\_channel \quad (3.24)$$

$$\cup$$

$$\overrightarrow{M(s_0,T)_5} = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} \le Th_{i-1}^{chk}),$$

$$Channel\_k(i)\_enable, Drop\_t$$

$$\cup$$

$$\overrightarrow{M(s_0,T)_6} = Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} > Th_{i-1}^{chk}), Channel\_k(i)\_enable, Drop\_t$$

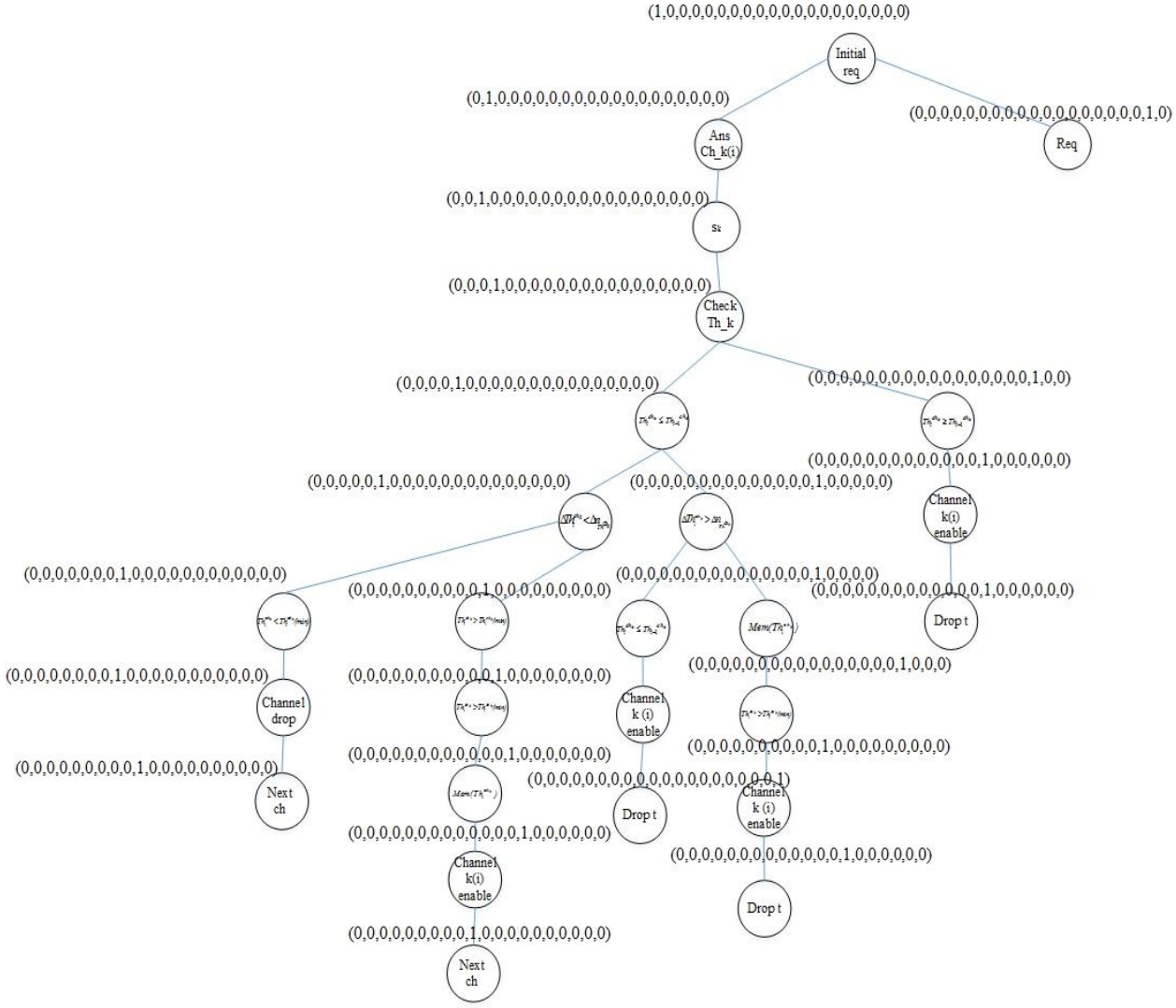


Рис. 3.11. Дерево достижимости спецификации протокола, определяющего канал связи с достаточной пропускной способностью

В процессе анализа ветвей дерева достижимости были сформированы следующие последовательности состояний, соответствующие определению ненагруженных каналов связи:

$$Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)),$$
$$Mem(Th_i^{chk}), Channel\_k(i)\_enable, Next\_channel$$
$$\cup$$
$$Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)), \quad (3.25)$$
$$Mem(Th_i^{chk}), Channel\_k(i)\_enable, Next\_channel$$

$$Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} \le Th_{i-1}^{chk}),$$
$$Channel\_k(i)\_enable, Drop\_t \quad (3.26)$$
$$\cup$$
$$Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k(Th_i^{chk} > Th_{i-1}^{chk}), Channel\_k(i)\_enable, Drop\_t$$

Ветви дерева достижимости (3.24) содержат последовательность состояний, которые приводят к определению ненагруженных каналов связи, ветви (3.25) содержат последовательность действий установления ненагруженных каналов связи, которые требуют дополнительной проверки по истечению времени T_{to}.

В процессе проверки соответствия были обнаружены следующие расхождения:

- множество состояний модели реализации не совпадает с множеством состояний модели спецификации. Состояниями, которые отсутствуют или недостижимы в модели реализации являются: *Next Ch, Drop t,* $Th_i^{ch_k} \le Th_{i-1}^{ch_k}$;

- в ветвях дерева достижимости модели реализации содержаться последовательности состояний, которые имеют отличия от ветвей модели требований спецификации:

$(\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} \le Th_{i-1}^{chk}), Channel\_k(i)\_enable$;

$(Th_i^{chk} > Th_{i-1}^{chk}(min)), Mem(Th_i^{chk}) \vee (Th_i^{chk} \le Th_{i-1}^{chk}), Channel\_k(i)\_enable, Drop\_t$;

При построении контрпримера в соответствии с правилами, приведенными в п.3.2.3 место (состояние) возникновения расхождений, а также сформированы следующие цепочки:

$$P_{контр1} = (Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k + (Th_i^{chk} < Th_{i-1}^{chk}) + (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)), Mem(Th_i^{chk})) || (Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k + (Th_i^{chk} \le Th_{i-1}^{chk}) + (\neg(\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)), Mem(Th_i^{chk}));$$

$$P_{контр2} = (Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k (Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)) + (Th_i^{chk} \le Th_{i-1}^{chk}) + Channel\_k(i)\_enable) || (Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k (Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)) + (Th_i^{chk} \le Th_{i-1}^{chk}) + \varnothing); \quad (3.27)$$

$$P_{контр3} = (Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k (Th_i^{chk} > Th_{i-1}^{chk}), Channel\_k(i)\_enable) || (Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k (Th_i^{chk} > Th_{i-1}^{chk}), Channel\_k(i)\_enable + Drop\_t + \varnothing.$$

Вывод порождающей формальной грамматики и контпримеры, построенные на основе дерева достижимости частично совпадают. Контрпример, построенный для деревьев достижимости имеет ряд расширений: вывод дополнительных последовательностей, приводящих к терминальным состояниям, но не имеющий строгих ограничений.

Последовательности, приведенные на основе правил формальных грамматик, имеют следующее преимущество: они позволяют формализовать условие срабатывания переходов. Однако при этом происходит отбрасывание цепочек, которые потенциально могут содержать ошибку.

3.3.1 Верификация процесса формирования канала связи в сетях, функционирующих на основе протокола OpenFlow

В беспроводных сетях связи, функционирующих на основе протокола OpenFlow, при формировании канала связи между пользователем A (который подключен к коммутатору Sw1) и пользователем B (который подключен к коммутатору Sw2) может возникнуть ошибка установления соединения, связанная с настройкой фиксированного тайм-аута между рассылкой управляющих сообщений [2, 3].

В соответствии с требованиями протокола OpenFlow правила передачи фиксируются в таблицах переадресации коммутаторов. Обновление таблиц переадресации, с целью проверки их актуальности, происходит через фиксированный временной интервал. В случае изменения своего местоположения пользователя, новый коммутатор Sw3, к которому в текущий

момент подключен пользователь, генерирует запрос контроллеру. Контроллер обрабатывает запрос и формирует новое управляющее сообщение, которое в последующем передает коммутатору Sw1 и Sw3.

Однако в некоторых реализациях коммутаторов [16, 17], изменение записей в таблице переадресации происходит только по истечению тайм-аута. До истечения тайм-аута коммутатор Sw1 продолжает передавать данные по на определенный порт коммутатора Sw2, что приводит к потере данных.

При этом правило переадресации не изменяется пока, не будет получено новое управляющее сообщения от контролера. Правило переадресации остается в коммутаторе тех пор, пока тайм-аут не истечет. Таким образом, пакеты не доходят до оконечного узла. По истечению тайм-аута генерируется новое управляющее сообщение, при этом определяется новое положение оконечного устройства пользователя B. Структурная схема процесса переадресации приведена на рис. 3.12.

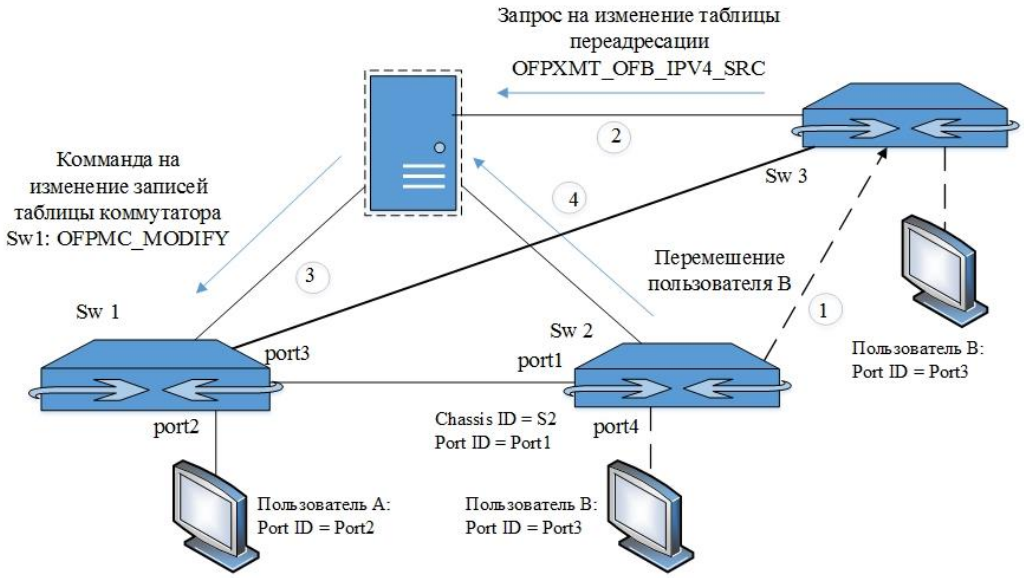


Рис. 3.12. Процесс переадресации сообщений при перемещении пользователя B от коммутатора Sw2 к коммутатору Sw3

Возможна повторная передача сообщений при формировании нового управляющего потока. Однако, при предоставлении онлайн сервисов или приложений (например, потокового видео, skype трансляции) восстановление

потерянных пакетов невозможно. Потеря пакетов в сети наблюдается до тех пор, пока не истечет тайм-аут.

При обнаружении такого типа ошибок E-сеть модели реализации и спецификации протокола (в данном случае процесса изменения таблицы переадресации) должна содержать управляющее место, атрибутом которого является значение времени тайм-аута.

Таблица модификации потока сообщений, в соответствии с спецификацией протокола OpenFlow v.1.3.0 может иметь следующий набор команд [70]:

```
enum ofp_flow_mod_command {
OFPFC_ADD (OFPFF_CHECK_OVERLAP) = 0, /* New flow. */
OFPFC_MODIFY = 1, /* Modify all matching flows. */
OFPFC_MODIFY_STRICT = 2, /* Modify entry strictly matching wildcards and priority. */
OFPFC_DELETE = 3, /* Delete all matching flows. */
OFPFC_DELETE_STRICT = 4, /* Delete entry strictly matching wildcards and priority. */ }
```

При этом команды OFPFC_MODIFY_STRICT и OFPFC_DELETE_STRICT могут быть отфильтрованы по значению cookie, если поле содержит cookie_mask, отличное от нуля: (flow entry.cookie & flow mod.cookie mask) == (flow mod.cookie & flow mod.cookie mask) &value(cookie_mask)!=0;

Требования, относящиеся к модификации таблиц переадресации, могут быть представлены с помощью формализмов ACSR следующим образом:

- проверка наличия перекрытия и конфликта между пришедшим сообщением и сообщением, содержащимся в базе данных:

$$OVERLAP\_CHECK(i):OFPFC\_ADD\left(OFPFF\_CHECK\_OVERLAP, true\right)\rightarrow \\ \longrightarrow Deny\_Request(i):ofp\_error\_msg(i) \quad , \quad (3.28)$$

- добавление записи в таблицу переадресации, если перекрытие не обнаружено. Сброс таймера OFPFF_RESET_COUNTS:

$$OVERLAP\_CHECK(i):OFPFC\_ADD\left(OFPFF\_CHECK\_OVERLAP, false\right)\rightarrow \\ \longrightarrow Add\_entry(i):OFPFF\_RESET\_COUNTS(Table) \quad , \quad (3.29)$$

Отправление запроса на изменение направления потока контролеру.

$$OVERLAP\_CHECK(i):OFPFC\_ADD\left(OFPFF\_CHECK\_OVERLAP,\ false\right)\rightarrow$$
$$\longrightarrow OFPET\_FLOW\_MOD(i)\longrightarrow Add\_entry(i):OFPFF\_RESET\_COUNTS(Table);$$

После сброса таймера (OFPFF_RESET_COUNTS) происходит обновление полей куки, тайм-аут, флагов, счетчиков, продолжительности действия:

$$OFPFF\_RESET\_COUNTS(Table(i))\rightarrow(cookie(i):reset)\cup(idle\ timeout(i):reset)$$
$$\cup(hard\ timeout(i):reset)\cup(flags(i):reset)\cup(counters(i):reset).$$

- удаление записи. Запись может быть удалена без подтверждения или удалена с подтверждением. Если в поле OFPFF_SEND_FLOW_REM содержится метка, то должен быть сгенерирован запрос на удаление сообщения:

$$(OFPFF\_SEND\_FLOW\_REM,\ true)\rightarrow Crete\_mes\_ofp\_delete(i). \qquad (3.30)$$

При использовании команды OFPFC_DELETE_STRICT обязательно фильтруется информация о получателе сообщения и значение выходного порта (out_port). Таким образом, выполняется полная сверка полей таблицы переадресации:

$$OFPFC\_DELETE\_STRICT(i):(entry(i)=exist\_table\_entry(j))\cup$$
$$(field\_entry(i)==table\_entry(j))\rightarrow Delete\_entry(i);$$

Удаление записей в таблице может также осуществляться на основании значения куки:

$$OFPFC\_DELETE\_STRICT(i):(entry.cookie(i)\cup flowmod.cookiemask)=$$
$$=(flowmod.cookie(i)\cup flowmod.cookiemask))\rightarrow Delete\_cookie(i)$$

- обновление таблиц переадресации в случае тайм-аута. По истечению времени, указанному в поле тайм-аут, коммутатор посылает повторный запрос контролеру. Управляющее сообщение от контролера, которое получено в ответ на запрос, может влиять на изменение записей (конечного получателя, порта, т.д.):

$$OFPFC\_MODIFY(i)^{Time\_out} : Sent\_mes\_flow\_modify(i) \rightarrow \\ \rightarrow receive\_MessegeFlowMode(i) \rightarrow\rightarrow (Delete\_entry(i) \cup modify\_entry(i)), \quad (3.31)$$

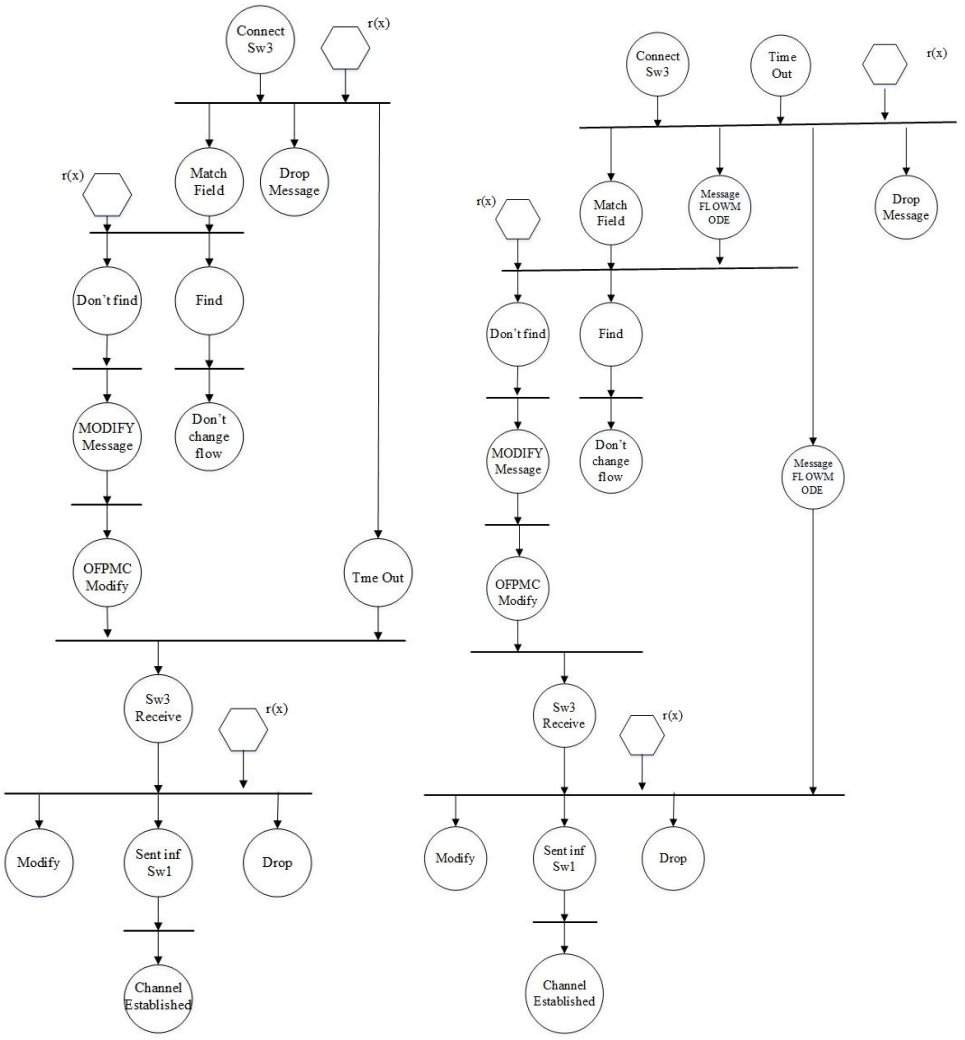
- изменение полей таблицы происходит только в случае получения управляющего сообщения от контролера:

$$rec(MessegeFlowMode(i) \perp (OFPMC\_Modify(Modify(i) \cap Delete(i)). \quad (3.32)$$

Модель Е-сети, которая соответствует процессу обновления переадресации при смене местоположения оконечного пользователя таблиц для фрагмента сети, представленного на рис.3.14 приведена на рис. 3.15 а). Модель Е-сети, которая соответствует требованиям спецификации OpenFlow v.1.3.0, соответствующим процессу обновления таблиц переадресации, приведена на рис. 3.15 б).

Состояние *Connect Sw3* является начальным состояние функционирования фрагмента протокола и соответствует смене местоположения пользователя В (подключению к коммутатору Sw3). При этом происходит сверка полей (состояние *Match_Field*) таблицы переадресации коммутатора Sw3 и пришедшего от пользователя В пакета либо отбрасывание пакета – *Drop Message*. Состояние *Find* соответствует нахождение соответствия в таблице переадресации, состояние *Don't change flow* соответствует автоматическому перенаправлению потока данных. Состояние *Don't find* соответствует отсутствию совпадений мета-данных поле сообщения, коммутатор принимает решение о необходимости изменения сообщения/обработки контролером - *MODIFY Message*. Состояние *OFPFC_modify* соответствует добавлению в таблицу переадресации и модификации полей пакетов, принадлежащих данному потоку, *Time Out* – истечение времени жизни записи в таблице переадресации, соответствующей данному потоку. Состояние *Receive Sw3* соответствует получению коммутатором управляющего сообщения от контролера и изменению таблицы

переадресации (*Modify*), отбрасыванию пакетов пользователя B (состояние *Drop*) и передачи информации о изменении потока коммутатору Sw1 (состояние *Sent inf Sw1*). После того, как данные таблиц переадресации коммутатора Sw1 и Sw3 изменены, происходит формирование нового потока и установление канала связи (*Channel Established*).



а) б)

Рис. 3.13. E-сеть модели реализации (а), E-сеть модели спецификации (б)

Начальным состоянием модели спецификации также является состояние *Connect Sw3*, однако в требованиях спецификации введено дополнительное состояние *Time Out*, которое соответствует периодическому обновлению таблиц переадресации и, следовательно, формированию запросов об изменении топологии сети/смене режима передачи данных (*Message FLOWMODE*).

Дерево достижимости для модели реализации приведено на рис. 3.16 для модели спецификации - на рис. 3.17.

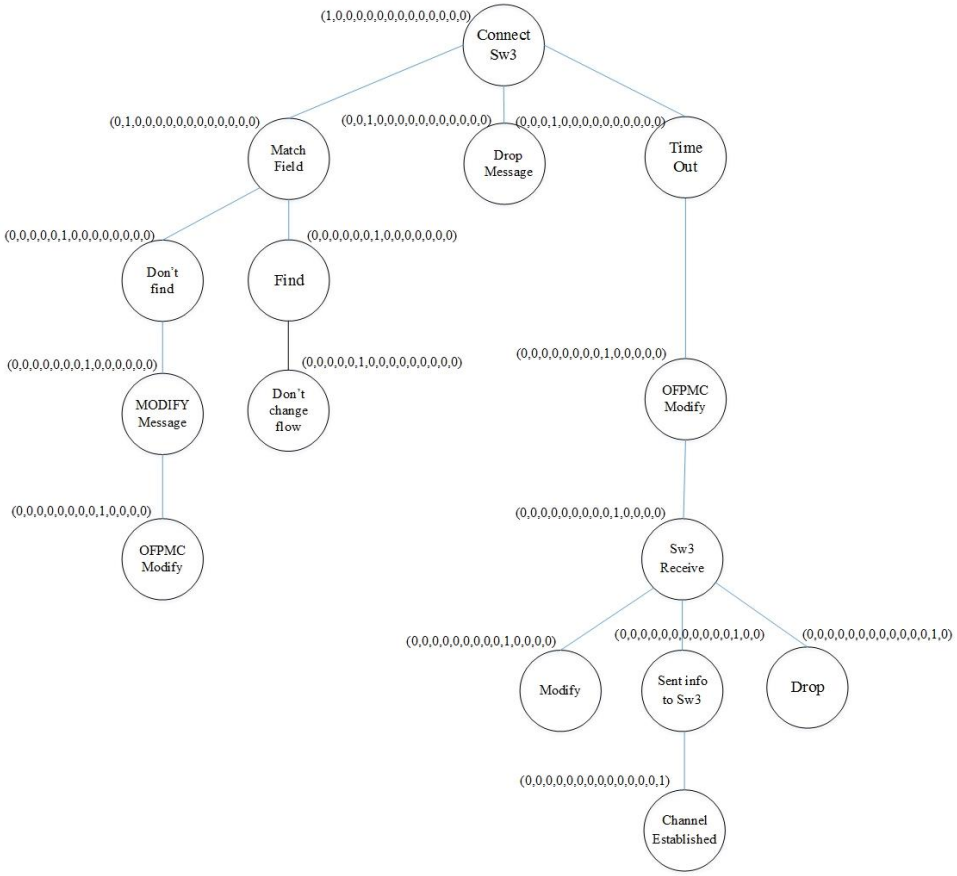


Рис. 3.14. Дерево достижимости модели реализации для процесса изменения канала связи с помощью протокола OpenFlow

Для дерева достижимости, приведенного на рис.3.14, могут быть построены следующие цепочки, приводящие к терминальным состояниям:

$$M(T_f)_{реализация} = \begin{cases} (ConnectSw3), Drop\_Message \\ (ConnectSw3), Match\_field, Find, Don't\_Change\_flow; \\ (ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, OFPMC\_Modify; \\ (ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, Sw3\_Receive, Drop; \\ (ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, Sw3\_Receive, \\ Sent\_inf\_Sw3, Channel\_Established; \\ (ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, Sw3\_Receive, Modify; \end{cases} \quad (3.33)$$

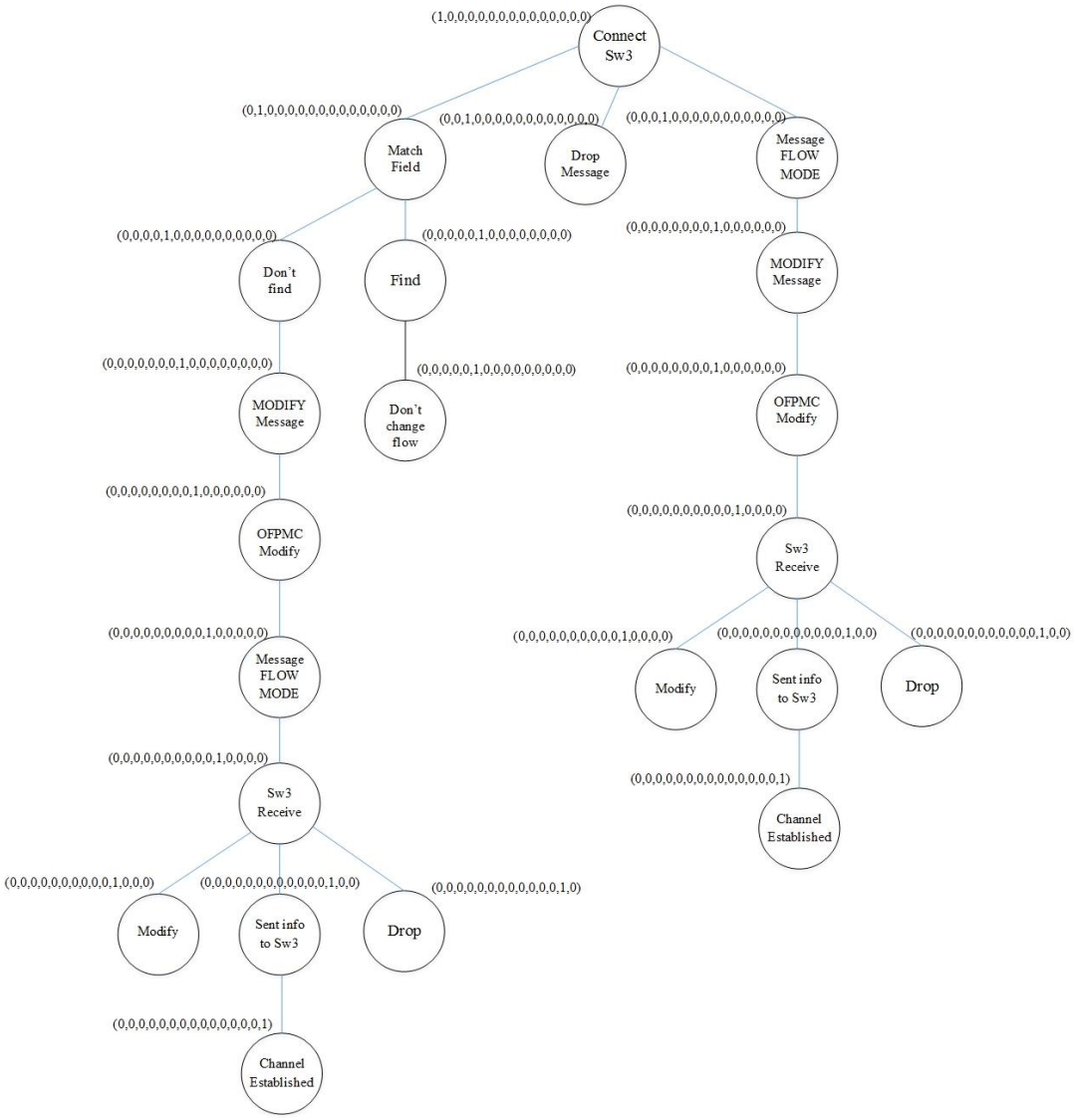


Рис. 3.15. Дерево достижимости модели спецификации для процесса изменения канала связи с помощью протокола OpenFlow

Для дерева достижимости, приведенного на рис.3.15, могут быть построены следующие цепочки, приводящие к терминальным состояниям:

$$M(T_f)_{спецификация} = \begin{cases} (ConnectSw3), Drop\_Message \\ (ConnectSw3), Match\_field, Find, Don't\_Change\_flow; \\ (ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, OFPMC\_Modify; \\ (ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, Sw3\_Receive, OFPMC\_Modify, \\ Message\_FLOWMODE, Drop; \\ (ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, OFPMC\_Modify, \\ Message\_FLOWMODE, Sw3\_Receive, Sent\_inf\_Sw3, Channel\_Establised; \\ (ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, OFPMC\_Modify, \\ Message\_FLOWMODE, Sw3\_Receive, Modify; \end{cases} \quad (3.34)$$

В результате проверки соответствия множества ветвей дерева достижимости, приходящих к терминальным состояниям, модели реализации и модели спецификации формирования канала связи между конечным пользователем A и пользователем B, обнаружены следующие расхождения:

$(ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, OFPMC\_Modify,$

$(ConnectSw3), Time\_Out$

$(ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, Sw3\_Receive, OFPMC\_Modify, Message\_FLOWMODE$

$(ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, OFPMC\_Modify, Message\_FLOWMODE,$,

$(ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, OFPMC\_Modify, Message\_FLOWMODE,$.

Состояния $Message\_FLOWMODE$ и $OFPMC\_Modify$ только модели спецификации. Они указывают на несоответствие реализации процесса формирования канала связи требованиям спецификации. В соответствии с правилами построения контрпримеров, предложенными в п. 3., могут быть построены следующие цепочки:

$$P_{контр} = (ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, Sw3\_Receive + \varnothing, \quad (3.35)$$

последовательность состояний принадлежит модели реализации;

$$P_{контр} = (ConnectSw3) + Time\_Out, \quad (3.36)$$

последовательность состояний принадлежит модели реализации;

$$\begin{aligned} P_{контр} = {} & ((ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, OFPMC\_Modify, \\ & Message\_FLOWMODE, Sw3\_Receive, Drop) \cup ((ConnectSw3), Match\_field, Don't\_Find, \\ & MODIFE\_Message, OFPMC\_Modify, Message\_FLOWMODE, Sw3\_Receive, Modify) \cup \\ & \cup ((ConnectSw3), Match\_field, Don't\_Find, MODIFE\_Message, OFPMC\_Modify, \\ & Message\_FLOWMODE, Sw3\_Receive, Sent\_inf\_Sw3) \end{aligned} \quad (3.37)$$

совокупность последовательностей состояний, принадлежащих модели реализации.

Построенные контрпримеры указывают на то, что состояния $OFPMC\_Modify,Message\_FLOWMODE$, которые соответствуют требованиям спецификации, не совпадают с состояниями модели реализации, что приводит к необходимости модификации процесса формирования канала связи (применение оборудования, поддерживающего одинаковые версии протокола OpenFlow не младше1.3.0), и повторной верификации.

3.4 Верификация OpenFlow Discovery Protocol

Для эффективного управления и предоставления услуг в сетях, построенных на основе концепции SDN контролер должен иметь актуальную информацию о состоянии и топологии сети. Таким образом, механизм определения сетевой топологии имеет решающее значение. Протокол определения сетевой топологии посредством протокола OpenFlow (OpenFlow Discovery Protocol, OFDP) на сегодняшний день не стандартизирован. Однако большинство сетевых операционных систем (NOX, RYU, Beacon) поддерживают сообщения формата OFDP. Его функциональной основой является Link Layer Discovery Protocol [5, 33, 40].

В функциональные обязанности контроллера не входит генерирование запросов определения топологи. OpenFlow коммутаторы инициализируют соединение с контролером посредством сообщений packet_in, что дает контролеру посную информацию о элементах сети. В случае добавления новых узлов коммутатор также посылает сообщение об изменении структуры сети.

Традиционный LLDP, как правило, реализуется с помощью Ethernet коммутаторов. Они посылают и получают пакеты LLDP [51]. LLDP пакеты генерируются через определенный промежуток времени и передаются через все порты коммутатора. При этом коммутатор хранит информацию о соседних устройствах в MIB, но не перенаправляет её дальше. Любой LLDP кадр должен

содержать три обязательных записей: chassis ID (идентификатор шасси), port ID (идентификатор порта), time to live (предписанное время жизни)

Функционирование OFDP несколько отличается от LLDP. Учитывая ограниченный набор действий OpenFlow коммутатора (коммутатор не может самостоятельно отправлять и модифицировать новые входящие (packet_out) сообщения, в том числе и OFDP), распространение информации о сетевой топологи выполняется контроллером. Процесс определения сетевой топологи приведен на рис. 3.16.

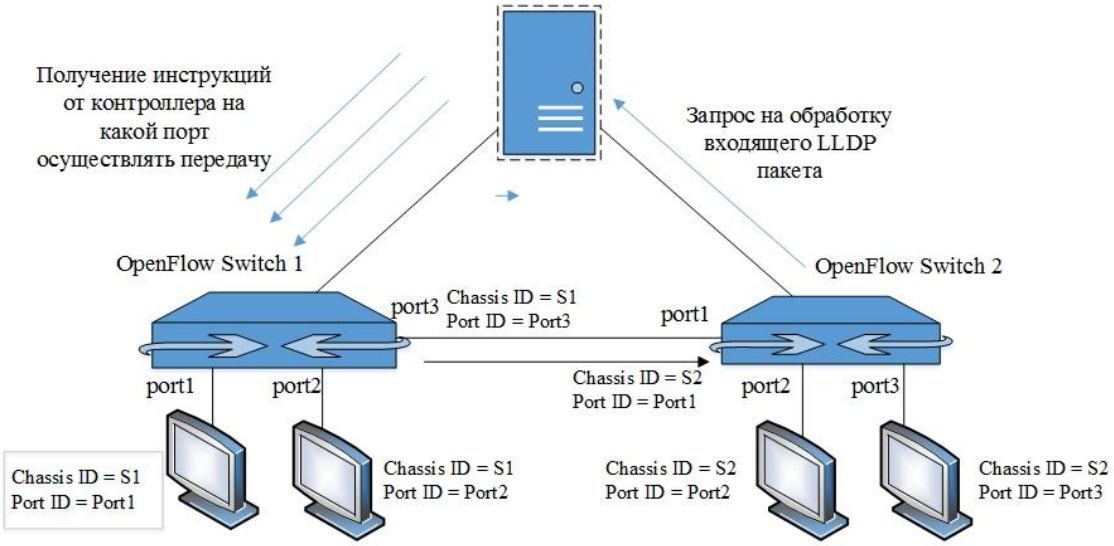


Рис. 3.16. Механизм установления сетевой топологи посредством OFDP

Изначально SDN контролер формирует индивидуальный LLDP пакет для каждого порта коммутатора S1: для порта 1 (Chassis ID = S1 Port ID = Port1), для порта 2 (Chassis ID = S1 Port ID = Port2), для порта 3 (Chassis ID = S1 Port ID = Port3). После того, как LLDP пакеты сформированы и разосланы, осуществляется сбор информации о топологии сети. Например, коммутатор 1 (S1) через порт 3 посылает широковещательный запрос на обнаружение соседних сетевых устройств. Коммутатор 2 (Sw2) получает данный запрос на порт 1. Далее Sw2 перенаправляет полученный от Sw1 пакет на обработку контроллеру. Поля пакета содержат следующую информацию: мета-данные, такие как ID коммутатора и входного порта (Chassis ID = S1 Port ID = Port3), через который был получен пакет, время жизни, тип полезной нагрузки, версию протокола OpenFlow. На основании полученной информации, контроллер

делает вывод, что существует связь между (S1, порт 2) и (S2, port1) и добавляет информацию об изменении топологии в MIB. Таким образом, происходит сбор полной актуальной информации о сетевой топологи.

Спецификация OFDP не стандартизирована, при формировании требований к протоколу большинство разработчиков оборудования OpenFlow основываются на требованиях LLDP.

Основные требования LLDP могут быть заданы следующими формализмами алгебры коммуникационных распределенных ресурсов:

- протоколом LLDP предусматривается передача данных только в одном направлении. То есть LLDP-устройства не обмениваются информацией в режиме запрос–ответ, а также не подтверждают ее получение:

$$C : Send(LLDP\_pack_i) \rightarrow (Sw : Receive(LLDP\_pack_i) \cup Sw : \neg Reply(LLDP\_pack_i)); \quad (3.38)$$

$$Sw : Send(LLDP\_pack_i) \rightarrow (C : Receive(LLDP\_pack_i) \cup Sw : \neg Reply(LLDP\_pack_i));$$

- каждый LLDP-пакет должен содержать четыре обязательных TLV:
- chassis ID TLV: идентифицирует шасси устройств, MAC-адресс устройства;
- port ID TLV: идентифицирует порт, через который передается LLDP-пакет;
- TTL TLV: указывает отрезок времени в секундах, в течение которого полученная информация актуальна (end of TLV: определяет конец TLV);
- LLDP-пакеты передаются периодически и сохраняются в течение определенного времени. Рекомендованная IEEE частота передачи составляет 30с, но TLV может регулироваться в зависимости от требований компаний-разработчиков.

$$Sw : Receive(LLDP\_pack_i) \rightarrow (C : Store(LLDP\_pack_i))^{t>x} : delete \mid x = 30s; \quad (3.39)$$

Оборудование хранит полученные данные в информационной базе MIB.

$$C : Send(LLDP\_pack_i) \rightarrow (Sw : Receive(LLDP\_pack_i)) : (Sw : MIB_{sw}(LLDP\_pack_i));$$

$$Sw : Send(LLDP\_pack_i) \rightarrow (C : Receive(LLDP\_pack_i)) : (C : MIB_{sw}(LLDP\_pack_i)). \quad (3.40)$$

Она актуальна в течение отрезка времени, определяемого значением поля Time to Live (TTL):

$$((MIB : (LLDP\_pack_i)^{t<x} : store) \cup ((MIB : (LLDP\_pack_i)^{t>x} : Erase\&Modify). \quad (3.41)$$

В качестве исходных данных выступает модель Е-сети, приведенная на рис 3.17. Начальным состоянием модели реализации процесса установления сетевой топологии является состояние *Sw1,p1*, которое соответствует формированию и рассылке широковещательного запроса всему сетевому оборудованию, поддерживающему протокол OpenFlow и находящемуся в зоне обслуживания данного контроллера, состояния *Sw2,p3… Swi, pi* соответствует получению запроса от Sw1,p1 и его дальнейшей обработке, состояние *Pac_in* – свидетельствует о перенаправлении пришедшего от Sw1 пакета контроллеру (при отсутствии совпадении с имеющимися записями в таблице переадресации), состояние *Mod* соответствует модификации записи в таблице переадресации, состояние *Del* – удалению записи, содержащей параметры пришедшего пакета, из таблицы переадресации, состояние *Cont* соответсвует обработке запроса контроллером, состояние *Cont* всегда следует за состоянием*Pac_in*, состояние *MIB* соответствует занесению информации о коммутаторе *Sw1* в базу данных контроллера, состояния *Store* и *Erase&Modify* соответствуют добавлению или удалению и модификации информации в MIB соответственно, состояние *Cont_proc* соответствует успешному формирование управляющего сообщения и передаче его коммутатору, состояние *TVL_end* соответствует истечению времени установленного таймеромTTL TLV, состояние *Reply* соответствует посылке ответного сообщения коммутаторуSw1,p1.

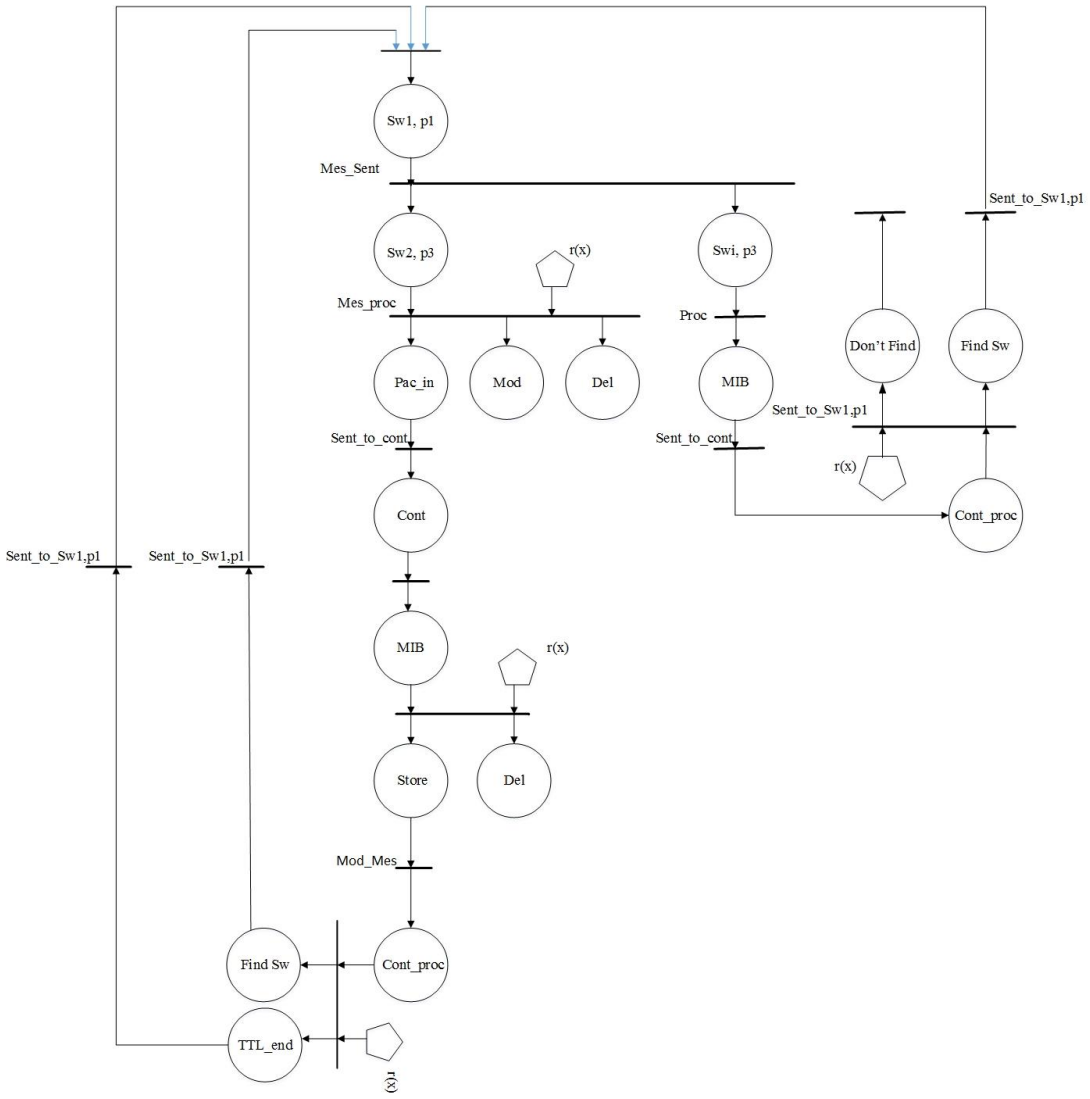


Рис. 3.17. E-сеть модели реализации процесса установления сетевой топологии между коммутатором Sw1, p1 и другим оборудованием сети (Sw2 … Swi)

На основе требований спецификации протокола LLDP сформирован ряд требований, предъявляемых к функционированию протокола OpenFlow при построении топологии сети. Начальным требованием является формирование и посылка одним коммутатором сети (Sw1) широковещательного запроса другим OpenFlow коммутаторам:

$$Sw1, p1(Mes_i) \xrightarrow{Mes\_send} (Sw_2, p_1) : receive(Mes_i) + ... + (Sw_i, p_3) : receive(Mes_i). \quad (3.42)$$

Коммутатор, получивший сообщение, осуществляет проверку соответствия полей сообщения с записями, уже существующими в таблице

переадресации. Если найдено совпадение поля $Match$ входящего пакета с одной из записей таблиц переадресации. Осуществляется модификация и последующая передача сообщения узлу назначения (destination node):

$$(Sw_2, p_1) : receive(Mes_i) \to (((Sw_2, p_1)^{Match_i = Match_j} : Mod(Mes_i)) : sent(Mes_i)) \cup \\ \cup ((Sw_2, p_1)^{Match_i = Match_j} : Del(Mes_i)) . \quad (3.43)$$

В случае отсутствия совпадений – пакет перенаправляется контроллеру:

$$(Sw_2, p_1) : receive(Mes_i) \to ((Sw_2, p_1)^{Match_i \neq Match_j} : Mod(Mes_i)) : Pac\_in(Mes_i) . \quad (3.44)$$

Контроллер при получении входящего LLDP пакета, заносит данные TLV в MIB, где они хранятся на протяжении определенного времени, по истечению которого происходит удаление или модификация данных:

$$Pac\_in \to Cont : receive(Mes_i) \to MIB ; \\ (MIB^{U>x} : Store) || (MIB^{U<x} : Erase \& Modify) . \quad (3.45)$$

Контроллер модифицирует сообщение и передает данные коммутатору отправителю и коммутатору получателю, учитывая номера исходящих и входящих портов:

$$Cont : Mod(Mes_i) \to Send(Sw1, p1) || Send(Swi, pi) . \quad (3.46)$$

Контроллер формирует управляющее сообщение, которое в последующем передается коммутатору. Если время, установленное таймером TTL истекло, то сообщение удаляется и процедура установления сетевой топологии повторяется заново:

$$(Cont\_proc)^{t<TTL} : Find\_Sw;$$
$$(Cont\_proc)^{t>TTL} : TTL\_end; \quad (3.47)$$

На основе приведенных выше формализмов построен фрагмент Е-сети спецификации процесса установления сетевой топологии (рис. 3.18):

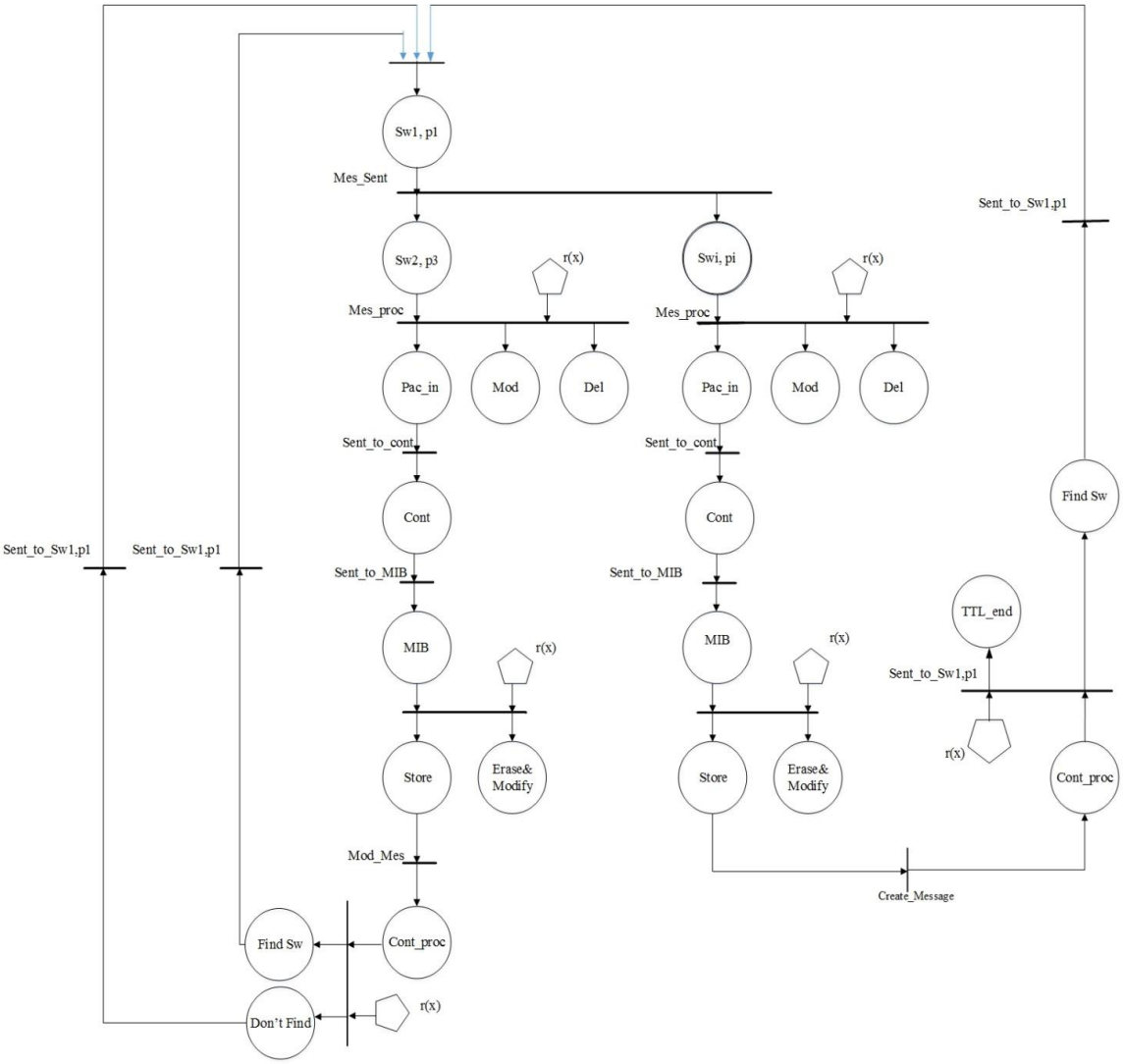


Рис. 3.18. Е-сеть модели спецификации процесса установления сетевой топологии между коммутатором Sw1, p1 и оборудованием сети

Дерево достижимости Е-сети модели реализации процесса установления сетевой топологии между коммутатором Sw1, p1 и другим оборудованием сети приведено на рис. 3.19 и дерево достижимости Е-сети модели спецификации приведено на рис. 3.20.

Начальной вершиной дерева достижимости является состояние *Sw1,p1*, начальная маркировка – *M(1, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)*.

Терминальными вершинами являются следующие состояния *Del, Erase&Modify, TTL_end, Find_Sw.*

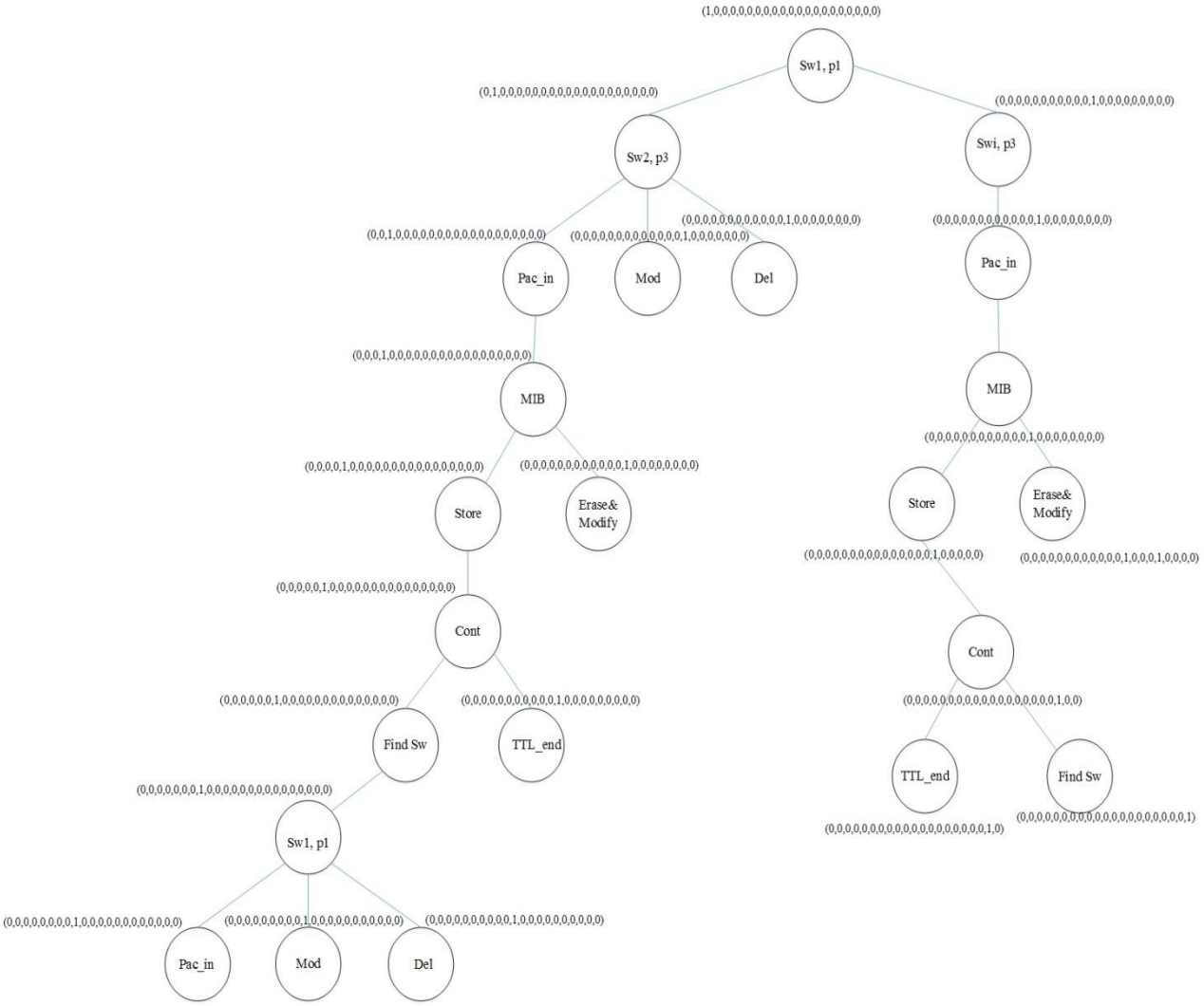


Рис. 3.19. Дерево достижимости модели реализации (а) и модели спецификации (б) процесса установления сетевой топологии

Для дерева достижимости модели реализации могут быть построены следующие цепочки смены состояний:

$$M(T_f)_{реализация} = \begin{cases} (Sw1, p1), (Sw2, p3), Mod; \\ (Sw1, p1), (Sw2, p3), Del; \\ (Sw1, p1), (Sw2, p3), Pac\_in, MIB, Del; \\ (Sw1, p1), (Sw2, p3), Pac\_in, MIB, Store, Cont, FindSw, (Sw1, p1), pac\_in; \\ (Swi, p3), Pac\_in, MIB, Cont, Don't\_Find, (Sw1, p1), pac\_in; \\ (Swi, p3), Pac\_in, MIB, Cont, FindSw, (Swi, p3), pac\_in; \end{cases} \quad (3.48)$$

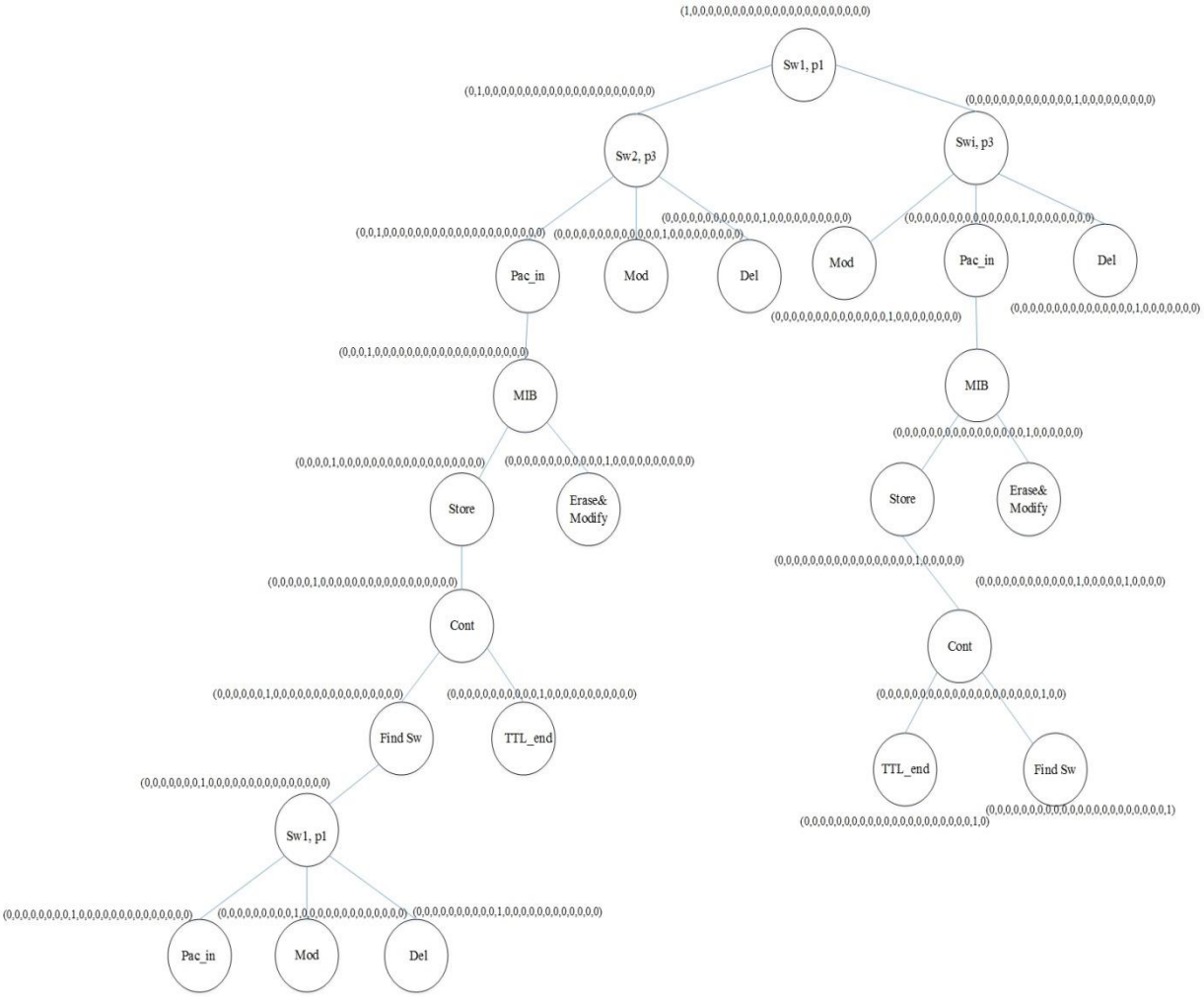


Рис. 3.20. Дерево достижимости модели спецификации процесса установления сетевой топологии

Для дерева достижимости модели спецификации могут быть получены следующие цепочки смены состояний:

$$M(T_f)_{спецификация} = \begin{cases} (Sw1,p1),(Sw2,p3),Mod; \\ (Sw1,p1),(Sw2,p3),Del; \\ (Sw1,p1),(Sw2,p3),Pac\_in,MIB,Del; \\ (Sw1,p1),(Sw2,p3),Pac\_in,MIB,Store,Cont,FindSw,(Sw1,p1),pac\_in; \\ (Sw1,p1),(Swi,p3),Mod; \\ (Sw1,p1),(Swi,p3),Del; \\ (Sw1,p1),(Swi,p3),Pac\_in,MIB,Del; \\ (Sw1,p1),(Swi,p3),Pac\_in,MIB,Cont,Don't\_Find; \\ (Sw1,p1),(Swi,p3),Pac\_in,MIB,Cont,FindSw,(Swi,p3),pac\_in; \end{cases} \quad (3.49)$$

В результате проверки эквивалентности последовательностей смены состояний ветвей дерева достижимости модели реализации и модели спецификации получены следующие несовпадения: $(Sw1, p1),(Swi, p3),Mod$, $(Sw1, p1),(Swi, p3),Del$ и $(Sw1, p1),(Swi, p3),Pac\_in,MIB,Cont,Don't\_Find$

Данные цепочки указывают на то, что модель реализации не соответствует модели спецификации.

Для данных цепочек в соответствии с правилами, предложенными в п. 3.2.3, могут быть построены следующие контрпримеры:

$$P_{контр} = P((Sw1, p1),(Swi, p3)) + Del + \varnothing; \quad (3.50)$$

$$P_{контр} = P((Sw1, p1),(Swi, p3)) + Mod + \varnothing; \quad (3.51)$$

$$P_{контр} = P((Sw1, p1),(Swi, p3)Pac\_in + MIB, Cont) + Don't\_Find + (Sw1, p1),Pac\_in;$$

Построенные контрпримеры указывают на то, что состояния протокола Del, Mod, $Don't\_Find$, указанные в требованиях спецификации, не совпадают с состояниями модели реализации. Это обуславливает необходимость расширениям пространства состояний модели реализации, а, следовательно, модификации реализации протокола.

3.5 Выводы по третьему разделу

1. Отличительной особенностью предлагаемого метода является расширение области исходных данных верификации. Помимо проверки соответствия полному набору требований спецификации предлагается также использовать набор шаблонов. Шаблон представляет собой требование, истинное в процессе всего времени функционирования протокола.

2. В рамках решения задачи проверки соответствия как протокола OpenFlow, так и фрагментов сети, функционирующих на его основе, требования спецификации, разработан новый метод верификации. На основе рассмотренных ситуаций выделены несколько областей проведения верификации (3.1 – 3.3). Области верификации отличаются степенью охвата

процедуры проверки. Использование такого подхода позволяет сократить пространство исследуемых состояний при рассмотрении различных сценариев поведения протокола, а, следовательно, избежать эффекта «комбинаторного взрыва».

3. Разработанный метод верификации базируется на классическом подходе метода Model Checking. При этом возникновение ошибки может быть обнаружено уже на первом шаге путем сравнения множества состояний реализации протокола и его спецификации. Локализация состояния или последовательности состояний, приводящих к возникновению ошибки, выполняется путем построения контрпримера, который позволяет установить поведение протокола, приводящее к расхождению со спецификацией.

4. Разработан метод построения контрпримера, который позволяет построить и алгебраически описать инварианты поведения протокола, а также учитывать образование циклов и тупиков. Метод построения в случае образования циклов основан на алгоритмах двойного обхода и поиска в глубину.

5. Приведен пример верификации процесса обнаружения каналов связи с недостаточной пропускной способностью. Верификация была проведена двумя методами: методом символьной проверки и разработанным методом. Результаты верификации показали, что разработанный метод позволяет выявить одинаковые ошибки, что и символьный метод, а также вывести дополнительные последовательности действий, приводящие некорректному поведению протокола (3.27).

7. В качестве примера реализации предложенного метода проведена верификация процесса формирования канала связи в беспроводных сетях, функционирующих на основе протокола OpenFlow и процесса сбора информации о текущей сетевой топологии посредством OpenFlow Discovery Protocol.

РАЗДЕЛ 4
ОБОБЩЕННАЯ МЕТОДИКА АНАЛИЗА И ВЕРИФИКАЦИИ ПРОТОКОЛОВ УПРАВЛЕНИЯ В ПРОГРАМНО-КОНФИГУРИРОВАННЫХ СЕТЯХ

4.1 Формирование методики анализа и верификации протокола OpenFlow

Анализ основных особенностей функционирования протокола OpenFlow, тенденций и проблем, возникающих на пути его развития, показал, что проверка корректности функционирования и верификация являются ключевыми этапами в процессе его проектирования [4-6, 15, 108].

В качестве модели жизненного цикла протокола OpenFlow предложено использовать спиральную модель. Существенной особенностью спиральной модели жизненного цикла является наличие нескольких прототипов реализации протокола на каждом этапе разработки. Таким образом, процесс анализа и верификации необходимо выполнять в одной плоскости для множества возможных реализаций. Такой анализ и проверка совместимости версий реализации в рамках каждого этапа позволит выявить расхождения различных версий протокола и сократить множество возможных ошибок [50, 99, 126].

Дополнения и модификации, приведенные для каждого этапа спиральной модели жизненного цикла в предыдущих разделах, позволили сформировать обобщенную методику анализа и верификации протокола OpenFlow.

Предлагаемая обобщенная методика анализа и верификации охватывает множество этапов жизненного цикла, а также позволяет учитывать основные особенности протокола OpenFlow.

Методика содержит следующие рекомендации:

1. На этапе анализа предметной области и формирования требований спецификации предложено использовать математический аппарат алгебры коммутационных распределенных ресурсов (ACSR). Методика формализации требований приведена в п.2.2. В рамках решения задачи проверки на

непротиворечивость требований спецификации («горизонтальная» проверка) разработан алгоритм, базирующийся на последовательном поэлементном сравнении формализмов ACSR с учетом их причинно-следственных связей. В результате проверки предложено выполненять следующие действия:

- если в ходе проверки на непротиворечивость в спецификации ошибки или противоречия были обнаружены, то спецификация направляется на доработку;

- если проверка выполнена успешно, то на основе полученных формализмов формируется множество требований спецификации, которые являются исходными данными для последующего этапа разработки.

2. С целью устранения ошибок, связанных с множеством прототипов и особенностями программной реализации протокола OpenFlow, нарушение которых приводит к некорректному функционированию протокола, в качестве завершающей стадии этапа проектирования предлагается выполнять анализ как функциональных, так и нефункциональных требований.

Анализ предложено проводить путем проверки соблюдения основных алгоритмических свойств Е-сети модели реализации протокола: активности, сохраняемости, ограниченности, безопасности, достижимости. В качестве средства анализа предложено применять модифицированное дерево достижимости.

3. На этапе тестирования предложено выполнять заключительную проверку соответствия реализации протокола требованиям спецификации посредством проведения формальной верификации. Для получения наиболее эффективного результата предложено модифицировать классический метод Model Checking: в качестве входных данных предложено использовать модель Е-сети реализации протокола и множество ACSR формализмов требований спецификации, включая множество возможных прототипов реализации[43, 45].

Метод проверки соответствия базируется на последовательном сопоставлении дерева достижимости модели реализации и модели спецификации протокола. Такой подход позволяет существенно сократить

эффект «комбинаторного взрыва» пространства исследуемых состояний [126]. Если в процессе верификации выявлены ошибки, то формируется контрпример: множество вариантов поведения протокола, приводящих к желаемому заключительному состоянию. На основании построенных поведенческих цепочек создаются рекомендации о том, какие изменения необходимо внести в реализацию протокола.

Структурная схема предлагаемой методики анализа и верификации протокола OpenFlow, приведена на рис. 4.1.

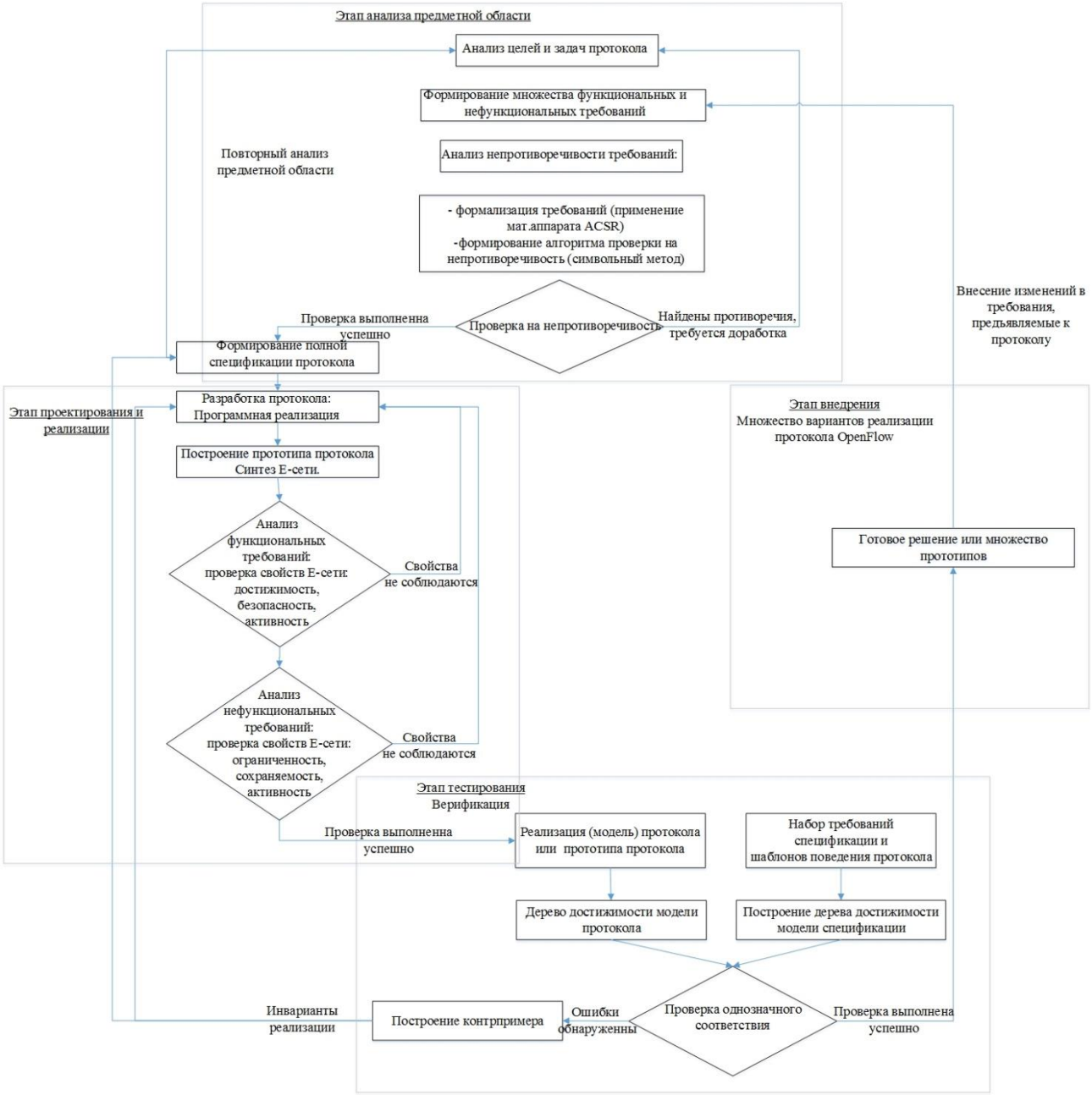


Рис.4.1. Методика анализа и верификации протокола OpenFlow в рамках спиральной модели жизненного цикла

Стоит отметить, что в рамках спиральной модели жизненного цикла протокола OpenFlow этап верификации может являться отдельным этапом, в случае, если необходимо выполнить проверку совместимости прототипов. Такой подход позволяет выявить ряд ошибок, связанных с несогласованностью спецификаций различных версий протокола OpenFlow, что невозможно при применении других методов [3,154, 155].

Применение на каждом последующем этапе в качестве входных данных результатов предыдущего этапа позволяет значительно сократить временные и материальные издержки процесса разработки. В рамках разработанной методики на этапе верификации предлагается применять модель Е-сеть протокола OpenFlow и дерево ее достижимости, которые были получены на предшествующих этапах, что позволяет существенно сократить временные затраты.

Возможность внесения изменений в требования спецификации без существенных последствий для дальнейших этапов, а также параллельная разработка, анализ корректности и проверка совместимости нескольких версий реализации протокола позволяют наилучшим образом адаптировать циклическую модель жизненного цикла для разработки протокола OpenFlow.

4.2 Оценка эффективности применения разработанной методики анализа и верификации в процессе проектирования протокола OpenFlow

Критерием оценки эффективности предлагаемой методики является отношение качества функционирования конечной реализации протокола к объему используемых ресурсов и времени процесса разработки. В качестве основного показателя эффективности предлагается использовать суммарную трудоемкость процесса разработки протокола, измеренную в человеко-часах. Выбор данного показателя обусловлен тем, что он позволяет оценить как сокращение времени разработки, так и снижение финансовых затрат на реализацию протокола [137, 151].

С целью оценки эффективности разработанной методики двум группам разработчиков было предложено выполнить проект «Flow Balance», который нацелен на определение ненагруженных каналов связи и вычислительных узлов, а также формирование пути передачи управляющих данных посредством протокола OpenFlow[81, 109].

Двум группам разработчиков было предложено использовать различный подход к разработке: одна группа разработчиков использовала каскадную модель жизненного цикла, а вторая - спиральную модель жизненного цикла.

Первая группа разработчиков на этапе определения и спецификации требований воспользовалась языком UML. Диаграммы UML являлись исходными данными для дальнейших этапов разработки [100, 103]. Методы и методики, применяемые в дальнейшем процессе разработки, были отведены на самостоятельный выбор разработчиков.

Вторая группа воспользовалась предлагаемой методикой анализа и верификации. На этапах определения требований и анализа предметной области применены формализмы алгебры коммуникационных распределенных ресурсов [26].

Время разработки в соответствии с техническим заданием, без этапа сопровождения готового решения, для каждой группы разработчиков составляло три месяца.

В рамках решения поставленной задачи как первой, так и второй группой разработчиков были выделены следующие основные этапы разработки, которые соответствуют как каскадной модели, так и спиральной модели жизненного цикла [13, 96, 97, 108]:

а) Анализ требований и формирование спецификации.

б) Проектирование фрагмента протокола OpenFlow в соответствии с сформированными требованиями.

в) Реализация и автономная отладка.

г)Интеграция и комплексная отладка.

В процессе разработки протокола проводилась оценка следующих индикаторов:

- количество возвратов с этапа разработки *j* на предыдущий этап *i* по причине обнаружения на этапе *j* ошибки, допущенной и не выявленной на этапе *i*;
- суммарное количество человеко-часов, затраченных на каждом этапе разработки фрагмента протокола OpenFlow с учетом возможных возвратов на предыдущие этапы.

Количество человеко-часов, затраченных разработчиками в процессе выполнения каждого этапа разработки (T_{total}) представляет собой сумму двух составляющих: количество человеко-часов, которые затрачены на прохождение определенного этапа в первый раз (T_{first}), и суммарное количество человеко-часов, которые затрачены при повторном возврате на этот этап ($T_{repeated}$): $T_{total} = T_{first} + T_{repeated}$.

На этапе формирования требований спецификации, в случае применения аппарата алгебры коммуникационных распределенных ресурсов, проведены дополнительные вычисления значений $T_{first(ACSR)}$ и $T_{repeated(ACSR)}$. Вычисления $T_{first(ACSR)}$ и $T_{repeated(ACSR)}$ позволяют оценить временные затраты на данном этапе при использовании предлагаемого подхода.

По результатам работы двух групп были получены следующие данные. Первая группа разработчиков в результате выполнения проекта затратила на проект 500 человеко-часов, в то время как вторая группа разработчиков затратила 425 человеко-часа.

На рис. 4.2 приведена гистограмма распределения человеко-часов, затраченных на каждом этапе разработки первой и второй группой разработчиков соответственно.

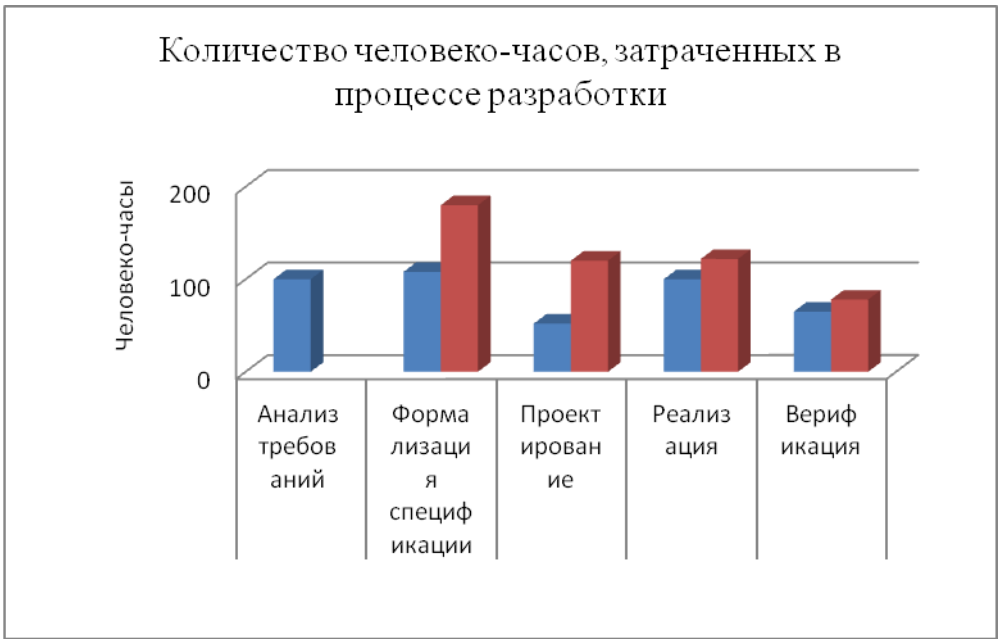


Рис. 4.2. Гистограмма распределения суммарного количества человеко-часов

Исходя из данных, приведенных на гистограмме, можно отметить, что количество человеко-часов, затраченных на ранних этапах (анализ требований и формализация спецификации) второй группой разработчиков больше, чем первой. Это обусловлено наличием дополнительного этапа и временем, потраченным на построение и проверку формальнных требований спецификации.

Количественное значение T_{first} и $T_{repeated}$ для двух групп разработчиков приведено в таблице 4.1.

Таблица 4.1

Количественное значение T_{first} и $T_{repeated}$ на этапе анализа предметной области и спецификации требований

Индикатор	Значения индикатора для первой группы (человеко-часы)	Значения индикатора для второй группы (человеко-часы)
T_{total}	180	208
$T_{first} / T_{first(ACSR)}$	82	164
$T_{repeated} / T_{repeated(ACSR)}$	98	44

В соответсвии с данными, приведенными в таблице 4.1, количество

человеко-часов, затраченных возвращение к первому этапу и на внесение повторных изменений в спецификацию при выполнении проекта второй группой разработчиков существенно ниже.

Полученные результирующие значения указывают на то, что применение разработанной методики анализа и верификации позволило повысить эффективность процесса разработки протокола OpenFlow на 17%. Количество часов, затраченных второй группой на ранних стадиях разработки (анализ предметной области и формализация спецификации) несколько выше. Однако, суммарное количество человеко-часов, затраченных второй группой разработчиков на всех совместных этапах разработки, меньше чем в первой группе.

Данный факт связан с тем, что большинство расхождений и неточностей в соответствии с этапами предлагаемой методики, устранено на этапе формализации требований. Уменьшение суммарного времени, затраченного на этапах формализации спецификации и проектирования, у второй группы связано с меньшим количеством возвратов на эти этапы.

Проверка нескольких прототипов в горизонтальной плоскости на каждом этапе разработки также позволяет расширить множество ошибок, возникновение которых возможно на данном этапе разработки, что нивелирует их появление на последующих этапах. Это позволяет утверждать, что разрабатываемый с помощью предложенной методики протокол определения требуемой пропускной способности принадлежащих агенту каналов связи содержит меньше ошибок и неточностей, чем в первой группе и не потребовал кардинального пересмотра на протяжении всех этапов разработки [98, 99].

В ряде случаев, этап верификации выполняется автономно: интеграция нескольких решений, обновление протокола OpenFlow, ввод в эксплуатацию нового OpenFlow-оборудования. В таком случае эффективность разработанного метода верификации должна быть оценена отдельно. В качестве оценки эффективности разработанного метода верификации приводится его сравнение с существующим методом SPIN [7, 91, 92]. Как предлагаемый метод

верификации, так и SPIN базируются на подходе Model Checking. Критериями оценки эффективности выступают время, затраченное в процессе верификации и количество выявленных ошибок.

4.3 Оценка эффективности разработанного метода верификации

В рамках оценки эффективности и корректности разработанного метода верификации исследовано несколько ситуаций функционирования протокола OpenFlow. Экспериментальные исследования проводились для фрагмента сети, структурная схема которого приведена на рис. 4.3. Имитационное моделирование было проведено с помощью программного продукта Mininet [59, 60].

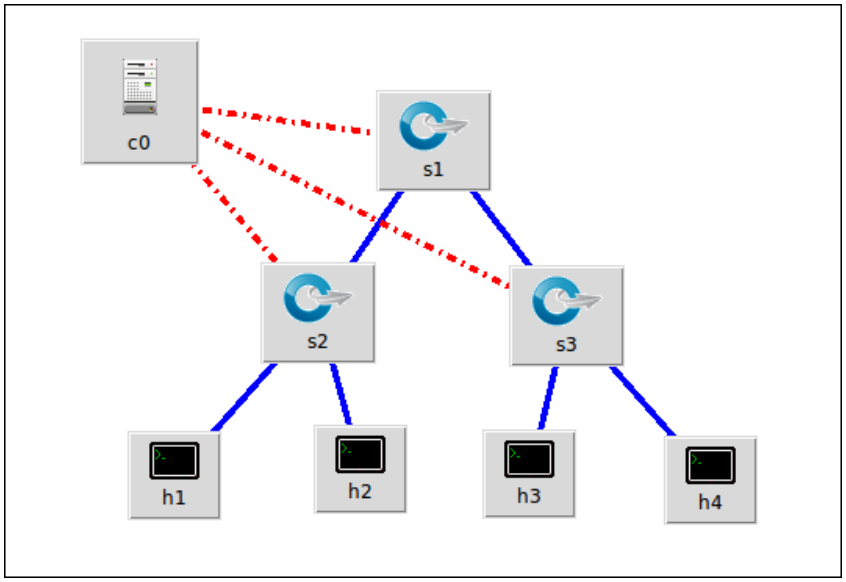


Рис. 4.3. Фрагмент экспериментальной сети

Основными компонентами экспериментальной сети являются: SDN контроллера (c1) с сетевой операционной системой POX [105, 108], трех OpenFlow коммутаторов (s1, s2, s3) и четырех конечных узлов, подключенных к коммутатору s2 (h1 h2) и s3 (h3 h4). Конфигурация сети выполнялась следующим образом:

```
*** Creating network
$ sudo mn --topo=single,8
c1
sudo mn --controller=remote, ip=192.168.12.129 --topo=tree,8
…
$ sudo ~/pox/pox.ru forwarding.l2_learning
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1, s2, s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
mininet> net
h1 h1-eth0:s2-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s3-eth1
s1 lo:  s1-eth3:s2-eth3 s1-eth2:s3-eth3
s2  lo:    s2-eth3:s1  -  s3-eth1:h1-eth0  s3-eth2:h2-eth0  s3-
eth3:s2-eth1
s3  lo:    s3-eth3:s1  -  s4-eth1:h3-eth0  s4-eth2:h4-eth0  s4-
eth3:s2-eth2
*** Configuring hosts
h1 h2 h3 h4
mininet>    h1    ifconfig    h1-eth0    192.168.12.5    netmask
255.255.255.0
mininet>    h2    ifconfig    h2-eth0    192.168.12.2    netmask
255.255.255.0
mininet>    h2    ifconfig    h3-eth0    192.168.23.2    netmask
255.255.255.0
mininet>    h4    ifconfig    h43-eth0    192.168.23.3    netmask
255.255.255.0
```

В процессе проведения эксперимента и выявления возможных ошибок в предложенной реализации фрагмента сети осуществлялся сбор статистических данных с помощью программы WireShark на протяжении интервала времени (30 мин). Количество запусков исследуемого фрагмента сети – 20 раз.

В результате эксперимента обнаружены следующие ошибки:

а) потеря ARP пакетов. Контроллер обрабатывает ARP запросы пришедшие от коммутатора как прокси-сервер. При этом TCP сообщение устанавливается, однако ARP запросы не обрабатываются (контроллер не генерирует ответ на данный тип запроса).

б) задержки при передаче TCP пакетов [84, 85, 99]. В случае изменения правил балансировки нагрузки и распределении ресурсов сети возникает

задержка обработки TCP пакетов. При отсутствии подтверждения о получении (ACK) отправитель генерирует новый пакет и направляет его обратно коммутатору. Возникновение такой ситуации привело к обработке сообщений в неправильном порядке, а, следовательно, снижению производительности и некорректной работе сети.

в) дублирование SYN-запросов. Повторная передача SYN-запроса возникла из-за превышения времени тайм-аут на коммутаторе. При получении и обработке повторного SYN-запроса контроллер может идентифицировать его как новый. В этом случае может быть сформирован новый маршрут передачи, что приводит к потере логической цепочке и разрыву связи.

Приведенные ошибки указывают на некорректное поведение протокола OpenFlow.

В процессе оценки эффективности разработанного метода верификации были рассмотрены следующие сценарии:

- сбор информации о топологии сети с помощью протокола OFDP (алгоритм функционирования и процесс верификации приведен в п.3.3);
- проверка доступности сетевых ресурсов в процессе функционирования протокола OpenFlow: каждый процесс, инициированный стороной клиента, обязательно получит доступ к разделяемому ресурсу в течении определенного промежутка времени. Формальное представление данного требования посредством ACSR приведено в п.2.3 и проверка выполнимости данного утверждения приведена в п.3.3.2.
- корректность обмена информацией между OpenFlow коммутаторами. s2 – инициирует установления канала связи с s3, необходимо подтверждение того, что каждое сообщение, генерируемое s2, будет отправлено без ошибки хотя бы один раз и будет принято s3 не более одного раза - отсутствие дублирования сообщений (формализм утверждения приведен в п.2.3). Дополнительным вариантом проверки являлось следующее условие: если исходящие сообщения от s2 отсутствуют, то отсутствуют и принятые

сообщения на s3 с заданными значениями полей для данного канала связи[105, 107, 108].

Модель E-сети, позволяющая отобразить поведение фрагмента исследуемой сети, приведена на рис. 4.4.

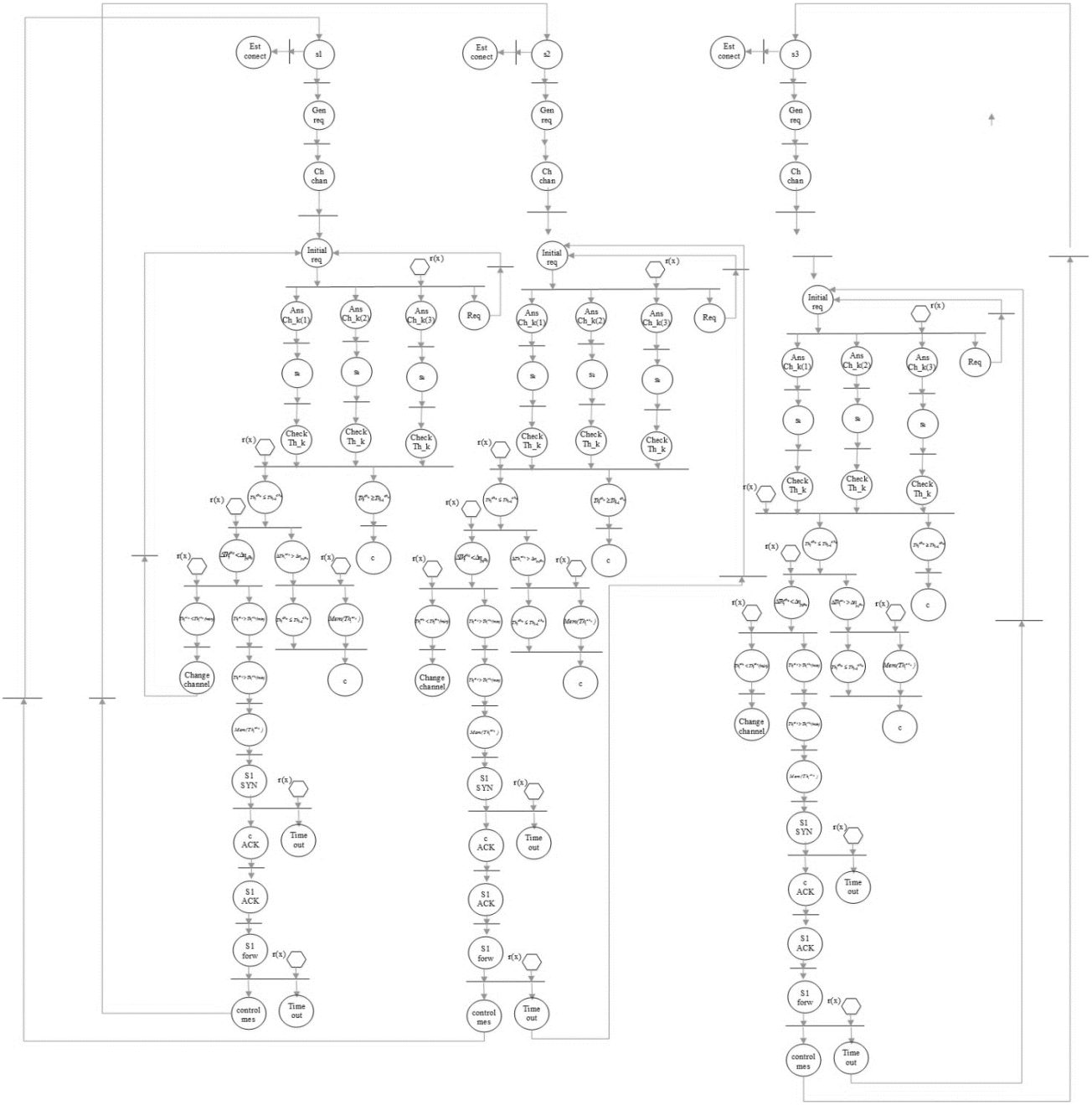


Рис. 4.4. Модель E-сети, отображающая поведение протокола OpenFlow для заданного фрагмента сети

Каждая приведенная выше ситуация, при которой были обнаружены ошибки, верифицировалась с помощью двух методов: разработанного метода верификации, и с помощью верификатора SPIN.

Предлагаемый метод верификации был апробирован на вычислительном узле со следующими параметрами: ОС – Ubuntu 14.04_64, RAM - 64 GB, тактовая частота процессора – 2,6 ГГц, количество ядер – 4.

Верификация алгоритма сбора информации о топологии сети приведена в п. 3.4. 3 с целью последующей балансировки нагрузки. В результате проведения верификации были обнаружены расхождения с требованиями спецификации (3.48-3.49) и построены следующие контрпримеры:

$$P_{контр} = P((Sw1, p1),(Swi, p3)) + Del + \varnothing; \quad (4.1)$$

$$P_{контр} = P((Sw1, p1),(Swi, p3)) + Mod + \varnothing. \quad (4.2)$$

Алгоритм, используемый в SPIN верификаторе, представлен следующим образом:

```
spin -a ds.pml
gcc -DNCORE=4 -DNP -o pan pan.c
pan -l > log.txt
spin -p -t ds.pml > trace.txt
del pan.* ds.pml.trail
```

Количество ошибок, обнаруженных в процессе верификации, разработанным методом и с помощью SPIN верификатора, не совпадает.

Для SPIN верификатора получены следующие характеристики:

```
state vector 24 bite, depth reached 65, errors 1
…
pan: elapced time 0,129 ms
pan: rate 2553.1915 states/second
```

Реализация алгоритма динамической балансировки нагрузки и оценки загруженности канала связи приведена на рис.3.10, модель Е-сети требований спецификации на рис. 3.11

Поэтапная проверка соответствия модели реализации требованиям спецификации приведена в п. 3.4.1. В результате проверки были выявлены следующие расхождения (3.25 -3.26) и построено множество контрпримеров:

$$P_{контр1} = (Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k + (Th_i^{chk} < Th_{i-1}^{chk}) + (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)),$$
$$Mem(Th_i^{chk})) || (Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k + (Th_i^{chk} \le Th_{i-1}^{chk}) + (\neg(\Delta Th_i^{Chk} < \Delta n_{c^k}),$$
$$(Th_i^{chk} > Th_{i-1}^{chk}(min)), Mem(Th_i^{chk}));$$
$$P_{контр2} = (Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k (Th_i^{chk} < Th_{i-1}^{chk}), (\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)) +$$
$$(Th_i^{chk} \le Th_{i-1}^{chk}) + Channel\_k(i)\_enable) || (Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k (Th_i^{chk} < Th_{i-1}^{chk}),$$
$$(\Delta Th_i^{Chk} < \Delta n_{c^k}), (Th_i^{chk} > Th_{i-1}^{chk}(min)) + (Th_i^{chk} \le Th_{i-1}^{chk}) + \varnothing);$$
$$P_{контр3} = (Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k (Th_i^{chk} > Th_{i-1}^{chk}), Channel\_k(i)\_enable) ||$$
$$(Initial\_req, Ans\_Ch\_k(i), sk, Check\_Th^k (Th_i^{chk} > Th_{i-1}^{chk}),$$
$$Channel\_k(i)\_enable + Drop\_t + \varnothing.$$

Одним из основных требований, предъявляемых при балансировке нагрузки и учитываемых верификатором SPIN, является недостижимость ситуации отказа в доступе, при которой канал связи всегда будет находиться в состоянии Change Channel.

Для проверки выполнимости данного требования верификатором SPIN применялась следующая последовательность команд:

```
spin -f "> ([] (State == Change Channel) & M(Change Channel)!=∅)" > ltl.nvr
spin -a -N ltl.nvr sw.pml
gcc -DNCORE=4 -o pan pan.c
pan -a -n > log.txt
spin -p -t sw.pml > trace.txt
del pan.* ltl.nvr sw.pml.trail
```

В результате проверки спецификации с помощью подхода SPIN обнаружено 2 ошибки.

```
state vector 530 bite, depth reached 1025, errors 2
…
pan: elapced time 0,501 ms
pan: rate 912.81139 states/second
```

Таким образом, при проверке соответствия реализации требованиям спецификации с помощью разработанного метода было выявлено большее количество ошибок.

В рамках проверки доступности сетевых ресурсов при функционировании протокола OpenFlow построена следующая модель E-сети (рис. 4.5).

Приведенная модель Е-сети моделирует следующее утверждение: каждый процесс, инициированный стороной клиента, обязательно получит доступ к разделяемому ресурсу в течении определенного промежутка времени (иными словами контроллер рано или поздно обработает запросы каждого коммутатора). Формализованные требования, предъявляемые к доступности сетевых ресурсов, приведены в п.2.3.

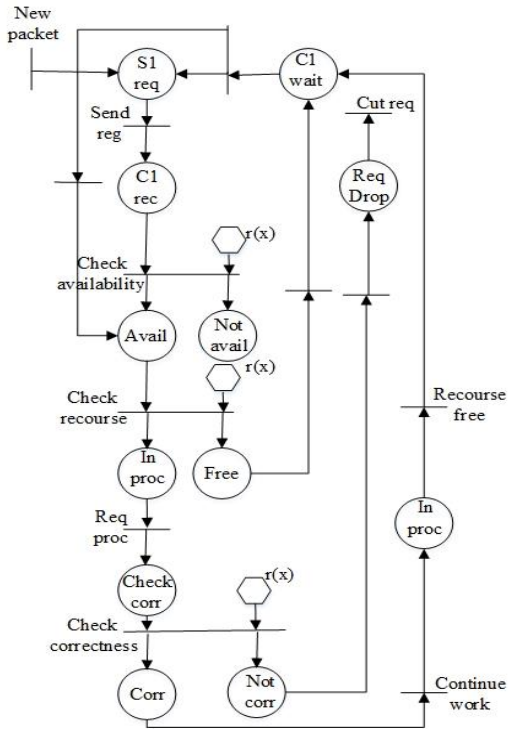


Рис.4.5. Модель Е-сети процесса проверки доступности сетевых ресурсов (доступность контроллера c1)

Начальным состоянием модели Е-сети процесса проверки доступности контроллера c1 является состояние *S1 reg* – формирование запроса коммутатора, состояние *C1 rec* указывает на доставку сообщения контроллеру, состояние *Avail* соответствует доступности сетевых ресурсов, *Not avail* – сетевой ресурс не доступен, состояние *In proc* указывает на обработку запроса, полученного от коммутатора, состояние Free символизирует освобождение сетевых ресурсов, состояние *Check corr* указывает на некорректную обработку запроса, *Corr* – обработка запросов выполненна успешно, *Not corr* – запрос обработан некорректно, состояние *Req drop* соответствует потере запроса, состояние *C1 wait* ожидание запроса от контроллера.

Модель Е-сети, соответствующая требованиям спецификации, представленным в (2.11-2.21) и [67, 73], приведена на рис.4.6.

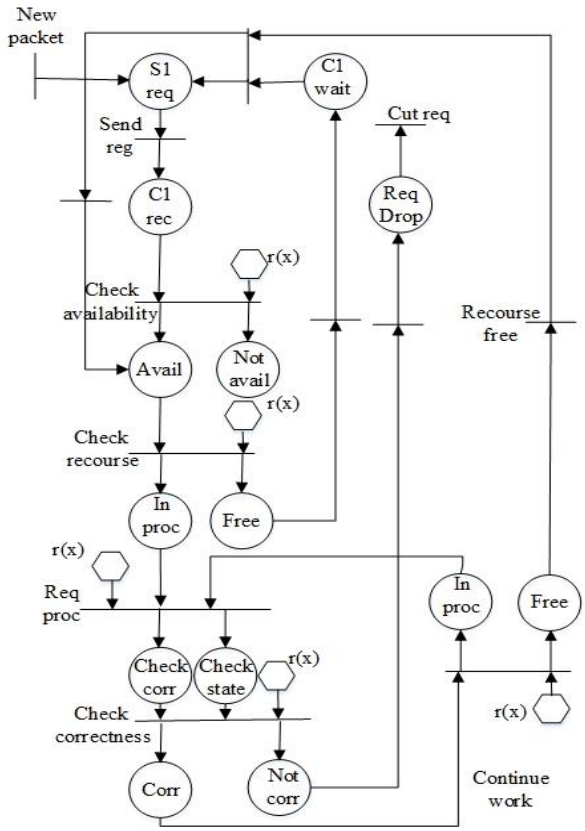
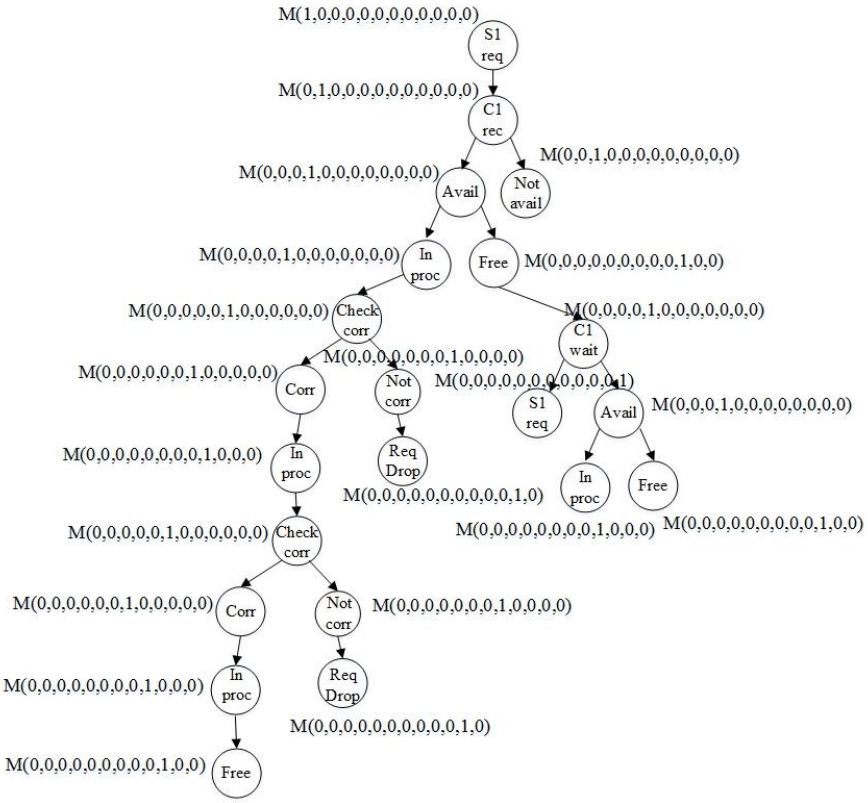


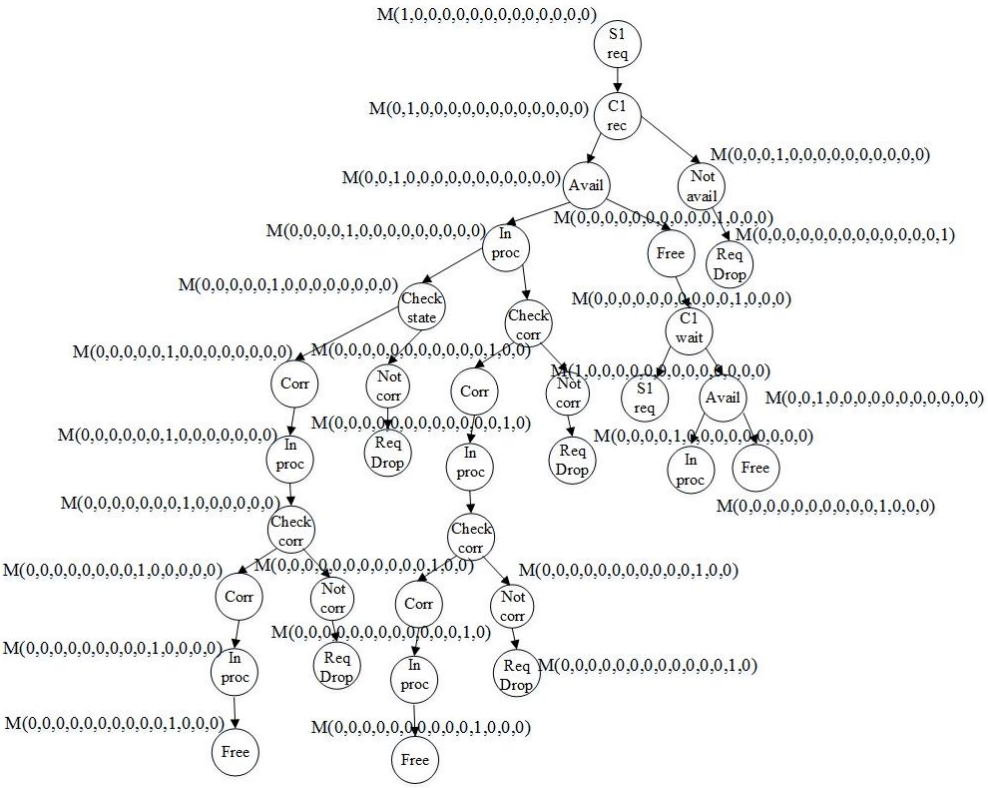
Рис.4.6. Модель Е-сети процесса проверки доступности сетевых ресурса (доступности контроллера с1)

Для модели реализации и спецификации выведены следующие деревья достижимости (рис 4.7).

Начальным состоянием модели Е-сети процесса проверки доступности контроллера с1 является состояние *S1 reg* – формирование запроса коммутатора, состояние *C1 rec* указывает на доставку сообщения контроллеру, состояние *Avail* соответствует доступности сетевых ресурсов, *Not avail* – сетевой ресурс не доступен, состояние *In proc* указывает на обработку запроса, полученного от коммутатора, состояние *Free* символизирует освобождение сетевых ресурсов, состояние *Check corr* указывает на некорректную обработку запроса, *Check state* – повторная проверка состояния обрабатываемого запроса, *Corr* – обработка запросов выполнена успешно.



а)



б)

Рис. 4.7. Дерево достижимости модели реализации (а) и модели спецификации (б) процесса проверки доступности сетевых ресурсов для заданного фрагмента сети

Not corr – запрос обработан некорректно, состояние *Req drop* соответствует потере запроса, состояние *C1 wait* ожидание запроса контроллером.

Данному начальному состоянию соответствуют следующие маркировки: M(1,0,0,0,0,0,0,0,0,0,0,0) – для дерева достижимости модели реализации, M'(1,0,0,0,0,0,0,0,0,0,0,0,0,0) – для дерева достижимости модели спецификации. Состояния *Free, Req Drop, In proc* принадлежат множеству заключительных состояний.

Для дерева достижимости модели реализации построено следующее множество цепочек, приводящих к терминальным состояниям:

$$M(T_f)_{реализация} = \begin{cases} S1\_req, C1, Avail, In\_proc, Check\_corr, Corr, Ip\_proc, Corr, Free \rightarrow \\ \rightarrow S1\_req, C1, Avail, (In\_proc, Check\_corr, Corr, Ip\_proc, Corr)^n, Free; \\ S1\_req, C1, Not\_avail; \\ (S1\_req, C1, Avail, Free, C1\_wait, S1\_req, S1\_req, C1, Avail, In\_proc, Check\_corr, \\ Corr, Ip\_proc, Corr, Free) \cup \\ \cup (S1\_req, C1, Avail, Free, C1\_wait, S1\_req, S1\_req, C1, Avail, Free) \rightarrow \\ \rightarrow S1\_req, C1, Avail, (In\_proc, Check\_corr, Corr, Ip\_proc, Corr)^m, Free \cup \\ (S1\_req, C1, Avail, Free)^l; \\ S1\_req, C1, Avail, In\_proc, Check\_corr, Corr, Ip\_proc, Corr, Free, Not\_avai; \\ S1\_req, C1, (Avail, In\_proc, Check\_corr)^k, (Corr, Ip\_proc, Not\_corr)^p, Drop\_req. \end{cases} . \quad (4.3)$$

Для дерева достижимости модели спецификации:

$$M(T_f)_{спецификация} = \begin{cases} S1\_req, C1, Avail, In\_proc, Check\_corr, Corr, Ip\_proc, Corr, Free \rightarrow \\ \rightarrow S1\_req, C1, Avail, (In\_proc, Check\_corr, Corr, Ip\_proc, Corr)^n, Free; \\ S1\_req, C1, Not\_avail, Drop\_req; \\ (S1\_req, C1, Avail, Free, C1\_wait, S1\_req, S1\_req, C1, Avail, In\_proc, Check\_corr, \\ Corr, Ip\_proc, Corr, Free) \cup \\ \cup (S1\_req, C1, Avail, Free, C1\_wait, S1\_req, S1\_req, C1, Avail, Free) \rightarrow \\ \rightarrow S1\_req, C1, Avail, (In\_proc, Check\_corr, Corr, Ip\_proc, Corr)^m, Free \cup \\ (S1\_req, C1, Avail, Free)^l; \\ S1\_req, C1, Avail, In\_proc, Check\_corr, Corr, Ip\_proc, Corr, Free, Not\_avail; \\ S1\_req, C1, (Avail, In\_proc, Check\_corr)^k, (Corr, Ip\_proc, Not\_corr)^p, Drop\_req; \\ S1\_req, C1, Avail, (In\_proc, Check\_corr, Corr, Ip\_proc, Check\_state, Corr, Free; \\ S1\_req, C1, Avail, In\_proc, Check\_corr, Corr, Ip\_proc, Check\_state, Not\_corr, Drop\_req; \\ S1\_req, C1, Avail, (In\_proc, Check\_corr)^k, (Corr, Ip\_proc, Check\_state, Not\_corr)^p, Free. \end{cases} \quad (4.4)$$

В результате проверки соответствия множества ветвей дерева достижимости, приходящих к терминальным состояниям, модели реализации и модели спецификации, обнаружены следующие расхождения:

$Ip\_proc, Check\_state, Corr; Ip\_proc, Check\_state, Not\_corr; (Ip\_proc, Check\_state, Not\_corr)^p;$
$S1\_req, C1, Not\_avail, Drop\_req.$

Основываясь на алгоритме, приведенном в п. 3.2.3 (3.9), сформированы следующие контрпримеры: $P_{контр_1} = S1\_req, C1, Not\_avail + \varnothing;$

$P_{контр2} = S1\_req, C1, Avail, In\_proc, Check\_corr, Corr, Ip\_proc + Check\_state + Corr;$
$P_{контр3} = S1\_req, C1, Avail, In\_proc, Check\_corr, Corr, Ip\_proc + Check\_state + Drop\_req;$
$P_{контр4} = S1\_req, C1, Avail, (In\_proc, Check\_corr)^y + Check\_state + Not\_corr.$

Основным требованием, учитываем при проверке доступности сетевых ресурсов, в частности, доступности контроллера c1, является необходимость достижения для каждого запроса состояния In proc, а для контроллера – состояний Avail и Free.

```
spin -f " ((C1:State == Avail || C1:State == Free) & (Mes:State==In proc) )" > ltl.nvr
spin -a -N ltl.nvr sw.pml
gcc -DNCORE=4 -o pan pan.c
pan -a -n > log.txt
spin -p -t sw.pml > trace.txt
del pan.* ltl.nvr sw.pml.trail
```

Результаты, полученные с помощью верификатора SPIN и разработанного метода верификации, совпали. При проверке с помощью SPIN было однаруженно 4 ошибки:

```
State vector 36 bite, depthreached 12, errors 4
pan: elapced time 0,129 ms
pan: rate 200 states/second
```

Таким образом, исследуемый алгоритм не отвечает предъявляемому требованию – возможна ситуация, при которой запрос коммутатора перейдет в состояние *Drop req* и не выйдет из этого состояния в дальнейшем. В целях устранения этой ошибки необходимо расширить множество состояний протокола сигнальными сообщениями *Check state* и *Drop Req.*

Сформировано выражение, которое позволяет учитывать зависимость времени выполнения процесса верификации от характеристик дерева

достижимости. В общем случае, время вывода дерева достижимости с одной ветвью может быть рассчитано с помощью следующей формулы:

$$T_{av} = d\frac{\overrightarrow{|M(t)_i|}}{\beta}\omega, \tag{4.5}$$

а среднее время вывода полного дерева достижимости:

$$T_{\overrightarrow{M(t)}} = \sum_{v=1}^{i} d_{\overrightarrow{M(t)}}\left(T_{av} + \frac{\overrightarrow{|M(t)_i|}}{\beta}\omega\right), \tag{4.6}$$

где d - длина ветви дерева достижимости; $d_{\overrightarrow{M(t)}}$ - средняя длинна ветви дерева достижимости; j количество возможных ветвей дерева достижимости;

T_{av} - среднее время вывода ветви дерева достижимости; $\overrightarrow{|M(t)_i|}$ - среднее количество состояний, задействованных в образовании цепочки с учетом образования циклов; ω- среднее время обработки правил управляющих переходов; β - коэффициент учета повторяющихся состояний ($\beta \geq 1$ - при наличии повторного прохождения состояний в ветви дерева достижимости).

В таблице 4.2 приведены временя выполнения процесса верификации и количество найденных ошибок для разработанного метода и верификатора SPIN.

Таблица 4.2

Значение времени верификации и количество обнаруженных ошибок, обнаруженных в процессе верификации

Верифицируемое требование/ критерии оценки	Разработанный метод		Метод SPIN	
	Время верификации (мс)	Количество обнаруженных ошибок	Время верификации (мс)	Количество обнаруженных ошибок
Сбор информации о топологии сети с помощью протокола OFDP	0,037	2	0, 040	1
Корректность обмена информацией между OpenFlow коммутаторами	0,501	3	0,585	2
Проверка доступности сетевых ресурсов в процессе функционирования протокола OpenFlow	0,129	4	0,130	4

Таким образом, разработанный метод верификации позволяет выявить большее количество ошибок, чем метод SPIN. Время, потраченное на верификацию в ряде случаев меньше времени, потраченного при применении SPIN. Для приведенного примера выигрыш в случае использования предлагаемого метода верификации составляет 22%.

4.4 Выводы по четвертому разделу

1. Разработанная методика методика анализа и верификации протоколов управления в прогрммно конфигурируемыъ сетях позволяет повысить эффективность процесса разработки на 17%. Повышение эффективности достигается следующим образом:

- применение спиральной модели жизненного цикла и проведение промежуточного анализа результатов, полученных на каждом этапе в двух плоскостях: «вертикальной» - проверка наличия неточностей и противоречий в рамках одного прототипа и «горизонтальной» - проверка наличия неточностей и противоречий, которые возникают в процессе совместного функционирования нескольких прототипов. Применение такого подхода существенно позволяет расширить круг обнаруженных ошибок и понизить количество возвратов на предыдущие этапы разработки.

- применение аппарата алгебры коммуникационных распределенных ресурсов на этапе формирования требований спецификации и применение метода проверки непротиворечивости требований в рамках спецификации.

- выполнение анализа функциональных и нефункциональных свойств модели протокола, построенной на основе аппарата Е-сетей, что позволяет установить корректность поведения протокола и эффективность распределения сетевых ресурсов;

- построение контрпримера в случае обнаружения ошибок в поведении реализации протокола и/или не соответствии его требованиям спецификации.

Контрпример позволяет выработать рекомендации по устранению как выявленных ошибок, так и их возможных причин.

2. В рамках оценки эффективности разработанной методики анализа и верификации проведены экспериментальные исследования. Целью исследований являлся анализ выбранных критериев эффективности при применении различных подходов к процессу разработки. Обязательным требованием исследования было применение одной из групп разработчиков предлагаемого подхода. Критерием эффективности выступали затраченные человеко-часы.

Данные, полученные в ходе исследования, позволилиустановить, что время, потраченное первой группой разработчиков на 75 человеко-часов больше, чем второй группой. При этом количество возвратов первой группой разработчиков к этапу формализации спецификации значительно выше. Применение предлагаемых рекомендаций позволило уменьшить время, потраченное на повторное выполнения каждого этапа, на 24 человека-часа.

4. С целью устранения подобных ошибок была проведена верификация экспериментального фрагмента сети двумя методами: разработанным методом верификации и посредством применения SPIN.

5. В рамках оценки вычислительной сложности разработанного метода верификации сформированы выражения (4.3, 4.4), которые позволяют учитывать зависимость времени выполнения процесса верификации отхарактеристик дерева достижимости.

6. В результате сравнения полученных данных, установлено, что предлагаемый метод позволяет выявить большее количество ошибок. Выигрыш от применения разработанного метода верификации составил 22%.

ВЫВОДЫ

В диссертационной работе предложено решение актуальной научно-прикладной задачи, которая заключается в совершенствовании процессов разработки и внедрения сетевых решений, построенных на основе концепции SDN. Предложенные в работе модели и методы, в качестве основы которых положены формализмы алгебры распределенных коммуникационных ресурсов, аппарат E-сетей, средства анализа структурных свойств моделей протоколов и формальной верификации, позволяют устранить ряд ошибок и неточностей, возникающих в процессе разработки протокола OpenFlow. Благодаря внедрению предлагаемых моделей и методов в процесс разработки также удалось сократить время разработки сетевых решений.

В соответствии результатами, полученными в ходе решении поставленной научно-прикладной задачи, сделан ряд следующих выводов:

1. Анализ архитектуры сетевых решений, базирующихся на основе концепции SDN, показал, что существующие проблемы функционирования SDN-сетей связанны с наличием нескольких версий спецификаций протокола OpenFlow, их разной программной реализацией и значительными функциональными отличиями используемых сетевых операционных систем SDN контроллеров. Анализ этапов разработки SDN-сетей показал, что раннее обнаружение и устранение противоречий в требованиях спецификации управляющих протоколов, в частности протокола OpenFlow и строгая проверка соответствия реализации предъявляемым начальным требованиям, позволяют ускорить процесс разработки, организовать бесшовное взаимодействие различных фрагментов сети и повысить показатели качества предоставления сервисов. В качестве модели жизненного цикла процесса разработки протокола OpenFlow выбрана спиральная модель. Ее основными преимуществами является возможность динамического обновления и изменения начальных требований к функционированию протокола, учет нескольких сценариев в процессе разработки, возможность внедрение подэтапов промежуточного анализа и упрощенный возврат на предыдущие этапы разработки.

2. В рамках совершенствования начальных этапов, в частности этапа формирования требований, предложено применение алгебры распределенных коммуникационных ресурсов. Аппарат ACSR включает множество вариаций взаимодействия темпоральных и логических операторов, что позволяет наиболее полно описать особенности поведения протокола OpenFlow, учитывая такие факты, как момент и условие наступления событий, время и длительность их выполнения, возможность выполнения параллельных процессов, как синхронной, так и асинхронной природы.

3. В качестве заключительной стадии этапа формализации спецификации предложено выполнять проверку требований на непротиворечивость. Данный тип проверки позволяет устранить множество неочевидных ошибок, и тем самым уменьшить количество повторных итераций в процессе разработки. Предложено два метода проверки непротиворечивостей требований в рамках спецификации протокола OpenFlow. Первый метод основан на нахождении всех формализмов спецификации, содержащих утверждение, непротиворечивость которого необходимо проверить и последовательной проверке хронологии их вхождений во все формулы ACSR. Второй метод основан на поиске противоречий в требованиях спецификации на основе оценки достижимости графа состояний спецификации протокола. Подход, основанный на построении графа состояний и анализе их достижимости, имеет следующее преимущество – высокую степень наглядностинаглядность.

4. В рамках совершенствования этапов проектирования и реализации протокола OpenFlow предложено ввести подэтап анализа и оценки корректности функционирования полученной реализации протокола и ее соответствия начальным требованиям. Разработан метод синтеза E-сети по формализмам требований спецификации, приведенным с помощью ACSR. Данный метод позволяет повысить адекватность модели реализации и автоматизировать процесс построения E-сети. Предлагаемый метод базируется на однозначном преобразовании процессов и состоянии формализмов ACSR в

переходы и позиции Е-сети.

5. Решение задачи оценки корректности функционирования протокола OpenFlow сведено к задаче проверки выполнимости функциональных и нефункциональных требований. Предложено проводить анализ таких алгоритмических свойств моделей Е-сетей, как ограниченность, достижимости, активность, сохраняемости для проверки соблюдения функциональных требований; безопасность, активность, сохраняемость для проверки нефункциональных требований. В качестве метода анализа основных алгоритмических свойств Е-сетей рассмотрены матричные уравнения, формальные грамматики и построение дерева достижимости. Дерево достижимости предложено в качестве основного инструмента анализа таких свойств как достижимость, активность, сохраняемость, ограниченность, а также устанавливать факт возникновения зацикливаний в процессе функционирования протокола и выявлять избыточность решений. Для устранения существующих недостатков (возникновения символа ω) в процессе вывода ветвей и адаптации предлагаемого метода в соответсвии с особенностями моделей Е-сетей разработан алгоритм построения модифицированного дерева достижимости.

6. В процессе анализа способов устранения ошибок, возникающих на разных этапах жизненного цикла протокола выявлено, что верификация является основным методом проверки соответствия реализации протокола OpenFlow его спецификации. Отмечено, что при верификации программных решений инфокоммуникационных протоколов наиболее широкое распространение получил метод «проверки на моделях» (Model Checking). Однако недостатком данного метода является экспоненциальный рост пространства исследуемых состояний – эффект «комбинаторного взрыва», что делает применения данного метода затруднительным. Предложена модификация метода Model Checking, которая базируется на построении и последовательной проверке эквивалентности ветвей дерева достижимости модели реализации и модели спецификации. В случае применения такого

подхода возникновение ошибки может быть обнаружено уже на начальных этапах проверки. Локализация состояния или последовательности состояний, приводящих к возникновению ошибки, выполняется путем построения контрпримера. Разработан метод вывода контрпримера, который позволяет построить и алгебраически описать инварианты поведения протокола, учитывая образование циклов и тупиков. Метод построения, в случае образования циклов, основан на алгоритмах двойного обхода и «поиска в глубину».

7. В рамках оценки эффективности предложенного метода проведет эксперимент, заключающийся в верификации фрагмента сети двумя методами различными методами - разработанным методом и методом SPIN. В результате сравнения данных, полученных в ходе выполнения эксперимента установлено, что предложенный метод позволяет выявить большее количество ошибок. Выигрыш от использования разработанного метода составил 22%.

8. В рамках оценки эффективности разработанной методики анализа и верификации проведены экспериментальные исследования. Обязательным требованием исследования было применение одной из групп разработчиков предлагаемого подхода. Критерием эффективности выступали затраченные человеко-часы. Данные, полученные в ходе исследования, позволилиустановить, что время, потраченное первой группой разработчиков на 75 человеко-часов больше, чем второй группой. При этом количество возвратов первой группой разработчиков к этапу формализации спецификации значительно выше. Применение предлагаемых рекомендаций позволило уменьшить время, потраченное на повторные выполнения каждого этапа, на 24 человека-часа. В целом отмечено, что предлагаемая методика анализа и верификации охватывает большинство этапов жизненного цикла протокола OpenFlow. Рекомендации и дополнения, внесенные для каждого из этапов, повысить эффективность процесса разработки на 17%.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Akyildiz I. F. A roadmap for traffic engineering in SDN-OpenFlow networks / Ian F. Akyildiz, Ahyoung Lee , Pu Wang, Min Luo, Wu Chou//Computer Networks. – 2014. – T. 71. – P. 1-30.

2. Al-Fares M.A Scalable, Commodity Data Center Network Architecture / M.Al-Fares, A.Loukissas, A.Vahdat // Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, (SIGCOMM ’08). – Seattle, WA, USA. – 2008. – P. 63–74.

3. Al-Shaer E. FlowChecker: Configuration Analysis and Verification of Federated Openflow Infrastructures / Al-Shaer E., Al-Haj S. // Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration, (SafeConfig ’10). – Chicago, Illinois, USA. – 2010. – P. 37–44.

4. Al-Shaer E. Network Configuration in a Box: Toward End-to-End Verification of Network Reachability and Security / Al-Shaer E., Marrero W., El-Atawy A. // 17th IEEE International Conference on Network Protocols (ICNP’09). Princeton, New Jersey, USA. – 2009. –P.123-132.

5. Architecture SDN [Електроннийресурс]// Open Networking Foundation. – 2014. – Режим доступа: https://www.opennetworking.org/

6. Baldoni M. Verifying the conformance of Web services to global interaction protocols: A first step / Baldoni M., Baroglio C., Martelli A. // International Workshop on Web Services and Formal Methods. – 2005. – p.27

7. Bengtsson J., Larsen K., Larsson F., Pettersson P., Yi W. UPPAAL — a tool suite for automatic verification of real-time systems // Hybrid Systems III, Lecture Notes in Computer Science. Vol. 1066. – Springer Berlin Heidelberg, 1996. – Pp. 232–243.

8. Bergstra J.A. Algebra of Communicating Processes with Abstraction / J.A. Bergstra, J.W. Klop // Journal of Theoretical Computer Science. – Vol.37. – 1985. – P.77–121.

9. Booch G. Object-oriented analysis and design with applications / Grady Booch // 3rd ed., Pearson Education, Inc. – 2009.– 717 p.

10. Boucadair M. [Электронный ресурс] Service Function Chaining: Framework & Architecture / Boucadair M., Jacquenet C. // Internet Draft (Intended Status: Standards Track) - 2014. – Режим доступа к статье: https://tools.ietf.org/search/draft-boucadair-sfc-framework-02

11. Bozga M. A modelchecking tool for real-time systems / Bozga M., Daws C., Maler O. // Formal Techniques in Real-Time and Fault- Tolerant Systems, Lecture Notes in Computer Science. Vol. 1486, Springer Berlin Heidelberg— 1998. – P. 298–302.

12. Brémond-Grégoire P. A Process Algebra of Communicating Shared Resources With Dense Time and Priorities / P. Brémond-Grégoire, I. Lee // Technical report, University of Pennsylvania. – January 1995. – 47 p.

13. Brémond-Grégoire P. ACSR: An algebra of communicating shared resources with dense time and priorities / P. Brémond-Grégoire, S. Davidson, and I. Lee //4th Intrenational Conference on Concurrency Theory Hildesheim, Germany. – August 23–26, 1993

14. Cai Z.Maestro: Balancing Fairness, Latency and Throughput in the OpenFlow Control Plane / Cai Z, Cox A.L, Eugene NgT.S. // Technical Report, Rice University: Houston, TX, USA–2011 – 78 p.

15. Chandra S.Software model checking in practice: an industrial case study / Chandra S., Godefroid P., Palm P. // In ICSE 02: International Conference on Software Engineering (ACM). –2002. – P. 431–441.

16. Chemeritskiy E.On QoS management in SDN by multipath routing / Chemeritskiy E., Smelansky R. // Proceedings 2014 international science and technologe conference «Modern Networking Technologies (MoNeTec)». Moscow, Russia, June 27-29, 2014. – P. 41—46.

17. Chemeritskiy E. On the Network Update Problem for Software Defined Networks / Chemeritskiy E., Zakharov V. // Proceedings of the 5-th Workshop

“Program Semantics, Specification and Verification: Theory and Applications”. – Moscow, Russia, June 4, 2014. – P. 26–37.

18. Chen P. The entity relationship model – Towards a unified view of data / P. Chen // ACM Trans. On Database System. – 1976. – №1(1). – P. 9-45.

19. Chung L.On Non-Functional Requirements in Software Engineering / Chung L., Prado Leite J. // Conceptual Modeling: Foundations and Applications, Lecture Notes in Computer Science. Vol. 5600. – Springer Berlin Heidelberg. – 2009. – P. 363–379.

20. Cisco Enterprise SDN: APIC-EM/ Andersen R.// Cisco public. – 2014 – 41 p.

21. Clarke E. M., Emerson E. A., Sistla A. P. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications / Clarke E. M., Emerson E. A., Sistla A. P. // ACM Transactions on Programming Languages and Systems, Vol. 8, No. 2. – April 1986. –P. 244-263.

22. Cleaveland R. Faster model checking for modal mu-calculus / R. Cleaveland, M. Klein, B. Steffen // Proc. Intern. Conf. On Computer-aided verification. – 1992 – P. 410-422.

23. Costas N. OpenFlow and SDN Technical Report / AtanasovaI. K., Costas N. // Technical Report CESGA-2014-001 — 2014. – 64 p.

24. DECODING SOFTWARE DEFINED NETWORKINGSunnyvale [Электронный ресурс] // Juniper Network Inc. – 2015. – 8 с. – Режим доступа: http://www.juniper.net/us/en/local/pdf/whitepapers/2000510-en.pdf -.

25. Egawa T. SDN standardization Landscape from ITU-T Study Group /13 Egawa T. // ITU Workshop on SDN. – Geneva, Switzerland, 4 June 2013. – 22 p.

26. Enns R. NETCONF configuration protocol [Электронный ресурс]/ R. Enns // Internet Draft, Internet Engineering Task Force. – December 2004. – Режим доступа: http://tools.ietf.org/html/rfc4741

27. Erickson D. The Beacon OpenFlow Controller / Erickson D. // In Proceedings of the ACM Workshop on Hot Topics in Software Defined Networks (HotSDN). – Hong Kong, China, 12–16 August 2013 - P. 13–18.

28. ETSIGSNFV 002[Электронный ресурс].Network functions virtualization (NFV):Architectural framework v1.1. ETSI// Technical Report. – October 2013. – Режим доступа: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01 60/gs NFV002v010101p.pdf

29. Feamster N.The Road to SDN: An Intellectual History of Programmable Networks / Feamster N., Rexford J., Zegura E. // ACM Queue.- 2013. – P. 20–40.

30. Fernandez M.P. Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive / M.P.Fernandez // In Proceedings of the International Conference on Advanced Information Networking and Applications (AINA), Barcelona, Spain, 25–28 March 2013. – P. 1009–1016.

31. Foster N. Frenetic: A Network Programming Language / N.Foster, R.Harrison, M. J.Freedman, C.Monsanto, J.Rexford, A.Story, D.Walker // Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming, (ICFP-2011), Tokyo, Japan, 2011. – P. 279–291.

32. Gutz S. Splendid Isolation: A Slice Abstraction for Software-defined Networks / S.Gutz, A. Story, C.Schlesinger, N.Foster // Proceedings of the First Workshop on Hot Topics in Software Defined Networks, (HotSDN ’12), Helsinki, Finland, 2012. – P. 79–84.

33. Haleplidis E.Software-Defined Networking: Experimenting with the Control to Forwarding Plane Interface / E.Haleplidis, S.Denazis, O. Koufopavlou // In Proceedings of the European Workshop on Software Defined Networks (EWSDN). – Darmstadt, Germany, 25–26 October 2012. – P. 91–96.

34. Handigol N. Reproducible network experiments using container-based emulation / N.Handigol, B.Heller, V.Jeyakumar // In Proceedings of the 8th International Conference on emerging Networking Experiments and Technologies - CoNEXT’12, December 10–13, 2012, Nice, France. – P. 253-264.

35. Henzinger T. HyTech: A model checker for hybrid systems / Henzinger T., Ho P.-H., Wong-Toi H. // Computer Aided Verification, Lecture Notes in Computer Science. Vol. 1254. —Springer Berlin Heidelberg, 1997. – Pp. 460–463.

36. Hoare C. A. R. Communicating Sequential Processes / C. A. Hoare // Commun. ACM. – 1978. —Aug. – Vol. 21, no. 8. – P. 666–677.

37. Hoffman L. Talking Model-Checking Technology / L. Hoffman // Communications of the ACM. – 2008. – V. 51. 77. – P. 110–112.

38. Hoffman L. Talking Model-Checking Technology / L. Hoffman // Communications of the ACM. – 2008. – V. 51. 77. – P. 110–112.

39. IBM Systems and Technology. IBM Software Defined Network for Virtual Environments[Электронный ресурс]. – Нью-Йорк, 2013.– 6 с.– Режим доступа:
https://researchweb.watson.ibm.com/haifa/dept/stt/papers/QCW03028USEN.PDF

40. IEEECS 2022 report [Элктронный ресурс] / H. Alkhatib, P. Faraboschi, E. Frachtenberg, H. Kasahara, D. Lange, P. Laplante, A. Merchant, D. MilojicicK. Schwan // IEEEComputerSociety, TechnicalReport. – 2014. – 32 p. – Режим доступа: http://web.cs.wpi.edu/~cs557/f14/papers/CybersecurityInitiative-online.pdf.

41. Infonetics Research, Inc. SDN and NFV Strategies: Global Service Provider Survey [Электронный ресурс]. – 2014 – 39 с. – Режим доступа: http://alu.us.neolane.net/res/img/286758382c7e061c52883e873cee02e6.pdf

42. Isaam Saad A Method of Behavioral Analysis of OpenFlow Protocol / Isaam Saad, Tkachova O. // Scholars Journal of Engineering and Technology, 2016. – Vol. 3(128). – C.92-98.

43. Isaam Saad, Tkachova O. Methods for specification and verification of complex Web-services / Infocommunications Science and Technology, 2014 First International Scientific-Practical Conference Problems of Telecommunication – 2014.

44. IssamS. Mathematical models for analysis Software-defined network / S. Issam,M.J.Salim, Tkachova O. // International Journal "Information Technologies & Knowledge", 2015 - Vol. 9 (2) – p.111-123.

45. Issam SaadVerification method of complex Web-services / S. Issam, O. Tkachova A. Raed // Journal of Future Internet, 2014, 1(1): 1-15

[электроннаяверсия] режимдоступаhttp://www.pakinsight.com/pdf-files/JFI-2014-1%281%29-1-15.pdf

46. ITU-T Recommendation H.248. Series H: Audiovisual and Multimedia Systems: Infrastructure of audiovisual services – Communication procedures: Gateway control protocol[Электронный ресурс]. – Geneva: International Telecommunications Union. – 2000. – Режим доступа: http://www.itu.int/ITU-T/recommendations/index.aspx

47. ITU-T Recommendation SG 11: Scenarios and signalling requirements for software defined BAN (SBAN) - working document[Электронный ресурс]// ITU-T, Tech. Rep. – July 2014. - Режим доступа: http://www.itu.int ITU-T, "Framework of software-defined networking," ITU-T, Tech. Rep., June 2014, recommendation ITU-T Y.3300. – Режим доступа:http://www.itu.int/rec/T-REC-Y.3300-201406-I/en

48. Kang N. Policy Transformation in Software Defined Networks / Kang N., Reich J., Rexford J., Walker D. // Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, (SIGCOMM '12), Helsinki, Finland, 2012. – P. 309–310.

49. Katchabaw M. Administrative Policies to Regulate Quality of Service Management in Distributed Multimedia Applications / M.Katchabaw, H.Lutfiyya, M.Bauer // Management of Multimedia Networks and Services, Lecture Notes in Computer Science. Vol. 2839. —Springer Berlin Heidelberg, 2003. – P. 341–354.

50. Kim H. Improving network management with software defined networking / H.Kim, N.Feamster // Communications Magazine, IEEE, 2013. – P. 114-119.

51. Kotani D., Suzuki K., Shimonishi H. A Design and Implementation of OpenFlow Controller handling IP Multicast with Fast Tree Switching / D.Kotani, K. Suzuki, H.Shimonishi // In Proceedings of the IEEE/IPSJ International Symposium on Applications and the Internet (SAINT), Izmir, Turkey, 16–20 July 2012. – P. 60–67.

52. Lawrence Shari. Software Engineering: Theory and Practice (2nd ed.) / Shari Lawrence. – Upper Saddle River, NJ: Prentice Hall, 2001. – 630 pages.

53. Lipton R. J. Reachability problem requires exponential space: Research Report. List of model checking tools [Электронныйресурс]. – 2015. – Режимдоступа: http://en.wikipedia.org/wiki/List_of_model_checking_tools.

54. Mai H.Debugging of the Data Plane with Anteater / Mai H., Khurshid A., Agarwal R. // Proceedings of the ACM SIGCOMM conference. – 2011. – P. 290-301.

55. Mattos D. OMNI: OpenFlow MaNagement Infrastructure / D. Mattos // In Proceedings of the International Conference on the Network of the Future (NOF), - Paris, France, 28–30 November 2011. – P. 52–56.

56. McClurg J.Efficient Synthesis of Network Updates / McClurg J., Hojjat H., Cerny P., Foster N. //ACM SIGPLAN Conference on Programming Language Design and Implementation,(PLDI'15). – Portland, USA, June 2015. – P. 85–97.

57. Megaco. Megaco Protocol Version 1.0. RFC 3015 / N. Greene, A. Rayhan. – [Действительный от 2000-10-11]. – Ottawa, November 2000 – 179 p.– (RFC 3015)

58. Milner R.Algebras for communicating systems / R.Milner // Proceedings of AFCET/SMF Joint Colloquium in Applied Mathematics. – 1978.

59. Mininet: An Instant Virtual Network on your Laptop (or other PC) [Элекронный ресурс] – Режим доступа: http://mininet.org/

60. Mininet: Introduction to Mininet [Элекронный ресурс] – Режим доступаhttps://github.com/mininet/mininet/wiki/Introduction-to-Mininet

61.

62. Misra J. Proofs of networks of processe / Misra J., Chandy K. // IEEE Transactions on Software Engineering SE-7. – Vol(4). – 1981. – P.417–426.

63. Muller J.P. Modelling Reactive Behaviour in Vertically Layered Agent Architecture / J.P. Muller, M. Pishel, M. Thiel // Intellegent Agents. Proc. of ECAI–94. – Berlin: Springer Verlag , 1994. – P. 261 –276

64. Murata T. Petri Nets: Properties, Analysis and Applications / T. Murata // Proceedings of the IEEE. – April 1989, vol. 77. –P. 541 – 580.

65. Nutt G. Evaluation Nets for Computer Systems PerformanceAnalysis / G. Nutt // FJCC, AFIPSPRESS. – 1972. – P. 279 – 286.

66. OF-CONFIG 1.2. OpenFlow Management and Configuration Protocol [Электронный ресурс] // Open Networking Foundation. – 2014. – Режим доступа: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf

67. OpenFlow Benchmarking [Электронный ресурс] // OpenFlow Wiki. – Режим доступа: http://www.openflow.org/wk/index.php/Oflops

68. OpenFlow Switch Consortium and Others. OpenFlow Switch Specification Version 1.2.0.[Электронный ресурс] // OpenFlow Switch Consortium and Others. – 2011. – Режим доступа: https://www.opennetworking.org/images/stories/downloads/sdn-resources/

69. OpenFlow Switch Specification (Series) [Электронный ресурс]// Open Networking Foundation. – 2014. – Режимдоступа: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow

70. OpenFlow Switch Specification. Version 1.3.0 (Wire Protocol 0x04) [Электронный ресурс]// Open Networking Foundation. – 2012 –Режимдоступа: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

71. Openflowtutorial. [Электронный ресурс]. – 2012. – http://www.openflow.org/wk/index.php/OpenFlow_Tutorial

72. Palmer C.Generating network topologies that obey power laws / C.Palmer, J.Steffan // Proceedings of the Global Telecommunications Conference, (IEEE GLOBECOM ’00). – Vol (1). – 2000. – Pp. 434–438.

73. Paul R, Network S.Virtualization and Software Defined Networking for Cloud Computing: A Survey. IEEE Commun. Mag. 2013, 51, 24–31.

74. PRISM. ProbabilisticSymbolModelChecker [Электронный ресурс]. – Режим доступа: http://www.prismmodelchecker.org/.

75. Process Algebras. Transactions on Petri Nets and Other Models of Concurrency // Lecture Notes in Computer Science. Vol. 5100. – 2008. – Pp. 54–70.

76. Project Floodlight: Open Source Software for Building Software-Defined Networks [Электронный ресурс]. – Режим доступа: http://www.projectfloodlight.org/floodlight/.

77. Quintero D. IBM Software Defined Environment/ Quintero D., Genovese W, Waon K. // IBM Redbook, IBM Corp. – 2015 – 820 с.

78. RajagopalK. Open Networking: Dell's Point of View on SDN[Электронный ресурс] / K. Rajagopal// Dell Inc – 2015. – 17 р.– Режим доступа: http://i.dell.com/sites/doccontent/business/large-business/en/Documents/Dell-Networking-SDN-POV.pdf

79. Randall S. A Guide to Artificial Intelligence with Visual Prolog / S. Randall. – Denver: Outskirts Press, 2010. – 192 p.

80. Reitblatt M. Abstractions for Network Update / M.Reitblatt, N.Foster, J. Rexford, D.Walker // In Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. – Helsinki, Finland, 13–17 August 2012. – P. 323–334.

81. Reitblatt M., Foster N., Rexford J., Walker D. Consistent Updates for Software-defined Networks: Change You Can Believe in! // Proceedings of the 10th ACM Workshop on Hot Topics in Networks, (HotNets-X). – Cambridge, Massachusetts, 2011. – P. 71–76.

82. Relationship of SDN and NFV // Open Networking Foundation [Электронный ресурс]. – Январь, 2015. – Режим доступа: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/onf2015.310_Architectural_comparison.08-2.pdf

83. Ros F. J. Five nines of southbound reliability in software-defined networks / F. J. Ros, P. M. Ruiz // In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, ser. HotSDN '14. – New York, NY, USA: ACM – 2014. – P. 31–36.

84. RTCP. A Transport Protocol for Real-Time Applications Version 1.0. / H. Schulzrinne. – [Действительна от 1996-01-01]. – Berlin, January 1996 — 75 p. – (RFC 1889).

85. RTCP. A Transport Protocol for Real-Time Applications Version 1.0. / H. Schulzrinne. – [Действительна от 2003-07-01]. – NY, July 2003. – 104 p. – (RFC 3550).

86. Salsano S. Information centric networking over SDN and OpenFlow: Architectural aspects and experiments on the OFELIA testbed / S Salsano, N Blefari-Melazzi, A Detti, G Morabito //Computer Networks. – 2013. – P. 79–104.

87. SDN-NFV Reference Architecture v1.0. Verizon Network Infrastructure Planning[Электронный ресурс]// Verizon – 2016 – 220р.– Режим доступа: http://innovation.verizon.com/content/dam/vic/PDF/Verizon_SDN-NFV_Reference_Architecture.pdf

88. Sezer S. Are we ready for SDN? Implementation challenges for Software-Defined Networks / S. Sezer, S. Scott-Hayward, P. Chouhan // Communications Magazine, IEEE, vol. 51, no. 7, July 2013. – P. 36–43.

89. SherwoodR. FlowVisor: A Network Virtualization Layer [Электронный ресурс] / SherwoodR., Gibb G , Yap K. K. , Appenzeller G. , Casado M., McKeown N.// Open Networking Foundation.–CA, USA, 2009. – 18 p. – Режимдоступа: http://archive.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf.

90. Software-Defined Networking. The New Norm for Net works [Электронный ресурс]// Open Networking Foundation. – 2012.–Режимдоступа: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

91. Specification and Description Language (SDL) COM X-R 26CCITT Recommendation Z.100 / Telecommunication Standardization Sector Of ITU — [Действительный от 2000-10-11]. —– 246 p. – (ITU-T Recommendation Z.100)

92. SPINverificationexamplesandexercises [Электронный ресурс]. – Режим доступа: http://spinroot.com/spin/Man/Exercises.html.

93. SPIN. FormalVerification [Электронный ресурс]. – Режим доступа: http://spinroot.com/spin/whatispin.html

94. Sridharan M., Greenberg A. NVGRE: Network Virtualization using Generic Routing Encapsulation [Электронныйресурс] / M. Sridharan, A. Greenberg // Internet Draft, Internet Engineering Task Force. – August 2013. – Режимдоступа: http://tools.ietf.org/id/draftsridharan-virtualization-nvgre-03.txt

95. Stirling C. P. Modal and temporal logics / C. P. Stirling // Handbook of Logic in Computer Science. –Oxford University Press, 1992. – Vol. 2. – P. 477–563.

96. Stirling C. P. Modal and temporal logics for processes / C. P. Stirling // LNCS 1043. – 1996. –P. 149–237.

97. Systems Using Temporal Logic Specifications // ACM Trans. Program. Lang. Syst. – Vol. 8, no. 2. – 1986. – P. 244–263.

98. Tarjan R. Depth first search and linear graph algorithms / R. Tarjan // SIAM Journal on Computing. –Vol. 1.–№ 2. –1972. – P.146–160.

99. TCP. Transmission control protocol / J. Postel. – [Действительна от 1981-09-01]. – California, sept. 1981. – 85 p. – (RFC 793)

100. Tkachova O.B. Method for OpenFlow protocol verification / O.B. Tkachova, Isaam Saad // Second International IEEE Conference «Problems of Infocommunications. Science and Technology» PICS&T-2015, 13-15 October 2015: proc. of the conf.– Kharkiv, Ukraine, 2015. – P.139-140.

101. TR-518 Relationship of SDN and NFV[Электронный ресурс]// Open Networking Foundation. - 2015 - 20 с.– Режим доступа: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/onf2015.310_Architectural_comparison.08-2.pdf

102. Vardi M.Y. An automata-theoretic approach to automatic program verification / M.Y. Vardi, P. Wolper // Proc. of the First Symposium on Logic in Computer Science. –1986. – P. 322-331.

103. Verssimo P.Intrusion-tolerant architectures: Concepts and design / P.Verssimo, N.Neves, M.Correia // In Architecting Dependable Systems, ser. Lecture

Notes in Computer Science, Eds. Springer - Berlin Heidelberg, vol. 2677. – 2003. – P. 3–36.

104. W3C Recommendation. SDL: Concepts and Abstract Syntax [Электронный ресурс]// World Wide Web Consortium. – 2004. – Режим доступа: https://www.w3.org/TR/rdf11-concepts/.

105. Yazici V. Controlling a software defined network via distributed controllers / V.Yazici, M. O. Sunay, and A. O.Ercan // in Proceedings of the Conference on Implementing Future Media Internet Towards New Horizons, ser. 2012 NEM SUMMIT. – Heidelberg, Germany: Eurescom GmbH. – Oct 2012. – P. 16–22.

106. Yeganeh S., On scalability of software-defined networking / Yeganeh S., Tootoonchian A., Ganjali Y. // Communications Magazine, IEEE, vol. 51, no. 2, 2015. – P. 136–141.

107. YuHunag C. A Novel Design for Future On-Demand Service and Security / C. YuHunag, T. MinChi, C. YaoTing, C. YuChieh, C. YanRen // In Proceedings of the International Conference on Communication Technology (ICCT). – Nanjing, China. – 11–14 November 2010. –P. 385–388.

108. Zheng H. Path Computation Element to Support Software-Defined Transport Networks Control / H.Zheng, X.Zhang // Internet Draft (Informational), 2014. Available online: https://datatracker.ietf.org/doc/draft-zheng-pce-for-sdn-transport/.

109. Брукс П. Метрики для управления ИТ-услугами / П. Брукс. – Москва, 2008. – 270 с.

110. Буханько А.Н. Алгоритмы управления каналами связи интеллектуального агента участка сети / А.Н. Буханько, В.М. Безрук, Е.В. Дуравкин // Вісник національного університету «Львівська політехніка». – 2009. – № 645. – С. 68 - 72.

111. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник. / А.М. Вендров – М.: Финансы и статистика, 2000. – 348 с.

112. Гинзбург С. Математическая теория контекстно-свободных языков / С. Гинзбург. – М.: Мир, 1970. – 326 с.

113. Гладкий А. В. Формальные грамматики и языки / А. В. Гладкий. – М.: Наука, 1973. – 368 с.

114. Гольдштейн А. Б. SoftSwitch / А.Б. Гольдштейн, Б. С. Гольдштейн // СПб.: БХВ. – Санкт-Петербург, 2006. – 368 с.

115. Захаров Н. Г. Синтез цифровых автоматов: Учебное пособие / Н. Г. Захаров, В. Н. Рогов. – Ульяновск: УлГТУ, 2003. – 136 с.

116. Иссам С. Метод верификации комплексных Web-сервисов / С. Иссам, Я. А. Раед, О.Б. Ткачева // Проблемителекомунікацій. – 2014. – № 1 (13). – С. 63 - 73. .: http://pt.journal.kh.ua/2014/1/1/141_tkacheva_soa.pdf

117. ИссамСаад Оценка масштабируемости уровня управления в программно-конфигугируемых сетях / ИссамСаад, ТкачеваЕ. Б. // Друк Материалы 1-ой Международной научно-технической конференции «Проблемы электромагнитной совместимости перспективных беспроводных сетей связи ЭМС-2015 - Харьков: ХНУРЭ, 2015. 27 мая 2015.

118. Иссам Саад Применение ASCR для верификации протоколов управления Software-Defined Network / Е.Б. Ткачева, Иссам Саад, Мохаммед ДжамалСалим // Радиотехника: Всеукр. межвед. научн. техн. сб., 2015. – Вип. № 180. – С.48-56.

119. Иссам Саад, Ткачева Е.Б. Архитектура SDN. Анализ основных проблем на пути развития / Е.Б. Ткачева, Е.В. Дуравкин, Иссам Саад // Системи обробки інформації: зб. наук. праць Харківського університету Повітряних Сил, 2015. – Вип. 3(128). – С.92-98.

120. Иссам Саад. Анализ и оценка масштабируемости уровня управления сетей, построенных на основе концепции SDN / Иссам Саад, О.Б. Ткачева, Мохаммед Джамал Салим // Системи обробки інформації збірник наукових праць. – Х.:Харківський університет Повітряних Сил імені І. Кожедуба, 2015 – Вип. 9(134), – 2015 –с.133-137.

121. Иссам Саад. Архитектура SDN. Анализ основных проблем на пути развития / Е.Б. Ткачева, Е.В. Дуравкин, Иссам Саад // Системи обробки інформації: зб. наук. праць Харківського університету Повітряних Сил, 2015. – Вип. 3(128). – С.92-98.

122. Иссам Саад. Метод анализа свойств программно-конфигурируемых сетей на базе аппарата Е-сетей / Материалы 19-го международного молодежного форума «Радиоэлектроника и молодежь в XXI веке». – Часть 4. – Харьков: ХНУРЭ, 2015.

123. Иссам Саад. Методы формального анализа программно конфигурируемых сетей / Иссам Саад, Ткачева Е. Б., Фаиз Мобхот // Всеукраїнськанауково-технічнаконференція «Проблемирозвиткуглобальноїсистемизв'язку, навігації, спостереженнятаорганізаціїповітряногоруху CNS/ATM» – Київ, ІнститутаеронавігаціїНАУ –2014. – с.62

124. Иссам Саад. Применение ASCR для верификации протоколов управления Software-Defined Network / Е.Б. Ткачева, Иссам Саад, Мохаммед ДжамалСалим // Радиотехника: Всеукр. межвед. научн. техн. сб.,2015. – Вип. 180. – С.48-56.

125. Иссам Саад. Применение аппарата Е-сетей для решения задач анализа и верификации программно-конфигурируемых сетей / Е.Б. Ткачева,Иссам Саад, Мохаммед ДжамалСалим // Вісник НТУ «ХПІ». Серія: Інформатика і моделювання, 2015. – Вип. 32 (1141). – С. 148–159.

126. КарповЮ.Г. Model Checking. Верификация параллельных и распределенных программных систем / Ю.Г. Карпов. – СПб.: БХВ-Петербург, 2010. – 552 с.

127. Карпов Ю.Г. Теория автоматов / Ю.Г. Карпов. – СПб.: Питер, 2003. – 208 с.

128. Кларк, Э. М. Верификация моделей программ. Model Checking / Э. Кларк, О. Грамберг, Д. Пелед. – М.: МЦНМО, 2002. – 416 с.

129. Коберн А. Современные методы описания функциональных требований к системам. / А. Коберн.: Пер. с англ. – М.: Издательство ЛОРИ, 2002. – 266 с.

130. Колчин А. Направленный поиск в верификации формальных моделей / А. Колчин // Теоретичні та прикладні аспекти побудови програмних систем TAAPSD'2007 – Бердянск. НаУКМА, Національний ун-т ім. Т.Г. Шевченка, Ін-т програмних систем НАН України. – 2007. – С. 256–258.

131. Коровченко Е.Б. Модели и методы анализа и верификации телекоммуникационных протоколов на основе Е-сетей и формальных грамматик / Диссертация на соискание ученой степени кандидата технических наук: 05.12.02. – Х. 2011, – 168 с.

132. Котов В.Е.Сети Петри / В.Е. Котов. – М.: Наука, 1984. – 160с.

133. Красотин А. А. Программно-конфигурируемые сети как этап эволюции сетевых технологий / А. А. Красотин, И. В.Алексеев // Моделирование и анализ информационных систем. – 2013. – Т. 20. – №. 4. – С. 110-124.

134. Кривый С.Л. Формальные методы анализа свойств систем / С.Л. Кривый, Л.Е. Матвеева // Кибернетика и системный анализ. —№2. – 2003.— С.15-36.

135. Кулямин, В.В. Методы верификации программного обеспечения / В.В. Кулямин // Всероссийский конкурсный отбор обзорно-аналитических статей по приоритетному направлению «Информационно-телекоммуникационные системы», 2008. – 117 с.

136. Липаев В. В. Обеспечение качества программных средств. Методы и стандарты / В. В. Липаев. – М.: Синтег, 2001. – 380 с.

137. Лосев Ю.И. Применение методов анализа Е-сетей к моделям СОД / Ю.И. Лосев, С.И. Шматков, Е. В. Дуравкин // Радиотехника: Всеукр. межвед. науч.-техн. сб. – 2002. – Вып.132. – С. 149 - 151.

138. Мещеряков С. В. Эффективные технологии создания информационных систем / С. В. Мещеряков, В. М. Иванов. – К.: Политехника, 2005. – 312 с.

139. Пентус А. Е., Пентус М. Р. Теория формальных языков: Учебное пособие / А. Е. Пентус, М. Р. Пентус. – М.: Издательство ЦПИ при механико-математическом ф-те МГУ, 2004. – 80 с.

140. Питерсон Дж. Теория сетей Петри и моделирование систем / Дж. Питерсон. – М.: Мир, 1984. – 264с.

141. Протокол управления мультимедийным шлюзом: MediaGatewayControlProtocol (MGCP) / M. Arango, A.Dugan, I. Elliot, C.Huitema, S.Pickett. – Ft. Lauderdale — [Действительный от 2009-10-10]. —– 134 p. – (RFC 2705)

142. Салман Амер Мусхин. Применение методов анализа Е-сетей к моделям СОД / Амер Мусхин Салман // Радиотехника: Всеукр. межвед. научн. техн. сб. 2008. – Вып. 155. – С. 159-163.

143. Смелянский Р. Программно-конфигурируемые сети [Електронный ресурс] / Р.Смелянский // Открытые системы. – 2012. – № 9. – Режим доступа:www.osp.ru/os/2012/09/13039421/.

144. Смелянский Р.Л. Программно-конфигурируемые сети [Електронный ресурс] / Р. Л. Смелянский // Открытые системы. – 2012. – № 9. – Режим доступу до статті: www.osp.ru/os/2012/09/13039421/.

145. Смелянский Р.Л. Технология программно-конфигурируемых сетей и виртуализация сетевых сервисов: новые возможности для телекоммуникаций / Р.Л. Смелянский // Центр прикладных исследований компьютерных сетей [Электронный ресурс] – режим доступа: http://arccn.ru/knowledge-base?pdf=53b3d96b0c961.pdf от 30 мая 2016

146. Советов Б.Я. Моделирование систем / Б.Я. Советов, С.А. Яковлев. – М.: Высш. шк., 1985. – 271 с.

147. Создание прототипа отечественной ПСК платформы управления сетевыми ресурсами и потоками с посощью сетевой операционной системы

(СОС) на основе анализа и оценки существующих сетевых операционных систем ПСК сетей. Отчёт о НИР (Заключительный) [Электронный ресурс] / Рук. Смелянский Р.Л. // МГУ им. Ломоносова. – Москва, 2014. – Режим доступа: http://en.cs.msu.ru/sites/cmc/files/scproj/4047_otchet_nir1_1.pdf

148. Создание прототипа отечественной ПСК платформы управления сетевыми ресурсами и потоками с посощью сетевой операционной системы (СОС) на основе анализа и оценки существующих сетевых операционных систем ПСК сетей. Отчёт о НИР (Заключительный) [Электронный ресурс] / Рук. Смелянский Р.Л. // МГУ им. Ломоносова. – Москва, 2013. – 252 с. – Режим доступа: http://en.cs.msu.ru/sites/cmc/files/scproj/4047_otchet_nir1_0.pdf

149. Стивенс У. Р. UNIX: разработка сетевых приложений / У. Р. Стивенс. – М.; СПб.: Питер, – 2003. – 1088 с.

150. Таненбаум Э. Распределенные системы. Принципы и парадигмы. / Таненбаум Э., М. ван Стеен. – СПб.: Питер, 2003. – 877с.

151. Ткачева Е.Б. Применение аппарата Е-сетей для решения задач анализа и верификации программно-конфигурируемых сетей / Е.Б. Ткачева,Иссам Саад, Мохаммед ДжамалСалим // Вісник НТУ «ХПІ». Серія: Інформатика і моделювання, 2015. – Вип. 32 (1141). – С. 148–159.

152. Фатрелл Р. Управление программными проектами. Достижение оптимального качества при минимуме затрат / Р. Фатрелл, Д. Шафер, Л. Шафер.: Пер. с англ. – М.: Вильямс, 2004. – 1136 с.

153. Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования / М. Фаулер, К. Скотт.: Пер. с англ. – М.: Мир, 1999. – 192 с.

154. Хоар Ч. Взаимодействующие последовательные процессы / Ч. Хоар.: Пер. с англ. – М.: Мир, 1989. —264 с.

155. Чистяков Г.А. Метод и машина логического вывода для формальной верификации параллельных алгоритмов / Диссертация на соискание ученой степени кандидата технических наук: 05.13.11, 05.13.15. – Киров, 2014. – 221 с.

156. Чистяков, Г.А. Формирование контрпримера при верификации алгоритмов с помощью методов логического вывода / Г.А. Чистяков // Вестник Астраханского государственного технического университета. Серия: управление, вычислительная техника и информатика. – №3. – 2014. – С. 50-57.

157. Шварц М. Сети связи: протоколы, моделирование и анализ / М. Шварц.: Пер с англ. – М.: Наука, 1992. – 336 с.

«ЗАТВЕРДЖУЮ»

Перший проректор
Харківського національного
університету радіоелектроніки
_____ Ключник І.І.
"___" _____ 2016 р.

АКТ

про впровадження в навчальний процес
Харківського національного університету радіоелектроніки
результатів дисертаційної роботи аспіранта кафедри
«Телекомунікаційні системи» Іссам Саада
на тему«Моделі і методи аналізу та верифікації протоколів управління у програмно–конфігурованих мережах, що базуються на алгебрі комутаційних розподілених ресурсів та графах досяжності»

Комісія у складі: завідувача кафедрою «Телекомунікаційних систем» професора, д.т.н. Поповського В.В., професора, д.т.н. Агєєва Д.В., декана факультету ТКВТ, доцента кафедри «Телекомунікаційні системи» к.т.н. Снігурова А.В., підтверджує, що результати дисертаційної роботи Іссам Саада на тему «Моделі і методи аналізу та верифікації протоколів управління у програмно–конфігурованих мережах, що базуються на алгебрі комутаційних розподілених ресурсів та графах досяжності» використовуються в навчальному процесі Харківського національного університету радіоелектроніки на кафедрі «Телекомунікаційних систем та мереж», а саме:

- наукові положення і результати дисертаційної роботи були використані при підготовці лекційних курсів із дисциплін «Інформаційні системи та Інтернет» та «Менеджмент та оптимізація телекомунікаційних систем та мереж», які викладаються на кафедрі «Телекомунікаційні системи».

В.В. Поповський

Д. В. Агєєв

А.В. Снігуров

«ЗАТВЕРДЖУЮ»

Перший проректор
Харківського національного
університету радіоелектроніки
_____ Ключник І.І.
"___" _____ 2016 р.

АКТ

про впровадження результатів дисертаційної роботи Іссам Саада за темою «Моделі і методи аналізу та верифікації протоколів управління у програмно–конфігурованих мережах, що базуються на алгебрі комутаційних розподілених ресурсів та графах досяжності», представлену на здобуття наукового ступеня кандидата технічних наук за спеціальністю 05.12.02 – телекомунікаційні системи та мережі.

Комісія у складі: д.т.н., проф., проф. каф. ТКС Євсєєвої О.Ю., наукового керівника НДР № 299-1; к.т.н., доц. каф. ТКС Ткачової О.Б., відповідального виконавця НДР № 299-1; к.т.н., с.н.с., доц. каф. ТКС Єременко О.С., виконавця НДР № 299-1 підтверджує, що ряд наукових та практичних результатів дисертаційної роботи Іссам Саада на тему «Моделі і методи аналізу та верифікації протоколів управління у програмно–конфігурованих мережах, що базуються на алгебрі комутаційних розподілених ресурсів та графах досяжності» використані в держбюджетній НДР № 299-1 «Підвищення масштабованості технологічних рішень щодо забезпечення якості обслуговування в конвергентних телекомунікаційних системах». Зокрема, було використано наступні результати:

- математична модель фрагменту програмно-конфігурованої мережі та методи аналізу розподілення мережевих ресурсів, зокрема модифікований метод побудови графів досяжності, що дозволило вирішити задачу перевірки збереження функціональних та нефункціональних властивостей протоколу OpenFlow;
- узагальнена методика аналізу та верифікації протоколу OpenFlow. Рекомендації та доповнення кожного етапу розробки, запропоновані у дисертаційній роботі, дозволили підвищити ефективність процесу розробки та впровадження.

Вказані результати використані в процесі наукових та експериментальних досліджень при виконанні держбюджетної НДР № 299-1 (№ ДР 0115U002432), в яких автор дисертації був виконавцем.

Голова комісії О.Ю. Євсєєва

Члени комісії О.Б. Ткачова

О.С. Єременко