МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ М.В. ЛОМОНОСОВА

На правах рукописи

ЧЕМЕРИЦКИЙ ЕВГЕНИЙ ВИКТОРОВИЧ

ИССЛЕДОВАНИЕ МЕТОДОВ КОНТРОЛЯ ФУНКЦИОНИРОВАНИЯ ПРОГРАММНО-КОНФИГУРИРУЕМЫХ СЕТЕЙ

Специальность 05.13.11 —
Математическое обеспечение вычислительных машин,
комплексов и компьютерных сетей

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
д.ф.-м.н., член-корр. РАН
Смелянский Р.Л.

Москва – 2015

Содержание

Введение

Вклад глобальной компьютерной сети Интернет в развитие телекоммуникационных технологий сложно переоценить. Быстрый и точный поиск по огромному количеству доступной в сети информации, удобное общение с помощью электронной почты, интерактивных текстовых, аудио и видео конференций и социальных сетей, доступ к удалённым вычислительным ресурсам и хранилищам данных – эти и другие возможности глобальной сети привлекали, и продолжают привлекать в неё новых и новых пользователей. По состоянию на начало 2015 года к Интернет подключено порядка 40% жителей планеты, и это единственная в своём роде компьютерная сеть столь внушительного размера.

Одним из объяснений отсутствия других сетей сравнимого масштаба может служить эмпирический закон Меткалфа [1]. Пытаясь убедить покупателей персональных компьютеров укомплектовывать их сетевыми картами, Роберт Меткалф, прародитель широко распространённого сегодня стандарта Ethernet, заметил, что каждый из объединённых локальной сетью компьютеров может получать доступ к дорогостоящему оборудованию, подключённому к другим компьютерам сети. Поскольку сеть из n машин предоставляет каждой из них возможность использовать ресурсы других $n-1$ машин, учёный предложил оценивать суммарную ценность сети произведением $n(n-1)$, что асимптотически пропорционально квадрату n^2 от количества её абонентов. Несмотря на то, что масштабы современных сетей могут значительно отличаться от масштабов локальных сетей, в терминах которых рассуждал Меткалф, выведенная им зависимость продолжает сохранять свою актуальность [2]. Таким образом, связывание одного и того же количества абонентов с помощью одной крупной сеть значительно выгоднее, чем с использованием нескольких независимых сетей меньшего масштаба.

Благодаря своему уникальному положению и стремительному росту сеть Интернет уже долгие годы является основной движущей силой прогресса в области сетевых технологий. Практически на всём протяжении эволюционного развития компьютерных сетей наибольшее внимание получали исследования, результаты которых могли найти наиболее

широкое практическое применение – то есть, были так или иначе приспособлены к использованию в условиях сети Интернет. Фундаментальные исследования, предлагавшие альтернативные способы построения компьютерных сетей, не проходили естественный отбор, вызывая значительно меньшую заинтересованность как в научном сообществе, так и в индустрии. Таким образом, те идеи, которые были заложены в основу сети Интернет, во многом определили облик современных компьютерных сетей, оказав значительное влияние на правила их проектирования, внутреннее устройство и принципы работы коммутационного оборудования, методы и средства их администрирования.

В попытке разъяснить архитектурные решения, заложенные в сеть Интернет, Дэвид Кларк, возглавлявший разработку стека протоколов TCP/IP с 1981 по 1988 годы, сформулировал список из семи основополагающих принципов [3], которыми руководствовались её создатели:

1. *Отказоустойчивость.* При возникновении отказа сеть должна автоматически восстанавливать свою работоспособность, не вынуждая взаимодействующих с её помощью агентов заново устанавливать соединения. В частности, при выходе из строя коммуникационного устройства, сеть должна попытаться соединить отправителей и получателей проходящих через него потоков данных с помощью альтернативных маршрутов;

2. *Разнообразие сервисов передачи данных.* Разные сетевые приложения предъявляют разные требования к характеристикам соединения: пропускной способности, максимальной задержке и надёжности при передаче данных. Причём приёмы, используемые для выполнения разных требований, могут противоречить друг другу. Например, обеспечение надёжности путём повторной передачи потерянных данных увеличивает задержку. Таким образом, сеть должна поддерживать несколько разных типов сервиса;

3. *Поддержка широкого диапазона сетей.* Интернет должен объединить множество самых разных сетей, построенных с разными целями (военные и гражданские), обладающих разными скоростями передачи и задержками (проводные и спутниковые) и способных передавать данные порциями разной длины;

4. *Распределённое управление сетевыми ресурсами.* Интернет должен обеспечить возможность обмена данными между пользователями, подключёнными к разным сетям.

Однако, поскольку разные сети могут принадлежать разным организациями, Интернет должен сохранять за этими организациями право их независимого управления;

5. *Рентабельность.* Подключение к сети Интернет должно обладать низкой стоимостью. Интернет не должен предоставлять дополнительных услуг, если их реализация повышает стоимость его обслуживания, но не требуется для достижения поставленных перед его разработчиками целей;

6. *Расширяемость.* Интернет должен предоставлять возможность простого подключения к сети. Таким образом, сеть должна предъявлять как можно меньше требований к аппаратуре и программному обеспечению, запущенному на подключенных к ней машинах. Согласно этому принципу, например, прокладывание маршрута передачи данных должно быть реализовано не их отправителем, а инфраструктурой сети;

7. *Учёт использованных сетевых ресурсов.* Интернет должен предоставлять возможности для сбора статической информации по количеству переданных пользователями данных и количеству сетевых ресурсов, которые были на это затрачены.

Кларк отмечает, что принципы из приведённого списка располагаются в порядке уменьшения их значимости. Если при принятии какого-нибудь архитектурного решения принципы вступали друг с другом в противоречие, то утверждалось решение, удовлетворяющее принципу с наибольшим приоритетом. Изменение порядка принципов могло бы привести к созданию совершенно другой сети. Так как сеть Интернет изначально разрабатывалась для военных, то наибольший приоритет отдавался живучести сети – она должна была сохранять работоспособность даже при серьёзных повреждениях. Если бы сеть разрабатывалась для коммерческого использования, наиболее важным могла бы стать необходимость учёта использованных ресурсов, и компьютерные сети, возможно, были бы лучше приспособлены к выполнению требований современных пользователей.

Наивысший приоритет принципа отказоустойчивости привёл к тому, что в современных компьютерных сетях не существует единого центра управления, который позволял бы однозначно определить, по каким *правилам обработки* будет обслуживаться конкретный пакет: через какие выходные порты сети будут направлены его копии, какими заголовками они будут обладать, сколько пакетов, принадлежащих к одному потоку данных, могут быть переданы в единицу времени. Выход из строя такого центра управления привёл бы к полной неработоспособности сети, что было недопустимым. Вместо этого сети строятся на базе множества коммутационных устройств с высокой степенью автономности – каждое

такое устройство может работать независимо от других устройств, самостоятельно определяя правила, по которым оно будет обрабатывать поступающие на него пакеты. Таким образом, при отказе некоторых компонентов сети коммутационные устройства, оставшиеся в рабочем состоянии, смогут сохранить её работоспособность с помощью выбора надлежащих правил обработки пакетов.

Современные коммутационные устройства вырабатывают свои правила коммутации пакетов по заложенным в них алгоритмам, значительная часть которых, зачастую, реализована в железе, и не может быть изменена. Работа наиболее примитивных алгоритмов сводится к выбору того или иного набора предустановленных инструкций в зависимости от некоторых свойств обрабатываемого пакета. Например, списки доступа, запрещающие передачу пакетов, заголовки которых соответствуют заданным шаблонам, как правило, известны уже на этапе развёртывания сети, и могут быть заданы статически. В общем же случае вырабатываемые устройством инструкции зависят не только от его настроек и свойств поступающих на него пакетов, но и от контекста, в котором оно работает – его сетевого окружения. Именно анализ окружения позволяет устройствам осмысленно изменять свои правила обработки пакетов таким образом, чтобы передавать их в обход вышедших из строя компонентов сети, или вырабатывать подходящие правила для обслуживания вновь подключившихся абонентских машин, адреса которых неизвестны заранее.

Каждое устройство имеет собственное *локальное представление сети* и использует сразу несколько способов, чтобы поддерживать его в актуальном состоянии. Они наблюдают за поведением соседних узлов, отслеживая их подключение и отключение по электрическому сигналу на соответствующих линиях связи, или же анализируя поступающий через эти линии пользовательский трафик. Например, самообучающийся коммутатор использует наблюдение для того, чтобы определить соответствие между адресами подключённых к нему устройств и номерами портов, через которые с ними можно связаться. Однако большинство задач обслуживания пакетов в масштабах сети требуют координированной работы множества устройств и, как правило, не могут быть решены с помощью одного лишь пассивного наблюдения. Для обеспечения необходимого поведения сети отдельные устройства должны явным образом обменяться информацией о своих настройках, каждое из них должно поддерживать представление о своём окружении в согласованном состоянии и использовать совместимые алгоритмы выработки правил передачи пакетов.

Правила обмена релевантной информацией между устройствами и способы её применения для выработки согласованных правил обработки пакетов определяются множеством разнообразных *служебных протоколов*, каждый из которых ставит своей целью решение

некоторого подмножества задач администрирования сети. Например, протоколы маршрутизации позволяют устройствам динамически вырабатывать такие правила обработки пакетов, чтобы поступающие в сеть потоки трафика передавались через неё по наиболее эффективным маршрутам. Протоколы управления группами способны подключать абонентские машины к ведущимся в сети вещательным трансляциям и обеспечить доставку копий пакетов трансляции на каждую из выбравших её машин. С помощью протоколов резервирования можно устанавливать квоту ресурсов, выделенных на обслуживание каждого конкретного потока, и заставить коммутационные устройства сбрасывать пакеты, при превышении заявленных квот.

Подобные протоколы отчасти компенсируют отсутствие единого центра управления, автоматически адаптируя поведение сети, построенной на базе независимых коммутационных устройств, под разнообразные изменения в её конфигурации и, тем самым, существенно снижая объём работы сетевых инженеров. Однако, использование множества служебных протоколов отнюдь не делает сетевое администрирование простым. Сложившаяся архитектура сетей вынуждает инженеров управлять поведением сети не напрямую, а опосредовано. Устанавливая те или иные настройки, они вынуждены использовать абстракции и интерфейсы сразу нескольких протоколов, просчитывать последствия их взаимодействия между собой и предвосхищать правила обработки, которые будут выработаны отдельными устройствами для обслуживания поступающих в сеть пакетов. Чтобы определить судьбу конкретного пакета инженерам приходится преодолевать сразу несколько *уровней косвенности* устоявшейся системы управления сетью, переходя между абстракциями уровней её стека:

1. *Настройки устройств* $\rightarrow$ *Поведение протоколов.*

   За годы развития распределённых систем было разработано множество алгоритмов, позволяющих наделить компьютерные сети способностями для самоорганизации и адаптации под окружение, в котором они оказались. Однако, даже самые совершенные алгоритмы автоматического конфигурирования коммутационных устройств не в состоянии использовать особенности конкретной компьютерной сети, и в полной мере учитывать те требования, которые к ней предъявляются. Поэтому современные служебные протоколы можно рассматривать как реализации эвристических алгоритмов для построения как можно более эффективных правил обработки пакетов: каждый из них может применяться к широкому диапазону сетей, однако в некоторых

случаях использование эвристики может приводить к не самому лучшему решению, в некоторых – к решению, которое и вовсе нарушает ограничения задачи.

Для решения указанной проблемы сетевым администраторам часто приходится подгонять поведение служебных протоколов под конкретную сеть с помощью множества низкоуровневых настроек. Такие настройки, как правило, специфичны для каждого протокола, и их осмысленное изменение требует детального понимания используемых им абстракций и механики его работы. Например, протокол STP [4] разрывает циклы, превращая топологию сети в остовное дерево – связный ациклический подграф, содержащий все вершины исходного графа. Корню построенного остовного дерева соответствует коммутатор с наименьшим номером, а ветвям – кратчайшие пути между выбранным корнем и остальными коммутаторами. Протокол предоставляет возможность изменения номера для каждого из коммутаторов вручную, однако, зависимость между указанными опциями и множеством отключённых протоколом линий связи нетривиальна;

2. *Поведение протоколов → Локальное представление сети.*

   В формировании локальных представления о сетевом окружении, которые поддерживаются каждым коммутационным устройством, как правило, принимает участие сразу несколько служебных протоколов. При обнаружении изменения локального представления одного из устройств, протоколы пытаются привести сеть к согласованному состоянию, внося необходимые правки в локальные представления других устройств. Нередко это вызывает новую волну обновлений, в том числе, и посредством других протоколов. Таким образом, в процессе настройки оборудования может возникнуть так называемый *эффект бабочки* [5], когда даже небольшое изменение в конфигурации одного из компонентов сети приводит к значительным переменам в её поведении. В результате, вычисление конфигураций устройств, соответствующих стабильному состоянию сети, может потребовать просчёта множества замысловатых сценариев их взаимодействия между собой.

   Неочевидные зависимости, порождённые множеством связывающих их служебных протоколов, зачастую приводят к необходимости поиска сложных обходных решений даже для стандартных инженерных задач. Поскольку количество переданного и принятого трафика для разных абонентов, как правило, различается, то наиболее выгодные прямые и обратные маршруты между абонентами нередко проходят через разные узлы сети. При известном профиле трафика логичной представляется

возможность оптимизировать существующие пути передачи данных, заставив протоколы маршрутизации использовать разные пути для передачи данных в разные стороны. Однако, при несовпадении прямых и обратных путей протоколы многоадресной передачи, могут построить неэффективное дерево для тиражирования пакетов, тем самым, сведя попытку оптимизации на нет;

3. *Локальное представление сети → Правила обработки пакетов.*

   Информация, заданная при начальной конфигурации устройства и накопленная им в процессе анализа своего сетевого окружения, представляется в виде множества разнообразных таблиц. Например, таблица маршрутизации сопоставляет адресам из указанного диапазона адрес следующего узла на пути к образованной ими подсети, а таблица классификации по типам обслуживания связывает значения метки качества сервиса с определённым приоритетом обработки относительно пакетов с метками других типов. Таким образом, обработка пакета сводится к последовательному просмотру указанных таблиц, поиску в них подходящих ему записей, и применению построенных по этим записями элементарных инструкций.

   Несмотря на кажущуюся простоту описанного подхода к выводу правил обработки пакетов, определить конкретные действия, которые будут предприняты коммутатором при обслуживании заданного пакета, бывает затруднительно. Для поиска в таблицах часто используются признаки, которые тяжело анализировать непосредственно. Сетевые инженеры регулярно допускают ошибки в битовых масках списках доступа, блокируя легитимный трафик или, наоборот, допуская прохождение запрещённого трафика и, тем самым, приводя к угрозе безопасности. Поскольку применение инструкций из одной таблицы способно повлиять на выборку записей из последующих таблиц, то инженеры вынуждены учитывать порядок просмотра таблиц, отслуживая изменения контекста пакета вручную. Отсутствие стандартных низкоуровневых интерфейсов для манипуляций над правилами обработки не позволяет инженерам абстрагироваться от внутреннего устройства коммутационных устройств, различные модели и марки которых построены на базе разных аппаратных решений, используют разное количество и типы таблиц, просматривают их в разном порядке и имеют собственные представления для одних и тех же данных.

4. *Правила обработки пакетов → Поведение сети.*

На заключительном этапе анализа конфигурации инженерам необходимо построить суперпозицию правил обработки для каждого из коммутационных устройств сети и определить, как будут обслуживаться пакеты в масштабах сети. По указанным правилам они делают выводы об особенностях и характерных свойствах, которыми она обладает: какие пакеты будут доставлены их адресатам, а какие – сброшены, какова максимальная интенсивность потока данных, которую способна выдержать сеть в указанном направлении, какие коммутационные устройства сети являются её узкими местами и требуют замены на более производительные аналоги.

Полученная информация, тем не менее, не позволяет судить о множестве аспектов обслуживания пакетов, которые зависят не только от установленных правил передачи пакетов, но и, например, от загруженности сети. К свойствам указанного типа относятся, в частности, доля пропускной способности канала, доступная для использования каждому из проходящих через неё потоков данных, и *сквозная задержка* обработки пакета – время, которое сеть затрачивать на передачу пакета через свою инфраструктуру. Расписание передачи через каналы и сквозная задержка каждого конкретного пакета зависит от множества пакетов-соперников, конкурирующих с ним за доступ к вычислительным ресурсам сети.

Между тем, перед инженерами чаще ставится не задача предсказания выбора правил обслуживания пакетов по заданному набору настроек для её коммутационных устройств, а более сложная, обратная задача: предложить такие настройки отдельных устройств, при которых сеть удовлетворяла бы предъявляемым к ней требованиям – *политикам маршрутизации пакетов*. Приведём некоторые примеры типичных политик маршрутизации:

1. Сеть должна обеспечивать своих пользователей теми услугами, для оказания которых она была построена:
   - Пользователи должны иметь возможность использования сетевого принтера из любой точки офиса;
   - Приватная сеть компании должна обеспечивать соединения надлежащего качества для голосовой или видео связи между отделами компании;
   - Соединение с хранилищем данных должно иметь повышенную надёжность;
2. Действующие в сети правила обслуживания пакетов должны удовлетворять различным административным ограничениям, продиктованных, например, требованиями безопасности компании-владельца сети:

- Инженеры не должны допускать возможность прямого подключения к внутренним почтовым серверам компании из внешних сетей;
- Весь трафик, передающийся между офисами компании через внешние сети, должен подвергаться обязательному шифрованию;
- При возникновении перегрузок трафик привилегированных пользователей должен получать большее количество сетевых ресурсов;

3. Сеть должна использовать доступные вычислительные ресурсы эффективно:
   - Зацикливание пакетов, приводящее к их многократной обработке на коммутационных устройствах, должно быть исключено;
   - Обслуживание каждого пакета должно задействовать как можно меньшее количество коммутационных устройств – сеть должна стремиться передать его по кратчайшему маршруту;
   - Коммутационные устройства не должны работать «в холостую» – сеть должна предусматривать возможности для временного отключения части коммутационных устройств при низкой загруженности сети.

Политики маршрутизации, действующие в разных сетях, порой значительно отличаются друг от друга: те качества, которыми непременно должна обладать одна сеть, зачастую, могут быть недопустимыми для другой. Отдельные политики маршрутизации формулируются в разных терминах и, зачастую, способны вступать друг с другом в неявные противоречия, или же вовсе быть несовместимыми друг с другом. Уровень абстракции политик маршрутизации, как правило, существенно превосходит уровень обслуживания отдельных пакетов на коммутационных устройствах, поэтому одна и та же политика может быть реализована множеством способов.

При современном уровне развития технологий автоматический синтез настроек коммутационного оборудования по множеству политик маршрутизации оказался слишком трудным для механического решения. Существующие инструментальные средства не могут учесть всё многообразие политик маршрутизации, обнаружить и разрешить противоречия между ними, и предложить эффективную конфигурацию сети, которая бы им удовлетворяла. Поэтому сетевым инженерам приходится самостоятельно транслировать высокоуровневые политики маршрутизации в соответствующие им настройки оборудования, фактически изменяя эвристики служебных протоколов под условия работы в заданной сети.

Построение множества настроек коммутационного оборудования вручную – тяжёлая и утомительная работа. Поэтому были предприняты неоднократные попытки если не автоматизировать процесс построения таких настроек, то, по-крайней мере, решить более простую задачу и построить аналитические инструментальные средства для обнаружения некоторых классов допущенных при настройке ошибок ещё до тестирования на реальной сети. Например, средство FlowChecker [6] предоставляет возможности для проверки множества правил обслуживания пакетов на соответствие произвольному множеству требований достижимости. Средство Margrave [7] позволяет обнаруживать несоответствие между двумя различными конфигурациями сетевого экрана и, тем самым, позволяет убедиться в корректности проведённых оптимизаций. Существует средство автоматизированной проверки конфигурации устройств, взаимодействующих по протоколу BGP, относительно произвольных политик маршрутизации, заданных выражениями специализированного формального языка [8].

Подобные средства анализа конфигураций предлагают инженерам специфицировать требования, которые предъявляются к сети на некотором формальном языке, позволяющем охватить подмножество политик маршрутизации. Как правило, подобные языки используют абстракции, в которых сравнительно просто выражаются такие стандартные требования к сети, как достижимость между отдельными её узлами и правила перераспределение трафика при возникновении перегрузок. Соответствие конфигурации сети заданным требованиям проверяют путём анализа свойств некоторой её модели. Обычно в качестве такой модели выступает совокупность правил, которые применяются отдельными коммутационными устройствами при обслуживании пакетов – такой низкий уровень абстракции позволяет судить о поведении сети безотносительно множества функционирующих в ней служебных протоколов и их настроек. Если свойства сети, представленной определённым набором правил обслуживания пакетов, в точности соответствуют тем требованиям, которые к ней предъявляются, то средство заключает, что ошибок в заданной конфигурации нет. В противном случае, делается вывод о наличии ошибки.

Долгое время развитие подобных средств существенно сдерживалось сложившейся архитектурой сети. Во-первых, их разработчикам приходилось искать способы для получения глобального состояния сети, каждое коммутационное устройство в составе которой способно асинхронно изменять свои правила обслуживания пакетов независимо от остальных устройств. Во-вторых, построение глобального состояния сети в терминах правил обслуживания пакетов требовало средств автоматического синтеза указанных правил из настроек множества взаимодействующих между собой служебных протоколов, что, как

это было показано выше, не является тривиальной задачей. Однако, решение указанных проблем значительно упростилось вместе со сравнительно недавним появлением концепции Программно-Конфигурируемых Сетей (ПКС) [9].

Сети указанного вида пересматривают приоритеты перечисленных Кларком базовых принципов построения компьютерных сетей, делая больший упор не на повышении их надёжности, а на упрощении управления. ПКС лишают свои коммутационные устройства способности самостоятельно вырабатывать правила обработки пакетов. Устройства обслуживают пакеты лишь по тем правилам, которые были загружены в них *контроллером* – своеобразной операционной системой сети, отвечающей за согласованное управление множеством её устройств. Тем самым, концепция ПКС отделяет *контур управления* (Control Plane), осуществляющей выработку надлежащих правил обслуживания пакетов, и *контур данных* (Data Plane), занимающегося непосредственным применением этих правил к проходящим через сеть пакетам трафика.

Контроллер представляет собой, вообще говоря, распределённую программу, которая имеет доступ к каждому коммутационному устройству сети и предоставляет единый программный интерфейс (API) для централизованного управления этими устройствами. Поверх контроллера может быть запущено множество *управляющих приложений*, которые, подобно служебным протоколам в сетях с традиционной архитектурой, призваны адаптировать поведение сети к её окружению.

Централизация ПКС избавила разработчиков анализаторов конфигураций сети от необходимости самостоятельно собирать слепок её глобального состояния – контроллер осуществляет централизованное управление всеми коммутационными устройствами сети, и заведомо способен предоставить всю необходимую для этого информацию. Одновременно с этим явное выделение контура данных позволило получить доступ непосредственно к правилам обработки пакетов, которые в сетях с традиционной архитектурой приходилось выводить на основании настроек служебных протоколов самостоятельно. Таким образом, концепция ПКС дала мощный толчок к разработке новых и развитию существующих методов анализа и проверки свойств сети по её заданной конфигурации. Исследованиям в указанном направлении и посвящена настоящая работа.

Целью данной работы является исследование и разработка методов и средств для оценки свойств ПКС по известной конфигурации её компонентов и проверке соответствия этих свойств предъявляемым к сети требованиям.

Любое свойство ПКС можно условно отнести к одному из двух типов [10]: *логическим свойствам* (qualitative) или *свойствам производительности* (quantatitative). К логиче-

ским относятся такие свойства, для описания которых не требуются численные измерения. Как правило, такие свойства исследуют причинно-следственные связи между происходящими в сети событиями. Применяемая при их описании абстракция логических часов [11] позволяет зафиксировать порядок событий, но не даёт возможности определить интервал между этими событиями: для логических свойств не имеет принципиального значения, произойдёт ли следующее событие в течение последующей секунды или года.

Свойства производительности, напротив, исследуют численный характеристики ПКС: они не могут быть выражены одними лишь логическими формулами и требуют интенсивного использования арифметических операций. Для свойств данного типа привязка событий к шкале времени, как правило, имеет принципиальное значение. Без прямой зависимости между произошедшими в сети событиями и временем нельзя определить скорость передачи данных, среднюю загруженность коммутационных устройств или длительность восстановления сети после сбоя.

В настоящей работе рассматривается по одному классу базовых свойств ПКС для каждого из перечисленных типов: свойства достижимости и максимальная сквозная задержка передачи пакета через сеть, соответственно. Свойства достижимости здесь понимаются в широком смысле – они описывают возможности пакетов изменять свои *состояния*, перемещаясь между отдельными элементами сети и модифицируя свои заголовки. Таким образом, открывается возможность исследования не только маршрутов передачи пакетов через топологию сети, но и, например, способности сети сохранить оригинальные заголовки пакетов при их доставке получателю и не допустить смешивания трафика от разных групп пользователей.

Такие свойства достижимости, как отсутствие безусловного сброса пакетов легитимного трафика или циклической передачи пакетов между коммутационными устройствами, соответствуют политикам маршрутизации, которые действуют в подавляющем большинстве сетей. Однако количество подобных общепринятых политик маршрутизации сравнительно мало: значительно большее их число специфично для конкретной сети, и не может быть задано заранее. Поэтому в настоящей работе рассматривается в некотором смысле универсальный метод исследования свойств достижимости, который позволил бы сетевым инженерам проверять конфигурации ПКС на соответствие произвольным требованиям к свойствам из данного класса. Для построения метода, обладающего указанными качествами, предполагается создать специализированный язык спецификации произвольных свойств достижимости, а так же разработать механизмы для автоматической проверки конфигурации ПКС на соответствие произвольным спецификациям этого языка.

Второй класс свойств изучает своевременность доставки передаваемых через сеть данных. Особую важность эти свойства приобретают в контексте распределённых приложений *реального времени*, к которым, в частности, относятся разнообразные интерактивные сетевые сервисы. Поскольку для комфортной работы пользователей подобные приложения должны успевать реагировать на их запросы в течение заданного временного интервала, то приложения нуждаются в соединениях, время передачи пакетов через которые гарантированно не превосходит определённой величины.

Своевременность доставки проверяется на основании вычисления оценок для задержек передачи данных. В отличие от свойств достижимости, свойства из указанного класса не отличаются разнообразием – каждое из них представляет собой ограничение максимальной задержки передачи пакета для одного из потоков трафика.

Таким образом, для достижения поставленной цели в рамках настоящей работы было необходимо решить следующие задачи:

1. Предложить формальный язык для спецификации политик маршрутизации, выражающих свойства достижимости в пространстве состояний пакетов;

2. Построить математическую модель ПКС и алгоритм для проверки модельного представления ПКС на соответствие спецификациям предложенного языка;

3. Разработать математическую модель ПКС и алгоритм оценки для максимальной сквозной задержки при передаче пакетов через инфраструктуру сети;

4. Выполнить программную реализацию и провести экспериментальное исследование предложенных алгоритмов.

Научная новизна:

1. Впервые формализм логики первого порядка, дополненной оператором транзитивного замыкания, был использован в качестве языка для спецификации свойств достижимости в ПКС. Для данного языка были разработаны оригинальные методы проверки соответствия конфигурации ПКС его произвольным спецификациям;

2. Для исследуемого класса сетей впервые был предложен алгоритм вычисления достижимой верхней оценки для задержки передачи пакета, обладающий полиномиальной сложностью.

Теоретическая и практическая значимость. Результаты настоящей работы могут быть использованы в качестве базы для разработки новых методов и средств исследования разнообразных свойств компьютерных сетей по их конфигурациям. Распространение инструментальных средств указанного вида способно снизить затраты на обслуживания сетей и повысить продуктивность сетевых администраторов, наделив их возможностью заблаговременно выявлять несоответствия между настройками сетевого оборудования и требованиями, предъявляемыми к поведению сети.

Основные положения, выносимые на защиту:

1. Разработан метод проверки свойств достижимости ПКС на соответствие заданным политикам маршрутизации.

2. Предложен метод построения верхних оценок для сквозной задержки передачи пакетов через инфраструктуру ПКС.

Достоверность полученных результатов обеспечивается использованием хорошо проработанных методов проверки моделей на соответствие спецификациям языка формальной логики (model checking) [12; 13] и широко известного аппарата сетевого исчисления (network calculus) [14; 15]. Теоретические результаты подтверждаются проведёнными экспериментальными исследованиями и находятся в соответствии с результатами, полученными другими авторами.

Апробация работы. Основные результаты работы докладывались на:

1. Четвёртом международном семинаре
   «Программные семантики, спецификации и верификация (PSSV-2013)»;

2. Международной научной конференции
   «Управление и виртуализация в современных сетях (Сети 2014: SDN & NFV)»;

3. Международной научно-практической конференции
   «Инструменты и методы анализа программ (TMPA-2014)»;

4. Международном семинаре
   «Formal Foundations for Networking (Dagstuhl Seminar 15071)».

Личный вклад автора заключается в выполнении основного объема теоретических и экспериментальных исследований, изложенных в диссертационной работе, включая разработку теоретических моделей и алгоритмов, построение их программных реализаций,

создание оригинальных методик экспериментальных исследований, проведение исследований, анализ и оформление результатов в виде публикаций и научных докладов.

Публикации. Основные результаты по теме диссертации изложены в 11 печатных изданиях [16—26], 8 из которых изданы в журналах, рекомендованных ВАК [17; 18; 20—24; 26].

Объем и структура работы. Диссертация состоит из введения, двух глав, заключения и трёх приложений. Полный объём диссертации составляет 200 страниц с 18 рисунками и 5 таблицами. Список литературы содержит 131 наименование.

Глава 1

Исследование свойств достижимости

1.1 Формальные методы в ПКС

В данной главе представлены результаты, полученные в ходе разработки нового средства для проверки соответствия поведения ПКС политикам достижимости с помощью формальных методов. Предполагается, что поведение сети целиком определяется её конфигурацией: топологией сети и правилами обработки, которые хранятся в таблицах отдельных коммутационный устройств и применяются ими при обслуживании пакетов. Прочие классы свойств ПКС, которые, например, зависят от текущей загруженности оборудования, находятся за рамками настоящей работы.

Раздел 1.2 содержит подробное описание устройства ПКС сетей, построенных на базе протокола OpenFlow, а так же интерфейса для управления коммутационным оборудованием, которым он обеспечивает контроллер.

В разделе 1.3 приведены примеры типичных требований политик маршрутизации пакетов. В этом же разделе рассматривается взаимосвязь между требованиями политик маршрутизации и проверками конфигурации оборудования, которые необходимо выполнить, чтобы убедиться в их выполнении.

Успех применения формального метода для решения той или иной задачи проверки соответствия свойств системы заданным требованиям во многом зависит от выбора нескольких ключевых позиций по следующему списку:

Язык спецификаций – способ строго описания тех свойств исследуемой системы, которые необходимо верифицировать;

Модель исследуемой системы – упрощённое описание системы, которое позволяет абстрагироваться от несущественных деталей и сосредоточиться лишь на тех её особенностях, которые могут повлиять на результаты проверки;

Метод верификации – алгоритм, который позволяет доказать, что модель системы удовлетворяет множеству свойств, описанных на выбранном языке спецификаций.

Несколько возможностей для подобного выбора существует и при верификации ПКС. Краткое сравнение способов спецификации требований к поведению ПКС и способов описания их конфигураций приведено в разделах 1.4 и 1.5 соответственно.

Забегая вперёд отметим, что выбранные на основании проведённых обзоров язык спецификации политик маршрутизации и модель ПКС оказались хорошо совместимы друг с другом, и способ проверки их согласованности очевидным образом сводится к вычислению логической формулы от набора двоичных переменных. Поэтому дополнительный обзор методов верификации здесь не приводится.

В качестве формальной модели для задания конфигурации ПКС предлагается использовать реляционную модель, отношения которой выражены с помощью двоичных решающих диаграмм OBDD. Описание указанной модели содержится в разделе 1.6.

В разделе 1.7 содержится подробное описание предложенного языка для спецификации политик маршрутизации на основе логики первого порядка с оператором транзитивного замыкания FO[TC]. Здесь содержится как математическое описание языка, так и описание его грамматики, а так же несколько примеров выраженных с его помощью требований политик маршрутизации.

Раздел 1.8 включает в себя описание разработанного программного прототипа системы проверки конфигураций ПКС. Результаты его экспериментального исследования приводятся в разделе 1.9.

1.2 Основные положения протокола OpenFlow

Основными компонентами Программно-Конфигурируемых Сетей (ПКС) являются коммутаторы [1], ответственные за обслуживание передаваемого через сеть трафика, и контроллеры, основная обязанность которых состоит в управлении коммутаторами.

[1] В мире ПКС нет чёткой границы между коммутацией и маршрутизацией: любые сетевые устройства, занимающиеся передачей пакетов через сеть, принято называть коммутаторами.

Концепции ПКС предполагают, что взаимодействие между коммутаторами и контроллерами происходит по специальным сетевым протоколам, которые скрывают внутреннее устройство коммутационного оборудования, предоставляя контроллерам стандартный интерфейс управления. На сегодняшний день предложено уже несколько подобных протоколов, например [27; 28]. Однако наибольшее распространение из них получил OpenFlow [29]. Основные положения указанного протокола перечислены ниже.

Некоторые подробности, упомянутые в описании актуальной версии спецификации протокола OpenFlow, но несущественные для построения математической модели протокола и анализа его поведения, опущены.

1.2.1 Коммутатор

Коммутатор – сетевое устройство, снабженное несколькими *портами*. Каждый порт имеет один входной и один выходной буфер. Порты коммутатора соединены с портами других коммутаторов физическими каналами связи. *Порт управления* прямо или опосредовано соединен с узлом контроллера – по этому каналу происходит обмен OpenFlow сообщениями между коммутатором и контроллером.

Коммутатор снабжен набором *таблиц коммутации* (flow tables), образующих *конвейер коммутации* (pipeline). Пакет, поступивший во входной буфер одного из портов (ingress port), передается на конвейер коммутации и обрабатывается этим конвейером, а затем либо поступает в выходной буфер одного из портов (egress port) передачи данных или управления (control port), либо сбрасывается. Описанную операцию будем называть *коммутацией пакета*.

Пакеты, поступившие в выходной буфер порта, пересылаются по подключённому к нему каналу передачи данных во входной буфер порта, находящегося на другом конце этого канала. Пакеты, поступившие в выходной буфер порта управления, пересылается контроллеру. Каждую из перечисленных операций будем называть *пересылкой пакета*.

Каждый порт коммутатора имеет уникальный номер, который выступает в роли имени порта. Кроме того, при выполнении некоторых действий над пакетами в качестве имени порта могут выступать служебные имена `ALL`, `CONTROLLER` и `IN_PORT`:

- Если коммутатор направляет пакет в порт по имени `ALL`, то копия этого пакета направляется в выходные буфера всех портов коммутатора, за исключением того, в который этот пакет поступил;

- Если коммутатор направляет пакет в порт по имени `CONTROLLER`, то этот пакет направляется в выходной буфер того порта, который подключен к каналу управления коммутатора;
- Если коммутатор направляет пакет в порт по имени `IN_PORT`, то этот пакет направляется в выходной буфер того же порта, через входной буфер которого он поступил.

1.2.2 Пакеты

Пакеты – это элементарные структуры данных, автономно циркулирующие в сети под воздействием операций коммутации и пересылки. Каждый пакет представляет собой битовую строку, из которой можно выделить две части: *заголовок* (header) и *полезную нагрузку* (payload). Операция коммутации не изменят полезной нагрузки пакета, но способна изменять его заголовок; операция пересылки не изменяет ни заголовка, ни нагрузки.

Заголовок пакета состоит из нескольких *полей* (fields). Как правило, в этих полях указывается идентификаторы сетевых протоколах, которые должны обрабатывать пакет, и используемая ими служебная информация. Например, первые версии протокола OpenFlow выделяли среди полей заголовка адреса отправителей и получателей для протоколов Ethernet, IP, TCP, UDP, а так же номер VLAN. Более новые версии OpenFlow выделяют значительно большее число полей или даже предлагают возможность задавать произвольное разбиение заголовка на поля.

В процессе коммутации к заголовку пакета могут быть добавлены *служебные поля* (metadata), которые предназначены для передачи служебной информации внутри конвейера, и сбрасываются при поступлении пакета в один из выходных буферов коммутатора. Например, одно из таких служебных полей зарезервировано для хранения номера того порта, во входной буфер которого поступил полученный коммутатором пакет. Состав и размер служебных полей определяется техническими характеристиками конкретного OpenFlow коммутатора.

1.2.3 Конвейер коммутации пакетов

Конвейер коммутации пакетов – это конечная последовательность таблиц коммутации. Все таблицы конвейера занумерованы идущими подряд натуральными числами, начиная с 0. Таблица коммутации с номером 0 считается стартовой таблицей конвейера.

Каждый пакет, прибывший во входной буфера порта, присоединенного к каналу пересылки данных, передаётся на вход стартовой таблицы конвейера. Пакету сопоставляется

два множества: множество служебных полей, куда сразу же заносится номер входного порта, и изначально пустое множество *действий* (action set). В последствии, при прохождении пакета через таблицы конвейера размеры и некоторые элементы сопутствующих ему множеств могут изменяться.

На выходе из таблицы с номером i пакет может быть:

- либо передан на вход таблицы с номером $j, j > i$;
- либо отправлен в входные буферы одного или нескольких портов коммутатора (включая, быть может, порт, присоединенный к каналу управления);
- либо сброшен.

Если пакет не передается на вход очередной таблицы, то к нему примеряются (в определенной последовательности) все действия из накопленного множества действий, сопутствующего пакету. Если эта последовательность действий оканчивается отправкой пакета на некоторые порты приема/передачи данных коммутатора, то служебные поля пакета сбрасывается, а сам он направляется в выходные буферы соответствующих портов. Если же пакет отправляется в порт управления, то служебные поля передаются вместе с ним. Если указанная последовательность не содержит действия отправления пакета в какой-либо порт коммутатора, то пакет сбрасывается.

1.2.4 Правило коммутации

Записями каждой таблицы коммутации являются *правила коммутации* (flow entry), каждое из которых, в свою очередь, состоит из:

- шаблона (pattern, match fields);
- приоритета (priority);
- списка инструкций (instructions);
- таймеров (timeouts);
- счетчиков (counters).

Шаблон – это список *масок полей*. Каждая такая маска имеет вид $\langle field, mask \rangle$, где $field$ – наименование некоторого поля заголовка пакета, а $mask$ – строка, состоящая из

символов $0, 1$ (двоичные символы) и $*$ (символ неопределенности). Полями шаблона могут быть как основные, так и служебные поля заголовка пакета.

Будем говорить, что заголовок пакета *совместим* (matches) с маской $\langle field, mask \rangle$, если все двоичные символы строки $mask$ совпадают с соответствующими битами поля $field$ в заголовке пакета. Правило коммутации *применимо* к пакету (иными словами – заголовок пакета подпадает под шаблон правила), если заголовок пакета совместим со всеми масками шаблона.

Приоритет – это натуральное число, указывающее степень значимости правила и использующееся для их избирательного применения: из нескольких правил таблицы, применимых к одному и тому же пакету, для его обработки выбирается правило с максимальным приоритетом.

В списке инструкций допустимы инструкции следующих типов (но не более одной инструкции каждого типа):

$ApplyActions(act_1, act_2, \ldots, act_k)$ – поочерёдно применить к пакету все действия из списка $act_1, act_2, \ldots, act_k$ не изменяя при этом накопленного множества действий, сопутствующего пакету. Если среди действий этого списка встречается передача пакета в выходные буферы, то в соответствующий порт направляется копия пакета с тем заголовком, который был преобразован предшествующими действиями списка;

$ClearActions$ – опустошить сопутствующее пакету множество действий;

$WriteActions(act_1, act_2, \ldots, act_k)$ – объединить множество действий $\{act_1, act_2, \ldots, act_k\}$ и накопленное множество действий, сопутствующее пакету;

$WriteMetadata(mask)$ – изменить содержимое служебного поля пакета;

$GotoTable(table)$ – передать пакет на вход таблицы с номером $table$.

Действие (action) – это элементарная операция обработки пакета, которая применяется к нему либо в процессе прохождения пакета через таблицы конвейера (в результате вызова инструкции *ApplyActions*), либо по окончании обслуживания пакета конвейером (в том случае, когда пакет не передается на вход очередной таблицы конвейера и выполняются все действия накопленного к этому моменту множества действий, ассоциированного с пакетом). Существует два основных типа действий:

- действия модификации заголовка – вносят изменения в поля заголовка пакета;
- действия коммутации – направляют пакет в выходной буфер заданного порта.

Множество накопленных пакетом действий не может содержать несколько действий коммутации. Если таких действий в списке нет, то его выполнение приведёт к сбросу пакета. Если действие коммутации есть, то оно выполняется в последнюю очередь.

Таймер – это натуральное число, которое определяет максимальную продолжительность пребывания правила в таблице правил коммутатора. С каждым правилом коммутации ассоциировано, как минимум, два таймера: максимальное время жизни (hard timeout) и максимальное время простоя (idle timeout). Если хотя бы один из них истекает, то правило удаляется из таблицы, и коммутатор оповещает об этом событии контроллер.

С каждым правилом коммутации также ассоциировано множество *счетчиков*, которые служат для учета разнообразной статистической информации об использовании правила. Например, подсчитывают количество обработанных им пакетов и битов данных.

1.2.5 Таблица коммутации пакетов

Таблица коммутации пакетов – это список правил коммутации, снабженный процедурой выбора подходящих правил для обработки пакетов, поступающих на вход таблицы. Обработка очередного пакета проводится на основании следующих положений:

- В таблице выделяются все правила, применимые к данному пакету, и из них выбирается правило с наибольшим приоритетом. Здесь возможны два случая:
  - Если таких правил несколько, и произошла коллизия, то поведение коммутатора не определено. Заметим, что протокол OpenFlow позволяет запретить использовать таблицы с пересекающимися правилами, но не делает этого по умолчанию;
  - Если такое правило единственно, то список его инструкций применяется к обрабатываемому пакету, и этот пакет покидает таблицу. При этом происходит обновление показателей счетчиков и, возможно, некоторых таймеров приписанных данному правилу;
- Если применимые к пакету правила в таблице отсутствует, то в зависимости от реализации коммутатора протокол допускает следующие исходы:
  - Таблица коммутации может содержать специальное *правило по умолчанию* (table miss/default flow entry), которое применимо к любому пакету и используется, если в таблице не нашлось других правил;
  - Если правило по умолчанию в таблице отсутствует, то пакет, не подпадающий под шаблон ни одного из правил таблицы, сбрасывается.

Состав таблицы коммутации может изменяться в следующих случаях:

- Истёк один из таймеров какого-либо правила коммутации. В этом случае правило, срок активности которого истек, удаляется из таблицы. Об этом событии оповещается контроллер.

- На коммутатор поступает команда, требующая добавить в таблицу новое правило. В этом случае правило, являющееся параметром такой команды, добавляется в таблицу коммутации.

- Контроллер требует изъять из таблицы все правила с определенными шаблонами. В этом случае из таблицы удаляются все правила, обладающие указанному признаку.

1.2.6 Контроллер

Контроллер управляет содержимым таблиц правил подвластных ему коммутаторов. Протокол OpenFlow предполагает, что основной причиной того или иного изменения содержимого таблиц коммутаторов является реакция контроллера на события из сети, и предлагает несколько типов уведомлений, чтобы коммутаторы могли оповестить контроллер о подобных событий, а так же несколько типов команд, с помощью которых контроллер мог бы вносить модификации в таблицы коммутаторов. Перечислим основные из них:

$PacketIn$ – уведомление контроллера о пакете, обработка которого требует его вмешательства. Они отправляются контроллеру всякий раз, когда одно из правил таблицы $table$ коммутатора $switch$ отправляет пакет в выходной буфер порта `CONTROLLER`. Наиболее часто это происходит при установке правила, передающего пакеты на контроллер, в качестве правила по умолчанию. Сообщения указанного типа имеют вид $packet\_in(switch, table, packet)$;

$FlowRemoved$ – уведомление контроллера ПКС об удалении из таблицы $table$ коммутатора $switch$ правила с шаблоном $pattern$, приоритетом $priority$ и значениями счётчиков $counters$. Сообщения указанного типа описываются следующей сигнатурой: $flow\_removed(switch, table, pattern, priority, counters)$;

$ModifyState$ – три разновидности команд изменения содержимого таблиц коммутации:

- $add(switch, table, pattern, priority, timeout, instructions)$ – ввести правило с заданными параметрами в заданную таблицу коммутатора. Если в таблице ком-

мутации содержится правило с тем же шаблоном и приоритетом, то старое правило коммутации при этом удаляется;

- $delete(switch, table, pattern, priority)$ – изъять из заданной таблицы указанного коммутатора все правила с выбранным приоритетом, шаблоны которых могут быть вложены в некоторый заданный шаблон;
- $modify(switch, table, pattern, priority, instructions)$ – предписывает коммутатору $switch$ найти в своей таблице $table$ все правила с приоритетом $priority$, шаблоны которых могут быть вложены в шаблон $pattern$, и заменить их списки инструкций на список $instructions$;

PacketOut – команда, предписывающая коммутатору $switch$ последовательно применить к пакету $packet$ все действия из списка $list\_of\_actions$. Сообщения данного типа имеют вид: $packet\_out(switch, packet, list\_of\_actions)$.

ReadState – команда, предписывающая коммутатору $switch$ отправить контроллеру статистическую информацию (показания счетчиков) всех правил таблицы $table$, шаблоны которых вкладываются в некоторый заданный шаблон $pattern$. Сигнатура сообщений указанного вида следующая: $query(switch, table, pattern)$;

Barrier – команда, заставляющая коммутатор выполнять все последующие команды лишь после того, как будет завершено выполнение всех предшествующих команд;

1.3 Требования к поведению ПКС

Политики маршрутизации способны предъявлять разнообразные требования к поведению ПКС. Чтобы судить об адекватности и удобстве использования того или иного языка спецификаций необходимо сначала рассмотреть и классифицировать наиболее типичные из этих требований. Данный раздел содержит одну из их возможных классификаций.

В основу предложенной классификации положено свойство *локальности* политик маршрутизации в двух измерениях: пространстве и времени. Указанное свойство политик выступает своеобразным индикатором сложности их требований:

- Для проверки выполнения более простых политик маршрутизации достаточно рассмотреть меньшую часть конфигурации ПКС, чем для проверки сложных (локальность в пространстве);

- Выполнение более простых политик может быть показано с использованием меньшего числа состояний сети, в то время как выполнение более сложных – нуждаться в рассмотрении большего их числа (локальность во времени).

Необходимо отметить, что задача проверки более сложных свойств ПКС потребует большей выразительной мощности от языка спецификации политик маршрутизации и более сложной формальной модели сети. Рассмотрение свойств ПКС от простых к сложным позволяет прийти к компромиссу между шириной диапазона проверяемых свойств ПКС и сложностью используемых для этого формальных методов.

1.3.1 Классификация свойств поведения ПКС

Классификацию свойств ПКС можно провести по следующим категориям:

1. локальные (local) и глобальные (global) свойства;
2. статические (static) и динамические (dynamic) свойства.

Коротко охарактеризуем каждый из перечисленных классов:

Локальные свойства – это свойства отдельных элементов сети, не зависящие от их внешнего окружения. Такие свойства соответствуют требованиям, которые предъявляются к отдельным компонентам сети (действиями, инструкциям, таблицам коммутации, коммутаторам, портам, каналам передачи данных), а также к пакетам и их последовательностям. Локальные свойства определяются правилами работы соответствующих элементов сети, которые, в свою очередь, задаются в терминах отношений и функций над множествами *элементарных объектов*: заголовков пакетов, а так же идентификаторов коммутаторов и их портов;

Глобальные свойства характеризуют сеть как единую систему, состоящую из взаимодействующих компонентов. Они определяются правилами взаимодействия между отдельными компонентами сети и описываются в терминах локальных свойств этих компонентов;

Статические свойства касаются поведения сети коммутаторов, содержание таблиц которых остается неизменным на протяжении некоторого времени её работы.

Динамические свойства – это свойства поведения сети, для описания и проверки которых необходимо учитывать происходящие в сети преобразования таблиц коммутации. Поскольку выполнимость динамических свойств поведения сети предполагает

выполнимость статических свойств в обусловленные моменты или промежутки времени, то динамические свойства поведения систем называют также *темпоральными свойствами*.

Как это уже было отмечено во введении, при рассмотрении динамических свойств феномен времени может проявляться в двух различных аспектах. В некоторых случаях интерес может представлять выполнимость тех или иных свойств системы в последовательные моменты времени, когда происходят изменения ее состояния. При этом длительность промежутков времени, в течение которых происходят эти изменения, не имеет существенного значения. Известно, например, что команды обновления таблиц коммутации, поступающие от контроллера в таблицы коммутаторов по каналам управления, выполняются коммутаторами поочередно, чередуясь при этом с выполнением операций обработки пакетов. Поэтому поведение сети должно удовлетворять определенным (статическим) требованиям корректности и безопасности поведения как по завершении выполнения последовательности команд обновления таблиц коммутации, так и на каждом этапе (шаге) обновления таблиц коммутации. В этом случае время может быть представлено последовательностью (линейно упорядоченным множеством) моментов, в которые происходят события в сети – поступление на вход контроллера сообщений от коммутаторов и поступление на входы коммутаторов команд от контроллеров. Динамические свойства поведения сети в такой модели времени называются *свойствами дискретного времени* (discrete).

В то же время, каждое правило коммутации может иметь определенный срок активности, а процедуры выполнения и загрузки каждого правила в таблицах коммутации выполняются с некоторой задержкой. Поэтому интерес может представлять не только сама последовательность осуществления событий в сети, но и размеры интервалов времени между моментами, в которые эти события происходят. Динамические свойства такого рода называются свойствами *реального времени* (real time).

Далее в разделе приводятся неформальные описания типичных свойств поведения сети согласно предложенной классификации.

1.3.2 Элементарные объекты ПКС

Для описания функциональных возможностей компонентов ПКС вводится совокупность элементарных объектов, включающая следующие три множества:

- множество H заголовков пакетов;

- множество W имен (номеров) коммутаторов;
- множество P имен (номеров) портов в коммутаторах.

Для упрощения обозначений будем полагать, что все коммутаторы сети имеют одинаковое устройство и, следовательно, идентичный набор портов. Кроме того, будем полагать, что каждый порт коммутатора является либо *входным портом* (способен лишь принимать пакеты), либо *выходным портом* (способен лишь передавать пакеты). Каналы передачи данных соединяют выходные порты одних коммутаторов с входными портами других. К числу выходных портов каждого коммутатора относится также выделенный порт $CONTROLLER$, подключённый к каналу управления, и порт $DROP$, в который направляются все сброшенные пакеты.

Точкой сети (point) будем называть пару $\langle switch, port \rangle$, где $switch \in W$ и $port \in P$. Если порт одной точки сети соединен каналом связи с портом другой точки из этой же сети, то такую точку будем называть *внутренней точкой сети* (internal point), а иначе будем называть её *пограничной точкой сети* (boarder point). Пограничная точка может быть подключена либо к хосту, либо к устройству за пределами сети. Множество $W \times P$ всех точек обозначим символом L.

Пару $\langle header, point \rangle$, где $header \in H$, $point \in L$, будем называть *состоянием пакета* в сети (packet state). Множество $H \times L$ состояний пакета в сети обозначим символом S. Пару $\langle header, port \rangle$, где $header \in H$, $port \in P$, будем называть *локальным состоянием пакета* в коммутаторе (packet location). Множество $H \times P$ локальных состояний пакета в коммутаторе обозначим символом Z.

Функциональные возможности всех компонентов сети определяются в терминах функций и отношений на множествах пакетов, точек сети, локальных состояний пакетов в коммутаторе и глобальных состояний пакетов в сети. В этом разделе приводится описание типов тех функций и отношений, которые вычисляются компонентами сети. Этого достаточно для формулировки свойств поведения сети. Строгое определение самих функций и отношений приводится в разделе, посвященном описанию формальной модели ПКС.

1.3.3 Локальные свойства пакетов и точек

Свойства заголовков пакетов

Такие свойства выражаются одноместными предикатами на множестве заголовков пакетов H. К этой группе относятся требования, предъявляемые к полям заголовков пакетов, такие как:

- MAC-адрес получателя пакета (не) подпадает под заданный шаблон;
- для передачи пакета должна использоваться широковещательная рассылка;
- если IP-адрес отправителя пакета подпадает под заданный шаблон, то номер VLAN пакета должен лежать в заданном диапазоне значений;

Свойства пакетов можно использовать для формализации понятия потока пакетов. *Поток* – это упорядоченное множество пакетов, поля заголовков которых подпадают под некоторый заданный шаблон (например, шаблон, в котором полностью определены все поля, относящиеся к адресам отправителей, но не специфицированы адреса получателей). Указанный шаблон может быть назван характеристикой потока.

Отношения между заголовками пакетами

Для описания используются двухместные предикаты на множестве заголовков пакетов H. К этой группе свойств относятся требования к полям заголовков пар пакетов, такие как:

- заголовки пары пакетов подпадают под один и тот же заданный шаблон;
- IP-адрес получателя одного пакета совпадает с IP-адресом отправителя другого пакета;
- два пакета (не) принадлежат одному и тому же потоку;
- один из пакетов следует непосредственно за другим пакетом в одном и том же потоке;
- один из пакетов подтверждает успешное получение другого пакета.

Свойства локальных состояний пакета в коммутаторе

Выражаются с помощью одноместных предикатов на множестве локальных состояний пакетов Z. Наиболее важным свойством локальных состояний пакета в коммутаторе является свойство *сочетаемости* с шаблоном (pattern matching). В общем случае каждый

шаблон $pattern$, используемый в правилах коммутации, представляет собой одноместный предикат (свойство) на множестве Z локальных состояний пакета в коммутаторе. Если локальное состояние пакета на коммутаторе $\langle header, port \rangle$ подпадает под шаблон $pattern$, то для обозначения этого факта будет использоваться запись $\langle header, port \rangle \in pattern$.

Отношения между шаблонами

Свойства указанного типа выражаются предикатами на множестве всех подмножеств (булеане) $\mathcal{P}(Z)$ множества Z. Отношения между шаблонами используются для проверки корректности таблиц коммутации. В частности, к этой группе свойств относятся следующие требования:

- два шаблона не имеют пересечений, то есть любое локальное состояние пакета в коммутаторе, подпадающее под один шаблон, не подпадает под другой;
- шаблон покрывается заданным набором шаблонов, то есть любое локальное состояние пакета в коммутаторе, подпадающее под указанный шаблон, также подпадает под один или несколько из шаблонов указанного набора;
- заданный набор шаблонов охватывает все множество заголовков пакетов, то есть всякое локальное состояние пакета подпадает под один из шаблонов указанного набора.

Свойства точек

Для выражения используются предикаты на множестве L точек сети. Свойства из данной группы используются для выделения подмножества среди портов коммутаторов, например:

- свойство точки быть входной (выходной) точкой сети;
- свойство точки быть пограничной (внутренней) точкой сети;
- хост с заданным MAC-адресом подключен к заданной точке.

Отношения между точками

Выражаются двухместными предикатами на множестве $L \times L$. К этой группе свойств относятся требования, предъявляемые к топологии сети, например:

- пара точек принадлежат одному и тому же коммутатору;
- одна из точек сети соединена с другой точкой каналом связи (отношение смежности).

В терминах отношений между точками сети определяется её топология и такие свойства, как связность и отсутствие циклов.

1.3.4 Локальные свойства элементов коммутационного конвейера

К числу этих свойств и отношений принадлежат те, которые описывают отдельные правила обработки пакетов, таблицы коммутации, а так же состоящие из них коммутационные конвейеры. Будем использовать запись A для обозначения множества действий над пакетами с множеством заголовков H в коммутаторе с множеством портов P. Множество локальных состояний пакета будем обозначать записью $Z = H \times P$

Каждое действие $a \in A$ модификации заголовков пакетов вычисляет функцию $f_a : H \to H$. Действия коммутации пакетов описываются константами типа P. Каждое действие $b \in A$ коммутации пакетов определяет порт коммутатора $f_b \in P$, в который должны быть направлены пакеты в результате выполнения этого действия.

Запись A^* обозначает совокупность всех конечных последовательностей действий. В каждой конечной последовательности действий $\alpha = a_1, a_2, \ldots, a_n$ действие a_i применяется к той копии пакета, которая была получена в результате поочерёдного применения всех предшествующих ему действий: $a_1, a_2, \ldots, a_{i-1}$. Поэтому в результате применения всей последовательности α к исходному пакета может быть построено сразу несколько его копий, причём эти копии могут обладать разными заголовками и быть направлены на разные выходные порты коммутатора.

Каждая конечная последовательность действий $\alpha = a_1, a_2, \ldots, a_n$ вычисляет:

- функцию $f_\alpha : H \to H$, которая для каждого заголовка *header* определяет заголовок, который будет получен после применения последовательности действий α;
- отношение $R_\alpha \subseteq H \times Z$, определяющее для каждого заголовка пакета непустое множество тех локальных состояний пакета в коммутаторе, которые образуются в результате выполнения последовательности действий α.

Каждое правило коммутации *rule* с шаблоном *pattern* и списком инструкций:

$$ApplyActions(\alpha), ClearActions(), WriteActions(\beta), GotoTable(id),$$

где $\alpha, \beta \in A^*$, $ClearActions() \in \{true, false\}$, $id \in \mathbb{N}$, применяется к тем пакетам, которые поступили в определенный входной порт коммутатора, и текущие заголовки которых при этом соответствуют шаблону правила *pattern*. Правило *rule* применяет к пакету последовательность действий α и в зависимости от значения индикатора $ClearActions$ либо формирует новую последовательность накопленных действий β, либо приписывает эту последовательность к накопленной ранее последовательности действий. В заключение правило *rule* направляет пакет с тем заголовком, который образуется в результате применения последовательности действий α на вход таблицы с номером *id*. Таким образом, правило *rule* вычисляет:

- бинарное отношение $R_{rule} \subseteq Z \times Z$, определяющее для каждого заголовка пакета непустое множество тех локальных состояний пакета в коммутаторе, которые образуются в результате выполнения последовательности действий α;
- функцию $f_{rule} : H \to H \times A^* \times \mathbb{N}$, которая определяет:
  - заголовок пакета, модифицированный последовательностью действий α;
  - накопленную последовательность действий, образующуюся в результате применения правила;
  - номер таблицы коммутации, на вход которой подается пакет после применения правила.

Таблица коммутации – это конечное множество пар вида $\langle rule, priority \rangle$, где *rule* – правило коммутации, а *priority* – натуральное число (приоритет правила). Каждая таблица Tab коммутации реализует отношение R_{tab} и функцию f_{tab}, имеющие те же типы, что и соответствующие функции правил коммутации. Для поступившего на вход таблицы пакета эти отношения описывают результат применения к нему одного из тех правил таблицы, шаблоны которых соответствуют заголовку пакета, а приоритет – максимален.

Свойства таблиц коммутации пакетов

Свойства из заданного класса выражаются предикатами на множествах таблиц коммутации, то есть на множестве пар вида $\langle R_{tab}, f_{tab} \rangle$. К их числу относятся следующие свойства и требования, предъявляемые к таблицам коммутации:

- к каждому пакету, поступившему на вход заданной таблицы коммутации пакетов, применимо (с учетом приоритетов или без учета) в точности одно правило (требование отсутствия коллизий правил коммутации);

- ни один пакет, поступивший на вход заданной таблицы коммутации пакетов, не утрачивается, то есть хотя бы одна его копия поступает либо в некоторый выходной порт, либо на вход другой таблицы коммутации (свойство отсутствия потери пакета);
- для любого правила заданной таблицы коммутации существует пакет, к которому, в случае поступления его на вход таблицы, применимо это правило (свойство отсутствия бесполезных правил);
- ни один пакет, поступивший на вход заданной таблицы, не направляется в некоторый заданный порт коммутатора;
- заголовок всякого пакета, который одна из двух заданных таблиц направляет на вход другой таблицы, удовлетворяет некоторому заданному свойству заголовков пакетов;
- отношение коммутации пакетов R_{tab} является функциональным отношением (требование однозначности);
- две таблицы коммутации пакетов (функционально) эквивалентны (отношение эквивалентности).

Конвейер коммутации пакетов – это такое конечное множество $Pipeline = \{Tab_i \mid 0 \leq i \leq N\}$ пронумерованных натуральными числами таблиц коммутации пакетов, что для каждого номера i соответствующая ему таблица $Tab_i \in Pipeline$ не содержит ни одного правила с инструкцией вида $Goto\_Table(j)$, где $j \leq i$. Конвейер коммутации пакетов вычисляет отношение $R_{Pipeline} \subseteq Z \times Z$, которое определяется как композиция отношений и функций, вычисляемых составляющими его таблицами. Для всякого пакета с заголовком $header$, поступившего во входной порт $port$ коммутатора, в конвейере формируется монотонно возрастающая последовательность натуральных чисел $tr = 0, i_1, i_2, \ldots, i_k$, не превосходящих количества $N \in \mathbb{N}$ таблиц в конвейере коммутации, которая называется *траекторией* пакета в конвейере $Pipeline$. Этот пакет обрабатываете в конвейере последовательно таблицами $Tab_0, Tab_{i_1}, Tab_{i_2}, \ldots, Tab_{i_k}$.

Свойства конвейеров коммутации пакетов

Свойства указанного вида выражаются предикатами на множестве $R_{Pipeline}$ отношений коммутации пакетов в конвейере. К их числу относятся следующие свойства и требования, предъявляемые к конвейерам коммутации:

- хотя бы одна траектория хотя бы одного пакета проходит через заданную таблицу заданного конвейера (свойство отсутствия недостижимых таблиц в конвейера);
- траектория всякого пакета, локальное состояние которого удовлетворяет заданному свойству, (не) проходит через заданную таблицу заданного конвейера;
- длина траектория всякого пакета в заданном конвейере не превосходит заданной величины;
- все пакеты, заголовки которых удовлетворяют заданному свойству, имеют одну и ту же траекторию в заданном конвейере;
- траектории всех пакетов проходят через одну и ту же таблицу (свойство bottleneck);
- нет ни одной таблицы (отличной от стартовой), через которую проходят как траектории пакетов, поступивших во входной порт $port_1$, так и траектории пакетов, поступивших во входной порт $port_2$ коммутатора;
- всякий пакет, траектория которого проходит через заданную таблицу конвейера коммутации, поступает в эту таблицу с без накопленных ранее действий.

В том случае, если известны технические характеристики коммутаторов (например, оценки времени срабатывания правила, времени выполнения действия и так далее) к конвейерам можно предъявлять требования, связанные с быстродействием обработки пакетов, например:

- любая копия пакета, поступившего на вход конвейера, проходит свою траекторию за время, не превосходящее заданной величины (свойства соблюдения директивных сроков коммутации пакетов).

Каждый коммутатор $switch$, в котором работает конвейер коммутации пакетов $Pipeline$, реализует *отношение коммутации* пакетов $R_{switch} \subseteq S \times S$, которое определяется так: состояния пакетов $\langle header_1, port_1, switch\rangle$ и $\langle header_2, port_2, switch\rangle$ находятся в отношении R_{switch} тогда и только тогда, когда пакет с заголовком $header_1$ после поступления во входной порт $port_1$ обрабатывается конвейером так, что он достигает выходного порта $port_2$ с заголовком $header_2$.

Свойства коммутаторов

Выражаются предикатами на множестве S. К этой группе свойств относятся требования, предъявляемые к отношению коммутации пакетов:

- в заданный выходной порт коммутатора могут поступать только те пакеты, заголовки которых удовлетворяют заданному свойству;
- ни один пакет, поступивший в заданный входной порт коммутатора, не сбрасывается (не отправляется контроллеру;
- все пакеты, состояния которых удовлетворяют заданному свойству (например, подпадают под один и тот же шаблон или принадлежат одному и тому же потоку), при поступлении в какой-либо входной порт коммутатора, направляются в один и тот же выходной порт коммутатора;
- все пакеты, заголовки которых удовлетворяют заданному свойству, при поступлении в какой-либо входной порт, коммутируются широковещательно.

1.3.5 Глобальные свойства

На основе отношений связности между точками T_{Net} и отношений коммутации пакетов R_{switch} определенного для всех коммутаторов $switch \in C$ сети, можно ввести следующие понятия. *Отношением одношаговой маршрутизации* называется такое бинарное отношение R_{hop} на множестве состояний пакетов S, что для каждой пары состояний пакетов $\langle header_1, port_1, switch_1\rangle$ и $\langle header_2, port_2, switch_2\rangle$ коммутатору $switch_1$ принадлежит такой порт $port$, для которого выполняются:

- отношение коммутации:

$$\langle\langle header_1, port_1, switch_1\rangle, \langle header_2, port, switch_1\rangle\rangle \in R_{switch_1}$$

- отношение пересылки пакетов:

$$\langle\langle port, switch_1\rangle, \langle port_2, switch_2\rangle\rangle \in T_{Net}$$

Маршрутом в сети Net будем называть такую конечную последовательность $s_1, s_2, \ldots, s_{2i-1}, s_{2i}, \ldots, s_n$ состояний пакетов $s_j = \langle header_j, port_j, switch_j\rangle$, $1 \leq j \leq n$, что для любого $i, 1 \leq i < n$ она удовлетворяет условию одношаговой маршрутизации пакетов $\langle s_i, s_{i+1}\rangle \in R_{hop}$.

Для заданной сети Net отношением маршрутизации R_{Net} будем называть бинарное отношение на множестве состояний пакетов S. Два состояния пакетов $s_1, s_2 \in S$ находятся в отношении маршрутизации $\langle s_1, s_2 \rangle \in R_{Net}$ тогда и только тогда, когда в сети существует маршрут из состояния пакетов s_1 в состояние пакетов s_2. Отношение R_{Net} является транзитивным замыканием одношаговой маршрутизации пакетов R_{hop}. В терминах отношения маршрутизации можно описывать статические свойства поведения сети, то есть свойства маршрутизации пакетов в сети коммутаторов, таблицы коммутации пакетов которых остаются неизменными.

1.3.6 Статические свойства ПКС

Статические свойства выражаются предикатами различной местности на множестве состояний пакетов S. Примерами статических свойств могут служить следующие свойства:

- всякий пакет, заголовок которого подпадает под заданный шаблон, и поступивший в сеть через заданную входную точку, обязательно покинет её через определённую выходную точку (которая может зависеть, например, от IP-адреса получателя, указанного в заголовке пакета), причём его заголовок будет соответствовать другому заданному шаблону (свойство достижимости);

- ни один пакет с некоторым заданным IP-адресом отправителя (подозрительного отправителя), поступивший на пограничную точку, не достигнет ни одного из некоторого заданного подмножества работающих в сети хостов (свойство безопасности);

- каждый пакет, поступивший в любую пограничную точку, не может циркулировать по сети бесконечно долго (свойство отсутствия достижимых циклов);

- длины маршрутов, по которым движутся поступившие в пограничные точки сети пакеты, не превосходят заданной величины;

- всякий маршрут любого из пакетов, заголовок которого подпадает под некоторый заданный шаблон, не проходит через некоторые выделенные точки сети (свойство безопасности);

- маршруты двух пакетов, заголовки которых подпадают под некоторые заданные шаблоны, не пересекаются в случае поступления этих пакетов в некоторые заданные входные точки сети (свойство безопасности);

- каждый пакет, заголовок которого подпадает под один из заданных шаблонов, при поступлении в заданную пограничную точку никогда не будет отправлен контроллеру;
- для любого пакета, состояние которого удовлетворяет некоторому свойству, не существует двух разных маршрутов.

В том случае, если известны технические характеристики коммутаторов и каналов передачи данных (например, задержка, вероятность потери и искажения содержимого пакета), и эти характеристики обладают свойством аддитивности, то требования к поведению сети, описывающие качество соединений могут быть так же выражены в терминах маршрутов (например, предельно допустимое время задержки передачи потоков и максимальная вероятность безошибочной доставки сообщений).

1.3.7 Динамические свойства ПКС

Поведение коммутаторов может изменяться в результате поступления на коммутаторы по каналам управления от контроллера некоторого *потока команд* (control stream) $stream = \{r_1, r_2, \ldots, r_k\}$, вносящих изменения в таблицы коммутации. Каждая команда может удалить или добавить правило в таблицу коммутации пакетов одного из коммутаторов сети или потребовать от контроллера применить заданную последовательность действий к заданному пакету. Таким образом, для заданной сети, находящейся под управлением контроллера каждая команда r_i изменяет отношение коммутации пакетов R_{switch} коммутатора $switch \in C$, наименование которого указано в заголовке команды. Поскольку изменение отношения R_{switch} одного из коммутаторов может привести к изменению отношения R_{Net} одношаговой маршрутизации сети, то можно считать, что каждая команда r_i реализует всюду определенную функцию $apply[r_i]$, преобразующую отношения R_{Net}.

На разные коммутаторы команды контроллера могут поступать в порядке, отличном от того, в котором они были отправлены. Отношение порядка в этом потоке определяется, в частности, командами $barrier$. Поэтому поток команд следует рассматривать как некоторое частично упорядоченное множество команд $(stream, \leq)$. Частичный порядок $\leq$ в потоке команд $stream$ определяется множеством факторов, таких как топология системы каналов управления , технические характеристиками этих каналов и самих коммутаторов. Любая линеаризация частично упорядоченного потока команд $stream$ задает допустимую последовательность срабатываний команд этого потока $\alpha = r_{i_1}, r_{i_2}, \ldots, r_{i_k}$ в коммутаторах. Таким образом, для некоторого исходного отношения одношаговой маршрутизации

R_{Net} всякая допустимая последовательность срабатываний команд α, порождает последовательность отношений одношаговой маршрутизации $R_0, R_1, \ldots, R_k$, в которой $R_0 = R_{Net}$ и $R_{j+1} = apply[r_i](R_j)$ для всех $j, 1 \leq j < k$.

При описании динамических свойств поведения сети под воздействием потока команд предполагается, что сеть Net определяется множеством точек сети и отношением маршрутизации пакетов R_{Net}, а поток команд задан частично упорядоченным множеством $(stream, \leq)$.

Динамические свойства сети

Выражаются предикатами различной местности на множестве S. Примерами могут служить следующие свойства:

- для любой допустимой последовательности команд заданного потока после каждого выполнения очередной её команды сеть сохраняет заданное статическое свойство (например, достижимость, безопасность, отсутствие циклов и так далее);

- для любой допустимой последовательности команд заданного потока команд все пакеты заданного (произвольного) потока пакетов, поступившие на заданную (произвольную) внешнюю входную точку сети, в том же порядке доставляются на другую заданную (некоторую) внешнюю выходную точку;

- если поведение сети обладает некоторым заданным статическим свойством, то после выполнения любой допустимой последовательности команд заданного потока поведение сети будет обладать некоторым другим заданным статическим свойством;

В более общем случае для определения свойств сети необходимо принимать во внимание модель той программы, которое управляет коммутаторами сети. Далее такую модель программы будем называть моделью контроллера. Ограничимся рассмотрением лишь того случая, когда всякое сообщение, передающиеся контроллеру от коммутаторов сети, содержит данные о коммутаторе-отправителе и пакет, для обработки которого в таблицах этого коммутатора не было обнаружено подходящего правила. Таким образом, можно считать, что на вход модели контроллера подаются состояния пакетов. На выходе – контроллер вырабатывает поток команд. При этом достаточно рассмотреть случай, когда алфавит выходного потока состоит из команд типа *add*, *delete* и *barrier*.

В качестве простейших моделей контроллера могут быть использованы табличная и автоматная модель. Табличная модель контроллера задается конечной функцией (таблицей)

M, которая сопоставляет каждому состоянию пакета $s = \langle switch, port, header \rangle$ некоторый поток команд $M(s) = (stream, \leq)$. Автоматная модель контроллера представляет собой некоторую абстрактную машину (автомат) M, которая в зависимости от своего внутреннего состояния q способна для каждого состояния пакета $s = \langle switch, port, header \rangle$ построить поток $(stream, \leq)$ и перейти в новое внутреннее состояние q'. Машина такого вида может быть представлена как тройка $M = \langle Q, \phi, \psi \rangle$, где

- Q – множество внутренних состояний,
- $\phi : (Q \times S) \to A$ – функция выходов,
- $\psi : (Q \times S) \to Q$ – функция переходов.

Табличная модель контроллера – это частный случай автоматной модели, в которой множество внутренних состояний Q состоит из единственного состояния q_0, в котором машина генерирует потоки команд на основании таблицы, задающей функцию выходов. Содержимое таблицы постоянно, выходная последовательность команд контроллера не зависит от истории его функционирования.

В том случае, если определена автоматная модель контроллера, то глобальное состояние ПКС – это пара $\langle R_{Net}, q \rangle$, где R_{Net} – текущее отношение одношаговой маршрутизации, а q – внутреннее состояние контроллера.

Модель контроллера позволяет рассматривать ПКС как единую систему, глобальное состояние которой определяется множеством состояний пакетов, пребывающих в точках сети, отношением одношаговой маршрутизации, зависящим от текущего заполнения таблиц коммутаторов, а также набором внутренних состояний моделей для множества управляющих ими контроллеров. Эта система может изменять своё состояние вследствие таких факторов как:

- поступление сообщений контроллеру от коммутатора: в результате контроллер изменяет свое внутреннее состояние и вырабатывает последовательность команд;
- выполнение коммутатором команды, поступившей от контроллера: в результате изменяется содержимое таблицы коммутации и в соответствие с этим изменяется отношение одношаговой маршрутизации;
- истечение срока активности одного из правил коммутации в таблице контроллера: в результате изменяется содержимое таблицы коммутации и отношение одношаговой маршрутизации.

В рамках модели функционирования сети под управлением контроллера можно сформулировать некоторые новые требования к поведению сети.

Динамические свойства сети под управлением контроллера

- при любом начальном глобальном состоянии модели системы, если во все внешние входные порты будут поступать пакеты, удовлетворяющие заданному свойству состояний пакетов, то система рано или поздно перейдет в такое глобальное состояние, в котором сеть будет удовлетворять заданному статическому свойству (требование живости (liveness property));

- при любом начальном глобальном состоянии системы, в котором сеть удовлетворяет некоторому заданному статическому свойству, система никогда не достигнет такого глобального состояния, в котором сеть удовлетворяет другому заданному статическому свойству (требование безопасности поведения системы (safety property)).

1.3.8 Свойства реального времени

В том случае, когда известны оценки временных параметров, определяющих длительность выполнения отдельных операций (коммутация пакетов, пересылка пакетов по каналам связи, срабатывание команд модификации таблиц коммутации, пересылка команд по каналам управления, выработка потоков команд контроллерами), можно формулировать требования к срокам выполнения различных задач по доставке отдельных пакетов. При построении моделей коммутатора и контроллера в этом случае целесообразно воспользоваться вычислительными моделями машин (автоматов) реального времени.

Свойства реального времени сети под управлением контроллера:

- любой пакет, заголовок которого подпадает под некоторый заданный шаблон, в случае поступления в произвольную внешнюю входную точку сети, достигнет некоторого выходной внешней точки сети за время, не превосходящее заданной величины (статическое свойство реального времени);

- при любом начальном глобальном состоянии системы, в котором сеть удовлетворяет некоторому заданному статическому свойству, любой пакет, заголовок которого подпадает под некоторый заданный шаблон, в случае поступления на любую входную внешнюю точку сети, достигнет некоторой внешней выходной точки сети за время, не превосходящее заданной величины (динамическое свойство реального времени);

- если в каждую входную внешнюю точку сети поступает поток пакетов (входной поток) определенной интенсивности, то каждый из потоков будет доставлен на соответствующую ему выходную внешнюю точку сети, и при этом интенсивность каждого выходного потока будет не меньше заданной величины.

1.4 Выбор языка спецификаций поведения ПКС

1.4.1 Требования к языка спецификации

Как это было показано в разделе 1.3, свойства поведения ПКС могут быть разбиты на несколько классов: локальные, глобальные статические, глобальные динамические свойства и свойства реального времени. Поэтому удобным представляется использовать сразу несколько языков спецификаций, каждый из которых позволял бы наилучшим образом описывать некоторое подмножество указанных классов свойств: обладал бы достаточной выразительной мощностью, был бы прост в понимании и имел небольшую сложность решения соответствующих задач анализа.

Необходимо отметить, что свойства каждого последующего класса из приведённого выше списка требуют использования более выразительного языка спецификаций и более сложной модели ПКС. Поэтому желательным свойством построенного множества языков является его иерархичность: языки, предназначенные для описания более сложных свойств, должны синтаксически включать в себя языки описания более простых свойств.

В основу выбора языков спецификации, используемых для описания каждого из указанных классов свойств, были положены следующие критерии:

1. *Простота описания политик маршрутизации.* Язык спецификаций должен позволять как можно более удобно описывать требования корректного и безопасного поведения ПКС из соответствующего ему класса;

2. *Алгоритмическая сложность задачи верификации.* В рамках данной работы для проверки соответствия поведения ПКС требованиям политик маршрутизации предлагается использовать метод верификации на моделях (model checking). Поэтому сложность решения соответствующей задачи, проверки выполнимости формул языка спецификаций на моделях, является одним из определяющих факторов выбора такого языка для каждого из перечисленных ранее классов свойств;

3. *Согласованность с моделями ПКС.* Язык спецификации политик маршрутизации должен хорошо сочетаться с формальными моделями тех компонентов сети, поведение которых он позволяет описывать. Желательно, чтобы указанные модели могли служить интерпретациями для его формул;

4. *Возможность использования существующих наработок.* При выборе подходящих языков спецификаций политик маршрутизации целесообразно рассмотреть известные языки для описания поведения распределенных систем. Такие языки достаточно выразительны, и, вероятно, позволяют описать значительную часть выделенных свойств ПКС. Кроме того, существует множество разнообразных систем верификации, которые реализуют эффективные процедуры проверки выполнимости формул таких языков на моделях.

Далее представлено краткое описание перспективных логик, которые могли бы использоваться в качестве основы для разрабатываемого языка спецификации политик маршрутизации. Подробное описание синтаксиса и семантики их формул на конечных системах переходов (моделей Крипке) приведено в приложении A.

1.4.2 Темпоральные логики

Темпоральные логики применяются для выражения утверждений, использующих понятие времени. Логики данного типа широко применяются, например, при формальной верификации распределённых систем, где принципиальное значение имеют правила упорядочивания последовательностей действий, выполняемых их отдельными компонентами. Темпоральные логики, в частности, использовались для верификации моделей бортовых вычислительных систем в среде моделирования DYANA [30].

Темпоральные логики позволяют записывать утверждения о том, что некоторое выделенное состояние системы *когда-нибудь* будет пройдено или что состояние ошибки *никогда* не будет достигнуто. Подобного рода свойства, в которых упоминаются события, происходящие *когда-нибудь* или *никогда*, специфицируются при помощи *темпоральных операторов*. Различные темпоральные логики отличаются друг от друга набором используемых ими темпоральных операторов и семантикой их использования.

Одной из наиболее известных и выразительных темпоральных логик является логика CTL*, формулы которой описывают свойства размеченных систем переходов – ориентированных графах, вершины которых могут быть промаркированы метками. Размеченные

системы переходов (модели Крипке) являются простейшей моделью вычислений распределенных систем, в терминах которых можно определить семантику (множество вычислений) других моделей вычислений, например, сетей Петри и алгебр процессов.

В логике CTL* можно выделить два подмножества, которые чаще всего используются в качестве языков спецификации поведения вычислительных систем: *логику ветвящегося времени* (CTL) и *логику линейного времени* (LTL). При этом все три логики: CTL*, CTL и LTL – имеют разные выразительные мощности. Любые формулы логик CTL и LTL могут быть представлены формулами CTL*. Однако, существуют формулы LTL и, следовательно, CTL*, для которых нет эквивалентных формул CTL, и наоборот, некоторые формулы логик CTL и CTL* нельзя выразить формулами LTL.

В статье [31] показано, что задача проверки выполнимости $M, s \vDash \phi$ темпоральной формулы ϕ в состоянии s конечной модели Крипке M:

- Обладает линейной сложностью относительно размера формулы ϕ и размера (количества состояний и переходов между ними) размеченной системы переходов M и оценивается величиной $O(|\phi|\,|M|)$ в логике ветвящегося времени CTL;
- Является PSPACE-полной и разрешима за время $O(2^{|\phi|}\,|M|)$ в логике линейного времени LTL и логике CTL*.

Необходимо отметить, что для каждой из рассмотренных логик разрешима задача проверки непротиворечивости конечного множества формул – эта задача является PSPACE-полной. Указанное свойство может быть полезно для выявления противоречивых политик маршрутизации, которые заведомо не могут быть одновременно выполняться ни при какой конфигурации сети.

Для перечисленных логик были разработаны эффективные процедуры проверки выполнимости формул на моделях и созданы такие известные инструментальные средства верификации как SPIN [32], NuSMV [33] и UPPAAL [34].

Существует несколько примеров использования темпоральных логик и в контексте ПКС. Авторы статей [35—37] применяют темпоральную логику CTL для описания некоторых статических свойств поведения ПКС с заданной конфигурацией. В статье [38] логика линейного времени LTL используется для описания свойств всевозможных конфигураций сети, которые могут возникнуть в процессе её модификации. В этих статьях формулы темпоральных логик CTL и LTL интерпретируются на системах переходов, в которых в качестве состояний системы используются состояния пакета, а в роли переходов выступали правила коммутации пакетов.

Описанный сценарий использования темпоральной логики оправдан, если цель верификации – проследить корректность перемещения отдельных пакетов в сети с течением времени. Однако, как это было показано в разделе 1.3, инженеры часто налагают ограничения не на последовательности прохождения пакетов через определённые точки сети, а скорее на само отношение маршрутизации. Применение темпоральных логик для спецификации подобных свойств заставляет эти логики выступать в несвойственной им роли.

1.4.3 Пропозициональное μ-исчисление

Пропозициональное μ-исчисление представляет собой мощный язык, выразительность которого превосходит темпоральную логику CTL* (как и её фрагменты CTL и LTL): для каждой формулы CTL* можно построить эквивалентную формулу μ-исчисления, но существуют такие формулы μ-исчисления, которые не могут быть выражены в логике CTL*.

Повышенный интерес к μ-исчислению обусловлен эффективными процедурами верификации моделей, которые он предлагает за счёт использования операторов неподвижной точки. Указанные операторы позволяют построить *символьные* алгоритмы верификации, работающие не в терминах отдельных состояний модели, а в терминах их подмножеств.

Формулы пропозиционального μ-исчисления, как и рассмотренные ранее логики, интерпретируются на размеченных системах переходов (моделях Крипке). Однако метками в них могут быть промаркированы не только вершины, но и дуги графа. Поэтому при работе с μ-исчислением вместо одной системы переходов бывает удобно использовать сразу несколько таких систем, каждая из которых включает в себя все дуги, промаркированные одинаковыми метками.

Задача проверки выполнимости $M, s \vDash \phi$ формул μ-исчисления на конечных моделях разрешима за время $O(|M|^d|\phi|)$, где d – глубина чередования операторов наибольшей и наименьшей неподвижности точки в формуле ϕ. Задача проверки противоречивости конечного множества формул данной логики является EXPTIME-полной [39].

1.4.4 Логика транзитивного замыкания

Поскольку наиболее важные статические свойства ПКС связаны с отношением маршрутизации, целесообразно рассмотреть и такие логики, в которых присутствуют специальные средства для выражения требований достижимости, ацикличности и связности в конечных системах переходов. К числу таких логик, в частности, относится логика пер-

вого порядка с оператором транзитивного замыкания или просто *логика транзитивного замыкания* $FO[TC]$.

Логика первого порядка с оператором транзитивного замыкания ($FO[TC]$) используется для формального определения свойств достижимости в структурах с одним или несколькими бинарными отношениями. Такими отношениями, в частности, могут быть выражены изменения состояний пакетов при их перемещении или коммутации.

Семантика $FO[TC]$ задается на размеченных системах переходов (моделях Крипке). В статьях [40; 41] показано, что и темпоральные логики (CTL, LTL, CTL* и μ-исчисление) транслируются в логику $FO_2[TC]$ – фрагмент логики транзитивного замыкания, в формулах которой позволяется использовать не более двух свободных переменных. Однако при трансляции из CTL* и μ-исчисления размер полученной формулы $FO_2[TC]$ может расти экспоненциально относительно размеров исходной формулы.

Задача проверки выполнимости формулы логики транзитивного замыкания $FO_2[TC]$ на конечной модели является NLOGSPACE-полной [42]. Это, в частности означает, что она может быть решена за полиномиальное время. Таким образом, логика $FO_2[TC]$ не только более выразительна, нежели темпоральные логики, но и одновременно имеет меньшую сложность проверки выполнимости формулы на конечных системах переходов.

Основным недостатком логики $FO_2[TC]$ является алгоритмическая неразрешимость задачи проверки противоречивости её формул – язык спецификаций, построенный на её основе, не будет позволять определить тот факт, что совокупность требований политик маршрутизации не допускает ни одной корректной конфигурации ПКС.

1.4.5 Выводы

На основании характеристик рассмотренных логик для построения языка спецификации политик маршрутизации, описывающих глобальные статические свойства ПКС, представляется целесообразным воспользоваться логикой транзитивного замыкания. Вместе с тем, темпоральные логики очень удобны для описания требований, предъявляемых к программам реконфигурирования сетей коммутаторов – динамическим свойствам поведения ПКС. Таким образом, перспективный язык спецификаций политик маршрутизации может состоять из нескольких согласованных уровней: находящийся на нижнем уровне фрагмент языка $FO[TC]$ будет служить описанию статических свойств сети, а расположенный выше фрагмент одной из темпоральных логик – использоваться для задания формулирования её динамических свойств.

1.5 Выбор формальных моделей ПКС

В этом разделе приводится перечень требований, предъявляемых к тем математическим моделям ПКС, которые целесообразно использовать для решения задач анализа и верификации политик маршрутизации. Перечисленные требований используются для обоснования выбора наиболее перспективных формальных моделей – двоичных решающих диаграмм, которые будут использоваться для представления отношений коммутации и маршрутизации, а так же конечных автоматов – для моделирования поведения контроллера, взаимодействующего с сетью из коммутаторов.

1.5.1 Требования к формальным моделям ПКС

В процессе постановки требований к перспективным математическим моделям необходимо принять во внимание, что ПКС представляет собой иерархическую систему, состоящую из множества разнородных компонентов. В соответствии со спецификациями OpenFlow, приведёнными в разделе 1.2, указанные компоненты могут быть упорядочены по возрастанию их структурной сложности:

- пакеты и шаблоны – элементарная база для более сложных компонентов;
- действия над пакетами и инструкции (наборы действий);
- правила коммутации пакетов, сообщения и команды контроллера;
- таблицы и конвейеры коммутации пакетов, управляющие приложения контроллера;
- сеть коммутаторов, связанных между собой каналами передачи данных;
- ПКС – сеть коммутаторов, находящихся под управлением контроллера.

Поведение ПКС складывается из поведений отдельных ее компонентов. Поэтому для построения модели(ей) ПКС необходимо разработать спектр согласованных между собой моделей всех составляющих ее компонентов, руководствуясь следующими требованиями:

Выразительные возможности моделей. Модель каждого из перечисленных компонентов должна охватывать основные положения, определяющие ее устройство и функционирование согласно спецификации OpenFlow. Однако степень точности соответствия модели указанным положениям может варьироваться в зависимости от прочих требований, которые к ней предъявляются. Так, например:

- при построении модели таблиц коммутации с целью устранения неоднозначности удобно предполагать, что в случае применимости к пакету с заданным заголовком нескольких правил коммутации (с одинаковым приоритетом), выполняются инструкции всех применимых правил;
- при построении модели управляющего приложения контроллера с целью упрощения можно предполагать, что они работают недетерминированно, то есть способны вырабатывать различные последовательности команд в ответ на одно и то же сообщение, поступившее от коммутатора.

Композиционные свойства моделей. Модели составных компонентов ПКС, устройство и функционирование которых определяется множеством более простых компонентов, должны строиться посредством композиции их моделей. Например:

- модели таблиц коммутации должны быть образованы за счет композиции моделей содержащихся в них правил;
- модели сетей коммутаторов должны быть образованы на основе композиции моделей образующих её коммутаторов с учётом топологии сети.

Согласованность с языками спецификации. Для целей настоящей работы наибольший интерес представляют математические модели, способные служить интерпретациями для выражений тех логических языков, которые будут использоваться при спецификации требований к поведению соответствующих компонентов ПКС. Если указанное условие выполнено, и модель M пригодна для интерпретации выражений языка L, а его формула ϕ выражает интересующее нас свойство модели, то задача проверки этого свойства сводится к задаче проверки выполнимости формулы ϕ на модели M: $M \vDash \phi$.

Разнообразие уровней абстракции моделей. При моделировании тех компонентов ПКС, поведение которых может быть очень сложным (например, конвейерных коммутаторов, политик маршрутизации, ПКС в целом), целесообразно использовать, по меньшей мере, две модели – *конкретную модель* и *абстрактную модель*.

Конкретная модель может иметь сложное математическое устройство, отражающее многие особенности поведения моделируемого компонента ПКС. Конкретные модели вычислительных систем обычно сравнительно легко построить по описанию самих систем. Однако высокая точность моделирования обычно сопряжена с тем, что анализ поведения конкретной модели может быть трудной математической задачей.

Абстрактная модель может быть устроена гораздо проще за счет пренебрежения (или упрощения) некоторыми деталями поведения моделируемого компонента, и поэтому задачи анализа абстрактной модели могут иметь гораздо более простые и эффективные решения. Однако в этом случае остается открытым вопрос о выборе (или построении) корректной абстрактной модели. Основанием для такого выбора может служить конкретная модель анализируемой системы.

Таким образом, между абстрактной и конкретной моделями должна быть установлена взаимосвязь, позволяющая:

- конструировать абстрактные модели компонентов ПКС на основе конкретных моделей этих компонентов;
- переносить некоторые (строго очерченные) результаты анализа поведения абстрактных моделей компонентов ПКС на конкретные модели этих компонентов.

Примерами подобных отношений между математическими моделями могут служить отношения гомоморфизма в алгебре, отношения симуляции в моделях Крипке (размеченных системах переходов) и отношения абстракции-конкретизации в теории абстрактных интерпретаций.

Компактность описания моделей. Поскольку сложность анализа и верификации моделей вычислительной системы существенно зависит от размеров их описаний, то для наиболее перспективны те модели, описания которых наиболее лаконичны. Традиционным методом сокращения размеров моделей является их описание *символьными методами*. Указанные методы представляют внутренние структуры моделируемой системы не с помощью явного перечисления элементов модели и связей между ними, а посредством формул, диаграмм и других компактных символьных конструкций. Однако не все конструкции подобного рода допускают эффективное выполнение тех операций, которые используются в процессе построения, анализа и верификации моделей. Поэтому перспективные формальные модели компонентов ПКС должны обладать удобным символьным представлением, которое бы хорошо сочетались с перечисленными алгоритмами.

Эффективность решения задач анализа моделей. При выборе конкретной модели компонента ПКС решающим фактором является её способность максимально точно описывать особенности поведения этого компонента. Но при выборе абстрактных

моделей основное внимание нужно уделять не выразительной мощности, а разнообразию операций, использование которых они допускают, накладным расходам на применение эти операций, а так же сложности некоторых задач анализа построенных на их основе вычислений. Так, проблема достижимости состояний, тотальности и эквивалентности часто используются в качестве своеобразного критерия практической пригодности модели для проверки поведения вычислительных систем. Неразрешимость или высокая вычислительная сложность этих задач, как правило, свидетельствует об отсутствии эффективных алгоритмов и для решения задачи верификации её специальных свойств.

Инструментальная поддержка моделей. Среди абстрактных моделей вычислений, которые можно использовать для эффективной верификации компонентов ПКС, наибольший интерес представляют те из них, для которых существуют программно-инструментальные средства анализа их поведения. Адаптация этих средств для проверки специальных свойств поведения компонентов ПКС может оказаться гораздо более простой задачей, нежели разработка оригинального средства верификации.

1.5.2 Выбор моделей для компонентов ПКС

Моделируемые системы можно разделить на *трансформационные* и *реагирующие* [43].

Трансформационная система может быть представлена в виде преобразователя, который запускается, принимает входные данные, генерирует на их основе выходные данные и завершает свою работу. Такие системы не имеют внутреннего состояния и не способны изменять своё поведение с течением времени, поэтому их удобно задавать отношениями между входными и выходными данными.

В контексте ПКС в качестве трансформационных систем можно рассматривать:

- Действия и инструкции, которые выполняют преобразование заголовков пакетов;
- Правила, таблицы и конвейеры, которые определяют отношение коммутации;
- Сеть коммутаторов, которая реализует отношения пересылки и маршрутизации.

Реализованные трансформационными системами отношения удобно представлять, например, явным образом – таблицей, или символьными методами – булевыми формулами или двоичными решающими диаграммами. Табличное представление отношений коммутации было использовано, в частности, средствами верификации сети NetPlumber [44],

булевы формулы – средством Anteater [45], двоичные решающие диаграммы – средствами ConfigChecker [46], FlowChecker [6] и APVerifier [47].

На практике отношения коммутации обычно имеют достаточно большие размеры, чтобы сделать их явное табличное представление громоздким и неэффективным по сравнению с записями, которые предлагают символьные методы. При этом лаконичность записей, полученных с помощью различных символьных методов, в значительной степени зависит от конкретного отношения коммутации, и может меняться от конфигурации к конфигурации. Для отношений из некоторых классов длина их представления в виде конъюнктивой нормальной формы имеет экспоненциально больший размер, чем соответствующие им двоичные решающие диаграммы, и наоборот. В то же время, было замечено, что множества правил обработки пакетов на коммутаторах обычно обладают высокой степенью избыточности и, как правило, могут быть эффективно представлены двоичными диаграммами небольшого размера [48]. Кроме того, некоторые алгоритмически значимые операции (например, проверки эквивалентности) над булевыми формулами имеют большую вычислительную сложность, нежели над двоичными диаграммами. Поэтому именно двоичные решающие диаграммы были выбраны в качестве наиболее перспективной модели для перечисленных выше компонентов ПКС – более подробное рассмотрение указанного формализма приводится в разделе 1.5.3.

В отличие от трансформационных систем, реагирующие системы имеют внутреннее состояние, которое может изменяться под действием поступающих данных. Таким образом, результат каждого последующего взаимодействия с реагирующей системой может зависеть от произошедших до этого обменов данными.

В качестве реагирующих систем, в частности, могут выступать:

- Коммутаторы, изменяющие своё поведение после получения команд контроллера;
- Управляющие приложения контроллера, которые призваны динамически адаптировать ПКС под состояние сетевого окружения и выдавать коммутаторам новые последовательности команд после получения каждого нового сообщения из сети;
- Сеть ПКС, которая изменяет конфигурацию своего контура данных, реагируя на попытки переслать через неё новые пакеты данных.

При описании подобных систем удобно рассуждать в терминах изменения состояний, поэтому для их моделирования часто используют математические структуры, представляющие собой разнообразные обобщения конечных автоматов, например, модели Крип-

ке, автоматы реального времени, алгебры процессов и сети Петри. Описание устройства (синтаксиса) и правил функционирования (семантики) перечисленных формализмов приведено в приложении Б.

Модель системы переходов конечного автомата получила широкое использование в области формальной верификации, и служит основой метода верификации систем на модели (model checking). В частности, модель конечных автоматов используется для представления сети коммутаторов в статье [49].

Системы переходов хорошо поддаются автоматической верификации, но подвержены проблеме *взрыва состояний*, когда число состояний системы становится таким большим, что при текущем развитии вычислительной техники его невозможно перебрать за приемлемое время. Причём вместе с увеличением выразительной мощности автоматных моделей указанная проблема лишь усугубляется. Ввиду высокой вычислительной сложности задачи верификации алгебр процессов и сетей Петри их бывает сложно использовать даже для моделирования коммуникационных протоколов, содержащих лишь несколько тысяч состояний [50]. Поэтому перспективы построения практически применимого средства верификации ПКС, обладающих несравнимо большим количеством состояний, с помощью указанных методов сомнительны. В литературе так же отсутствуют свидетельства практической эффективности верификации компьютерных сетей с помощью конечных автоматов и сетей Петри со встроенными средствами хронометрирования и контролем времени. Размеры и сложность успешно анализируемых с использованием указанных методов моделей так же существенно уступают характеристикам современных ПКС [51; 52].

Таким образом, наиболее перспективной является простейшая автоматная модель ПКС – когда единственным автоматом в сети является контроллер, состояния которого определяются совокупностью действующих в сети отношений коммутации. Ввиду общеизвестности модели конечных автоматов подробное рассмотрение указанного формализма в настоящей работе не приводится.

1.5.3 Двоичные решающие диаграммы

Упорядоченные двоичные разрешающие диаграммы (Ordered Binary Decision Diagrams, OBDD) – являются канонической формой представления булевых функций [53]. Часто они оказываются значительно более компактными, нежели такие традиционные представления как дизъюнктивная нормальная форма и конъюнктивная нормальная форма. Кроме того, известны эффективные алгоритмы выполнения логических операций,

которые способны работать с OBDD непосредственно, не требуя предварительной трансляции в менее лаконичные математические структуры. По этим причинам OBDD нашли широкое применение во многих приложениях, относящихся к автоматическому проектированию, в том числе в символьном имитационном моделировании, верификации комбинационных логических схем и, совсем недавно, в верификации параллельных систем с конечным числом состояний. Дональд Кнут назвал OBDD одной из самых фундаментальных структур данных, разработанных за последние 25 лет, так как она позволила эффективно решать проблемы, которые ранее представлялись практически неразрешимыми [54].

Двоичная разрешающая диаграмма – это корневой ориентированный ациклический граф, вершины которого разбиты на два класса – терминальные вершины и нетерминальные вершины со следующими параметрами:

1. Каждая нетерминальная вершина v помечена переменной $var(v)$ и имеет две вершины-последователя: $low(v)$ и $high(v)$.

2. Каждая терминальная вершина помечена либо 0 либо 1.

3. Всякая двоичная разрешающая диаграмма B с корнем в вершине v определяет булеву функцию $f_v(x_1, \ldots, x_n)$ следующего вида:

   - В случае, когда v – терминальная вершина мы считаем, что:

     (a) если $value(v) = 1$, то $f_v(x_1, \ldots, x_n) = 1$;

     (b) если $value(v) = 0$, то $f_v(x_1, \ldots, x_n) = 0$.

   - Если v – нетерминальная вершина и $var(v) = x_i$, считаем: $f_v(x_1, \ldots, x_n) = ((\neg x_i \wedge f_{low(v)}(x_1, \ldots, x_n)) \vee ((x_i \wedge f_{high(v)}(x_1, \ldots, x_n))$

Чтобы называться упорядоченной, двоичная решающая диаграмма (OBDD), должна удовлетворять двум ограничениям на ее структуру:

1. По каждому пути из корня в терминальную вершину переменные должны следовать в одном и том же порядке;

2. В диаграмме не должно быть изоморфных подграфов или избыточных вершин.

Размер OBDD очень сильно зависит от упорядочения переменных. Вообще говоря, отыскать эффективно наилучший порядок расположения переменных невозможно. В частности, можно показать, что даже проверка того, будет ли предложенный порядок

оптимальным, является NP-полной задачей [55]. Более того, существуют последовательности булевых функций, размер OBDD для которых растет экспоненциально с увеличением числа переменных независимо от порядка их расположения [56]. Было разработано несколько эвристик поиска хороших упорядочиваний переменных в случаях, когда такого рода порядок существует. Если булева функция представлена логической схемой, то эвристики, основанные на обходе графа схемы в глубину с возвратом, обычно дают хорошие результаты.

OBDD – это один из наиболее удобных символьных способов описания булевых функций. Главные достоинства его состоят в том, что все операции булевой алгебры (конъюнкция, дизъюнкция, отрицание, эквиваленция, квантификация переменных) могут быть воспроизведены на OBDD за время, пропорциональное квадрату от размера исходных OBDD и позволяют построить диаграммы, размер которых также не превосходят размера исходных OBDD. В статье [53] представлен единый алгоритм для вычисления всех шестнадцати логических операций над OBDD. Эти алгоритмы реализованы в многочисленных пакетах программ и библиотеках для работы с OBDD, например [57—59].

1.5.4 Выводы

Проведённый сравнительный анализ моделей вычислений показал, что наиболее перспективными из них для целей настоящей работы являются следующие формализмы:

1. Двоичные разрешающие диаграммы – эта модель будет использоваться для символьного представления правил, таблиц и конвейеров коммутации в виде отношений на множестве состояний пакетов, выраженных двоичными наборами;

2. Конечные автоматы – будут моделировать поведение контроллера, взаимодействующего с сетью коммутаторов, и, в конечном счёте, определяющего поведение ПКС.

Непосредственное описание моделей для компонентов ПКС, построенных на базе выбранных формализмов, приводятся в следующем разделе.

1.6 Формальная модель ПКС

В этом разделе представлены описания двух математических моделей ПКС, которые могут быть эффективно использованы при решении задачи их верификации.

Первая из этих моделей охватывает лишь сеть коммутаторов с заданными таблицами правил OpenFlow. Эта модель не охватывает контроллер и не учитывает возможность изменения таблиц коммутации. Она предназначена для проверки статических свойств ПКС, и поэтому она была названа *статической моделью*. Достоинствами модели являются:

- Простота устройства. Семантика модели определяется бинарными отношениями на множестве состояний пакетов – булевых векторов. Указанные отношения могут быть естественным образом представлены булевыми функциями;

- Композиционные свойства модели. Семантика составных компонентов модели определяется на основе семантики ее более мелких компонентов при помощи простых средств композиции булевых функций;

- Расширяемость модели. Для ясности изложения в данном разделе приводится полное описание лишь наиболее важных элементов ПКС: состояний пакетов, действий, инструкций и таблиц коммутации. Тем не менее, из этого описания вполне ясно, каким образом можно расширить статическую модель при необходимости охватить такие средства коммутации пакетов, как конвейеры и групповые таблицы;

- Сочетаемость с языками спецификации. Статическая модель может служить интерпретациями для выражений логики транзитивного замыкания – перспективного логического языка спецификации, выбранного в разделе 1.4.

Статическая модель ПКС является развитием аналогичной модели сети коммутаторов, рассмотренной в статьях [37; 45; 46; 60; 61].

Вторая модель предназначена для описания поведения контроллера. В этой модели контроллер представлен в виде автомата, получающего сообщения от коммутаторов на входе и генерирующего последовательности команд для их реконфигурации на выходе. Множество допустимых сообщений в модели контроллера ограничено лишь заголовками тех пакетов, которые могут достичь контроллера, будучи отправленными из какой-либо внешней точки сети. Автоматная модель контроллера не учитывает его реакцию на сообщения, содержащие данные о применении правил коммутации и сведения о работоспособности элементов ПКС, так же не отражает возможности для коммутации отдельных пакетов без установки соответствующих правил, которая допускается протоколом OpenFlow.

Поскольку в рамках указанной модели проявляются динамические аспекты поведения ПКС, то она была названа *динамической моделью*. Достоинствами модели являются:

- Простота описания контроллера. Представление контроллера в виде конечного автомата позволяет моделировать множество одновременно запущенных на нём управляющих приложений. При этом конечный автомат приложения может быть получен путём анализа его кода;

- Сочетаемость с языками спецификации. Предложенная модель построена на основе формализма конечных автоматов и может служить интерпретацией для выражений темпоральных логик, которые были признаны наиболее перспективными для описания динамических свойств ПКС 1.4;

Динамическая модель ПКС является оригинальной; наиболее близким ее аналогом может служить модель ПКС, рассмотренная в статьях [35; 36; 62; 63].

1.6.1 Статическая модель ПКС

Статическая модель ПКС предназначена для проверки локальных свойств таблиц коммутации, конвейеров коммутации и коммутаторов сети, а также для верификации статических свойств ПКС. Эта модель строится в рамках следующих допущений:

- все компоненты сети: коммутаторы, порты коммутаторов, каналы связи – функционируют исправно, передающиеся через сеть пакеты не теряются;

- состав сети, а также содержимое таблиц коммутации остаются неизменными;

- взаимодействие коммутаторов с контроллерами не учитывается;

Приведённая здесь статическая модель так же не охватывает некоторые продвинутые средства коммутации пакетов, которые допускают спецификации протокола OpenFlow. В частности, предполагается, что:

- каждый коммутатор имеет единственную таблицу правил;

- каждое правило коммутации состоит из шаблона и инструкции типа *ApplyActions*;

- поскольку правила коммутации не учитывают приоритеты, то к каждому пакету применяются все правила таблицы, шаблоны которых совместимы с его заголовком;

- допускаются только действия по модификации заголовков и коммутации пакета;

- каждый заголовок пакета представляет собой одно-единственное поле.

Однако, семантика не рассмотренных средств коммутации может быть задана посредством бинарных отношений по аналогии с тем, как это сделано для основных компонентов ПКС.

Заголовки пакетов представляются двоичными векторами $\mathbf{h} = (h_1, h_2, \ldots, h_N)$; отдельные компоненты заголовков будем использовать с помощью записи вида $\mathbf{h}[i],\ \mathbf{h}[i] = h_i, 1 \leq i \leq N$. Множество всех заголовков обозначим символом $\mathcal{H},\ \mathcal{H} = \{0,1\}^N$.

Порт коммутатора представляется вектором $\mathbf{p} = (p_0, p_1, p_2, \ldots, p_k)$, для его компонент будем использовать записи вида $\mathbf{p}[i],\ \mathbf{p}[i] = p_i, 0 \leq i \leq k$. Если $\mathbf{p}[0] = 1 (\mathbf{p}[0] = 0)$, то порт $\mathbf{p}$ считается входным (выходным). Будем полагать, что все коммутаторы имеют одинаковое число портов. Множество всех (входных, выходных) портов коммутатора обозначим записью $\mathcal{P}$ (соответственно $\mathcal{IP}, \mathcal{OP}$).

Выходной порт $\mathbf{p} = (0, 0, \ldots, 0)$ будем считать портом сброса пакетов и обозначать записью *drop*. Выходной порт $\mathbf{p} = (0, 1, 1, \ldots, 1)$ будем считать портом канала управления и обозначать записью *octr*; через этот порт коммутатор отправляет сообщения контроллеру. Входной порт $\mathbf{p} = (1, 1, 1, \ldots, 1)$ – это входной порт канала управления; он обозначается записью *ictr*, и через этот порт коммутатор получает команды от контроллера.

Индивидуальным именем каждого коммутатора служит двоичный вектор $\mathbf{w} = (w_1, w_2, \ldots, w_m)$. Множество всех имен коммутаторов обозначим записью $\mathcal{W}$.

Пары $\langle \mathbf{h}, \mathbf{p} \rangle,\ \mathbf{h} \in \mathcal{H}, \mathbf{p} \in \mathcal{P}$, назовем *локальными состояниями пакетов*, а пары $\langle \mathbf{p}, \mathbf{w} \rangle,\ \mathbf{p} \in \mathcal{P}, \mathbf{w} \in \mathcal{W}$, – *точками сети*. Тройки $\langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle,\ \mathbf{h} \in \mathcal{H}, \mathbf{p} \in \mathcal{P}, \mathbf{w} \in \mathcal{W}$, назовем *состояниями пакетов*. Множество состояний пакетов обозначим буквой $\mathcal{S}$.

Шаблоном заголовка назовем троичный вектор $\mathbf{z} = (\sigma_1, \sigma_2, \ldots, \sigma_N)$, где $\sigma_i \in \{0, 1, *\}, 1 \leq i \leq N$, а *шаблоном порта* – подобного же рода троичный вектор $\mathbf{y} = (\delta_1, \delta_2, \ldots, \delta_k)$. Шаблоны используются как для выбора правил коммутации пакетов, так и для модификации заголовков пакетов.

Мы рассматриваем два типа действий: *действие коммутации* $OUTPUT(\mathbf{y})$, где $\mathbf{y} \in \mathcal{OP}$, и *действие модификации заголовка* $SET\_FIELD(\mathbf{z})$, где $\mathbf{z}$ – шаблон заголовка. Конечная последовательность действий называется *инструкцией*.

Правило коммутации пакетов определяется тройкой $\mathbf{r} = \langle \mathbf{z}, \mathbf{y}), \alpha \rangle$, где $\mathbf{z}$ и $\mathbf{y}$ – это шаблоны заголовка и порта, а α – инструкция.

Топология сети и функциональность коммутаторов описываются бинарными отношениями на множествах точек и локальных состояний пакетов; эти отношения задаются посредством квантифицированных булевых формул. В указанных формулах мы будем использовать две параметризованные вспомогательные функции $U_\sigma(u, v)$ и $E_\sigma(u)$, где $\sigma \in \{0, 1, *\}$, а u, v – двоичные векторы:

- если $\sigma = *$, то $U_\sigma(u, v) = u \equiv v$ и $E_\sigma(u) = 1$;
- если $\sigma \in 0, 1$, то $U_\sigma(u, v) = u \equiv \sigma$ и $E_\sigma(u) = u \equiv \sigma$.

Действие $a = OUTPUT(\mathbf{y})$ отправляет пакеты без изменения их заголовков во все выходные порты, имена которых подпадают под шаблон $\mathbf{y} = (\delta_1, \delta_2, \ldots, \delta_k)$. Действие $b = SET\_FIELD(\mathbf{z})$ изменяет заголовки пакетов в соответствии с шаблоном $\mathbf{z} = (\sigma_1, \sigma_2, \ldots, \sigma_N)$: бит заголовка $\mathbf{h}[i]$ остается неизменным, если $\mathbf{z}[i] = *$; в противном случае $\mathbf{h}[i]$ изменяет свое значение на $\mathbf{z}[i]$. Семантика обоих действий описывается бинарными отношениями на множестве локальных состояний пакетов $\mathcal{H} \times \mathcal{P}$:

$$R_a(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) = \bigwedge_{i=1}^{N} (\mathbf{h}[i] \equiv \mathbf{h}'[i]) \wedge \bigwedge_{i=0}^{k} U_{\delta_i}(\mathbf{p}'[i], \mathbf{p}[i])$$

$$R_b(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) = \bigwedge_{i=1}^{N} U_{\sigma_i}(\mathbf{h}'[i], \mathbf{h}[i]) \wedge \bigwedge_{i=0}^{k} (\mathbf{p}[i] \equiv \mathbf{p}'[i])$$

Инструкция α вычисляет последовательную композицию составляющих ее действий. Если α – пустая последовательность, то пакет коммутируется в порт $drop$, то есть сбрасывается. Поэтому будем полагать по умолчанию, что всякая инструкция завершается действием коммутации пакета. Семантика инструкции α описывается бинарным отношением R_α, которое определяется так:

1. если инструкция α пустая, то $R_\alpha = \mathbf{false}$;
2. если $\alpha = a, \beta$, то отношение R_α задается следующими формулами в зависимости от действия a:
   - если a – это действие коммутации пакета, то
   $$R_\alpha(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) = R_a(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) \vee R_\beta(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle)$$
   - если a – это действие модификации заголовка пакета, то
   $$R_\alpha(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) = \exists \mathbf{h}'' \left(R_a(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}'', \mathbf{p} \rangle) \wedge R_\beta(\langle \mathbf{h}'', \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) \right)$$

Правило коммутации пакетов $r = \langle \mathbf{z}, \mathbf{y}, \alpha \rangle$ применяет инструкцию α ко всем тем пакетам, локальные состояния которых подпадают под соответствующие шаблоны заголовка $\mathbf{y}$ и порта $\mathbf{z}$. Семантика правила определяется следующим бинарным отношением R_r на множестве локальных состояний пакетов $\mathcal{H} \times \mathcal{P}$:

$$R_r(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) = precond_r(\langle \mathbf{h}, \mathbf{p} \rangle) \wedge R_\alpha(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle)$$

где предикат

$$precond_r(\langle \mathbf{h}, \mathbf{p} \rangle) = \bigwedge_{i=1}^{N} E_{\sigma_i}(\mathbf{h}[i]) \wedge \bigwedge_{i=0}^{k} E_{\delta_i}(\mathbf{p}[i])$$

играет роль *предохранителя* правила r.

Таблицей коммутации tab называется пара $\langle D, \beta \rangle$, где $D = \{r_1, r_2, \ldots, r_n\}$ – множество правил коммутации, а β – инструкция умолчания. Коммутатор применяет правила таблицы ко всем пакетам, поступающим на его входные порты. Если ни одно из правил множества D нельзя применить к пакету, то этот пакет обрабатывается инструкцией β. Семантика таблицы коммутации tab задается следующим бинарным отношением:

$$\begin{aligned} R_{tab}(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) = \quad & \textstyle\bigvee_{i=1}^{n} R_{r_i}(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) \vee \\ & \neg\big(\textstyle\bigwedge_{i=1}^{n} precond_{r_i}(\langle \mathbf{h}, \mathbf{p} \rangle)\big) \wedge R_{\beta}(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) \end{aligned}$$

на множестве локальных состояний пакетов $\mathcal{H} \times \mathcal{P}$. Множество всевозможных таблиц коммутации для заданной сети обозначим записью Tab.

Топология сети полностью определяется *отношением пересылки* пакетов $T \subseteq (\mathcal{OP} \times \mathcal{W}) \times (\mathcal{IP} \times \mathcal{W})$. Хотя в рамках нашей модели допустимы любые отношения указанного выше типа, для реальных сетей отношение T должно быть инъективной функцией. Точки, вовлеченные в отношение T, называются *внутренними точками* сети; все прочие точки сети считаются её *внешними точками*. Будем использовать записи In и Out для обозначения множеств внешних входных и внешних выходных точек сети. Предполагается, что внешние точки сети подключены к внешним устройствам (контроллерам, серверам, сетевым шлюзам и так далее), которые находятся вне сферы влияния контроллера ПКС. Пакеты поступают в сеть через внешние входные точки и покидают сеть через внешние выходные точки.

Для заданного множества коммутаторов $\mathcal{W}$ и топологии T *конфигурацией сети* называется всюду определенная функция $Net : W \to Tab$, ассоциирующая с каждым коммутатором сети некоторую таблицу коммутации пакетов. Семантика сети с заданной конфигурацией Net определяется отношением

$$\begin{aligned} R_{Net}(\langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle, \langle \mathbf{h}', \mathbf{p}', \mathbf{w}' \rangle) = \quad & \big(C_{Net}(\langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle, \langle \mathbf{h}', \mathbf{p}', \mathbf{w}' \rangle) \wedge Out(\mathbf{p}', \mathbf{w}')\big) \vee \\ & \exists p'' \ \big(C_{Net}(\langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle, \langle \mathbf{h}', \mathbf{p}'', \mathbf{w} \rangle) \wedge T(\langle \mathbf{p}'', \mathbf{w} \rangle, \langle \mathbf{p}', \mathbf{w}' \rangle\big) \end{aligned}$$

на множестве состояний пакетов $\mathcal{S} = \mathcal{H} \times \mathcal{P} \times \mathcal{W}$, где

$$C_{Net}(\langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle, \langle \mathbf{h}', \mathbf{p}', \mathbf{w}' \rangle) = \big(\bigvee_{\mathbf{w} \in \mathcal{W}} R_{Net(\mathbf{w})}(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle)\big) \wedge \bigwedge_{i=1}^{m} (w_i \equiv w_i')$$

Если отношение $R_{Net}(\mathbf{s}, \mathbf{s}')$ выполняется для пары состояний пакетов $\mathbf{s} = \langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle$ и $\mathbf{s}' = \langle \mathbf{h}', \mathbf{p}', \mathbf{w}' \rangle$, то пакет с заголовком $\mathbf{h}$, поступающий на порт $\mathbf{p}$ коммутатора $\mathbf{w}$, может быть за один скачок отправлен либо на входной порт $\mathbf{p}'$ коммутатора $\mathbf{w}'$, либо на устройство, подключенное к внешнему выходному порту $\mathbf{p}'$ коммутатора $\mathbf{w}$.

Семантика статической модели ПКС Net определяется отношением одношаговой маршрутизации пакетов $R_{Net}(\langle \mathbf{w}, \mathbf{p}, \mathbf{h} \rangle, \langle \mathbf{w}', \mathbf{p}', \mathbf{h}' \rangle)$, которое составляет реализуемые сетью правила коммутации пакетов, а также свойствами $In(\mathbf{w}, \mathbf{p})$ и $Out(\mathbf{w}, \mathbf{p})$, описывающими множество внешних точек сети. Располагая этими отношениями можно проследить маршруты всех потоков, проходящих через сеть при заданной ее конфигурации.

1.6.2 Динамическая модель ПКС

Динамическая модель ПКС предназначена для проверки свойств поведения всей ПКС в целом как системы, состоящей из контроллера и множества коммутаторов, с которыми он взаимодействует. Приведённая в данном разделе динамическая модель строится в рамках следующих допущений:

- коммутаторы передают контроллеру лишь сообщения с состояниями пакетов, которые были отправлены в управляющий порт в результате применения правил коммутации;
- коммутаторы сети могут получать от контроллеров только команды добавления новых и удаления существующих правил из их таблиц коммутации;
- множество команд добавления правил коммутации, отправленных контроллером на коммутаторы, выполняется ими в некотором заранее описанном порядке;
- контроллер обрабатывает очередное поступающее на него сообщение сообщение только после того, как коммутаторы выполнили все те команды, которые были выработаны контроллером в результате обработки предыдущего сообщения.

В рамках перечисленных допущений, контроллер – это реагирующая программа, которая получает сообщения от коммутаторов по каналам управления и вырабатывает отклики, которые изменяют содержимое таблиц коммутации. Отправляя сообщение контроллеру, коммутатор выражает требование модифицировать его таблицу коммутации: это сообщение означает, что коммутатор не имеет подходящего правила для обработки поступившего пакета. Само сообщение представляет собой состояние данного пакета: совокупность его заголовка, имени коммутатора и порта, на который этот пакет поступил.

Контроллер может вырабатывать два типа команд: для добавления и удаления правила. Команда $add(\mathbf{w}, r)$, где $\mathbf{w} \in \mathcal{W}$ и r – это правило коммутации, требует вставить правило r в таблицу коммутатора $\mathbf{w}$. Команда $del(\mathbf{w}, \mathbf{z}, \mathbf{y})$, где $\mathbf{w} \in \mathcal{W}$, и $\mathbf{z}, \mathbf{y}$ – шаблоны порта и пакета, требует удалить из таблицы коммутатора $\mathbf{w}$ все правила $r = \langle \mathbf{z}', \mathbf{y}', \alpha \rangle$ в тех случаях, когда шаблоны этих правил $\mathbf{z}', \mathbf{y}'$ покрываются шаблонами $\mathbf{z}, \mathbf{y}$ соответственно.

Обозначим буквой $\mathcal{C}$ множество всех возможных команд. Команды обоих типов изменяют конфигурацию сети; мы будем использовать запись $Net' = update(cmd, Net)$ для обозначения того, что команда cmd преобразовала конфигурацию Net в конфигурацию Net'. Если $\omega = cmd_1, cmd_2, \ldots, cmd_n$ – это конечная последовательность команд, то выражение $update(\omega, Net)$ будет считаться сокращенной записью композиции из нескольких преобразований конфигурации:

$$update(\omega, Net) = update(cmd_n, update(\ldots, update(cmd_2, update(cmd_1, Net))\ldots))$$

Формальной моделью контроллера служит дискретный преобразователь $A = (\mathcal{S}, \mathcal{C}, Q, q_0, \Delta)$, в котором

- $\mathcal{S}$ и $\mathcal{C}$ – входной и выходной алфавиты соответственно,
- Q – множество внутренних состояний управления контроллера,
- $q_0,\ q_0 \subseteq Q$ – начальное состояние управления, и
- $\Delta,\ \Delta \subseteq Q \times mathcalS \times \mathcal{C}^* \times Q$ – отношение переходов.

Четверка $(q, \mathbf{s}, \omega, q_0)$ из Δ означает, что контроллер A после получения сообщения $\mathbf{s}$ в состоянии управления q может выработать конечную последовательность команд ω и перейти в состояние управления q_0.

Для каждой конфигурации сети Net контроллер A может получать только такие сообщения, которые были индуцированы пакетами, поступившими на внешние точки сети. Рефлексивно-транзитивное замыкание R^*_{Net} отношения одношаговой коммутации R_{Net} позволяет определить множество сообщений $Event(Net)$, допустимых в конфигурации Net следующим образом:

$$Event(Net) = \{\langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle :\ Visible(\mathbf{h}, \mathbf{p}, \mathbf{w})\}$$

$$Visible(\mathbf{h}, \mathbf{p}, \mathbf{w}) = \exists \mathbf{h}', \mathbf{p}', \mathbf{w}', \mathbf{h}'' \left(In(\langle \mathbf{p}', \mathbf{w}' \rangle) \wedge R^*_{Net}(\langle \mathbf{h}', \mathbf{p}', \mathbf{w}' \rangle, \langle \mathbf{h}'', \mathbf{p}, \mathbf{w} \rangle) \wedge R_{Net}(\langle \mathbf{h}'', \mathbf{p}, \mathbf{w} \rangle, \langle \mathbf{h}, octr, \mathbf{w} \rangle)\right)$$

Динамическая модель ПКС определяется множествами $\mathcal{W}, \mathcal{P}, \mathcal{H}$ коммутаторов, портов и заголовков пакетов, отношением пересылки пакетов T и контроллером A. *Частичный прогон* ПКС $M = (\mathcal{W}, \mathcal{P}, \mathcal{H}, T, A)$ представляет собой последовательность (конечную или бесконечную) вида

$$run = (Net_0, q_0) \xrightarrow{\mathbf{s}_1} (Net_1, q_1) \xrightarrow{\mathbf{s}_2} \cdots \xrightarrow{\mathbf{s}_i} (Net_i, q_i) \xrightarrow{\mathbf{s}_{i+1}} (Net_{i+1}, q_{i+1}) \xrightarrow{\mathbf{s}_{i+2}} \cdots$$

где для каждого $i,\ 0 \leq i$,

- Net_i – сетевая конфигурация, q_i – состояние контроллера A, и $\mathbf{s}_i$ – состояние пакета,
- $\mathbf{s}_{i+1} \in Event(Net_i)$,
- отношение переходов контроллера A содержит такую четверку $(q_i, \mathbf{s}_{i+1}, \omega_i, q_{i+1})$, что $Net_{i+1} = update(\omega_i, Net_i)$.

Пары (Net_i, q_i) мыслятся как состояния ПКС, а состояния пакетов $\mathbf{s}_{i+1}$ играют роль сообщений, отправленных контроллеру. *Полный прогон* – это частичный прогон, который либо является бесконечным, либо завершается в таком состоянии ПКС (Net_i, q_i), что $Event(Net_i) = \emptyset$. Для заданной ПКС M и конфигурации сети Net_0 запись $Run(M, Net_0)$ будет обозначать множество всех полных прогонов M, начинающихся парой (Net_0, q_0).

В следующем разделе приводится описание разработанных языков спецификации политик маршрутизации. Предложенные статическая и динамическая модели ПКС используются в качестве интерпретаций для выражений указанных языков.

1.7 Язык спецификации политик маршрутизации

В данном разделе приводится описание иерархии из трёх языков спецификации политик маршрутизации, построенных на основе проведённого анализа требований, предъявляемых к ПКС 1.3, обзора перспективных логических языков спецификации 1.4, а так же построенной ранее математической модели ПКС 1.6.

Предложенные языки образуют иерархию, в которой выражения языка нижележащего уровня выступают в роли атомарных высказываний для языка вышележащего уровня:

- Язык первого уровня $\mathcal{L}_0$ предназначен для описания свойств простейших элементов сети – отдельных состояний пакетов и их множеств – например, шаблонов правил коммутации. Поскольку семантика этих компонентов сети определяется двоичными векторами, математическую основу этого языка составляет булева алгебра.

- Язык второго уровня $\mathcal{L}_1$ описывает глобальные статические требования, значительная часть которых представляет собой ограничения на отношения коммутации и маршрутизации. Для удобства описания подобных требований язык $\mathcal{L}_1$ предлагает использовать фрагмент логики первого порядка, расширенной оператором транзитивного замыкания.

- Язык третьего уровня $\mathcal{L}_2$ предназначен для описания динамических требований к ПКС. Эти требования описывают желаемую реакцию контроллера на сообщения, поступающие от коммутаторов. В качестве основы для данного языка была выбрана логика линейного времени LTL; в качестве атомарных высказываний которой выступают формулы $\mathcal{L}_1$, описывающие свойства конфигурации сети.

1.7.1 $\mathcal{L}_0$: язык описания состояний пакетов

Рассмотрим множество переменных $Var = \{X_1, X_2, \ldots\}$, принимающих значения из множества состояний пакетов $\mathcal{S} = \mathcal{H} \times \mathcal{P} \times \mathcal{W} = \{0,1\}^{N+k+m}$. Спецификацией состояния пакета назовем всякую булеву формулу ϕ над множеством булевых переменных $\{X_i[j] : X_i \in Var, 1 \leq j \leq N+k+m\}$ и связок $\neg, \wedge$. Множество всех таких формул обозначим записью $\mathcal{L}_0$.

Нетрудно видеть, что формулы из $\mathcal{L}_0$ позволяют выразить любую булеву функцию на множестве состояний пакетов или, что то же самое, выделить любое подмножество указанных состояний. Таким образом, $\mathcal{L}_0$ образует базовый язык, формулы которого могут использоваться при задании отношений, специфицирующих поведение коммутирующих элементов сети.

1.7.2 $\mathcal{L}_1$: язык спецификации статических свойств

Для задания отношения пересылки, коммутации и маршрутизации пакетов предлагается использовать язык $\mathcal{L}_1$, основанный на логике транзитивного замыкания $FO[TC]$. В языке $\mathcal{L}_1$ используются единственный двухместный предикатный символ R и два одноместных предикатных символа I и O; это наименьший язык, удовлетворяющий следующим требованиям:

1. если $\phi \in \mathcal{L}_0$, то $\phi \in \mathcal{L}_1$;

2. если $X, Y \in Var$, то атомарные выражения $R(X,Y)$, $I(X)$, $O(Y)$ принадлежат $\mathcal{L}_1$;

3. если ψ_1 и ψ_2 – формулы из $\mathcal{L}_1$, и $X \in Var$, то формулы $(\neg\psi_1)$, $(\psi_1 \wedge \psi_2)$ и $(\exists X \psi_1)$ так же принадлежат $\mathcal{L}_1$;

4. если $\psi(X, Y)$ – формула из $\mathcal{L}_1$, содержащая не более двух свободных переменных, то $TC(\psi(X, Y)) \in \mathcal{L}_1$.

Семантика языка $\mathcal{L}_1$ определяется с помощью отношений из статической модели сети, описание которой было приведено в предыдущем разделе. Пусть Net – некоторая конфигурация сети, и $\mathbf{s} = \langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle$ и $\mathbf{s}' = \langle \mathbf{h}', \mathbf{p}', \mathbf{w}' \rangle$ – пара состояний пакетов. Тогда

1. $Net \models I(X)[\langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle]$ тогда и только тогда, когда $\langle \mathbf{p}, \mathbf{w} \rangle \in In$;

2. $Net \models O(X)[\langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle]$ тогда и только тогда, когда $\langle \mathbf{p}, \mathbf{w} \rangle \in Out$;

3. $Net \models R(X, Y)[\mathbf{s}, \mathbf{s}']$ тогда и только тогда, когда $(\mathbf{s}, \mathbf{s}') \in R_{Net}$;

4. $Net \models TC(\psi(X, Y))[\mathbf{s}, \mathbf{s}']$ тогда и только тогда, когда существует конечная последовательность состояний пакетов $\mathbf{s}_0, \mathbf{s_1}, \ldots, \mathbf{s_l}$, где $0 < l$, что $\mathbf{s_0} = \mathbf{s}$ и $\mathbf{s}_l = \mathbf{s}'$, и для каждого i, $1 \leq i \leq l$ выполнено условие $Net \models \psi(X, Y)[\mathbf{s}_{i-1}, \mathbf{s}_i]$.

Отношение выполнимости для прочих формул $\mathcal{L}_1$ определяется стандартным образом.

Как это было показано в работах [40; 41], любые формулы темпоральных логик LTL, CTL, CTL* и μ-исчисления можно транслировать в логику транзитивного замыкания $FO[TC]$. Следовательно построенный на базе указанной логики язык $\mathcal{L}_1$ позволяет выражать, в частности, любые статические свойства сети, которые можно задать формулами темпоральных логик. Что касается алгоритмической сложности использования языка $\mathcal{L}_1$ – в процессе разработки и исследования характеристик $\mathcal{L}_1$ была доказана следующая

Теорема 1. Задача верификации модели сети $Net \models \psi$ относительно замкнутых формул языка $\mathcal{L}_1$ является PSPACE-полной.

Доказательство. В работе [64] было показано, что задача верификации моделей программ относительно формул $FO[TC]$ является NLOG-полной, и, следовательно, принадлежит классу сложности PSPACE. Обоснование PSPACE-трудности этой задачи основывается на том, что заголовок пакета может мыслиться как ограниченная лента машины Тьюринга. Команды машины Тьюринга, оперирующей на ограниченной ленте, моделируются правилами коммутации пакетов. Для моделирования самой машины Тьюринга достаточно одного коммутатора с двумя портами, соединенными друг с другом каналом связи. Тогда вопрос о завершаемости вычисления машины Тьюринга с ограниченной

лентой, являющийся PSPACE-полной задачей ([65], оказывается равносильным вопросу о сбрасываемости пакета, поступившего на входной порт такого коммутатора. □

1.7.3 $\mathcal{L}_2$: язык спецификации динамических свойств

В качестве математической основы языка $\mathcal{L}_2$ спецификаций динамических свойств поведения ПКС предлагается использовать фрагмент языка темпоральной логики линейного времени $FO[TC]$, атомарными высказываниями которого выступают замкнутые формулы языка $\mathcal{L}_1$.

Синтаксис языка $\mathcal{L}_2$ определяется следующим образом. Пусть $\mathcal{L}_0(X)$ и $\mathcal{L}_1(X)$ – множества всех тех формул языков $\mathcal{L}_0$ и $\mathcal{L}_1$ соответственно, в которых единственная свободная переменная X. Тогда $LTL(\mathcal{L}_1)$ – это наименьший язык, удовлетворяющий следующим требованиям:

1. если $\psi(X) \in \mathcal{L}_1(X)$, то $\psi(X) \in LTL(\mathcal{L}_1)$;
2. если $\Phi(X), \Psi(X) \in LTL(\mathcal{L}_1)$, то формулы $(\neg\Phi(X))$, $(\Phi(X) \wedge \Psi(X))$, $(\mathbf{X}\ \Phi(X))$ и $(\Phi(X)\ \mathbf{U}\ \Psi(X))$ принадлежат $LTL(\mathcal{L}_1)$.

Оценка истинности формул из $LTL(\mathcal{L}_1)$ производится на бесконечных последовательностях сетевых конфигураций $\{Net_i\}_{i=1}^{\infty}$ для заданного состояния пакета $\mathbf{s}$:

- если $\psi(X) \in \mathcal{L}_1(X)$, то $\{Net_i\}_{i=1}^{\infty} \models \psi(\mathbf{s})$ тогда и только тогда, когда $Net_1 \models \psi(\mathbf{s})$,
- $\{Net_i\}_{i=1}^{\infty} \models \mathbf{X}\ \Phi(\mathbf{s})$ тогда и только тогда, когда $\{Net_i\}_{i=2}^{\infty} \models \Phi(\mathbf{s})$,
- $\{Net_i\}_{i=1}^{\infty} \models \Phi(\mathbf{s})\ \mathbf{U}\ \Psi(\mathbf{s})$ тогда и только тогда, когда $\{Net_i\}_{i=k}^{\infty} \models \Psi(\mathbf{s})$ для некоторого $k, 1 \leq k$, и $Net_j \models \Phi(\mathbf{s})$ для каждого $j, 1 \leq j < k$,
- семантика связок $\neg$ и $\wedge$ определяется обычным образом.

Язык спецификации политик маршрутизации $\mathcal{L}_2$ состоит из множества выражений вида $\phi(X) \Rightarrow \Phi(X)$, где $\phi(X) \in \mathcal{L}_0(X)$, а $\Phi(X)$ – темпоральная формула языка $LTL(\mathcal{L}_1)$. Семантика этих выражений определяется посредством отношения выполнимости на прогонах формальных моделей ПКС. Для заданного прогона run модели ПКС M и выражения $\phi(X) \Rightarrow \Phi(X)$ языка $\mathcal{L}_2$ отношение $run \models \phi(X) \Rightarrow \Phi(X)$ имеет место тогда и только тогда, когда $\{Net_i\}_{i=n}^{\infty} \models \Phi(\mathbf{s}_n)$ для каждого $n,\ 1 \leq n$, удовлетворяющего условию $\phi(\mathbf{s}_n) = 1$.

Политика маршрутизации PFP задаётся ограничением ψ на множество допустимых начальных конфигураций сети, представляющим собой замкнутую формулу языка $\mathcal{L}_1$, и конечным множеством $\{\phi_1(X) \Rightarrow \Phi_1(X), \ldots, \phi_n(X) \Rightarrow \Phi_n(X)\}$ выражений языка $\mathcal{L}_2$. Считается, что ПКС M реализует политику маршрутизации PFP в том случае, когда для любой сетевой конфигурации Net_0, удовлетворяющей условию $Net_0 \models \psi$, каждый прогон run из множества $Run(M, Net_0)$ удовлетворяет соотношению $\phi_i(X) \Rightarrow \Phi_i(X),\ 1 \leq i \leq n$. Таким образом, задача верификации моделей ПКС состоит в том, чтобы проверить для заданной формальной модели ПКС M, удовлетворяет ли она спецификации политики коммутации PFP.

Если контроллер ПКС представляет собой конечный автомат, сформулированная задача верификации разрешима. Это объясняется тем, что конечная сеть может иметь лишь конечное число конфигураций, и поэтому все прогоны конечной модели ПКС можно представить в конечной размеченной системе переходов. Таким образом, указанная задача сводится к хорошо известной проблеме верификации конечных моделей программ для темпоральной логики LTL. Однако использовать на практике такой подход затруднительно, поскольку размер пространства состояний этой системы переходов оценивается двойной экспонентой от размера описания модели ПКС.

На момент завершения настоящей работы более эффективный способ преодоления этого препятствия не найден. Тем не менее, в рамках работы было построено программно-инструментальное средство верификации сетевых конфигураций относительно их спецификаций на языке $\mathcal{L}_1$. Описание реализованного программно-инструментального средства приводится в следующем разделе.

1.8 Программная реализация верификатора ПКС

В данном разделе приводится описание верификатора ПКС – инструментального средства для проверки конфигурации ПКС на соответствие политикам маршрутизации, заданных множеством спецификаций политик маршрутизации. Указанное средство включает в свой состав три компонента:

Конструктор моделей строит множество диаграмм OBDD, предназначенных для описания ПКС согласно семантике статической модели из раздела 1.6;

Анализатор спецификаций предназначен для разбора выражений языка спецификации политик маршрутизации $\mathcal{L}_1$, представленного в разделе 1.7;

Верификатор ПКС позволяет убедиться в том, что свойства заданной конфигурации ПКС удовлетворяют множеству предъявляемых к ней требований, или, наоборот, обнаружить факт их нарушения.

Реализации каждого из указанных средств были выполнены на языке C++. Далее приводится описание их внутреннего устройства и основных принципов работы.

1.8.1 Построение модели ПКС по её конфигурации

Разработанный конструктор моделей принимает на вход файл, в котором содержится информация о топологии ПКС и правилах протокола OpenFlow, находящихся в таблицах её коммутаторов. На основании указанных данных конструктор модели строит множество двоичных решающих диаграмм OBDD, моделирующих поведение ПКС. Для работы с диаграммами OBDD используется библиотека BuDDy [57].

Как это было показано в разделе 1.5.3, упорядоченные двоичные решающие диаграммы OBDD могут рассматриваться как бинарные деревья с фиксированным количеством *ярусов*, вершины каждого из которых размечены одной из переменных моделируемого отношения. При этом соответствие между ярусами и переменными является взаимно однозначным: каждой переменной сопоставлен ровно один ярус бинарного дерева, а каждому ярусу дерева – ровно одна переменная моделируемого этим деревом отношения.

В целях повышения эффективности BuDDy объединяет деревья всех созданных диаграмм в единый ациклический направленный граф.

Регистры переменных

Представленные в разделе 1.6 модели ПКС предлагают описывать отдельные компоненты ПКС с помощью наборов булевых переменных, выражающих множества состояний пакетов, и отношений, определяющих взаимосвязи между такими множествами. Двоичные диаграммы OBDD представляют указанные отношения в форме булевых функций от нескольких независимых наборов двоичных переменных, представляющих состояния пакетов битовыми строками. Каждый такой набор переменных будем называть *регистром*.

Поскольку увеличение количества регистров переменных приводит к потенциальному увеличению размеров диаграмм OBDD и снижению эффективности операций над ними, то число использованных в рамках разработанной реализации регистров было ограничено тремя – ровно три регистра необходимо для вычисления транзитивного замыкания двухместного отношения. Для представления тех отношений, которые моделируют отдельные

компоненты ПКС, или же являются результатами промежуточных вычислений спецификаций языка $\mathcal{L}_1$, вообще говоря, достаточно и двух регистров.

Формирование отношений

Построение двоичных диаграмм OBDD, предназначенных для моделирования поведения ПКС, производится в несколько этапов по формулам, описанным в разделе 1.6, от простых к сложным. Далее приводится последовательное описание практических аспектов реализации указанных этапов.

Правила коммутации. Предполагается, что правила задаются парами из *шаблона* и набора *трафаретов*. И шаблоны, и трафареты представляют собой троичные вектора, длины которых соответствуют длине битового представления локального состояния пакета, а элементы – принадлежат множеству $\{0, 1, *\}$.

Сопоставление с шаблоном используются при выборе правил, применимых к пакетам, находящимся в заданном состоянии – подробный алгоритм выполнения указанной операции был приведён в разделе 1.6.

Наложение трафарета применяется при вычислении локального состояния, которым будет обладать пакет, обработанный по выбранному правилу:

- если в некоторой позиции трафарета находится символ «*», то соответствующий бит локального состояния остаётся неизменным после преобразования,
- если там находится 0 или 1 – этот бит переписывается указанным значением.

Необходимо отметить, что текущая версия спецификации OpenFlow [29] позволяет задавать побитовые маски для сопоставления лишь некоторых полей, шаблоны для значительного количества полей заголовков должны целиком состоять из нулей и единиц, или же из wildcard-символов «*». Спецификации так же не позволяют модифицировать конкретные биты полей заголовка. Иначе говоря, указанное правило применяется и ко всем полям трафаретов. Таким образом, абстрактное представление правил с помощью шаблонов и трафаретов является несколько более гибким, чем их описания, приведённые спецификациями протокола OpenFlow.

Коммутаторы. В рамках работы сделано предположение о том, что:

- коммутаторы имеют единственную таблицу правил,

- все правила в этой таблице имеют равные приоритеты,
- если к пакету применимо сразу несколько правил с перекрывающимися шаблонами, то коммутатор исполняет инструкции всех этих правил.

Вычисление отношения OBDD, моделирующего поведение коммутатора, производится на основе отношений, соответствующих отдельным правилам преобразования состояний пакетов из его таблицы коммутации:

1. Отношения отдельных правил таблицы объединяются между собой;
2. Те состояния пакетов, которые остаются непокрытыми правилами из таблицы, используются для формирования правила по умолчанию;
3. Формируется итоговое отношение преобразования локальных состояний пакетов, которое моделирует поведение таблицы;
4. Указанное отношение проецируется на конкретный коммутатор.

Важно понимать, что поведение коммутатора, поддерживающего несколько таблиц и приоритеты правил, воспроизводимо с помощью коммутатора с единственной таблицей правил и без поддержки приоритетов, хотя количество использованных правил при этом может существенно отличаться. Таким образом, хотя перечисленные ограничения и способны привести к трудностям при анализе конфигураций некоторых ПКС, они не приводят к снижению качества верификации.

Основные отношения: In, Out и R_{step}. На данном этапе происходит построение отношения пересылки пакетов $T \subseteq (\mathcal{OP} \times \mathcal{W}) \times (\mathcal{IP} \times \mathcal{W})$. Для каждого канала передачи данных, содержащегося в топологии сети, в отношение T добавляется пара связей между точками, которые этот канал связывает между собой.

Полученное в результате отношение связности используется для задания отношений, идентифицирующих множества входных (In) и выходных (Out) внешних точек сети:

- $In(\langle \mathbf{p}, \mathbf{w} \rangle) = \exists \langle \mathbf{p}', \mathbf{w}' \rangle \ (T(\langle \mathbf{p}, \mathbf{w} \rangle, \langle \mathbf{p}', \mathbf{w}' \rangle))$;
- $Out(\langle \mathbf{p}, \mathbf{w} \rangle) = \exists \langle \mathbf{p}', \mathbf{w}' \rangle \ (T(\langle \mathbf{p}', \mathbf{w}' \rangle, \langle \mathbf{p}, \mathbf{w} \rangle))$;

Необходимо отметить, что в указанных предположениях кодировка каждого входного внешнего порта отличается от кодировки соответствующего ему выходного внешнего порта лишь первым битом. Поэтому отношения In и Out могут быть вычислены друг

из друга. Например, отношение In можно представить через отношение Out по формуле $In(\langle \mathbf{p}, \mathbf{w} \rangle) = \exists \mathbf{p}[0](Out(\langle \mathbf{p}, \mathbf{w} \rangle)) \wedge \mathbf{p}[0] = 1$. Таким образом, суммарная сложность вычислений может быть несколько снижена.

Отношение отдношаговой маршрутизации R_{step} вычисляется на основе отношений, моделирующих поведение коммутаторов сети, и отношения пересылки пакетов T:

1. Отношения для отдельных коммутаторов объединяются между собой;
2. Набор переменных, соответствующий выходному состоянию пакета после его обработки на коммутаторе, заменяется состоянием, связанным с ним отношением пересылки пакетов, там где это возможно.

Отношение маршрутизации R^*_{step}. Отношение маршрутизации вычисляется как транзитивное замыкание отношения R_{step}. Содержательно, оно представляет собой отношение достижимости на множестве состояний пакетов и включает в себя всевозможные пары состояний, для которых существует последовательность преобразований, переводящая пакет из первого состояния во второе.

Расчёт транзитивного замыкания осуществляется путём итеративной композиции отношения до тех пор, пока не будет достигнуто состояние неподвижной точки. Поскольку отношение одношаговой маршрутизации R_{step} может содержать цепочку преобразований, переводящую пакет через каждое возможное состояние пакета, то в худшем случае число итераций при вычислении транзитивного замыкания может достигать числа переменных в кодировке состояния пакета. Однако, пакеты, как правило, не проходят через один и тот же коммутатор дважды, поэтому количество итераций ограничивается логарифмом от количества коммутаторов вдоль самого длинного маршрута в сети.

Порядок переменных в отношениях OBDD

Как это было показано в разделе 1.5.3, упорядоченные двоичные решающие диаграммы OBDD могут рассматриваться как бинарные деревья с фиксированным количеством *ярусов*, вершины каждого из которых размечены одной из переменных моделируемого отношения. При этом соответствие между ярусами и переменными является взаимно однозначным: каждой переменной сопоставлен ровно один ярус бинарного дерева, а каждому ярусу дерева – ровно одна переменная моделируемого этим деревом отношения.

Основное преимущество указанного представления заключается в том, что подобные деревья, зачастую, могут иметь значительное количество одинаковых поддеревьев, кото-

рые можно совместить друг с другом. Таким образом, появляется возможность провести *редукцию*, заменив множество изоморфных поддеревьев одним единственным их экземпляром и, тем самым, сократив исходные размеры двоичной диаграммы.

Для каждого конкретного отношения структура соответствующего ему дерева однозначно определяется порядком переменных, в соответствии с которым были размечены его ярусы. Удачный выбор порядка переменных способен увеличить количество редуцированных поддеревьев, уменьшить итоговый размер полученного дерева и, в конечном счёте, снизить вычислительную сложность использующих его алгоритмов, в то время как неудачный – привести к противоположным результатам. Таким образом, выбор правильного порядка переменных является одним из наиболее важных аспектов практического применения аппарата OBDD.

Хотя поиск наилучшего порядка переменных и является NP-полной задачей [55], а её решение актуально лишь для конкретного отношения и, вообще говоря, изменяется при каждой его модификации, это не помешало существенно сократить размеры OBDD для разработанных моделей ПКС за счёт упорядочивания переменных с учётом использования некоторых простых эвристик.

Было замечено, что большинство коммутационных устройств изменяет лишь малую часть полей заголовков пакетов. Так, в традиционных компьютерных сетях маршрутизаторы обычно изменяют лишь MAC адреса, а коммутаторы – и вовсе не вносят каких-либо модификаций в заголовки пакетов. Разработчики управляющих приложений контроллера так же стараются не модифицировать заголовки пакетов без лишней на то необходимости, тем самым, упрощая логику маршрутизации. Таким образом, можно, например, создать у пользователей ПКС иллюзию подключения к одному большому коммутатору, который передаёт пакеты от отправителя к получателю в неизменённом виде [66].

В то же время, всякая булева функция $f(X, Y)$, описывающая поведение тех компонентов сети, которые оставляют i-й бит кодировки состояния в неизменном виде, может быть представлена в виде $((X_i \wedge Y_i) \vee (\neg X_i \wedge \neg Y_i)) \wedge f(X, Y)$. Первая часть указанного выражения вызывает ветвление в графе двоичной диаграммы для функции f: он обязан иметь два одинаковых подграфа между вершинами на ярусах, размеченных переменными X_i и Y_i. Чем ближе друг к другу расположены эти ярусы, тем большая часть этих поддеревьев может быть редуцирована, и, наоборот, чем дальше они расположены, тем больше вершин будет включать в себя граф двоичной диаграммы. Таким образом, для уменьшения размера двоичных диаграмм, моделирующих компоненты ПКС, порядок переменных должен быть таким, чтобы те переменные, которые соответствуют одним и тем же эле-

ментам кодировки состояния пакета, но находятся в разных регистрах, были расположены последовательно друг за другом.

1.8.2 Анализ спецификаций

На вход анализатору подаётся текстовый файл, содержащий множество определений спецификации. Для разбора его содержимого в соответствии с грамматикой, приведённой в разделе 1.8.2, используется метод рекурсивного спуска [67]. Результатом успешного синтаксического разбора файла спецификации является абстрактное синтаксическое дерево (Abstract Syntax Tree, AST), вершины которого размечены конструкциями языка.

Полученное дерево преобразуется к последовательности деревьев, описывающих правила вычисления спецификаций. Листовыми вершинами таких деревьев могут быть константы, отношения, моделирующие отдельные компоненты ПКС, или же определения из файла спецификаций, которые уже были преобразованы и проанализированы ранее. В качестве внутренних вершин дерева выступают введённые языком $\mathcal{L}_1$ логические связки, кванторы всеобщности и существования, а так же оператор транзитивного замыкания.

За этапом построения множества деревьев вычисления для политик маршрутизации следует этап оптимизации. Деревья вычисления претерпевают изменения, позволяющие повысить эффективность их последующей обработки. На данном этапе, в частности, запускается алгоритм изменения порядка операндов коммутативных операторов (`and` и `or`), позволяющий снизить количество промежуточных диаграмм OBDD, которые необходимо запомнить в процессе вычисления итогового результата выражения. Указанный алгоритм работает в полной аналогии с алгоритмов разметки регистров Сети-Ульмана [67].

Полученные в результате деревья вычислений передаются на вход верификатору ПКС.

Грамматика языка спецификаций

Спецификация ⟨Spec⟩ политик маршрутизации состоит из определений ⟨Defintion⟩ [2]:

$$
\begin{aligned}
\langle \text{Spec} \rangle &\models \langle \text{Definition} \rangle\ \langle \text{Spec} \rangle \mid \bot \\
\langle \text{Definition} \rangle &\models \langle \text{Prefix} \rangle : \langle \text{Signature} \rangle := \langle \text{Expression} \rangle\ ; \\
\langle \text{Prefix} \rangle &\models \texttt{aux} \mid \texttt{main}
\end{aligned}
$$

[2] Здесь и далее описание грамматических конструкций языка приводится в форме Бэкуса-Наура.

Определение ⟨Definition⟩ начинается с указания его типа: вспомогательное ⟨aux⟩ или основное ⟨main⟩. В обязанности верификатора входит проверка соответствия конфигурации сети лишь основным определениям. Вспомогательные определения служат лишь повышения удобства задания спецификации. Верификатор может анализировать выполнимость вспомогательных определений и использовать полученные результаты в качестве промежуточных расчётов, необходимых для анализа основных определений. Однако, результаты про В процессе своей работы верификатор может изменить внутреннее представление спецификации, удалив существующие или создав новые вспомогательные определения.

⟨Signature⟩ ⊨ ⟨Name⟩ `()` | ⟨Name⟩ `(` ⟨Arglist⟩ `)`

⟨ArgList⟩ ⊨ ⟨Name⟩ | ⟨Name⟩ `,` ⟨ArgList⟩

Описание каждого определения ⟨Definition⟩ включает в себя сигнатуру ⟨Signature⟩ и тело ⟨Expression⟩. Сигнатура состоит из имени ⟨Name⟩ определения и набора ⟨ArgList⟩ именованных аргументов. При этом и имя определения, и имена аргументов представляют собой символьные идентификаторы, правила конструирования которых определяются стандартным для языков программирования способом. Идентификатор представляет собой последовательность из символов латинского алфавита, подчёркивания и цифр. При этом не допускается, чтобы первым символом указанной последовательности была цифра.

⟨Expression⟩ ⊨ ⟨Disjunct⟩ | ⟨Disjunct⟩ `or` ⟨Expression⟩

⟨Disjunct⟩ ⊨ ⟨Conjunct⟩ | ⟨Conjunct⟩ `and` ⟨Disjunct⟩

⟨Conjunct⟩ ⊨ `not` ⟨Conjunct⟩ | `(` ⟨Expression⟩ `)` | ⟨Atomic⟩
| `Closure {` ⟨Range⟩ `} [`⟨Name⟩ `,` ⟨Name⟩ `:` ⟨Expression⟩ `]`
| ⟨Quantor⟩ `[` ⟨Name⟩ `:` ⟨Expression⟩ `]`

⟨Atomic⟩ ⊨ ⟨Signature⟩ | ⟨Projection⟩ `==` ⟨Projection⟩

⟨Range⟩ ⊨ `+` | `*` | ⟨int⟩ `:` ⟨int⟩

⟨Quantor⟩ ⊨ `Exists` | `Forall`

Тело определения конструируется из переменных, соответствующих состояниям пакетов, с помощью дизъюнкции `or`, конъюнкции `and` и отрицания `not`, кванторов существования `Exists` и всеобщности `Forall` и оператора замыкания `Closure`. Для изменения приоритета логических операций допускается использование круглых скобок. Кроме того, в теле определения могут использоваться атомарные выражения: операторы равенства или подстановки ранее данных определений по заданной сигнатуре.

Внутри тела определения в качестве свободных переменных могут выступать лишь те из них, которые были объявлены заранее. Переменная внутри выражения считается объявленной лишь в том случае, если её имя было указано в качестве одного из аргументов сигнатуры определения, или же если она была указана в качестве привязанной переменной одного из объемлющих кванторов (`Exists` или `Forall`) или оператора замыкания `Closure`.

Определение считается объявленным лишь в том случае, если каждая из используемых в его теле сигнатур соответствует уже объявленному выше определению. Таким образом, язык спецификаций не допускает, например, рекурсивного использования определений.

Язык подразумевает наличие нескольких определений, которые считаются объявленными всегда. Их сигнатуры и семантика приводятся в таблице 1.1.

Таблица 1.1: Набор заданных заранее определений языка спецификаций.

Сигнатура	Описание семантики
True()	Тождественная истина
False()	Тождественная ложь
T(X,Y)	Отношение связи между портами коммутаторов
In(X)	Множество пакетов, которые могут поступить на вход сеть
Out(X)	Множество пакетов, которые могут покинуть сеть
R_step(X,Y)	Отношение одношаговой маршрутизации
R_tc(X,Y)	Транзитивное замыкание отношения R_step

$$
\begin{aligned}
\langle \text{Projection} \rangle &\models \langle \text{Slice} \rangle \mid \langle \text{Constant} \rangle \\
\langle \text{Slice} \rangle &\models \langle \text{Field\_Slice} \rangle \mid \langle \text{Field\_Slice} \rangle \texttt{ [} \langle \text{int} \rangle \texttt{ : } \langle \text{int} \rangle \texttt{]} \\
\langle \text{Field\_Slice} \rangle &\models \langle \text{Name} \rangle \mid \langle \text{Name} \rangle \texttt{ . } \langle \text{Field\_Name} \rangle \\
\langle \text{Field\_Name} \rangle &\models \texttt{w} \mid \texttt{d} \mid \texttt{p} \mid \texttt{h} \mid \texttt{t} \mid \texttt{r} \mid \texttt{a}
\end{aligned}
$$

Оператор равенства служит для указания поэлементного равенства двух заданных наборов двоичных переменных. В качестве операндов при этом могут выступать всевозможные сегменты срезы состояния пакета. Язык предусматривает набор предопределённых сегментов срезов состояния, таких как номер коммутатора, номер порта и тело заголовка. Кроме того, язык предусматривает возможность указания ещё более узкого диапазона переменных с помощью относительного битового сегмента среза, в котором могут ука-

зываться смещения байтов относительно исходного сегмента среза состояния. Например, выражение типа `X.h[1:2]` соответствует второму байту заголовка пакета.

$$\begin{aligned} \langle \text{Constant} \rangle &\models \texttt{"} \langle \text{HexPairList} \rangle \texttt{"} \\ \langle \text{HexPairList} \rangle &\models \langle \text{hdigit} \rangle \langle \text{hdigit} \rangle \texttt{ . } \langle \text{HexPairList} \rangle \end{aligned}$$

Помимо сегментов срезов, в сравнениях допускается участие целочисленных констант. Каждая константа представляет собой обрамлённую двойными кавычками последовательность из пар шестнадцатеричных цифр, разделённых между собой точками. Кроме того, разработанный язык спецификаций позволяет вставлять комментарии между любыми нетерминальными символами грамматики. Язык поддерживает как однострочные так и многострочные комментарии в стиле языка C++. Комментариями считаются любые последовательности символов, начинающиеся с `//`, и заканчивающихся символом перевода строки, а так любые последовательности, начинающаяся с `/*` и заканчивающихся комбинацией `*/`.

Примеры спецификаций

В данном разделе приводится листинг нескольких простых спецификаций состояния сети. Следующая спецификация, например, включает в себя два определения. Результатом вычисления первого из них является множество всех состояний пакетов, которые могут поступить в сеть и достигнуть точки выхода в цикл, то есть такого состояния, в котором пакет может оказаться повторно. Второе определение призвано проверить, существуют ли в сети циклы по состояниям.

```
// find all states leading to a packet loop
aux: lead_to_state_cycle(Initial) := In(Initial) &
    Exist[y: R_tc(Initial,y) & Exist[z: R_tc(y,z) & y == z.a]];
main: no_state_cycles() := Forall[x: not lead_to_state_cycle(x)];
```

Следующее определение соответствует множеству состояний для тех поступивших в сеть пакетов, которые способны достигнуть коммутатора под номеров 7, не пересекая коммутатор под номером 16, за 8, 9 или 10 шагов коммутации (хопов). Необходимо отметить, что определение `unsecure_flow` – вспомогательное, и не будет вычисляться, если его сигнатура не будет явным образом использоваться в других определениях.

```
/* find all packets able to reach switch #7 in 8 to 10 hops
        without crossing switch #16 */
aux: unsecure_flow(X) := In(X) & Exist[y:
    Closure{8:10}[X,Y: R_step(X,y) & not y.w == "f0"] & y.w == "07"
];
```

Набор приведённых далее определений призван проверить, что заданный канал передачи данных используется не более чем одним потоком в предположении о том. При этом делается допущение, что поток можно однозначно идентифицировать по первым семи байтам заголовка пакета.

```
// find all packets that goes through the link
//      connected to port #63 of switch #8
aux: use_port(P) := P.w == "08" & P.p == "3f";
aux: use_link(P) := use_port(P) | Exist[Q: T(Q,P) & use_port(Q)];
aux: reach_channel(state) := In(state) &
    Exist[y: R_tc(state,y) & use_link(y)];

// tracked channel is not used
aux: idle_channel() := Forall[x: not reach_channel(x)];

/* link can be used by at most one flow */
main: link_reserved() := idle_channel() |
    Exist[x: reach_channel(x) &
        Forall[y: not reach_channel(y) | x.h[1:7] == y.h[1:7]]
    ];
```

Следующее определение агрегирует множество введённых выше определений. Его истина соответствует выполнению всех описанных спецификаций, в то время как его ложное значение означает нарушение хотя бы одной из них.

```
// return true if no misconfiguration found
main: nothing_to_worry() := no_state_cycles()
    & not Exist[x: unsecure_flow(x)]
    & link_reserved();
```

1.8.3 Верификация конфигурации ПКС

Основная задача верификатора – определить соответствует ли состояние сети заданным спецификациям. Как это было отмечено ранее, предложенные в разделах 1.6 и 1.7 статическая модель ПКС и язык спецификации политик маршрутизации $\mathcal{L}_1$ хорошо сочетаются между собой: отношения статической ПКС могут служить интерпретациями формул языка $\mathcal{L}_1$. Поэтому указанная задача может быть сведена к задаче проверки выполнимости некоторых формул языка $\mathcal{L}_1$ на множестве отношений, моделирующих ПКС.

Именно указанный принцип и был заложен в основу разработанного средства проверки спецификаций: оно подставляет значения отношений, полученных на этапе анализа конфигурации сети, в деревья вычислений, полученные от анализатора спецификаций, и производит поочерёдное их вычисление:

- Если хотя бы одна формула выдаёт ложное значение, конфигурация сети нарушает требования политик маршрутизации и считается некорректной;
- Иначе, если все формулы выполняются – конфигурация корректна.

Результаты экспериментального исследования построенного комплекса программных средств для верификации ПКС рассматриваются в следующем разделе.

1.9 Экспериментальное исследование

В данном разделе приводится описание двух экспериментов, демонстрирующих функциональные возможности и характеристики производительности разработанного программно-инструментального средства верификации ПКС:

- Первый из них проверял работу верификатора на нескольких синтезированных конфигурациях ПКС. Правила генерации искусственных топологий и записей для таблиц коммутаторов описаны в разделе 1.9.1, результаты – в разделе 1.9.2;
- Второй эксперимент проводился с использованием выложенной в открытый доступ конфигурации магистральной сети кампуса Стенфордского университета. Результаты данного эксперимента приводятся в разделе 1.9.3.

1.9.1 Генерация конфигураций ПКС

Было использовано три типа синтетических топологий: PLOD, TREE и RING.

Первая из них – это топология со статистическим распределением рёбер графа между вершинами по «закону сильных» (Power Law Out Degree), при выполнении которого большинство рёбер инцидентно меньшинству вершин [68]. Графы, построенные по указанному принципу похожи на топологии реальных компьютерных сетей, в которых соблюдается иерархичность, и множество коммутационных устройств более низкого уровня связаны между собой лишь через устройства более высокого уровня иерархии.

Генерация PLOD топологии из n вершин производится по следующему принципу. Каждой вершине графа назначается собственный вес по формуле $\beta x^{-\alpha}$, где

- x – случайно выбранное натуральное число в диапазоне $[1; n]$;
- β – регулирует среднее количество инцидентных рёбер для каждой вершины;
- α – коэффициент равномерности распределения рёбер между вершинами: чем больше α, тем более неравномерно распределение.

Топологии типа TREE соответствуют традиционным многоярусным топологиям центров обработки данных [69]. Коммутаторы в таких топологиях связываются друг с другом по следующим правилам:

- каждый из коммутаторов яруса i соединяется с k коммутаторами яруса $i + 1$;
- каждый из коммутаторов яруса $i + 1$ связывается с l коммутаторами яруса i;
- количество коммутаторов на каждому последующем ярусе уменьшается: $k < l$.

Коэффициент дублирования k может рассматриваться как уровень устойчивости к отказам оборудования – топология будет сохранять свою связность при выходе из строя любых $k - 1$ коммутаторов. *Коэффициент ветвления* l управляет количеством альтернативных маршрутов – чем меньше l при фиксированном значении k, тем больше глубина дерева и более разветвлённая сеть маршрутов связывает любые хосты сети.

Топологии типа RING представляют собой кольцевые графы с заданным количеством вершин. Сети коммутаторов с такой топологией потенциально интересны необычной возможностью для маршрутизации. Для корректной работы подобной сети достаточно вставить в таблицы коммутаторов лишь те из них, которые предназначены для передачи трафика на подключённые к ним хосты. Передачу пакетов между коммутаторами можно организовать за счёт использования нестандартных правил по умолчанию: если правила по умолчанию будут передавать пакеты для удалённых абонентов на следующий коммутатор «кольца», то рано или поздно эти пакеты достигнут своих адресатов. Таким образом,

количество необходимых правил коммутации становится равным количеству обслуживаемых ей абонентских машин.

Наполнение таблиц коммутаторов правилами проводится единственным способом и проходит в два этапа: разметка топологии и генерация правил передачи потоков.

На первом этапе решается задача обеспечения маршрутизации пакетов по кратчайшему пути. Для этого рёбра графа размечаются случайно выбранными весовыми коэффициентами. Содержательно они соответствуют длинам линий передачи данных между коммутаторами. Построенная разметка используется для вычисления матрицы расстояний, собранной из весов наикратчайших маршрутов между каждой парой вершин графа.

На втором этапе производится установка правил коммутации отдельных потоков. Правила каждого потока вырабатываются согласно следующей процедуре:

1. Случайным образом выбираются две вершины графа и битовая строка фиксированной длины. Они играют роль отправителя, получателя и шаблона передающегося через сеть потока;

2. На основании построенной ранее матрицы расстояний производится вычисление кратчайшего маршрута передачи пакетов от отправителя к получателю;

3. В коммутаторы указанного маршрута заносятся правила для передачи пакетов, заголовки которых соответствуют выбранному шаблону, на следующий его элемент. Сами заголовки при этом не изменяются.

1.9.2 Эксперименты на синтетических конфигурациях

Разработанные генераторы синтетических конфигураций были использована для экспериментального исследования характеристик производительности построенных средств верификации и оценки практической применимости предложенного математического аппарата для решения задачи верификации ПКС.

В рамках исследования было проведено множество запусков верификатора. На протяжении всего эксперимента размерность состояния была фиксированной и составляла 96 битов. Из них 80 битов соответствовали заголовку пакета, 8 – номеру порта и 16 – имени коммутатора. Таким образом, максимальное число булевых переменных, участвующих в построении отношения OBDD, было равным 96×3 – произведению битовой длины состояния и количества используемых регистров.

Тип топологии, количество коммутаторов и число установленных маршрутов, напротив, варьировались. Количество проходящих через сеть потоков изменялось между 100, 1000 и 10000 штук. Число коммутаторов в различных топологиях составляло порядка 10, 100 и 300 штук: указанные значения соблюдались в точности при генерации топологии типа RING, и приближённо – при генерации топологий типа PLOD и TREE, структура которых не всегда позволяет построить сеть с заданным количеством вершин.

Переданный анализатору спецификаций файл описывал две политики маршрутизации: `s_loop` и `w_loop` – которые проверяли наличие циклов маршрутизации в пространстве состояний пакетов $\mathcal{S} = \mathcal{H} \times \mathcal{P} \times \mathcal{W}$ и в пространстве имён коммутаторов $\mathcal{W}$ соответственно:

```
// недопускать циклов в пространстве состояний пакетов
main: no_s_loop() := not Exist[X: In(X) and Exist[Y: R_tc(X,Y) and
        Exist[Z: R_tc(Y,Z) and Y == Z]
    ]];

// недопускать циклов в просранстве коммутаторов сети
main: no_w_loop() := not Exist[X: In(X) and Exist[Y: R_tc(X,Y) and
        Exist[Z: R_tc(Y,Z) and Y.w == Z.w and not Out(Z)]
    ]];
```

Для экспериментов использовалась машина с процессором Intel Xeon 2.4 Ghz и 12 Гб оперативной памяти. Поскольку ни в одном из экспериментов количество затраченной памяти не превышало 1,5 Гб, то её потребление было сочтено незначительным, а в качестве основной характеристики производительности используется время работы программы.

Как это видно из результатов экспериментов, представленных в таблице 1.2, время построения модели ПКС по заданной её конфигурации варьируется от долей секунд до нескольких минут и даже часов в зависимости от размеров топологии и количества проложенных через сеть маршрутов передачи данных. Наибольшую часть затраченного времени занимает на расчёт полного отношения маршрутизации R^*_{step}. Как это было указано в разделе 1.8 отношение R^*_{step} вычисляется методом итеративной композиции отношения одношаговой маршрутизации R_{step}. Каждая такая композиции порождает большую диаграмму OBDD, которая задействует переменные сразу из трёх регистров, и потребляет значительное количество вычислительных ресурсов при её дальнейшем использовании. Таким образом, неудивительно, что наибольшее время работы, более 9 часов, конструктор моделей продемонстрировал на топологии типа RING из 300 вершин и 10 000 маршрутов

Таблица 1.2: Показатели производительности верификатора на синтетической нагрузке.

Характеристики конфигурации			Время построения модели, [часы:][минуты]секунды			Свойства дерева R^*_{step}		Проверка, секунды	
Тип топологии	Количество потоков, шт	Количество вершин, шт	Отношение одношаговой маршрутизации	Итоговое отношение маршрутизации	Суммарное время работы конструктора	Число вершин дерева, тыс. шт	Порядок числа путей дерева	Циклы по состояниям	Циклы по топологии
PLOD	100	12	0.042	0.019	0.061	11	17	0.001	0.001
		101	0.077	0.128	0.206	22	17	0.002	0.004
		305	0.111	0.115	0.226	26	17	0.002	0.005
	1000	12	0.431	0.192	0.624	56	18	0.008	0.014
		101	0.715	1.816	2.532	139	18	0.036	0.052
		305	0.739	2.155	2.895	181	18	0.048	0.065
	10000	12	6.095	3.599	9.695	363	19	0.183	0.221
		101	10.914	24.357	35.271	835	19	0.521	0.741
		305	10.851	38.357	49.208	1164	19	0.532	0.750
TREE	100	14	0.044	0.019	0.064	12	17	0.001	0.001
		126	0.116	0.355	0.471	26	17	0.002	0.006
		254	0.138	0.497	0.636	30	17	0.003	0.007
	1000	14	0.431	0.166	0.897	59	18	0.008	0.014
		126	1.058	3.777	4.836	162	18	0.077	0.092
		254	1.285	7.401	8.688	199	18	0.081	0.103
	10000	14	6.294	3.163	9.458	373	19	0.213	0.292
		126	16.202	52.740	1 08.942	1005	19	0.679	0.885
		254	20.161	1 41.856	2 02.018	1304	19	0.986	1.242
RING	100	10	0.046	0.029	0.075	11	17	0.001	0.002
		100	0.315	6.694	7.010	22	18	0.003	0.008
		300	0.903	3 15.750	3 16.653	25	18	0.008	0.015
	1000	10	0.515	0.286	0.802	56	18	0.012	0.017
		100	3.550	1 18.387	1 21.973	151	19	0.129	0.166
		300	10.477	35 05.513	35 15.990	175	19	0.301	0.395
	10000	10	6.933	4.978	11.911	379	19	0.185	0.253
		100	56.675	20 35.105	21 31.781	1051	20	1.868	2.250
		300	2 52.455	9:06 39.428	9:09 31.885	1410	20	5.329	6.333

– здесь вычисление полного отношения маршрутизации потребовало девять композиций всё увеличивающегося в размерах отношения.

Необходимо отметить, что в реальные ПКС, как правило, включают меньшее количество коммутаторов, а их топология имеет куда более разветвлённую структуру. Таким образом, вычисление полного отношения маршрутизации будет требовать меньше операций композиции отношения одношаговой маршрутизации, да и указанное отношение будет иметь меньшие размеры. Например, вычисление отношения R^*_{step} на топологии типа PLOD из 305 вершин и 10 000 маршрутов потребовало лишь пять композиций и заняло порядка 38 секунд. Что интересно, сложность построенных решающих диаграмм R^*_{step} для топологий типа PLOD, TREE и RING отличаются между собой не так уж значительно: их размеры составили 1164, 1304 и 1410 тысяч узлов соответственно.

Проверка выполнения формул спецификации политик маршрутизации для построенных отношений оказалась гораздо менее затратной операцией, чем построение этих отношений. Результаты показали, что поиск достижимых циклов по состоянию пакета и по топологии сети требует несколько секунд даже на сложных конфигурациях ПКС. Таким образом, если политика маршрутизации не предполагает построения экзотических отношений, требующих транзитивного замыкания, то все использованные при её спецификации отношения потребуют не более двух регистров, и значение соответствующей формулы языка $\mathcal{L}_1$ будет вычислено относительно быстро.

1.9.3 Эксперименты на реальных данных

Разработанные средства были апробированы не только на синтетических данных, но так же и на конфигурации магистральной сети кампуса Стенфордского университета, выложенного в открытый доступ авторами статьи [44], которая представляет средство верификации сетей NetPlumber.

Топология указанной сети относится к рассмотренному ранее типу TRIE с некоторыми дополнительными рёбрами и состоит из 16 мощных маршрутизаторов, расположенных в разных зданиях кампуса. Поскольку сеть не относится к классу программно-конфигурируемых, и входящие в её состав маршрутизаторы не поддерживают протокол OpenFlow, то их конфигурации заданы в виде набора текстовых файлов, которые содержат распечатки содержимого внутренних таблиц и настроек для работающих на них протоколов и служб, в том виде, как они выводятся соответствующими командами диагностики оборудования. Общий объём открытых данных о конфигурации сети составляет порядка 90Mb.

Выложенные файлы конфигураций, в частности, перечисляют записи таблиц коммутации и маршрутизации, а так же списки интерфейсов с соответствующими метками (VLAN) и списками контроля доступа (ACL). Авторам удалось использовать эту информацию для построения эквивалентных наборов правил преобразования пакетов в стиле OpenFlow – соответствующий парсер конфигураций был так же выложен в открытый доступ.

Указанные наработки были использованы для построения конфигурации сети в нужном для разработанного конструктора моделей формате – при этом было воссоздано порядка 757 тысяч правил коммутации пакетов. Построенные правила учитывают девять полей заголовков: идентификатор виртуальной сети (VLAN), адреса отправителя и получателя на канальном (Ethernet), сетевом (IP) и транспортном (TCP и UDP) уровнях стека TCP-IP, а так же идентификаторы протоколов, инкапсулированных фреймами канального и сетевого уровней. Итоговая длина заголовка составила 120 битов, длина записи состояния пакета – 160 битов. Таким образом, число суммарное количество задействованных верификатором переменных с учётом использования трёх регистров составляло $160 \times 3 = 480$ штук.

Помимо отсутствия циклов по пространству состояний пакетов и пространству коммутаторов ПКС, множество политик маршрутизации включало в себя проверки на наличие путей длиннее одного, двух и трёх хопов. Спецификации указанных свойств совпадают с точностью до проверяемой длины пути `L`:

```
// недопускать маршруты, длина которых превышает L хопов
main: no_path_has_L_hops() := Exist[X: In(X) & Exist[Y: Out(Y) &
        R_tc(X,Y) & not Closure{1:L}[X,Y: R_step(X,Y)]
    ]];
```

Таблица 1.3: Производительность верификатора на сети Стенфордского университета.

Время построения модели, секунды			Свойства дерева R^*_{step}		Время верификации ПКС, секунды				
Отношение одношаговой маршрутизации	Полное отношение маршрутизации	Суммарное время работы конструктора	Число вершин дерева, тыс шт	Порядок числа путей дерева	Циклы по состояниям	Циклы по топологии	Пути длиннее одного хопа	Пути длиннее двух хопов	Пути длиннее трёх хопов
1.094	6.294	18.028	642	18	0.043	0.047	0.855	2.013	3.764

Результаты эксперимента с магистральной сетью Стенфордского университета, приведённые в таблице 1.3, продемонстрировали способность предложенных средств верификации сконструировать модель и провести проверку реальной ПКС за разумное время. Несмотря на большой объём записей в таблицах маршрутизаторов сети, разработанному конструктору удалось построить модель сети за 18 секунд, а верификатору – проверить соответствие сети заданным свойствам быстрее, чем за 7 секунд.

Важно отметить, что несмотря на значительную длину заголовков и, как следствие, существенно большую размерность состояния пакетов $\mathcal{S}$, нежели чем в экспериментах на синтетических данных (480 против 288), и конструктор моделей, и верификатор показали на них лучшие результаты. Этот факт позволяет заключить, что функционирующие в сети протоколы передачи данных и те правила установки заголовков, которые они используют, хорошо сочетаются с предложенным способом представления сети, и предложенный математический аппарат сможет успешно работать на практике.

Ещё одним важным с практической точки зрения результатом проведённого эксперимента можно считать то, что разработанному средству верификации, удалось подтвердить выводы полученные авторами оригинального исследования конфигурации сети Стенфорда, и обнаружить в ней несколько классов ошибок:

1. Сеть допускала зацикливание некоторых пакетов. Несмотря на то, что этого не удалось выявить при повседневном использовании, указанный недостаток мог, рано или поздно, привести к таким неприятным последствиям как нарушение качества установленных соединений, перегрузка оборудования или даже полное прекращение работы сети;

2. Установленные правила маршрутизации приводили к нерациональному использованию ресурсов. Поскольку топология сети принадлежит к классу двухъярусных деревьев, то длина кратчайшего маршрута передачи данных между любой парой абонентов сети не должна превышать трёх хопов. В то же время, в силу тех или иных ошибок, конфигурация содержала и более длинные маршруты – и это без учёта обнаруженных до этого циклов маршрутизации.

Таким образом, была продемонстрирована практическая польза от использования разработанных средств верификации: они смогли выявить значимые ошибки конфигурации, которые приводили к угрозе безопасности и потере эффективности работы сети, но не были обнаружены другими методами.

1.10 Выводы

В данной главе был представлен оригинальный язык спецификации политик маршрутизации, а так же математическая модель ПКС, в рамках которой была определена семантика его выражений. Язык имеет иерархическую структуру из трёх уровней: фрагмент каждого следующего уровня имеет большую выразительную мощность, и может быть использован для описания более сложных политик маршрутизации. Наиболее перспективным фрагментом разработанного языка является язык $\mathcal{L}_1$, основанный на логике первого порядка с оператором транзитивного замыкания $FO[TC]$, и обладающий достаточной выразительной мощностью для описания требований к глобальным свойствам статической конфигурации ПКС. Несмотря на то, что в работе была доказана принадлежность задачи проверки соответствия ПКС формулам языка $\mathcal{L}_1$ к классу PSPACE-трудных, указанная задача допускает своё эффективное практическое решение. Немаловажную роль здесь играет тот факт, что представленная в разделе 1.6 реляционная модель ПКС, на которой предлагается интерпретировать формулы $\mathcal{L}_1$, допускает компактное символьное представление конфигурации ПКС в виде набора двоичных решающих диаграмм OBDD.

На основе разработанного формального метода проверки ПКС относительно спецификаций языка $\mathcal{L}_1$ было построено программно-инструментальное средство выявления ошибок в конфигурации оборудования. Результаты представленного в разделе 1.9 экспериментального исследования показали, что разработанный математический аппарат обладает достаточно низкими требованиями к вычислительным ресурсам, чтобы эффективно решать указанную задачу на практике.

Необходимо подчеркнуть, что задача верификации конфигурации ПКС не нова, и на сегодняшний день для решения этой задачи уже было предложено несколько программно-инструментальных средств: FlowChecker [6], Anteater [45], Veriflow [61], NetPlumber [44], AP-Verifier [47]. Однако, указанные средства или не предлагают явно выделенного языка для спецификации политик маршрутизации и предназначены для проверки лишь предопределённого набора свойств (Anteater, Veriflow, AP-Verifier), или предлагают языки спецификаций, основанные на темпоральных логиках, которые менее удобны для описания свойств отношения маршрутизации, обладают меньшей выразительной мощностью, и одновременно предъявляют более высокие требования к вычислительным ресурсам (FlowChecker, NetPlumber).

Разработанное средство верификации обладает большей гибкостью: встроенный в него язык спецификации позволяет адаптировать верификатор под специфику конкретной се-

ти, и, тем самым, упростить применение формальной верификации к практическим задачам сетевого администрирования.

Глава 2

Оценка задержки передачи пакетов

В данной главе рассматривается задача проверки соответствия времени передачи данных через сеть с заданной конфигурацией тем ограничениям на максимальную задержку, которые предъявляются к соответствующим соединениям.

Время передачи отдельных пакетов через инфраструктуру сети существенно зависит от правил распределения её ресурсов, таких как пропускная способность физических линий связи и объёмы буферной памяти коммутаторов, между проходящими через сеть потоками данных. Поскольку правила распределения сетевых ресурсов влияют на характеристики отдельных соединений, то методы управления такими правилами традиционно называются *методами управления качеством сервиса*, а соответствующие им принципы работы коммутационного оборудования – *моделями сетевых сервисов*. Краткий обзор наиболее распространённых методов управления и моделей сетевых сервисов приводится в разделе 2.1. В нём, в частности, содержится краткое описание основных моделей сервиса современных компьютерных сетей: модели интегрированных сервисов (IntServ) и модели дифференцированных сервисов (DiffServ).

В разделе 2.2 приводится обзор существующих подходов к построению коммутаторов: перечисляются составляющие их структурные блоки, а так же способы их компоновки между собой. Особое внимание при этом уделяется зависимости времени обработки пакетов от внутреннего структуры.

В качестве математического аппарата для решения задачи оценки времени передачи данных через инфраструктуру сети выбрана теория сетевого исчисления (network calculus). Изложение основных результатов указанной теории разбито на два раздела. В разделе 2.3 содержатся базовые понятия, достаточные для применения сетевого исчисления в сетях, реализующих модель интегрированных сервисов. Вопросы разделения

ресурсов между потоками, рассмотрение которых необходимо при рассмотрении модели дифференцированных сервисов, затрагиваются в разделе 2.4. В этом же разделе приводится описание нескольких алгоритмов построения верхних оценок для задержки передачи данных через сеть.

В разделе 2.5 изложены алгоритмы для вычисления свёртки и обратной свёртки в алгебре min-plus. Указанные алгоритмы необходимы для того, чтобы обобщить рассмотренные ранее алгоритмы вычисления задержки на более широкий класс сетей.

Раздел 2.6 содержит описание альтернативного алгоритма построения задержек передачи данных. В отличие от изложенных ранее алгоритмов, активно применяющий операции над функциями в алгебре min-plus, данный алгоритм построен на идее использования метода линейного программирования.

Раздел 2.7 содержит краткое описание разработанных программных средств построения верхней оценки для времени передачи данных и результаты экспериментальной апробации данных средств.

2.1 Качество сервиса и методы его обеспечения

Как показывают история и практика использования компьютерных сетей, качество сетевого сервиса, которым хотят обладать пользователи, всегда превышает возможности существующей сетевой инфраструктуры. Более того, разрыв между совокупностью требований пользовательских приложений и доступной производительностью сети продолжает увеличиваться [70]. Как следствие, сетевые приложения вынуждены работать в условиях постоянной конкуренции за выделенные на их обработку ресурсы сети: пропускную способность каналов передачи данных, процессорное время и объём буферов на сетевых устройствах. Таким образом, проблема поиска справедливого и в то же время эффективного распределения доступных ресурсов между приложениями является одной из самых фундаментальных проблем компьютерных сетей.

За годы исследования обозначенной проблемы разработано множество разноплановых подходов к её решению как со стороны приложений, так и со стороны сетевой инфраструктуры. Примерами подобных подходов являются адаптация и приоритезация пользовательского трафика, динамическая маршрутизация, резервирование ресурсов. Причём каждый из этих подходов позволяет охватить лишь часть исходной задачи:

- Резервирование ресурсов – единственный механизм, предоставляющий приложениям хоть какие-то гарантии относительно качества сервиса. Например, зарезервированная для телеконференции пропускная способность не будет занята приложением резервного копированием данных.

- С другой стороны, только независимая маршрутизация потоков позволяет учесть требования приложения при построении пути передачи его данных через сеть. С её помощью трафик сетевой игры, для которой первостепенна скорость отклика, будет передаваться по пути с наименьшим временем передачи данных, пусть и с меньшей пропускной способностью.

Таким образом, ни один из существующих механизмов управления качеством сервиса не даёт комплексного решения проблемы. Композиция же сразу нескольких средств часто неэффективна и не всегда возможна без их модификации. Например, совмещение резервирования ресурсов и маршрутизации потоков требует, чтобы последняя считала зарезервированные ресурсы занятыми.

Кроме того, многие из существующих методов предъявляют дополнительные требования к функциональным возможностям и характеристикам производительности сетевых устройств, которые и без того должны обрабатывать миллионы пакетов ежесекундно. Например, резервирование ресурсов и маршрутизация потоков предполагают, что каждое сетевое устройство хранит информацию о всех обрабатываемых им потоках и сопоставляют с ней каждый входящий пакет [71]. Добавление такой логики к современному коммутационному оборудованию требует его существенного усложнения и соответствующего увеличения его стоимости. Поэтому поставщики сетевых услуг зачастую вынуждены отказываться от продвинутого управления качеством сервиса в пользу более низкой стоимости сетевой инфраструктуры.

Настоящая работа предлагает использовать для решения проблемы недавно появившуюся концепцию Программно-Конфигурируемых Сетей (ПКС), в основу которой заложены принципы централизованного управления сетевыми ресурсами и независимой коммутации потоков трафика [9]. Иными словами, в сетях ПКС автоматически выполняются те самые «неудобные» требования, которые препятствовали масштабному внедрению существующих методов управления качеством сетевого сервиса. Успешное развитие данной технологии делают актуальными задачу адаптации известных подходов к управления качеством сервиса к работе в новой для них среде, а так же задачу разработки нового комплексного механизма управления, охватывающего сразу несколько аспектов данной проблемы.

Методы оценки различных параметров качества сервиса и, в том числе, задержки передачи пакетов могут существенно различаться в зависимости от устройства сети. В данном разделе приводится краткий обзор наиболее важных методов обеспечения качества сервиса, а так же требований, которые эти методы предъявляют к устройству и принципам функционирования сетевой инфраструктуры.

2.1.1 Требования качества сервиса

В системотехнике и программной инженерии требования к свойствам разрабатываемой системы принято разделять на функциональные и нефункциональные [72]. Первые рассматривают систему как абстрактный автомат, описывая зависимость между исходными данными, поданными ему на вход, и результатами, полученными от него на выходе. Они определяют, что должна делать система, какие функции она должна выполнять [73]. Для некоторых требований, таких как удобство использования, детальность документации и конечная стоимость системы, автоматная модель неудобна. Их принято называть нефункциональными. Часть нефункциональных требований, описывающая как именно система должна работать с точки взаимодействующих с ней агентов, обычно называют качественными требованиями или требованиями качества сервиса (Quality of Service, QoS). Данный класс требований описывает такие свойства системы как её отзывчивость, надёжность, живучесть, безопасность и энергетическая эффективность [74].

Приведённая классификация требований применяется и в области компьютерных сетей. В качестве описываемой системы здесь выступает сетевая инфраструктура, а в качестве агентов – её пользователи, администраторы и владельцы. Их требования качества сервиса, в свою очередь, принято разделять ни пользовательские, административные и инфраструктурные [75].

Первая категория соответствует требованиям приложений к качеству предоставляемых им end-to-end соединений, например, их пропускной способности или времени передачи пакета [76]. При этом каждое приложение может предъявлять свой собственный набор требований качества сервиса независимо от других приложений. Для видеотрансляций наиболее важным требованием является пропускная способность: отставание допустимо, но дрожание картинки неприемлемо. Для биржевых роботов первостепенную роль играет время передачи данных. Обычно такие приложения оперируют небольшими объёмами сетевого трафика, но должны быстро реагировать на изменения котировок. Для интерактивных телеконференций существенны оба перечисленных параметра.

Административные требования определяют правила использования сетевой инфраструктуры для пользовательских приложений [75]. Обычно это требования безопасности, выраженные в виде набора запрещённых и, наоборот, обязательных к выполнению сценариев поведения. Например, корпоративная политика безопасности может запрещать доступ к почтовым серверам из внешней сети и предписывать передавать трафик между офисами компании только через зашифрованные туннели. К административным требованиям также можно отнести изоляцию трафика различных пользователей центра обработки данных и эксклюзивность их доступа к той части инфраструктуры, которую они арендовали.

Инфраструктурные требования качества затрагивают вопросы эффективности организации передачи данных с точки зрения сети в целом. Критериями эффективности могут служить сразу несколько различных, зачастую противоречащих друг другу, оценочных суждений. Естественным, например, представляется желание уменьшить энергопотребления сети. В случае небольшой загруженности сети, часть её элементов сети может быть обесточена, а проходящие через эти элементы маршруты – направлены обходными путями [77].

Другим примером инфраструктурного требования качества является устойчивость к скачкам нагрузки. Если в процессе построения маршрутов усреднить утилизацию сетевых устройств и каналов передачи данных, то вероятность перегрузки отдельного сегмента сети из-за резкого всплеска активности меньше, чем при обычной маршрутизации. К сходной стратегии выбора маршрутов может привести требование повышения отказоустойчивости сети. Чем меньше потоков передаётся через один элемент сети, тем меньше приложений узнают о его отказе [78].

Как показывают приведённые примеры, инфраструктурные требования могут существенно влиять на выбор маршрутов передачи данных. Однако в отличие от других категорий, требования данного вида редко используются в качестве ограничения для множества допустимых маршрутов. Например, если для обработки трафика приложения с заданным качеством сервисом требуется задействовать обесточенный канал передачи данных, то он будет включен, несмотря на общее увеличение энергопотребления. Вместо этого инфраструктурные требования, как правило, служат критериями оптимизации, на основании которых выбирается наиболее подходящий маршрут передачи данных. На практике такая задача обычно сводится к минимизации функционала, зависящего от абсолютного значения и распределения нагрузки по отдельным элементам сети.

2.1.2 Метрики качества сервиса

Сервис обмена данными, которым сеть обеспечивает функционирующие в ней приложения, можно оценивать в терминах различного рода метрик качества. Зачастую метрики соответствуют характеристикам производительности отдельных элементов сети. Например, время передачи пакета данных через предоставленное соединение может быть вычислено как сумма времён передачи пакета через каждый элемент соответствующего этому соединению маршрута. Таким образом, расчёт метрики для заданного соединения сводится к вычислению функции композиции для характеристик от нескольких элементов сети.

Для каждой метрики, вообще говоря, может применяться собственная функция композиции. Например, надёжность предоставленного соединения может быть вычислена как произведение вероятностей возникновения ошибки на каждом из элементом используемого им маршрута передачи данных, в то время как его задержка рассчитывается в виде суммы задержек отдельных элементов этого маршрута. Однако на практике обычно применяются лишь три типа метрик: минимаксные, аддитивные и мультипликативные. Для первых композиция определяется как минимум (или максимум) среди значений операндов. Аддитивные метрики используют в качестве композиции сложение, мультипликативные – умножение [79].

Помимо метрик, соответствующих характеристикам элементов сети, иногда используются и более сложные производные метрики, которые вычисляются на основе базовых. В частности, прослеживается тенденция к заданию пользовательских требований в терминах качества восприятия (Quality of Experience), что позволяет конечным потребителям охарактеризовать сервис в наиболее естественных для них метриках. Например, пользователю услуги IPTV может быть проще посетовать на длительность переключения каналов телевидения, чем заметить, что предоставленное ему соединение имеет слишком высокую задержку.

2.1.3 Связь между обеспечением качества и управлением ресурсами

Один из базовых принципов организации компьютерных сетей – использование единой инфраструктуры для одновременного обслуживания сразу нескольких независимых приложений. При этом если их трафик проходит через одни и те же элементы сети, то, в силу их конечной производительности приложения вынуждены конкурировать за право ис-

пользовать ресурсы этих элементов. Традиционно под критическими сетевыми ресурсами подразумевают пропускную способность каналов передачи данных, процессорное время и объём буферов коммутационных устройств. Однако список может расширяться. Например, в случае программно-конфигурируемых сетей, построенных на основе протокола OpenFlow [29], к списку критических ресурсов естественным образом добавляется также процессорное время контроллера, и размеры таблиц правил OpenFlow коммутаторов.

Качество предоставляемого конкретному приложению сервиса определяется количеством выделенных на его обслуживание ресурсов сети: чем больше ресурсов – тем выше качество. Задача построения end-to-end соединения с заданными метриками качества фактически эквивалентна задаче надлежащего распределения ресурсов сети между работающими в ней приложениями. При этом перераспределять ресурсы можно не только внутри отдельных элементов сети, но между разными элементами. Например, перенаправление потока данных эквивалентно освобождению ресурсов прежнего маршрута, и захвату ресурсов нового маршрута.

Замена исходной задачи поиска соединения с заданными характеристиками качества сервиса эквивалентной задачей распределения ресурсов порождает фундаментальную проблему соответствующего преобразования требований. Например, непонятно, каким образом можно преобразовать время передачи пакета данных в процессорное время, которое должен уделить трафику приложения конкретный коммутатор. При этом проблема только усугубляется, если использовать более удобные для пользователя метрики качества восприятия.

Универсальные механизмы обеспечения заданного качества сервиса, могут выполнять требования приложений лишь за счёт управления инфраструктурой сети. Поэтому они так или иначе должны решать задачу распределения ресурсов, и, следовательно, предоставлять способы преобразования требований приложений в инструкции по перераспределению ресурсов. На практике обозначенная проблема обычно решается с помощью последовательного пересчёта метрик качества сервиса для заданного соединения и последующей корректировки распределения ресурсов на основе полученных замеров.

2.1.4 Методы обеспечения качества

Адаптация приложения

Решение проблемы обеспечения заданного уровня качества сервиса в компьютерной сети может быть достигнуто различными способами и на разных этапах взаимодействия

между сетью и функционирующими в ней приложениями. Наименее требовательным к сетевой инфраструктуре способом реализации соответствующего средства является адаптация приложения. Данный метод обеспечения качества рассматривает возможности приложения для приспособления к тому уровню качества, который реально предоставляет ему сеть.

Адаптация позволяет подогнать требования качества приложения под требования качества восприятия, которыми руководствуется пользователь этого приложения. Например, единственным реальным требованием пользователя может быть возможность прослушивать радиопередачу в реальном времени. Тогда на ухудшение предоставленного сервиса целесообразно реагировать, соответствующим снижением качества звукопередачи, а не разрывом соединения и нарушением исходного пользовательского требования.

Достоинством описанного подхода является его полная инфраструктурная независимость. Если приложение умеет адаптироваться к предоставленному качеству, то оно может сделать это в любой сети, вне зависимости от принципов построения и характеристик её производительности. Все изменения происходят непосредственно внутри приложения. Однако данный метод не обладает свойством универсальности. Адаптации приложения неизбежно требует вмешательства в его логику и нетривиального изменения кода, которое невозможно проделать в автоматическом режиме. Кроме того, возможности для адаптации сильно ограничены и существенно меняются от приложения к приложению. Адаптация принципиально неприменима, например, к приложениям для передачи через сеть бинарных файлов.

Приоритезация трафика

Более универсальным подходом к обеспечению качества является приоритезация трафика. В отличие от адаптации, данный механизм зависит от инфраструктуры сети и работает в терминах управления ресурсами её коммутаторов. Приоритезация разделяет весь передаваемых трафик на несколько классов, сопоставляя каждому из них собственный приоритет. Далее ресурсы каждого коммутатора перераспределяются так, чтобы траффик с большим приоритетом получал большее количество его ресурсов. Маршруты передачи данных при этом не перестраиваются.

Примером использования приоритезации может служить перераспределение ресурсов сети между видеотрансляцией и резервным копированием файлов. Пусть эти приложения конкурируют за пропускную способность одного из каналов передачи. Тогда качество

видео можно улучшить, повысив приоритет её собственного трафика или же понизив приоритет трафика приложения резервного копирования.

На практике желаемое перераспределение ресурсов производится на коммутаторах за счёт планирования очерёдности обработки пакетов из разных потоков. В случае, если пакеты одного потока обрабатываются чаще, то он получает большее количество сетевых ресурсов и, как следствие, лучшее качество сервиса. Существует несколько различных подходов к буферизации пакетов внутри коммутатора, а так же множество алгоритмов активного управления очередью (Active Queue Management), осуществляющих выбор пакетов для обработки из его буферов. Эффективность работы таких алгоритмов является одним из центральных факторов, определяющих параметры качества сервиса проходящих через сеть соединений.

Приоритезация трафика требует от коммутаторов сети способности перераспределение ресурсы в зависимости от приоритетности трафика. Кроме того, трафик приложений должен помечаться соответствующими значениями приоритета при попадании пакетов в сеть. Такова минимальная цена за независимость механизма управления качеством от логики конкретных приложений.

QoS-маршрутизация

Существенным ограничением приоритезации является локальность его действия. Данный метод позволяет управлять ресурсами каждого конкретного коммутатора независимо друг от друга, однако он не предоставляет механизмов для согласованного управления сразу несколькими коммутаторами. Например, если один из коммутаторов сети перегружен, но при этом существует альтернативный пути передачи, не проходящий через данный коммутатор, то одна лишь приоритезация не позволит перераспределить ресурсы сети так, чтобы часть потоков данных воспользовалась обходными путями. Более мощным механизмом, обладающим необходимым глобальным видением сети и предоставляющим возможности для подобного перераспределения ресурсов, является QoS-маршрутизация.

Традиционные протоколы маршрутизации, такие как RIP, OSPF и EIGRP, также обладают механизмами для динамического детектирования и устранения заторов с помощью изменения части маршрутов и балансировки трафика по нескольким альтернативным путям (multipath routing). Однако, при балансировке они не учитывают требования качества сервиса отдельных потоков трафика. В результате, если затор образован несколькими нетребовательными приложениями, и одним приложением с жёсткими ограничениями ка-

чества, вполне вероятно, что часть пакетов последнего будет направлена альтернативным маршрутом с худшим качеством.

QoS-маршрутизация, напротив, осуществляет маршрутизацию каждого потока исходя из его требований качества и независимо от трафика других приложений. Маршрутизация на уровне потоков позволяет, например, направлять трафик приложений, запущенных на одних и тех же узлах сети по разным маршрутам и даёт средство для гранулярного контроля над распределением доступные сетевых ресурсов. Как и в случае с приоритезацией, увеличение точности контроля над распределением ресурсов требует усложнения коммутаторов. Реализация маршрутизации на уровне потоков предполагает, что коммутаторы способны запоминать правила маршрутизации для каждого из проходящих через него потоков и сопоставлять с ними поступающие пакеты. Описанную функциональность тяжело эффективно реализовать в сетях с традиционной архитектурой, и это всегда препятствовало широкому применению QoS-маршрутизации на практике. Маршрутизация потоков, однако, является одним из базовых принципов программно-конфигурируемых сетей на основе протокола OpenFlow, что делает эту платформу очень перспективной для развития данного метода обеспечения качества.

Резервирование ресурсов

Изменение нагрузки на отдельные элементы сети, вызванное непредсказуемым поведением пользователей, приводит к постоянному перераспределению ресурсов сети, и, как следствие, постоянному изменению метрик качества для существующих в сети соединений. В то же время многие приложения неспособны работать с использованием соединений, не соответствующих их требованиям качества. Например, соединение со слишком большим временем передачи пакета может сделать отзывчивость игры неприемлемо низкой. Для приложений данного типа нужны соединения с гарантиями качества. В тоже время ни один из рассмотренных выше способов обеспечения качества сервиса такие гарантии не предоставляет. Ни адаптация приложения, ни QoS-маршрутизация не приспособлены для сохранения качества соединения при перегрузке сети. Приоритезация позволяет изменить распределение ресурсов в условиях перегрузки, но так же не позволяет решить данную проблему. В самом деле, даже если назначить максимальный приоритет единственному потоку, а всем остальным потокам в сети назначить самые низкие приоритеты, то достаточное количество низкоприоритетных потоков всегда может вызвать нарушение требований качества высокоприоритетного потока.

Единственным механизмом управления качеством, способным дать приложению гарантии относительно построенного соединения, является резервирование ресурсов. При использовании резервирования для потока заранее прокладывается маршрут, часть ресурсов вдоль которого отдаются этому потоку в эксклюзивное пользование. Зарезервированные таким образом ресурсы не будут использоваться для обработки других потоков. Тем самым, приложение получает гарантию на сервис заданного качества.

Резервирование ресурсов, так же как и QoS-маршрутизация, работает на уровне отдельных потоков данных и предъявляет аналогичные требования к коммутаторам сети. Более того, для резервирования дополнительно требуется возможность явного выделения ресурсов коммутатора, что ещё больше затрудняет его реализацию в традиционных компьютерных сетях. Тем не менее, возможность получения гарантий качества подтолкнуло сообщество на разработку нескольких протоколов резервирования. Наиболее известный из них, протокол RSVP был разработан ещё в начале 1990х годов [71]. Однако, ни он, ни его последователи, протоколы RSVP2 и NSLP [80], не получили широкого распространения главным образом в силу чрезмерно высокой нагрузки, которой они подвергали коммутационные устройства сети. Важно отметить, что протоколы резервирования обычно не занимаются самостоятельным прокладыванием маршрутов передачи данных, а полагаются на маршруты, построенные другими протоколами, поэтому им, вообще говоря, не свойственны возможности маршрутизации с учётом потоков и, тем более, требований качества сервиса.

2.1.5 Выводы

Как это видно из приведённого описания существующих механизмов обеспечения качества, ни один из них не лишён недостатков. Возможности адаптации ограничены, к тому же данный подход требует вмешательства в логику работы сетевого приложения. Приоритезация трафика допускает лишь локальное управление ресурсами коммутатора, чего может быть недостаточно для действительно эффективного распределения ресурсов между сетевыми приложениями. QoS-маршрутизация использует недостающее приоритезации глобальное виденья сети, однако ни приоритезация, ни QoS-маршрутизация не дают гарантий относительно построенных соединений. Резервирование, напротив, предоставляет такие гарантии, но, вообще говоря, не позволяет так же эффективно управлять ресурсами, как это позволяет QoS-маршрутизация. При этом независимое использование описанных методов не способно дать того же результата, что их координированное применение.

Два наиболее перспективных метода управления качеством: QoS-маршрутизация и резервирование ресурсов – предполагают независимую маршрутизацию потоков. Поэтому перспективной представляется идея адаптировать данные методы к сетям SDN, которые строятся на том же базовом принципе. Более того, контроллер сети SDN располагает глобальным видением сети, необходимым QoS-маршрутизации для эффективного построения путей передачи данных. Данные особенности сетей SDN также позволяют надеяться дополнительное повышение эффективности этих методов, которое было недоступным в условиях сетей с традиционной архитектурой.

2.2 Организация обработки пакетов на коммутаторе

Данный раздел содержит описания методов организации внутреннего устройства и принципов функционирования коммутаторов, необходимые для проектирования адекватных моделей и методов оценки качества обслуживания, которое эти устройства предоставляют проходящим через них потокам данных.

2.2.1 Компоненты коммутатора

Изнутри аппаратный сетевой коммутатор представляет собой конвейер для параллельной обработки пакетов. На сегодняшний день известно множество различных конфигураций таких конвейеров. Разные модели коммутаторов реализуют разное количество стадий обработки пакетов, используют разные типы обработчиков, а так же различные последовательности их сопряжения. Далее приводится классификация указанных обработчиков.

Среди различных обработчиков будем выделять следующие основные классы:

Анализатор пакетов просматривает заголовки пакета, сопоставляет их с одним из известных ему потоков, и принимает решение о необходимости копировать тело пакета, изменять заголовки его копий, а так же пересылать полученные копии через выходы. Фактически анализатор реализует отношение трансформации на множестве состояний пакетов, введённом в работе [81]. Хорошей абстракцией анализатора пакетов является описание конвейера обработки пакетов, сформулированное спецификациями протокола OpenFlow [29];

Буферный блок используется для синхронизации других обработчиков конвейера, а так же для переупорядочивания поступающих на него пакетов. Буферный блок осу-

ществляет хранение пакетов, которые готовы перейти на следующую стадию конвейера, пока соответствующий обработчик ещё не готов их принять;

Коммутационная матрица является центральным элементом любого коммутатора. Она обладает множеством входов и выходов и пересылает пакеты между ними в соответствии с метаинформацией, полученной от анализатора. В зависимости от своей логики, матрица может отказываться обрабатывать пакеты в отдельных позициях поступившего к ней на вход вектора. При этом, в зависимости от конкретной реализации коммутатора, обработка пакеты может быть либо отложена на следующий такт, либо же пакет может быть сброшен.

Некоторые реализации указанных обработчиков могут работать в терминах целых пакетов, а некоторые – разбивать пакеты на более мелкие порции и снабжать каждую из которых собственным заголовком, образуя *ячейку* (cell) фиксированной длины. При всей простоте первых, они зачастую проигрывают в эффективности последним. Например, с помощью обработчиков ячеек возможно построение так называемых cut-through [82] коммутаторов. В отличие от store-and-forward коммутаторов, работающих в терминах целых пакетов, cut-through коммутаторы могут начать обрабатывать пакет сразу же после получения нескольких первых ячеек, в которых содержится заголовок. При этом, например, возможна ситуация, когда коммутатор уже начал передавать копии пакета через свои выходы, ещё не успев получить весь оригинальный пакет.

Коммутаторы, построенные на принципе cut-through, однако имеют и некоторые недостатки. Например, выгода от использования cut-through теряется, если выходной канал имеет меньшую пропускную способность, чем входной канал. Кроме того, коммутатор обязан пересылать данные пакетов без разрывов, поэтому хвостовые ячейки пакета, головные ячейки которого уже начали пересылаться, должны быть обработаны в первую очередь. Подобные требования накладывают дополнительные ограничения на конвейер, и необходимость их выполнения приводит к снижению общих показателей производительности коммутатора [83].

2.2.2 Анализатор пакетов

Анализатор пакетов представляет собой элемент с одним входом и множеством выходов. Способ обработки конкретного пакета определяется множеством его заголовков и заложенными в анализатор правилами трансформации пакетов.

Анализатор пакетов обычно представляет собой небольшой конвейер из нескольких стадий обработки, соответствующих различным заголовкам пакета. Например, поиск таблице MAC-адресов осуществляется по заголовку канального уровня L2, по таблицам маршрутизации и преобразования адресов NAT – по заголовку сетевого уровня L3, по таблицам контроля доступа ACL – по заголовку транспортного уровня L4. При этом зачастую анализ может быть завершён досрочно, после прохождения пакетом лишь нескольких первых стадий, и наиболее продвинутые реализации анализаторов используют эту возможность для снижения задержки. В результате, задержка обработки пакета на cut-through анализаторах зависит не только от длины пакета, но и от множества его заголовков.

Современные анализаторы предусматривают быструю аппаратную обработку большей части поступающих на них пакетов, например, с помощью кэширования правил в таблицах коммутации, позволяющих быстро найти инструкции для обработки пакета по его заголовку. Для обработки пакетов без программного принятия решения об их передаче обычно используют термин fast-path обработка. Аппаратные обработчики имеют задержку близкую к постоянной.

Если быстрый поиск инструкций закончился неудачей, либо же в его результате были найдены инструкции, для которых не реализовано аппаратной поддержки, происходит программная обработка пакета (этот случай обработки пакетов часто называют slow-path). При этом задержка анализа пакета сильно зависит от конкретного пакета, трудно предсказуема, и может значительно превышать задержку обработки пакета на fast-path. Для абсолютного большинства современных сетевых устройств и проходящих через них потоков данных частота появления slow-path пакетов значительно уступает частоте появления пакетов fast-path, поэтому при вычислении задержки передачи пакетов обработка slow-path рассматриваться не будет.

Помимо поиска инструкций для обработки пакета, анализатор так же записывает найденные инструкции контекст пакета, предназначенный для передачи данных между обработчиками внутри коммутационного конвейера. Например, в некоторых коммутаторах инструкции по перезаписи заголовков пакета исполняются непосредственно перед пересылкой через выход коммутатора. Кроме того, некоторые реализации анализаторов способны выполнять частичное выполнение инструкций самостоятельно. Некоторые анализаторы размножают пакеты, предназначенные для групповой передачи (multicast) или широковещания (broadcast).

2.2.3 Буферный блок

Буферный блок осуществляет временное хранение ячеек данных, необходимое для обеспечения взаимодействия разнообразных элементов коммутационного конвейера, которые могут обрабатывать ячейки с разными скоростями. Модуль буферного блока имеет один вход и один выход. В начале каждого такта блок считывает пакет со своего входа и пытается сохранить его в свою память. В ряде случаев, например, если в памяти блока нет свободного места, полученный пакет может быть сброшен. На каждом такте один из хранящихся буферным блоком пакетов помещаются на его выход. Если в течение этого такта следующий элемент коммутационного конвейера не произвёл считывание, то буферный блок может поменять своё решение, и поместить на выход другой пакет. В противном случае, пакет удаляется из памяти буферного блока.

Буферные блоки способны менять относительный порядок сохранённых в них пакетов, а так же сбрасывать одни пакеты чаще других, тем самым перераспределяя доступное время других обработчиков коммутационного конвейера. Указанный механизм управления ресурсами коммутатора позволяет, в частности, добиться уменьшения задержки передачи пакетов одних потоков путём ухудшения этого показателя для других потоков. Полученная в результате дифференциальная система положена в основу модели управления качеством DIffServ [84].

Наиболее простой с точки зрения аппаратуры реализацией буферного блока является очередь пакетов с дисциплиной обслуживания FIFO. Но такая очередь не позволяет переупорядочивать поступающие в буферный блок пакеты и, следовательно, не предоставляет возможностей для дифференциации качества сервиса между передаваемыми потоками. Поэтому буферный блок современного коммутатора, как правило, включает в свой состав несколько FIFO-очередей с ограничением размера, классификатор, который распределяет поступившие на вход буферного блока пакеты между его очередями, и планировщик, который выбирает один из хранящихся в них пакетов и помещает его на выход буферного блока.

Классификатор представляет собой простейший анализатор пакетов, который сопоставляет каждый пакет с одним из предопределённых классов обслуживания и направляет его пакет в соответствующую этому классу внутреннюю очередь. При этом одна очередь может быть сопоставлена сразу нескольким классам обслуживания.

Далее в работу вступает фильтр очереди, который исходя из размера и класса обслуживания пакета, а так же её текущей утилизации, помещает его в конец очереди, или же

осуществляет его сброс. Примером простейшего фильтра может служить Tail-Drop, который начинает сбрасывать пакеты при переполнения очереди. Более сложным примером фильтра является фильтр RED (Random Early Detection), который начинает сбрасывать случайные пакеты после того, как занятость буфера превысила заданное пороговое значение, постепенно увеличивая вероятность сброса вместе при уменьшении объёма свободной памяти. Обычно алгоритм RED вычисляет текущую вероятность сброса пакета по заданной утилизации очереди исходя из пропорции, построенной по двум крайним значениям: (1) до достижения порогового значения не должно быть сброшено ни одного пакета, (2) при полном заполнении очереди должны сбрасываться все направленные на неё пакеты.

На практике использование фильтра RED предпочтительнее, чем Tail-Drop, так как в случае появлении затора он позволяет избегать эффекта синхронизации активных TCP соединений [85], при котором сразу несколько отравителей производят одинаковые изменения своего окна передачи TCP. В результате, объём передаваемого трафика распределяется во времени крайне неравномерно, что приводит к резкому падению показателей эффективности сети.

Расширением алгоритма RED, позволяющим не только бороться с эффектом синхронизации, но и более точно перераспределять ресурсы коммутатора, является алгоритм WRED (Weighted RED). Указанный алгоритм предусматривает возможность задания собственных пороговых значений для каждого класса обслуживания, ячейки которого отображаются в одну и ту же очередь. Таким образом, при переполнении очереди ячейки менее важного класса обслуживания будут сбрасываться чаще, чем более важного.

Логика работы планировщика, который выбирает один из первых пакетов, хранящихся в очередях буферного блока, выходным, может строиться на самых разных принципах. Одним из стандартных подходов является приоритезация очередей. При этом первый пакет каждой очереди может быть выбран выходным лишь в том случае, если все более приоритетные очереди пусты. Таким образом, ячейки из более приоритетных очередей способны заблокировать обработку ячеек из менее приоритетных очередей. На практике такой ситуации не происходит потому, что почти весь трафик помещается в очереди с равными приоритетами, и лишь малая его часть проходит через очереди с большим приоритетом. Обычно большим приоритетом обладают служебные потоки, обработка которых необходима для корректного функционирования сети.

Для решения проблемы блокировки неприоритетных потоков обычно применяются различные алгоритмы из семейства Round-Robin, которые циклически перебирают пакеты из очередей буферного блока. Алгоритм WRR (Weighted Round-Robin) позволяет добить-

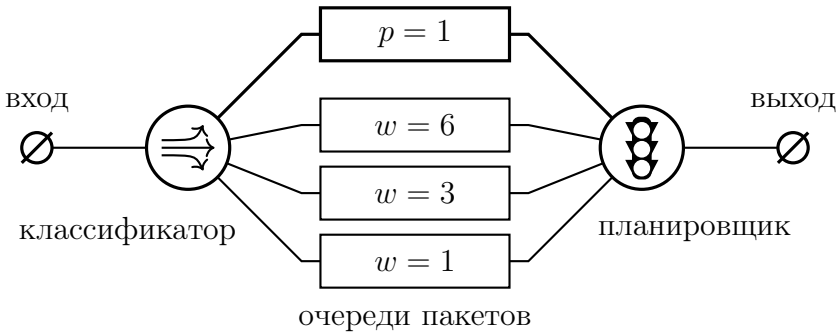

Рисунок 2.1: Типичное устройство буферного блока с поддержкой качества сервиса.

ся перераспределения ресурсов между существующими классами обслуживания путём назначения очередям весовых коэффициентов. Переключившись на новую очередь буферного блока, планировщик пакетов, работающий по указанному алгоритму, не переходит к следующей очереди, пока данная очередь не пуста, или же из неё уже было выбрано число пакетов, равное её весу. Таким образом, объём ресурсов коммутатора, выделенный для обработки потоков заданного класса обслуживания, пропорционален весу очередей, в которые будут размещены их пакеты.

Распространённой разновидностью Round-Robin алгоритмов является SRR (Shaped Round-Robin). Указанный алгоритм переходит к следующей очереди при выборе каждого очередного пакета, однако, имеет более сложный цикл работы, состоящий из нескольких раундов. В раунде с номером k участвуют лишь те очереди, которые имеют весовой коэффициент $w \geq k$. Если в начале очередного раунда не осталось ни одной непустой очереди, способной в нём участвовать, то алгоритм начинает работу с первого раунда. Таким образом, алгоритм SRR позволяет добиться равномерной передачи пакетов каждого класса обслуживания, и исключает возникновения импульсов, характерных для более простого в реализации алгоритма WRR.

На практике механизмы назначения приоритетов и весовых коэффициентов могут быть совмещены. Типичный буферный блок с поддержкой управления качеством изображён на рисунке 2.1. Как правило, такие блоки имеют единственную очередь с высоким приоритетом и сразу несколько очередей с одинаковым более низким приоритетом, но разными весами. При этом на каждом такте планировщик проверяет, есть ли пакеты в более приоритетной очереди, и в случае её пустоты продолжает выполнение реализованного в нём round-robin алгоритма для выборки из остальных очередей. В теории буферный блок может иметь и несколько групп очередей по приоритетам, выборка из которых будет производиться в случае пустоты всех более приоритетных очередей.

2.2.4 Коммутационная матрица

Коммутационная матрица отвечает за передачу пакетов между обработчиками на входах и выходах коммутатора, согласно предписаниям, полученным от анализатора пакетов. Конкретные реализации коммутационной матрицы могут быть построены на самых разных принципах. На практике в её качестве могут использоваться общая шина передачи данных, полная матрица, способная одновременно соединить каждый из входных портов с одним из выходных портов, многоярусные сети коммутационных элементов и разнообразные гибридные схемы на их основе [86]. Если матрица способна одновременно передавать k пакетов, то обычно говорят, что она имеет k плоскостей передачи. Например, общая шина имеет единственную плоскость передачи, а полная матрица с N входами и N выходами – ровно N таких плоскостей. Как правило, матрица предоставляет по одному входу на каждый вход коммутатора. Хотя существуют реализации, в которых каждому входу матрицы соответствует сразу несколько входов коммутатора, и, наоборот, когда каждому входу коммутатора сопоставлено сразу несколько входов коммутационной матрицы. Ограниченное число входов, выходов и плоскостей передачи матрицы зачастую не позволяют сразу же обрабатывать все поступившие на её входы пакеты. Таким образом, на каждом такте матрица должна осуществлять выбор входов и выходов, которые должны быть соединены и обслужены.

Для указанной проблемы поставлено множество разнообразных оптимизационных задач, и соответствующих им алгоритмов решения, нацеленных на поиск допустимого расписания передачи, лучшего с точки зрения одного из критериев эффективности [87; 88]. Наибольшие успехи были достигнуты в области алгоритмов планирования, предполагающих разбиение пакетов на ячейки фиксированной длины. Хорошо изучена задача поиска расписания, которое бы на каждом такте передавало максимальное количество ячеек, сводится к задаче поиска максимального независимого паросочетания на двудольном графе, каждая вершина которого соответствует входу или выходу коммутационной матрицы, а каждое ребро – заявке на передачу ячейки от заданного входа на заданный выход.

Алгоритм, корректно решающий описанную выше задачу, однако, не удовлетворяет условиям стабильности и справедливости, допуская бесконечную блокировку ячеек при специально подобранном входном расписании. Однако известны и постановки задачи с критерием оптимизации, лишённым указанного изъяна. Стабильность и справедливость гарантирует, например, решение задачи поиска независимого паросочетания, удовлетворяющего заявки ячеек с максимальным суммарным временем ожидания [87].

2.2.5 Методы буферизации

В каждый момент времени через один канал может передаваться единственный пакет. При появлении второго пакета, предназначенного для передачи через занятый выход, коммутатор может, например, выполнить его сброс. Таким образом, если, например, каждый пятый такт на такой коммутатор будет поступать по три пакета, адресованных на один и тот же выход, то лишь один из них будет передаваться, а остальные – будут сбрасываться. Причём эта ситуация сохраняется, даже при учёте того, что в остальные промежутки времени канал, за который эти пакеты конкурируют, будет простаивать.

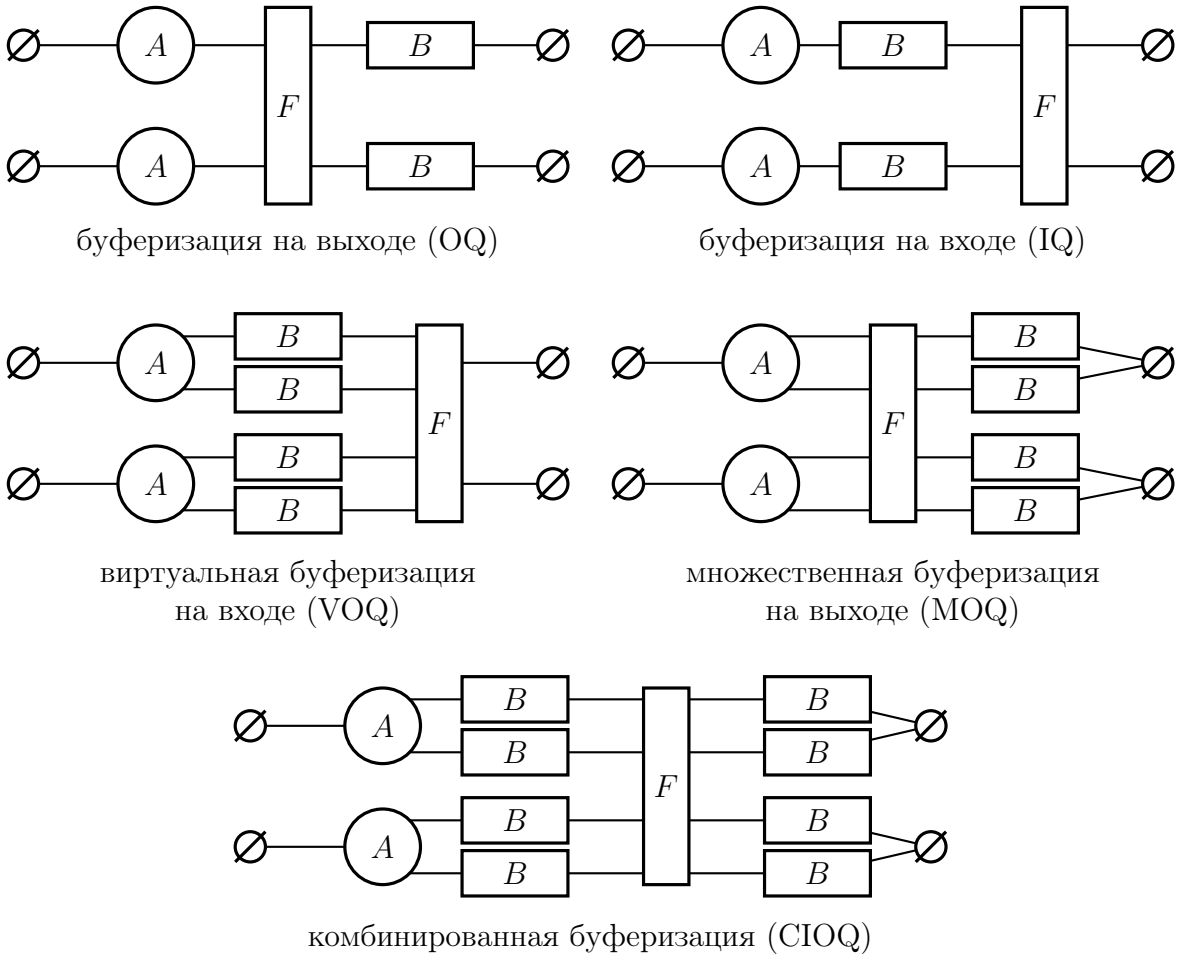

Рисунок 2.2: Схемы буферизации современных коммутаторов. Основные компоненты коммутатора: анализаторы пакетов, буферные блоки и коммутационные матрицы – обозначены символами A, B и F соответственно.

Повысить эффективность и обеспечить более высокую степень утилизации каналов передачи данных позволяет идея буферизация на выходе коммутатора (Output Queuing, OQ). Построенные на её основе коммутаторы помещают направленные на выходной порт

пакеты в буферную память, откуда эти пакеты изымаются сразу же при освобождении необходимого для их передачи канала.

Буферная память позволяет распределять во времени моменты, когда объём трафика, который необходимо передать через канал, превышает пропускную способность этого канала. Однако если объём трафика достаточно долго превышает пропускную способность канала, то неизбежно произойдёт насыщение буфера, и пакеты начнут сбрасываться, как и в коммутаторах без буферизации. При этом, чем большим буфером располагает коммутатор, тем большие пиковые нагрузки на выходной канал могут быть сглажены с его помощью.

Основным недостатком буферизации на выходе являются требования указанного принципа построения коммутаторов к скорости обработки пакетов, а так же скорости и объёму памяти выходных буферов. Чтобы успевать за пакетами, поступающими одновременно на N входных портов, коммутатор должен тратить на обработку пакета в N раз меньше времени, чем требует его сборка при получении пакета из кабеля. Такие же ограничения накладываются и на скорость записи пакета в буферную память. Кроме того, объём памяти каждого выходного буфера должен, как минимум, в N раз превышать максимальный размер пакета.

В современных компьютерных сетях некоторые каналы способны передавать по 40 Гб ежесекундно, а коммутаторы часто имеют более 48 портов. Таким образом, при буферизации на выходе требования к буферной памяти приближаются к пределам возможностей её современных образцов [88].

Альтернативой для буферизации на выходе является буферизация на входе (Input Queuing, IQ), при которой пакеты ожидают освобождения исходящего канала передачи данных в очередях на входных портах коммутатора. Такая организация буферизации также позволяет не сбрасывать пакеты при их одновременном поступлении на все входные порты коммутатора, но в то же время даёт возможность обрабатывать пакеты со скоростью, значительно уступающей суммарной скорости поступления пакетов по подключённым к нему каналам.

Наиболее простой в реализации является буферизация на входе, при которой каждому входному порту ставится в соответствие собственная очередь пакетов с дисциплиной обслуживания FIFO. При этом первый пакет очереди изымается лишь в том случае, если выходной порт, через который необходимо этот пакет передать, не занят. Таким образом, если пакет находится в хвосте очереди, то он блокируется даже в том случае, если нужный для его передачи канал в данный момент свободен. Указанная проблема блокировки

очереди её первым элементом (Head-of-Line Blocking) не позволяет добиться высокой утилизации исходящих каналов передачи данных.

Например, работа [89] демонстрирует, что пропускная способность коммутатора с описанной реализацией буферизации на входе может составлять менее 59% от аналогичного показателя для коммутатора с буферизацией на выходе даже при поступлении равномерно распределённого трафика.

Для буферизации на входе существуют более сложные реализации, которые не уступают по своей эффективности коммутаторам с буферизацией на выходе. Наиболее распространённой из них является буферизация виртуальных выходных очередей (Virtual Output Queue, VOQ). При указанной организации каждому входному порту коммутатора ставится в соответствие не одна, а сразу несколько очередей пакетов – по числу его выходных портов.

VOQ-коммутатор анализирует поступивший на входной порт пакет и сразу же помещает его в очередь, соответствующую требуемому для него выходному порту. Впоследствии пакет будет изъят из очереди, как только освободится соответствующий выходной канал. Известно, что VOQ-коммутатор гарантированно способен копировать поведение OQ-коммутатора, если он будет обрабатывать пакеты в 4 раза быстрее скорости их прохождения через подключённые к нему каналы, причём указанная оценка не зависит от числа входов и выходов коммутатора [88].

Множественная буферизация на выходе (Multiple Output Queuing, MOQ [90]) так же, как и обычная буферизация на выходе (OQ), располагает буферные блоки вслед за коммутационной матрицей. Однако, как и в случае VOQ, на каждом порту располагается равное их числу количество буферных блоков. При этом предполагается использование неблокирующей матрицы с достаточным количеством плоскостей передачи для того, чтобы обеспечить попарную связь любой комбинации входных и выходных портов;

При комбинированной буферизации (Combined Input-Output Queuing, CIOQ) буферные блоки располагаются как на входах, так и на выходах матрицы. При небольших нагрузках указанная схема позволяет достичь производительности метода OQ без использования большого количества буферных блоков и мощной коммутационной матрицы.

На практике наиболее распространёнными из рассмотренных схем являются OQ, CIOQ и VOQ. Коммутаторы с небольшой суммарной пропускной способностью портов обычно реализуют буферизацию OQ. Более мощные магистральные коммутаторы, как правило, строятся на базе CIOQ. Если же от коммутатора требуется высокая пропускная способность или низкая латентность, то применяется VOQ. Буферизация IQ обладает низкой

производительностью и интересна лишь с точки зрения теории, а метод множественной буферизации MOQ слишком дорог и на текущий момент не востребован.

2.3 Оценка задержки с помощью сетевого исчисления

В ходе проведённых исследований было рассмотрено несколько методов оценки для задержки передачи пакетов через сеть. При этом в частности, были проанализированы перспективы для использования модели на основе теории очередей [91]. Рассмотрена возможность получения верхних оценок для времени обработки ячеек ATM на планировщике с дисциплиной обслуживания WRR путём синтеза и анализа "наиболее неудачных" сценариев его работы [92]. Рассмотрен метод аналитического вычисления оценок сквозной задержки на основе систем стохастических дифференциальных уравнений, учитывающих протоколы перегрузки TCP конечных хостов [93].

Каждый из указанных методов обладает собственными недостатками. Например, предположение об использовании хостами единственного протокола перегрузки TCP в современных компьютерных сетях не выполняется. К тому же использование стохастических оценок задержки неприменимо для обеспечения гарантий выполнения требований приложения. Большое количество предлагает вычислять верхние оценки для сквозной задержки передачи пакетов путём построения верхних оценок для прохождения отдельных элементов сети и последующего суммирования этих оценок. Однако пакет, испытавший максимальную задержку при прохождении одного из элементов сети, зачастую никогда не может испытать максимальную задержку ещё и на следующем её элементе. Поэтому полученная указанным образом итоговая оценка для задержки при прохождении через сеть из нескольких коммутаторов оказывается слишком пессимистичной, даже при высокой точности оценки задержки каждого из элементов сети.

Поскольку теория сетевого исчисления (network calculus) [15; 94] рассматривает проблему эффективной композиции оценок для задержки отдельных элементов сети при получении итоговой оценки как наиболее приоритетную и, к тому же, позволяет получать детерминированные верхние оценки, то именно этот аппарат был выбран в качестве базы для разрабатываемого формального метода.

2.3.1 Основы теории сетевого исчисления

Сетевое исчисление изучает зависимость между характеристиками отдельных элементов коммутационной сети и качеством сервиса, которое предоставляется проходящим через эту сеть потокам данных. В основе данной теории лежит идея получения детерминированных оценок путём рассмотрения граничных сценариев функционирования отдельных частей сети и комбинирования этих сценариев между собой. Указанный подход может быть использован, например, для расчёта наихудшего качества сервиса, которое может быть предоставлено соединению при прохождении через сеть с заданными характеристиками, или, наоборот, для определения минимальных требований к элементам сети, при выполнении которых заданные потоки обязательно будут обслужены с надлежащим качеством.

При моделировании сети с помощью сетевого исчисления необходимо определить:

1. Разбиение коммутационной сети на обработчики (server/service element/IO-system). Разные обработчики могут моделировать элементы сети различного масштаба, начиная от физической линии передачи данных и очереди пакетов внутри сетевого устройства и заканчивая отдельными коммутаторами и даже целыми подсетями;

2. Описание потоков данных. Для каждого потока задаётся последовательность его передачи через обработчики и кривая нагрузки (arrival curve), которая описывает характерные для него ограничения на скорость поступления данных в сеть;

3. Описание обработчиков. Поведение каждого обработчика в сети задаётся кривой сервиса (service curve), которая определяет его производительность, а так же дисциплиной мультиплексирования, которая описывает правила распределения его ресурсов между проходящими через него потоками, если таких потоков несколько.

Для моделирования обработчиков и потоков данных сетевое исчисление использует накопительные (cumulative) функции времени, которые выражают общее количество информации, переданной этими модельными сущностями с момента начала отсчёта. Множество $\mathcal{F}$ всевозможных накопительных функций определяется как множество всевозможных неубывающих и непрерывных слева функций, определённых при неотрицательных значениях аргумента, графики которых проходят через начало координат.

Замечание: Область определения накопительных функций часто расширяют и на отрицательную полуплоскость: в этом случае функции должны обладать свойством каузальности, то есть принимать нулевые значения при любом отрицательном значении ар-

гумента [15]. Так же, вместо свойства непрерывности слева от функций из множества F иногда требуют её непрерывности справа [91], однако такой подход требует отдельного рассмотрения значения функции в точке ноль.

$$\mathcal{F} = \left\{ f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0} \cup \{+\infty\} \middle| \begin{array}{c} LeftContinuous(f) \wedge \\ NonDecreasing(f) \wedge Causal(f) \end{array} \right\}$$

$$NonDecreasing(f) = \forall t_1 \leq t_2 : \ f(t_1) \leq f(t_2)$$

$$LeftContinuous(f) = \forall t_0 \in \mathbb{R} : \lim_{t \to t_0 - 0} f(t) = f(t_0)$$

$$Causal(f) = t = 0 \to f(t) = 0$$

Множество $\mathcal{F}$ позволяет описывать объекты с использованием моделей трёх типов:

1. *Дискретные модели* (discrete model) предполагают, что моделируемый объект передаёт лишь конечные порции данных, причём передача каждой такой порции осуществляется мгновенно, и может происходить лишь в дискретный момент времени, соответствующий началу одного из тактов. Дискретным моделям соответствуют ступенчатые накопительные функции;

2. *Жидкостные модели* (fluid model), напротив, способны осуществлять передачу в произвольные моменты времени, однако они не допускают мгновенной передачи порций данных конечного размера. Жидкостные модели задаются непрерывными накопительными функциями;

3. *Непрерывные модели* (continuous model) не накладывают никаких дополнительных ограничений на моделируемые объекты. Поэтому произвольная дискретная или жидкостная модель также является и непрерывной.

Поскольку элементы коммутационной сети всегда имеют конечную гранулярность и оперируют в терминах битов, байтов и пакетов, то для их естественного описания лучше подходят дискретные модели. Использование жидкостных моделей, в свою очередь, часто даёт возможность упростить теоретические выкладки и, как правило, позволяет достичь большей вычислительной эффективности. В то же время, наибольшей практической ценностью обладают результаты, полученные для класса непрерывных моделей: они справедливы как для дискретных, так и для жидкостных моделей, и позволяют строить произвольные комбинации из моделей разной природы.

2.3.2 Отставание и задержка передачи данных

Функцией *прибытия* (*отправки*) потока данных для обработчика S называется такая накопительная функция $A \in \mathcal{F}$ ($D \in \mathcal{F}$), которая описывает зависимость суммарного количества данных потока, поступивших на этот обработчик (переданных этим обработчиком), от времени.

Передавая полученные данные, каждый обработчик S сопоставляет функции прибытия $A \in \mathcal{F}$ функцию отправки $D \in \mathcal{F}$, зависящую от функции A и свойств обработчика. Таким образом, обработчик может быть описан перечислением соответствующих ему пар вида $\langle A, D \rangle$ и задан отношением вида $S \subset \mathcal{F} \times \mathcal{F}$.

После получения данных обработчики способны временно удерживать их внутри себя. Зависимость объёма хранящихся в обработчике S данных от времени называется его *отставанием* (backlog) $b(t)$ и представляется в виде разности функций прибытия A и отправки D обслуживаемого им потока данных, $\langle A, D \rangle \in S$:

$$b(t) = A(t) - D(t)$$

Обработчик не может начать передачу данных до того, как их получил, поэтому в каждый момент времени t значение $A(t)$ его функции прибытия превышает значение $D(t)$ его функции отправки, а отставание любого обработчика неотрицательно.

Периодом отставания (backlogged/busy period) называется временной интервал, в течение которого отставание b(t) обработчика строго положительно. Если момент $t \in \mathbb{R}$ находится внутри периода отставания $[SBP(t); EBP(t)]$, то его границы задаются следующими формулами:

$$SBP(t) = \sup \{u \leq t | A(u) = D(u)\}$$

$$EBP(t) = \inf \{u \geq t | A(u) = D(u)\}$$

Замечание: В случае непрерывных функций прибытия и отправки указанные формулы могут быть упрощены с использованием max и min вместо операций sup и inf. Однако в общем случае такие упрощения неверны.

Задержкой (delay) $d(t)$ обработчика S называется время, которое порция данных, поступившая в S в момент времени t, пробудет внутри этого обработчика. Предполагая обслуживание данных по дисциплине FIFO, задержку обработчика S можно выразить через функции прибытия A и отправки D, где $\langle A, D \rangle \in S$, следующим образом:

$$d(t) = \inf \{\tau \geq 0 | A(t) \leq D(t + \tau)\rangle$$

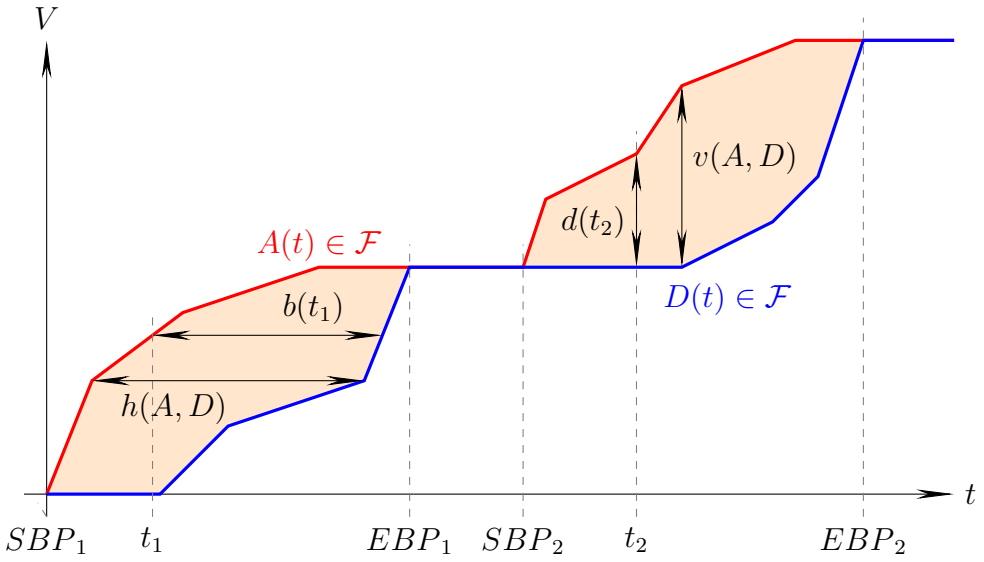

Рисунок 2.3: Отставание $b(t)$ и задержка $d(t)$ обработчика S при обслуживании потока, описанного парой накопительных функций $A, D \in \mathcal{F}$, $\langle A, D\rangle \in S$.

Если предположение об обработке данных потока по дисциплине FIFO не выполнено, то можно гарантировать лишь ограничение времени окончания обработки данных, поступивших в текущий период отставания, моментом завершения этого периода:

$$d(t) \leq EBP(t) - t = \inf\{u \geq t | D(u) \geq A(u)\}$$

Поскольку вне периодов отставания количество полученных и переданных данных совпадает, и полученную формулу можно переписать в следующем виде:

$$d(t) \leq \inf\{\tau \geq 0 | A(t+\tau) = D(t+\tau)\}$$

Для произвольного обработчика S и любой пары соответствующих этому обработчику функций прибытия A и отправки D, $\langle A, D\rangle \in S$, его отставание $b(t)$ ограничено сверху вертикальным отклонением $v(A, D)$ этих потоков. Более того, если обработчик S обслуживает данные по дисциплине FIFO, его задержка $d(t)$ ограничена сверху их горизонтальным отклонением $h(A, D)$:

$$\forall t \in \mathbb{R}: \; b(t) \leq v(A, D) = \sup_t \{A(t) - D(t)\}$$

$$\forall t \in \mathbb{R}: \; d(t) \leq h(A, D) = \sup_t \{\inf\{\tau \geq 0 | A(t) \leq D(t+\tau)\}\}$$

Графики вышеописанных функций изображены на рисунке 2.3.

2.3.3 Описание обработчиков с помощью кривых сервиса

Описание каждого обработчика путём перечисления пар из всевозможных функций прибытия и получающихся в результате их обслуживания функций отправки вряд ли мож-

но назвать практически применимым. Значительно более естественным методом описания обработчиков представляется их задание с помощью функций, отражающих зависимость между поступлением и передачей полученных данных. Содержательно подобного рода функции описывают сервис, с которым обработчик обслуживает поступивший на него поток данных, поэтому они называются *сервисными кривыми* (service curves). В литературе используется сразу несколько типов таких кривых [95].

Пусть накопительная функция производительности $P \in \mathcal{F}$ обработчика S выражает зависимость количества переданных им данных от времени при условии его полной загруженности. Рассмотрим ограничение, которое накладывает функция P на связь между функцией прибытия A и функцией отправки D обработчика S.

Величина отставания $b(t)$ в момент времени $t \in \mathbb{R}$ не может быть меньше разности между объёмом данных, поступивших на обработчик S, начиная с произвольного момента $s \leq t$, и объёмом данных, который он мог за это время обработать. Причём значение отставания не может быть отрицательным.

$$\forall s \leq t: \; b(t) \geq [(A(t) - A(s)) - (P(t) - P(s))]^+$$

Если в качестве u выбрать момент $SBP(t)$ начала текущего периода отставания, то указанное неравенство обратится в равенство. Поэтому его можно переписать в виде:

$$b(t) = \sup_{s \leq t} \left\{ [(A(t) - A(s)) - (P(t) - P(s))]^+ \right\}$$

Совмещая данную формулу с определением отставания b, получим оценку для функции передачи данных D:

$$D(t) = A(t) - \sup_{s \leq t} \left\{ [(A(t) - A(s)) - (P(t) - P(s))]^+ \right\}$$

$$D(t) = \inf_{s \leq t} \left\{ A(t) - [(A(t) - A(s)) - (P(t) - P(s))]^+ \right\}$$

$$D(t) = \inf_{s \leq t} \{ A(s) + (P(t) - P(s)) \}$$

Поскольку функция производительности P позволяет вычислять функцию отправки $D \in \mathcal{F}$ для произвольной функции прибытия $A \in \mathcal{F}$, то обработчик S может быть задан без явного перечисления всех удовлетворяющих ему пар вида $\langle A, D \rangle$.

Однако на практике количество обслуженных обработчиком данных часто зависит не только и не столько от времени, сколько от его состояния, точные изменения которого могут быть тяжело предсказуемыми. Например, при мультиплексировании потоков с разделением по времени (time division multiplexing) границы и размеры временного интервала,

выделенного под передачу данных потока, могут зависеть от характеристик передаваемого трафика. Поэтому для описания обработчиков обычно используется зависимость объёма данных, которые они потенциально способны обработать, не от текущего времени, а от длительности интервала их обслуживания.

Кривой производительности (variable capacity node) обработчика S называется такая функция $\beta_p \in \mathcal{F}$, что для каждого временного интервала длины $\tau \geq 0$, величина $\beta_p(\tau)$ не превышает объёма данных, который S способен обработать в течение этого интервала:

$$\forall t : \forall \tau : \; P(t+\tau) - P(t) \geq \beta_p(\tau)$$

Для кривой указанного соотношения существует графическая интерпретация. Функция $\beta_p \in F\mathcal{F}$ является кривой производительности элемента S, если его функция $P \in \mathcal{F}$ такова, что около произвольного участка её графика, соответствующего приращению аргумента $\tau > 0$, не может быть описано ни одного прямоугольника длины τ и высоты строго меньше $\beta_p(\tau)$, стороны которого были бы параллельны координатным осям.

Кривая производительности β_p позволяет построить нижнюю (наихудшую с точки зрения качества сервиса) оценку функции отправки D обработчика S по функции прибытия A следующим образом:

$$D(t) \geq \inf_{s \leq t} \{A(s) + \beta_p(t-s)\}$$

Подобную зависимость можно получить, используя и менее строгие предположения об обработчике. Например, вместо ограничения функции производительности P достаточно ограничить непосредственно функцию D.

Строгой кривой сервиса (strict service curve) обработчика S называется всякая такая функция $\beta_s \in \mathcal{F}$, что в процессе обслуживания потоков с функцией прибытия A и функцией отправки D, $\langle A, D \rangle \in S$, в течение каждого периода отставания $[s;t]$ элемент S обрабатывает по меньшей мере $\beta_s(t-s)$ данных:

$$D(t) - D(s) \geq \beta_s(t-s)$$

Поскольку функция отправки обработчика существенно зависит от функции прибытия, то вводить подобного рода ограничения вне периодов отставания не имеет смысла. Внутри же периодов отставания объём переданных обработчиком данных в точности совпадает с объёмом данных, которые он был способен обработать в соответствии со своей функцией производительности P:

$$D(t) - D(s) = P(t) - P(s)$$

Указанное выражение позволяет использовать строгую кривую сервиса β_s для получения нижней оценки функции отправки D, вид которой аналогичен оценке, построенной с использованием кривой производительности β_p:

$$D(t) \geq \inf_{s \leq t} \{A(s) + \beta_s(t-s)\}$$

В течение каждого периода отставания $[s;t]$ любая строгая кривая сервиса β_s обработчика S так же является и его кривой производительности.

$$D(t) - D(s) \geq \beta_s(t-s) \Rightarrow P(t) - P(s) \geq \beta_s(t-s)$$

В периоды же простоя, когда функция отставания b принимает лишь нулевое значение, кривая β_s не предъявляет к функции P никаких требований. Таким образом, ограничение строгой кривой производительности является более сильным, чем ограничение кривой сервиса. Если обозначить множество всех пар функций A и D, которые удовлетворяют ограничению функции β, с помощью $\mathcal{S}(\beta)$, то для каждого обработчика S выполнено соотношение $\mathcal{S}(\beta_p) \subseteq \mathcal{S}(\beta_s)$.

Для значительного количества результатов сетевого исчисления, однако, достаточно ещё более слабого ограничения на зависимость между функциями прибытия и отправки данных. *Нестрогой кривой сервиса* (service curve) обработчика S называется такая функция $\beta \in \mathcal{F}$, что при обработке каждого потока с функцией прибытия $A \in \mathcal{F}$ для функции их отправки $D \in \mathcal{F}$ выполняется неравенство:

$$D(t) \geq \inf_{s \leq t} \{A(s) + \beta(t-s)\}$$

Из указанного выражения следует, что каждая строгая кривая сервиса β_s и каждая кривая производительности β_p так же являются и его кривыми сервиса.

2.3.4 Описание потоков с помощью кривых нагрузки

Как и в случае функции производительности P обработчика, построение сколь-нибудь точной функции прибытия $A \in \mathcal{F}$ для заданного потока данных на практике может вызвать затруднения. Например, при передаче потока данных могут использоваться разнообразные алгоритмы предотвращения перегрузок (congestion avoidance), которые способны динамически изменять скорость пересылки данных в зависимости от текущей загруженности сети. Поэтому для описания потоков поступления данных используется тот же приём, что и для описания обработчиков: явная зависимость функции поступления A от времени заменяются зависимостью от длительности интервала передачи данных.

Кривой нагрузки (arrival curve) для потока с функцией прибытия $A \in \mathcal{F}$ называется такая функция $\alpha \in \mathcal{F}$, что для каждого временного интервала длины τ количество переданных в течение него данных этого потока не превышает величины $\alpha(\tau)$:

$$\forall t : \forall \tau : \ A(t+\tau) - A(t) \leq \alpha(\tau)$$

Кривая нагрузки ограничивает сверху скорость поступления данных потока на каждом интервале заданной длины. Указанное условие, в некотором смысле, противоположно ограничению кривой производительности. В геометрической интерпретации оно эквивалентно возможности "протащить" вдоль графика функции прибытия A каждый прямоугольник, стороны которого параллельны координатным осям, а высота h может быть выражена через его длину l по формуле $h = \alpha(l)$. Полученные ранее оценки задержки, отставания и связи входного и выходного потоков могут быть переписаны с использованием кривых сервиса и нагрузки в соответствии со следующими основными теоремами (доказательства этих теорем приведены в приложении B):

Теорема (Оценка отставания)**.** Пусть поток данных с функцией прибытия $A \in \mathcal{F}$, ограниченный кривой нагрузки $\alpha \in \mathcal{F}$, обслуживается обработчиком с кривой сервиса β. Тогда значение отставания обработчика не превышает вертикального отклонения между кривыми прибытия α и сервиса β:

$$\forall t \in \mathbb{R} : \ b(t) \leq v(\alpha, \beta) = \sup_{t \geq s \geq 0} \{\alpha(s) - \beta(s)\}$$

Теорема (Оценка задержки)**.** Пусть поток данных с функцией прибытия $A \in \mathcal{F}$, ограниченный кривой нагрузки $\alpha \in \mathcal{F}$, обслуживается обработчиком с кривой сервиса $\beta \in \mathcal{F}$ по дисциплине FIFO. Тогда значение задержки обслуживания этого потока не превышает горизонтального отклонения между кривыми прибытия α и сервиса β:

$$\forall t \in \mathbb{R} : d(t) \leq h(\alpha, \beta) = \sup_{t} \{\inf \{\tau \geq 0 | \alpha(t) \leq \beta(t+\tau)\}\}$$

Если дисциплина обслуживания неизвестна, то для задержки d(t) справедлива оценка:

$$d(t) \leq \inf \{\tau \geq 0 | \alpha(\tau) \leq \beta(\tau)\}$$

Теорема (Оценка выходного потока)**.** Если поток функцией прибытия $A \in \mathcal{F}$, ограниченный кривой нагрузки $\alpha \in \mathcal{F}$, поступает на обработчик с кривой сервиса $\beta \in \mathcal{F}$, то полученный в результате выходной поток ограничен кривой нагрузки $\alpha' \in \mathcal{F}$:

$$\alpha'(s) = \sup_{\tau \geq 0} \{\alpha(s+\tau) - \beta(\tau)\}$$

$$D(t) - D(s) \leq \alpha'(t-s)$$

Кривую нагрузки α', которой удовлетворяет исходящий из обработчика поток, часто называют *функцией обёртки* (envelope) этого потока, подчёркивая тем самым, что она накладывает ограничение на объём передаваемых им данных.

2.3.5 Основные понятия Min-Plus алгебры

Одна из основных идей сетевого исчисления заключается в преобразовании систем обработчиков к модели, удобной для аналитического исследования, с помощью min-plus алгебры. Вместо операций сложения и умножения, которые являются основными операциями традиционной алгебры, min-plus алгебра использует операции поточечного минимума (min) и сложения (plus). Полученная таким образом тройка $\langle F, min, plus \rangle$ образует идемпотентное полукольцо (dioid).

В линейной алгебре свёрткой двух функций f и g называется выражение:

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(s)g(t-s)ds$$

Тогда с учётом указанной замены операций сложения и умножения, в рамках min-plus алгебры *свёрткой* (convolution) функций f и g будет называться такая операция $\otimes$, что:

$$(f \otimes g)(t) = \inf_{0 \le s \le t} \{f(s) + g(t-s)\}$$

Свёртка позволяет сократить запись ограничений из определений для кривых нагрузки и сервиса:

$$\forall t, s \le t: \ A(t) - A(s) \le \alpha(t-s) \iff A = A \otimes \alpha$$

$$\forall t: \ D(t) \ge \inf_{0 \le s \le t} \{A(s) + \beta(t-s)\} \iff D \ge A \otimes \beta$$

Как это демонстрирует следующая теорема, указанный способ записи особенно удобен при анализе потоков, передающихся через последовательно соединённые обработчики.

Теорема (Композиция обработчиков)**.** Пусть два обработчика S_1 и S_2 образуют тандем, то есть, они соединены таким образом, что поток данных, исходящий из первого обработчика, попадает на обработчик S_2. При этом обработчику с номером $i \in \{1, 2\}$ соответствует кривая сервиса β_i. Тогда указанному тандему соответствует кривая сервиса $\beta = \beta_1 \otimes \beta_2$.

Доказательство. Заявленное свойство получается с помощью применения определения кривой сервиса к функциям отправки D_1 и D_2 данных, исходящих из S_1 и S_2, и равенства между функцией отправки D_1 и функцией прибытия A_2:

$$D_2 \ge A_2 \otimes \beta_2 = D_1 \otimes \beta_2 \ge (A_1 \otimes \beta_1) \otimes \beta_2 = A_1 \otimes (\beta_1 \otimes \beta_2) = A_1 \otimes \beta$$

□

Обратной свёрткой (deconvolution) называется такая операция $\oslash$, которая позволяет вычислить один аргумент f свёртки $w = f \otimes g$ по её значению и другому аргументу g. В алгебре min-plus формула вычисления обратной свёртки имеет вид:

$$(w \oslash g)(t) = \sup_{t \geq u \geq 0} \{w(t+u) - g(u)\}$$

Введение обратной свёртки позволяет более компактно переписать полученные ранее оценки для отставания, задержки и ограничения выходного потока. Пусть обработчик с кривой сервиса $\beta \in \mathcal{F}$ обслуживает поток данных, ограниченный кривой нагрузки $\alpha \in \mathcal{F}$. Тогда для отставания $b(t)$ выполнена формула:

$$b(t) \leq v(\alpha, \beta) = \sup_{s \geq 0} \{\alpha(s) - \beta(s)\} = (\alpha \oslash \beta)(0)$$

При дисциплине FIFO для задержки выполняется неравенство:

$$d(t) \leq h(\alpha, \beta) = \sup_{t} \{\inf \{\tau \geq 0 | \alpha(t) \leq \beta(t+\tau)\}\}$$

$$d(t) \leq \inf \left\{ \tau \geq 0 \middle| \sup_{t} \{\alpha(-\tau + t) - \beta(t)\} \leq 0 \right\} = \inf \{\tau \geq 0 | (\alpha \oslash \beta)(-\tau) \leq 0\}$$

Поток данных, передающихся обработчиком, ограничен кривой прибытия $\alpha'(t)$:

$$\alpha'(t) = \sup_{u \geq 0} \{\alpha(t+u) - \beta(u)\} = (\alpha \oslash \beta)(t)$$

Пусть символ $\odot$ обозначает одну из операций: свёртку $\otimes$ или обратную свёртку $\oslash$. Для каждой из них справедливы следующие свойства:

1. Коммутативность: $f \odot g = g \odot f$;
2. Ассоциативность: $(f \odot g) \odot h = f \odot (g \odot h)$;
3. Дистрибутивность по min: $\min(f, g) \odot h = \min(f \odot h, g \odot h)$

2.3.6 Регуляторы

Основные результаты сетевого исчисления справедливы для обработчиков и потоков данных, кривые сервиса и нагрузки которых описаны произвольными функциями из множества $\mathcal{F}$. В то же время для решения множества практических задач на множество $\mathcal{F}$ может быть удобным наложить дополнительные ограничения Ψ и аппроксимировать оригинальные кривые функциями из множества $G = \{f \in \mathcal{F} | \Psi \vdash f\}$. В частности, кусочно-линейные функции удобны при описании работы алгоритма «текущего ведра».

Рассмотрим указанный алгоритм подробнее. Пусть обработчик S использует алгоритм «текущего ведра» для шейпинга трафика. Внутри S находится контейнер объёма σ, который наполняется маркерами с постоянной скоростью ρ. Если контейнер заполнен, то вновь поступающие в него маркеры сбрасываются. Получаемые шейпером S данные сохраняются в его внутреннем буфере. В каждый момент времени шейпер S пересылает порцию данных, объём которой равен минимуму из объёма накопленных в контейнере маркеров и объёма данных, запасённых в его буфере.

В любой период отставания, когда в буфере шейпера S находятся данные, скорость передачи данных не уступает скорости ρ поступления маркеров в контейнер. Таким образом, шейпер можно описать строгой кривой сервиса $\beta_s(t) = [\rho t]^+ \in \mathcal{F}$:

$$\forall t: \; D(t) - D(SBP(t)) \geq \beta_s(t - SBP(t))$$

В качестве обычной кривой сервиса здесь подходит функция $\beta = [\rho t + \sigma]^+$:

$$D(t) = \inf_{s \leq t} \{A(s) + \beta(t - s)\} \tag{2.1}$$

Заметим, что для каждого значения s график функции $f_s(t) = A(s) + \beta(t - s)$ под знаком inf строится путём сдвига графика кривой сервиса β на s единиц вправо и на $A(s)$ единиц вверх. Множество всех функций f_s может быть построено с помощью дублирования графика функции β во всевозможные системы координат, полученных путём сдвига начала координат оригинальной системы в произвольную точку графика функции A. Формула вычисления функции отправки позволяет заключить, что график, совпадающий с нижней гранью построенного множества точек, и задаёт функцию D.

Рассмотрим рисунок 2.4. Его элементы a) и b) изображают график функции $A \in \mathcal{F}$ прибытия данных на шейпер S, и соответствующие ему строгую и обычную кривую сервиса: $\beta_s(t) = [\rho t]^+$ и $\beta = [\rho t + \sigma]^+$. Элементы c) и d) построены оценки для функции $D \in \mathcal{F}$ передачи данных, полученные при использовании строгой β_s и обычной β кривой сервиса. Оценка для функции отправки D, полученная с помощью строгой кривой сервиса β_s проигрывает в точности оценке, полученной с помощью обычной кривой сервиса β. Поскольку производительность шейпера не зависит от времени поступления данных, то полученная с помощью кривой сервиса β оценка функции отправки D – точная.

Рассмотрим элемент c) подробнее. Согласно функции A, первая порция данных поступает на шейпер с постоянной скоростью в течение периода $[t_1; t_3]$, причём к моменту t_1 контейнер с маркерами целиком заполнен. Так как скорость поступления пакетов превышает ρ, то число маркеров в контейнере постепенно уменьшается, пока контейнер не

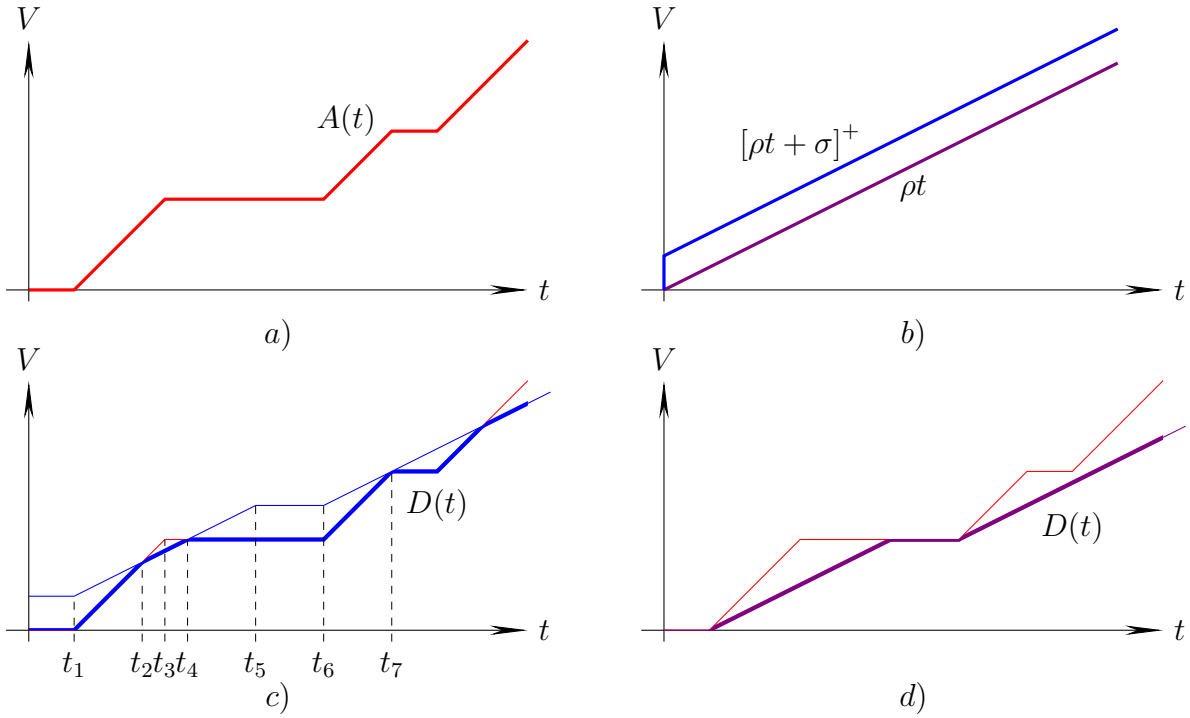

Рисунок 2.4: Построение функции отправки $D \in \mathcal{F}$, поставленной в соответствие функции прибытия $A \in \mathcal{F}$ шейпером, работающим по алгоритму “текущего ведра”.

станет пустым в момент времени t_2. При этом скорость передачи данных падает до ρ, начинается период отставания $[t_2; t_4]$, и часть полученных данных начинает запасаться в буфере шейпера. К моменту t_4 буфер данных опустошается, и количество маркеров в контейнере начинает расти, пока их объём не достигнет размера контейнера σ. Числу маркеров прошедших через контейнер соответствует функция $M \in \mathcal{F}$, график которой изображён тонкой сплошной линией.

Регулятором называется обработчик S, для которого может быть построена такая функция обёртки (envelope) $E \in \mathcal{F}$, что вне зависимости от функции прибытия A обслуживаемого им потока, объём данных, переданный им за каждый интервал длины u, не превышает величины $\alpha(u)$.

Рассмотрим комбинацию из регулятора с функцией обёртки $E \in \mathcal{F}$ и обработчика S. В указанной конфигурации поток данных A, поступающий на обработчик S и исходящий из регулятора, ограничен функцией обёртки: $A(t) - A(s) \leq E(t-s)$. Таким образом, функция обёртки $E \in \mathcal{F}$ регулятора одновременно является и кривой нагрузки для потока данных A, поступающего на присоединённый к его выходу обработчик.

2.3.7 RL-обработчики

Рассмотрим функцию $\delta_d \in \mathcal{F}$, которая принимает нулевое значение при $t \leq d$ и равна плюс бесконечности при $t > d$. Функцию δ_d часто называют *импульсом* или *сдвигом*, так как график её свёртки $w = f \otimes \delta_d$ с произвольной функцией $f \in \mathcal{F}$ получается сдвигом графика f на d единиц вправо.

$$\delta_d(t) = \Big\{0, t \leq d, +\infty, t > d.$$

$$(f \otimes \delta_d)(t) = \inf_{s \leq t} \{f(t-s) + \delta_d(s)\} = f(t-d)^+$$

RL-обработчиком (rate latency server) называется такой обработчик S, который передаёт поступающие на него данные с постоянной скоростью R и постоянным временем отклика T. Всякий RL-обработчик имеет кривую сервиса вида $\beta = R(t-T)^+$. Заметим, что функция сдвига позволяет переписать её в виде $\beta = Rt \otimes \delta_T$.

Пусть пара RL-обработчиков S_1 и S_2 с кривыми сервиса $\beta_i = R_i(t-T_i)^+$ образуют тандем S_1S_2. Тогда кривая сервиса β тандема S_1S_2 может быть вычислена по формуле:

$$\beta = \beta_1 \otimes \beta_2 = (R_1 t \otimes \delta_{T_1}) \otimes (R_2 t \otimes \delta_{T_2}) = (R_1 t \otimes R_2 t) \otimes (\delta_{T_1} \otimes \delta_{T_2}) =$$
$$\inf_s \{R_1 s + R_2(t-s)\} \otimes \delta_{T_1+T_2} = \min(R_1, R_2)t \otimes \delta_{T_1+T_2}$$

Таким образом, композиция двух RL-обработчиков β_1 и β_2 является RL-обработчиком с фиксированным временем отклика $T = T_1 + T_2$ и постоянной скоростью передачи данных $R = \min(R_1, R_2)$. Расчёт общей кривой сервиса для системы, состоящей из произвольного числа последовательно соединённых RL-обработчиков, производится в полной аналогии:

$$\beta = \beta_1 \otimes \beta_2 \otimes \cdots \otimes \beta_n = \min(R_1, R_2, \ldots, R_n)t \otimes \delta_{T_1+T_2+\cdots+T_n}$$

Прямым следствием является и обратное суждение. Тандем из двух RL-обработчиков S_1 и S_2 позволяет вычислить кривую сервиса $\beta_1 \in \mathcal{F}$ первого обработчика S_1 по общей кривой сервиса для их композиции $\beta = R(t-T)^+$ и кривой сервиса $\beta_2 = R_2(t-T_2)^+$ другого обработчика S_2:

$$\beta_1 = \min(R, R_2)(t - (T - T_2))^+$$

Аналогично строится и более общий случай из нескольких RL-обработчиков:

$$\beta_1 = \beta \oslash (\beta_2 \otimes \cdots \otimes \beta_n) = \min(R, R_2, \ldots, R_n)t \otimes \delta_{T-(T_2+\cdots+T_n)}$$

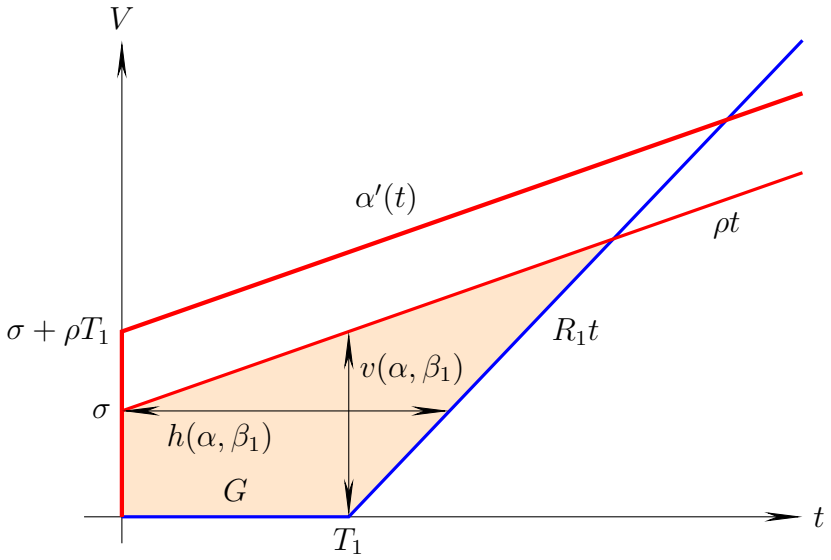

Рисунок 2.5: Вычисление верхних оценок для отставания $b(t)$, задержки $d(t)$ и кривой прибытия $\alpha'(t)$ выходного потока, ограниченного функций $\alpha(t) = \rho t + \sigma$ и RL-обработчика с кривой сервиса $\beta(t) = R(t - T)^+$.

2.3.8 Эффективная композиция обработчиков: принцип PBOO

Одним из базовых в сетевого исчисления является принцип «плати за всплеск один раз» (Pay Burst Only Once, PBOO), определяющий правила эффективной композиции обработчиков. В данном разделе этот принцип проиллюстрирован на примере тандема из двух RL-обработчиков S_i с кривыми сервиса $\beta_i = R_i(t - T_i)^+$, где $i \in \{1, 2\}$. Через тандем передаётся поток данных, ограниченный кривой прибытия $\alpha(t) = \rho t + \sigma$, такой что $\rho < \min(R_1, R_2)$, то есть в сети не возникает перегрузки. Вычислим верхнюю границу задержки $d(t)$ для передачи данных через указанную систему двумя способами:

1. Путём суммирования оценок для каждого обработчика:

$$D_{sum} = h(\alpha, \beta_1) + h(\alpha \otimes \beta_1, \beta_2)$$

2. С помощью кривой сервиса для тандема, рассчитанной заранее:

$$D_{pre} = h(\alpha, \beta_1 \otimes \beta_2)$$

Рассмотрим сначала первый случай. Приведём пример вычисления задержки передачи данных графическим методом. Поскольку при $t \in \mathbb{R}_{\geq 0}$ кривая прибытия $\alpha(t) = \rho t + \sigma$ является вогнутой, а кривая сервиса первого обработчика тандема $\beta_1(t) = R_1(t - T_1)^+$ – выпуклой, то заключённая между ними область координатной плоскости

$G = \{(x, y)|\beta_1(x) \leq y \leq \alpha(x)\}$ является выпуклой фигурой (см. рисунок 2.5). Таким образом, горизонтальное $h(\alpha, \beta_1)$ и вертикальное $v(\alpha, \beta_1)$ отклонения между этими кривыми равны наибольшим длинам сечений фигуры G в соответствующих направлениях. Из выпуклости фигуры G и кусочно-линейного характера функции α и β_1 следует, что её самые длинные сечения проходит хотя бы через одну из точек излома её границы.

Как это продемонстрировано на рисунке 2.5, наибольшее горизонтальное расстояние $h(\alpha, \beta)$ между кривыми α и β_1 достигается на горизонтальном сечении через точку $(0, \sigma)$. Наибольшее вертикальное расстояние $v(\alpha, \beta)$ – на вертикальном сечении через точку $(T_1, 0)$. Таким образом, задержка передачи при прохождении потоком RL-обработчика S_1 ограничена следующим выражением:

$$D_1 = h(\alpha, \beta_1) = \frac{\sigma}{R_1} + T_1$$

В соответствии с теоремой об ограничении выходного потока, поток данных, передаваемый обработчиком S_1 на обработчик S_2, ограничен кривой прибытия $\alpha' = \alpha \oslash \beta_1$. Построим формулу расчёта α' для кривых α и β_1 заданного вида:

$$\alpha'(t) = \alpha \oslash \beta_1 = \sup_{u \geq 0} \{\alpha(t + u) - \beta_1(u)\}$$

$$\alpha'(t) = \max\left(\sup_{0 \leq u \leq T_1} \{\rho(t + u) + \sigma\}, \sup_{u > T_1} \{\rho(t + u) + \sigma - R_1(u - T_1)\}\right)$$

$$\alpha'(t) = \max\left(\rho(t + T_1) + \sigma, \sup_{u > T_1} \{\rho t + \sigma + R_1 T_1 - u(R_1 - \rho)\}\right)$$

$$\alpha'(t) = \rho(t + T_1) + \sigma = \rho t + (\sigma + \rho T_1)$$

Расчёты максимального горизонтального отклонения для кривой прибытия α' и кривой сервиса β_2 производятся в полной аналогии с вычислениями, проведёнными для оценки задержки прохождения данных через элемент S_1. Таким образом, задержка прохождения данных через элемент S_2 ограничена выражением:

$$D_2 = h(\alpha \oslash \beta_1, \beta_2) = h(\sigma + \rho(t + T_1), \beta_2) = \frac{\sigma + \rho T_1}{R_2} + T_2$$

$$D_{sum} = D_1 + D_2 = \frac{\sigma}{R_1} + \frac{\sigma + \rho T_1}{R_2 + T_1 + T_2}$$

Теперь произведём оценку задержки с помощью предварительного расчёта общей кривой сервиса для тандема в соответствии с полученной ранее формулой композиции последовательности из RL-обработчиков:

$$D_{pre} = h(\rho t + \sigma, \beta_1 \otimes \beta_2) = \frac{\sigma}{\min(R_1, R_2)} + T_1 + T_2$$

Оценка D_{sum} задержки передачи данных, полученная с помощью бесхитростного суммирования, проигрывает в точности оценке D_{pre}, полученной в результате предварительного вычисления общей кривой сервиса для всего тандема. Причины такой разницы заключаются в том, что поэлементная схема оценки задержки учитывает всплеск проходящего через систему потока на каждом сервисном элементе. В формуле D_{sum} эта особенность представлена слагаемыми $\frac{\sigma}{R_1}$ и $\frac{\sigma}{R_2}$. Кроме того, при расчёте задержки прохождения данных через второй элемент поэлементная схема учитывает добавочный всплеск $\frac{\rho T_1}{R_2}$, который поток приобрёл при прохождения через первый обработчик. Теорема о композиции элементов позволяет избавиться от указанных изъянов, заплатив за всплеск потока единственный раз.

2.3.9 Моделирование ПКС

Сетевые исчисления позволяют представить каждый из коммутаторов сети в виде конвейера из обработчиков. В простейшем случае все компоненты сети могут быть представлены функциями из двух линейных сегментов. Внутренние компоненты коммутатора могут передавать друг другу пакеты с очень высокой скоростью. Они фактически не ограничены в пропускной способности и могут моделироваться импульсной функцией δ_d, где d – наибольшее время, которое пакет может находиться внутри обработчика. Внешние компоненты коммутатора, которые осуществляют непосредственную пересылку данных через физические линии должны дополнительно учитывать ограничения пропускной способностью этих линий. Поэтому моделирующие их поведение простейшие кривые сервиса могут иметь вид $\beta = R(t - T)$. Что касается ограничения активности потоков данных, то протокол OpenFlow позволяет устанавливать инструкции для шейпинга каждого из передающихся через сеть потоков по алгоритму «текущего ведра». В результате, каждый поток передачи данных можно моделировать функцией вида $\alpha = \rho t + \sigma$.

Таким образом, теория сетевого исчисления хорошо подходит для анализа ПКС сетей. Контроллер ПКС может выяснить конфигурацию оборудования, построить соответствующую этой конфигурации сеть из обработчиков, ограничить пропускные способности всех поступающих в сеть потоков и использовать аппарат сетевого исчисления для получения высокоточных оценок сквозной задержки. Однако приведённые в данном разделе теоретические выкладки пригодны для применения лишь в тех ПКС, где потоки не пересекаются между собой и не конкурируют за ресурсы обработчиков. Такое предположение выполняется при использовании модели интегрированных сервисов. Однако абсолютное

большинство современных сетей этим предположениям не соответствует. Отсюда возникает необходимость в более продвинутых алгоритмов построения оценок задержки. Обзор соответствующих результатов из области сетевого исчисления приводится в следующем разделе.

2.4 Мультиплексирование нескольких потоков

Если обработчик S обслуживает не один, а сразу $n \in \mathbb{N}$ потоков, то он определяется перечислением множества функций прибытия поступающих на него потоков данных и соответствующих им функций отправки: $S \subseteq F^n \times F^n$. Проекция обработчика на поток с номером $i \in \ltimes, i \leq n$ называется *остаточным обработчиком* (residual/left-over server) $S_i \subseteq F \times F$ для этого потока. Отношения остаточных обработчиков зависит от правил распределения ресурсов обработчика S между соответствующими им потоками. Такие правила принято называть правилами/политиками мультиплексирования обработчика S. Наиболее известными из них являются политики FIFO [96], GPS, P-GPS/WFQ [15], статические приоритеты [97], Round-Robin и его модификации WRR/DWRR/SRR [98], слепое мультиплексирование [99].

Следующая теорема устанавливает схему вычисления кривой сервиса, определяющей остаточный обработчик S_i, обслуживающий поток с номером i, по строгой кривой сервиса, соответствующей суммарной производительности обработчика S, и кривым нагрузки для остальных проходящих через него потоков.

Теорема (Вычисление остаточного обработчика)**.** Пусть на обработчик S со строгой кривой сервиса β поступают два потока данных, один из которых ограничен кривой прибытия α_1. Тогда если функция $\beta_2 = [\beta - \alpha_1]^+$ принадлежит классу $\mathcal{F}$, то она является одной из кривых сервиса остаточного обработчика S_2, который обслуживает другой поток данных.

Доказательство. По определению строгой кривой сервиса, обработчик S должен передавать не менее $\beta(u)$ данных в течение каждого временного интервала длины u внутри каждого его периода отставания. Поскольку количество данных первого потока ограничено сверху кривой прибытия α_1, то за каждый такой интервал обработчик S способен передать не менее $[\beta(u) - \alpha_1(u)]^+$ данных второго потока, вне зависимости от его политики мультиплексирования.

□

Замечание. Если вместо строгой кривой сервиса использовать нестрогую кривую, то результат теоремы, вообще говоря, не достигается, что можно продемонстрировать на следующем примере. Пусть обработчик S способен передавать один пакет данных в единицу времени. Обработчик получает пакеты двух потоков: F_1 и F_2. Поток F_1 присылает пакет в момент времени $t = 0$, и замолкает. Поток F_2 присылает свои пакеты в каждый момент $t \in \mathbb{N}$. Если обработчик S обладает нулевой задержкой и его политика мультиплексирования такова, что он обслуживает пакеты лишь второго потока, то функции получения и отправки для потоков F_1, F_2 и $F = F_1 + F_2$ имеют вид:

$$A_1(t) = \min(1, t) \quad A_2(t) = (t-1)^+ \qquad A(t) = t$$
$$D_1(t) = 0 \qquad D_2(t) = R_1(t) \qquad D(t) = t - 1$$

Таким образом, обработчик S обслуживает поступающий на него агрегированный поток данных в соответствии с нестрогой кривой сервиса $\beta = \delta_1(t)$. Если бы приведённая теорема выполнялась для нестрогой кривой сервиса, то поток F_1 получал бы $\beta_1 = [\delta_1(t) - t]^+$ ресурсов обработчика S, что заведомо неверно, так как S не обслуживал его пакеты.

Замечание. Оценка кривой сервиса для остаточного обработчика S_i, полученная с помощью доказанной теоремы, достигается лишь в том случае, когда породивший его обработчик S обслуживает соответствующий поток данных в последнюю очередь. Указанное условие выполняется, например, для потока с наименьшим приоритетом при мультиплексировании на основе статических приоритетов, или же при слепом мультиплексировании, когда обработчик волен обслуживать поступающие на него потоки в любом, в том числе и самом неудачном для каждого из них, порядке. На практике политики мультиплексирования пытаются распределять ресурсы обработчика между множеством поступающих на него потоков более равномерно.

Пусть поток данных представляет собой множество пакетов, пронумерованных в порядке их прибытия на сервисный элемент S. Пусть $a_n \geq 0$, $d_n \geq 0$ и l_n обозначают время прибытия, время отправки и длину пакета с номером n. Всякий обработчик S называется GR-обработчиком (guaranteed rate node) со скоростью r и временем отклика e для потока R, если для этого обработчика справедливы соотношения:

$$d_n \leq f_n + e \begin{cases} f_0 = 0 \\ f_n = \max\left(a_n, f_(n-1)\right) + \frac{l_n}{r} \end{cases}$$

Класс GR-обработчиков охватывает различные обработчики потоков с циклическими планировщиками, обработчики пакетов по дисциплине FIFO, обработчик потока с наивысшим приоритетом для обработчиков со статическими приоритетами.

Если RL-обработчику S соответствует минимальная кривая сервиса вида $\beta_{R,T} = R(t-T)^+$, и он обслуживает потоки данных по дисциплине FIFO, тогда он является GR обработчиком со скоростью R и задержкой T. Если обработчику S соответствует строгая кривая сервиса вида $\beta_{R,T}$, то он всегда является GR-обработчиком. Изучению взаимосвязей между различными классами обработчиков потоков посвящена, в частности, работа [100].

2.4.1 Индивидуальное и агрегированное планирование

Сетевые исчисления предлагают две альтернативных возможности оценки задержки передачи данных при их передаче через сеть из обработчиков: *индивидуальное* (per-flow) и *агрегированное* (aggregate) планирование (scheduling).

Индивидуальное планирование предполагает, что каждый обработчик сети способен разделять свои ресурсы между проходящими через него потоками таким образом, что они не будут оказывать друг на друга влияния. Иначе говоря, каждый обработчик должен гарантировать каждому передающемуся через него потоку обслуживание с определённой остаточной кривой сервиса, которая может быть построена в явном виде вне зависимости от остальных потоков. На практике индивидуальному планированию соответствует модель интегрированных сервисов (IntServ) [71], которая позволяет гарантировать надлежащее качество обслуживания потоков благодаря резервированию ресурсов коммутатора под конкретный поток.

В предположениях индивидуального планирования последовательная композиция обработчиков сети вдоль маршрута передачи потока позволяет свести задачу оценки сквозного качества обслуживания к задаче оценки качества при прохождении потока через единственный обработчик с заданной кривой сервиса. Её решение тривиально.

Агрегированное планирование наделяет обработчики значительно большей степенью автономности. Теперь каждый обработчик определяет несколько классов сервиса вне зависимости от проходящих через него потоков. При этом различий между данными потоков одного класса обслуживания не делается. Указанный подход предлагается, например, моделью дифференцированных ресурсов (DiffServ) [84].

Использование агрегированного планирования значительно усложняет применение сетевого исчисления по сравнению со случаем индивидуального планирования. При агрегированном планировании качество обслуживания каждого потока заведомо зависит от интенсивности остальных потоков того же класса обслуживания. Поэтому прямое приме-

нение теоремы о композиции здесь невозможно. К тому же сервисные элементы обычно не простаивают при наличии нагрузки (work-conserving). Их планировщики переходят к обработке следующего класса потоков, если нагрузка текущего класса ниже заложенной. Поэтому качество обслуживания каждого потока зависит ещё и от множества потоков других классов. Соответственно, кривая сервиса каждого остаточного обработчика априорно зависит от кривых прибытия множества поступающих на него потоков.

Для построения кривой сервиса обработчика необходимо знать кривые прибытия каждого из поступающих на него потоков. Такие кривые обычно известны лишь для поступающих в сеть потоков. Поэтому для вычислений необходимо дополнительное вычисление кривых прибытия для каждого потока и каждого пересекаемого им обработчика. При этом вычисление кривых прибытия требует знания кривых сервиса для обработчиков, через которые они передаются. Вычисление кривых сервиса требует знания кривых прибытия других поступающих на него потоков, и так далее. Тем самым, агрегированное планирование порождает транзитивные зависимости между потоками данных. Поэтому качество обслуживания одного потока может зависеть, в том числе, и от множества потоков, которые никогда не пересекали его маршрут.

Характерные для агрегированного планирования транзитивные зависимости между потоками могут порождать циклы, препятствующие построению сколь-нибудь точных оценок для длительности передачи данных. Такие сети называются нестабильными – они допускают бесконечный рост количества данных в буферах обработчиков и, как следствие, неограниченную задержку при передаче данных [14].

2.4.2 Стабильность при агрегированном планировании

Причина невозможности оценить качество обслуживания потоков внутри некоторых сетей обработчиков при агрегированном планировании кроется в циклических зависимостях между кривыми прибытия этих потоков. Сети без циклических зависимостей называются сетями с прямой связью (feed-forward networks) и заведомо стабильны. Если изучаемая сеть не относится к указанному классу, то существует несколько возможностей её к нему приведения. В простейшем случае граф передающихся через сеть потоков может быть приведён к одному из соответствующих ему остовных деревьев.

Результат сведения графа обработчиков к основному дереву отличается от результата применения протокола STP в современных компьютерных сетях, так как последний допускает двунаправленную передачу данных через каждый канал сети. Кроме того, протокол

STP не пытается уменьшить количество потерянных при расщеплении циклов ресурсов сети, и часто может быть неэффективен. В области сетевого исчисления разработаны и более эффективные алгоритмы разрывы циклов. Например, алгоритм Turn-Prohibition [101] предлагает запрещать не целые каналы передачи данных, а последовательности из двух таких каналов. Алгоритм пытается минимизировать количество неиспользуемых каналов и гарантирует, что число запрещённых каналов не превзойдёт одной трети от их общего числа.

При некоторых ограничениях сетевое исчисление применимо и к сетям с топологией, содержащей циклы [102]. Рассмотрим следующую модель сети обработчиков:

1. Сеть состоит из обработчиков, определяющих несколько классов обслуживания. Причём каждый обработчик с номером m является GR-обработчиком со скоростью r_m и задержкой e_m для агрегированного потока наивысшего класса обслуживания. Максимальная задержка e обработки данных задаётся формулой $e = \max_m \{e_m\}$;

2. В рамках модели рассматриваются только потоки наивысшего класса обслуживания. Каждый поток с номером i, поступающий в сеть, изначально ограничен кривой прибытия $\alpha_i = \rho_i t + \sigma_i$, и пересекает не более $h \in \mathbb{N}$ узлов сети;

3. Утилизацией u_m обработчика с номером m называется отношение суммы максимальных скоростей проходящих через него потоков к скорости их обработки $u_m = \frac{1}{r_m} \sum_{i \in m} \rho_i$. Символ $u = \max_m \{u_m\}$ обозначает максимальную утилизацию;

4. Отношение всплеска проходящего через обработчик m агрегированного потока наивысшего класса сервиса к скорости r_m его обработки $\tau_m = \frac{1}{r_m} \sum_{i \ inm} \rho_i$ называется нормализованным всплеском, $\tau = \max_m \{\tau_m\}$;

5. Максимальный размер передаваемого через сеть пакета равен $L_{\max}$.

Теорема (О стабильности при ограничении утилизации.)**.** В заданных предположениях из ограничения максимальной степени утилизации $u \leq \frac{1}{h-1}$ следует:

1. Максимальная задержка, которую будет испытывать передаваемый через сеть поток наивысшего класса сервиса, ограничена сверху значением hD;

2. Минимальный размер буфера обработчика m, необходимый для передачи всех поступивших в сеть пакетов, ограничен снизу значением $r_m D + L_m ax$;

$$D = \frac{e + \tau}{1 - (h-1)u}$$

Теорема (О критической утилизации.)**.** Если максимальная утилизация u превышает предельное значение $\frac{1}{h-1}$, тогда для любого наперёд заданного числа $D' > 0$ может быть построена сеть с максимальной задержкой передачи не меньшей D'.

Указанная теорема позволяет заключить, что, если верхняя оценка задержки для сетей общего вида и может быть построена, то перечисленных ограничений недостаточно для её вычисления. Возможно, подобная оценка зависит от размера или топологии сети обработчиков [102; 103]. Например, известны некоторые классы сетей, которые не являются сетями с прямой связью, но стабильны при утилизации, превышающей полученные оценки для сетей произвольного вида. Свойством стабильности обладает, в частности, однонаправленное кольцо [15].

В продолжении раздела поочерёдно рассматривается несколько различных алгоритмов вычисления задержки в сетях с прямой связью, реализующих агрегированное планирование, которые были представлены в статье [99]: анализ суммарного потока, анализ отдельного потока и анализ пересечения потоков.

2.4.3 Анализ Суммарного Потока (Total Flow Analysis)

Пусть задан тандем из двух последовательно соединённых обработчиков S_j, где $j = \{1, 2\}$, со строгими кривыми сервиса $\beta_j = R_j(t - T_j)^+$. Обработчики обслуживают каждый из проходящих потоков по дисциплине FIFO, однако их алгоритмы мультиплексирования потоков неизвестны. Через описанную систему обработчиков передаётся два потока данных $i = \{1, 2\}$ с функциями прибытия A_i^0, удовлетворяющими кривым нагрузки $\alpha_i^0 = \rho_i t + \sigma_i$. Функцию прибытия и кривую нагрузки потока i после его прохождения через обработчик с номером j будем обозначать через A_i^j и α_i^j соответственно. Построим верхнюю оценку задержки передачи данных для потока $i = 1$.

В зависимости от своего алгоритма мультиплексирования обработчик S_1 может обслуживать поступающие на него данных A_1^0 и A_2^0 в произвольном порядке в пределах текущего периода отставания (гарантируется лишь дисциплина FIFO для данных каждого потока). Таким образом, задержка $d_i^1(t)$ обслуживания каждого из этих потоков $i = \{1, 2\}$ на S_1 может быть ограничена длительностью наибольшего периода отставания при обслуживании их суммарного потока $A_1^0 + A_2^0$:

$$d_i^1(t) \leq \Delta_1 = \inf\left\{\tau \geq 0 \middle| (\alpha_1^0 + \alpha_2^0)(\tau) = \beta_1(\tau)\right\}$$

Аналогичная оценка задержки справедлива и для потоков данных, поступающих на S_2. При этом сумма кривых нагрузки $\alpha_1^1 + \alpha_2^1$, после прохождения потоками обработчика

S_1 может быть вычислена из кривых нагрузки α_1^0 и α_2^0 по теореме 3 (об оценке выходного потока):

$$d_i^2(t) \le \Delta_2 = \inf\left\{\tau \le 0 \middle| (\alpha_1^0 \oslash \beta_1 + \alpha_2^0 \oslash \beta_1)(\tau) = \beta_2(\tau)\right\}$$

Дистрибутивность обратной свёртки относительно операции min позволяет упростить вычисления, если потоки, как и в указанном примере, проходят через одну и ту же последовательность обработчиков:

$$\Delta_2 = \inf\left\{\tau \ge 0 \middle| \left((\alpha_1^0 + \alpha_2^0) \oslash \beta_1\right)(\tau) = \beta_2(\tau)\right\}$$

Верхняя оценка задержки потока через систему обработчиков может быть получена суммированием оценок задержки для каждого из элементов вдоль его маршрута:

$$d^{TFA} = \Delta_1 + \Delta_2$$

Проведём указанные вычисления для рассмотренной сети:

$$(\alpha_1^0 + \alpha_2^0)(\Delta_1) = \beta_1(\Delta_1)$$

$$(\rho_1 + \rho_2)\Delta_1 + \sigma_1 + \sigma_2 = R_1(\Delta_1 - T_1)^+$$

$$\Delta_1 = \frac{R_1 T_1 + \sigma_1 + \sigma_2}{R_1 - \rho_1 - \rho_2} = T_1 + \frac{T_1(\rho_1 + \rho_2) + \sigma_1 + \sigma_2}{R_1 - \rho_1 - \rho_2}$$

$$\left((\alpha_1^0 + \alpha_2^0) \oslash \beta_1\right)(\Delta_2) = \beta_2(\Delta_2)$$

$$\sup_{u \ge 0}\left\{(\rho_1 + \rho_2)(\Delta_2 + u) + \sigma_1 + \sigma_2 - R_1(u - T_1)^+\right\} = R_2(\Delta_2 - T_2)^+$$

$$(\rho_1 + \rho_2)(\Delta_2 + T_1) + \sigma_1 + \sigma_2 = R_2(\Delta_2 - T_2)^+$$

$$\Delta_2 = \frac{R_2 T_2 + T_1(\rho_1 + \rho_2) + \sigma_1 + \sigma_2}{R_2 - \rho_1 - \rho_2} = T_2 + \frac{(T_2 + T_1)(\rho_1 + \rho_2) + \sigma_1 + \sigma_2}{R_2 - \rho_1 - \rho_2}$$

$$d^{TFA} = T_1 + T_2 + \frac{T_1(\rho_1 + \rho_2) + \sigma_1 + \sigma_2}{R_1 - \rho_1 - \rho_2} + \frac{(T_2 + T_1)(\rho_1 + \rho_2) + \sigma_1 + \sigma_2}{R_2 - \rho_1 - \rho_2}$$

2.4.4 Анализ Отдельного Потока (Separated Flow Analysis)

Альтернативный способ вычисления задержки можно построить путём отделения целевого потока от остального трафика и применения теоремы о композиции обработчиков. При этом потоки данных, следующие по одному маршруту, как и в методе TFA, разделяются на группы, однако теперь целевой поток вычленяется в отдельную группу.

Задержка на каждом обработчике строится индивидуально для каждого потока. Так как данные каждого потока передаются по принципу FIFO, то для задержки может быть дана следующая оценка:

$$d^{SFA} = h(\alpha_1, [\beta_1 - \alpha_2]^+ \otimes [\beta_2 - (\alpha_2 \oslash \beta_1)]^+)$$

$$\beta_2 - (\alpha_2 \oslash \beta_1) = R_2(t - T_2) - (\rho_2 t + \rho_2 T_1 + \sigma_1) = t(R_2 - \rho_2) - R_2 T_2 - \rho_2 T_1 - \sigma_2$$

Сначала вычислим значение свёртки:

$$\begin{aligned}
\mathcal{C}(s) = &[\beta_1 - \alpha_2]^+ \otimes [\beta_2 - (\alpha_2 \oslash \beta_1)]^+ = \\
&[s(R_1 - \rho_2) - R_1 T_1 - \sigma_2]^+ + [(t - s)(R_2 - \rho_2) - R_2 T_2 - \rho_2 T_1 - \sigma_2]^+ = \\
&(R_1 - \rho_2)\left[s - \frac{R_1 T_1 + \sigma_2}{R_1 - \rho_2}\right]^+ + (R_2 - \rho_2)\left[t - s - \frac{R_2 T_2 - \rho_2 T_1 - \sigma_2}{R_2 - \rho_2}\right]^+ = \\
&(R_1 - \rho_2)\,[s - q_1]^+ + (R_2 - \rho_2)\,[q_2 - s]^+
\end{aligned}$$

$$q_1 = T_1 + \tfrac{\rho_2 T_1 + \sigma_2}{R_1 - \rho_2} \quad q_2 = t - \tfrac{T_2 + \rho_2(T_1 + T_2) + \sigma_2}{R_2 - \rho_2}$$

Рассмотрим значения s на каждом интервале, полученном с помощью разделения координатной оси в точках q_1 и q_2. При $s \le q_1$ обнуляется первое слагаемое $\mathcal{C}(s)$, при $s \ge q_2$ – второе. Сначала рассмотрим множество случаев, когда $q_1 \le q_2$:

$$\begin{aligned}
s \le q_1 \le q_2 \to \inf \mathcal{C}(s) &= \inf_{s \le q_1 \le q_2} \{(R_2 - \rho_2)(q_2 - s)\} = (R_2 - \rho_2)(q_2 - s) \\
q_1 \le s \le q_2 \to \inf \mathcal{C}(s) &= \inf_{q_1 \le s \le q_2} \{(R_1 - \rho_2)(s - q_1) + (R_2 - \rho_2)(q_2 - s)\} = \\
&\inf_{q_1 \le s \le q_2} \{s(R_1 - R_2) - q_1(R_1 - \rho_2) + q_2(R_2 - \rho_2)\}
\end{aligned}$$

$$\inf \mathcal{C}(s) = \begin{cases}
\begin{aligned} &q_1(R_1 - R_2) - q_1(R_1 - \rho_2) + q_2(R_2 - \rho_2) = \\ &\quad q_2(R_2 - \rho_2) - q_1(R_2 - \rho_2) = (R_2 - \rho_2)(q_2 - q_1) \end{aligned} & \text{при } R_1 \ge R_2 \\
\begin{aligned} &q_2(R_1 - R_2) - q_1(R_1 - \rho_2) + q_2(R_2 - \rho_2) = \\ &\quad q_2(R_1 - \rho_2) - q_1(R_1 - \rho_2) = (R_1 - \rho_2)(q_2 - q_1) \end{aligned} & \text{при } R_1 < R_2
\end{cases}$$

$$q_1 \le q_2 \le s \to \inf \mathcal{C}(s) = \inf_{q_1 \le q_2 \le s} \{(R_1 - \rho_2)(s - q_1)\} = (R_1 - \rho_2)(q_2 - q_1)$$

Таким образом, при $q_1 \le q_2$ для свёртки выполняется следующее условие:

$$\inf \mathcal{C}(s) = (\min(R_1, R_2) - \rho_2)\,(q_2 - q_1)$$

Теперь рассмотрим случай $q_1 > q_2$:

$$\begin{aligned}
s < q_2 < q_1 \to \inf \mathcal{C}(s) &= \inf_{s < q_2 < q_1} \{(R_2 - \rho_2)(q_2 - s)\} = 0 \\
q_2 < q_1 < s \to \inf \mathcal{C}(s) &= \inf_{q_2 < q_1 < s} \{(R_1 - \rho_2)(s - q_1)\} = 0 \\
q_2 \le s \le q_1 \to \inf \mathcal{C}(s) &= 0
\end{aligned}$$

Все рассмотренные случаи могут быть обобщены в виде выражения:

$$\inf \mathcal{C}(s) = (\min(R_1, R_2) - \rho_2)\,(q_2 - q_1)^+$$

$$\inf \mathcal{C}(s) = (\min(R_1, R_2) - \rho_2)\left(t - \left(T_2 + \frac{\rho_2(T_1 + T_2) + \sigma_2}{R_2 - \rho_2}\right) - T_1 - \frac{\rho_2 T_1 + \sigma_2}{R_1 - \rho_2}\right)^+$$

Таким образом, для оценки методом SFA справедлива формула:

$$d^{SFA} = \sup_t \left\{ \inf_\tau \left\{ \tau \le 0 \middle| \rho_1 t + \sigma_1 \le (\min(R_1, R_2) - \rho_2)\,(q_2 - q_1 + \tau)^+ \right\} \right\}$$

$$\tau \ge \frac{\rho_1 t + \sigma_1}{(\min(R_1, R_2) - \rho_2} + q_1 - q_2$$

$$\tau \ge T_1 + T_2 + \frac{t\,(\rho_1 + \rho_2 - \min(R_1, R_2)) + \sigma_1}{\min(R_1, R_2) - \rho_2} + \frac{\rho_2 T_1 + \sigma_2}{R_1 - \rho_2} + \frac{\rho_2(T_1 + T_2) + \sigma_2}{R_2 - \rho_2}$$

$$d^{SFA} = T_1 + T_2 + \frac{\sigma_1}{(\min(R_1, R_2) - \rho_2} + \frac{\rho_2 T_1 + \sigma_2}{R_1 - \rho_2} + \frac{\rho_2(T_1 + T_2) + \sigma_2}{R_2 - \rho_2}$$

Анализ отдельного потока (SFA) даёт более точную оценку, чем метод TFA. Однако полученная в результате вычислений формула включает в себя всплеск σ_2 второго потока дважды, что противоречит принципу PMOO: «плати за мультиплексирование один раз».

2.4.5 Анализ пересечения потоков (PMOO)

Проблема метода анализа отдельного потока SFA заключается в порядке вычислений. Анализ отдельного потока принимает в расчёт каждый из мультиплексируемых потоков на каждом из обработчиков. В то же время, данные стороннего потока, проходящего по одному участку маршрута с целевым потоком, способны повлиять на него лишь при мультиплексировании на входе в общую последовательность обработчиков. Появляется понятное желание использовать теоремы о композиции и остаточном обработчике в обратном порядке и вычислить задержку по формуле:

$$d^{PMOO} = h(\alpha_1, [(\beta_1 \times \beta_2) - \alpha_2]^+ \tag{2.2}$$

Однако даже если кривые сервиса β_1 и β_2 являются строгими, кривая сервиса, полученная в результате их свёртки, вообще говоря, строгой не является. Поэтому к ней неприменима теорема об остаточном обработчике, и, как следствие, указанное соотношение в общем случае не выполняется. Тем не менее, указанное соотношение справедливо при построении композиции из кусочно-линейных выпуклых строгих кривых сервиса [99]. Таким образом, метод PMOO позволяет оценить задержку передачи данных через модельную сеть следующим образом:

$$[(\beta_1 \otimes \beta_2) - \alpha_2]^+ = [\min(R_1, R_2)(t - T_1 - T_2) - \rho_2 t - \sigma_2]^+$$

$$d^{PMOO} = \sup_t \inf_\tau \left\{ \tau \geq 0 \middle| \rho_1 t + \sigma_1 \leq [(t+\tau)(\min(R_1, R_2) - \rho_2) - \min(R_1, R_2)(T_1 + T_2) - \sigma_2]^+ \right\}$$

$$\tau \geq \frac{t\rho_1 + \min(R_1, R_2)(T_1 + T_2) + \sigma_1 + \sigma_2}{\min(R_1, R_2) - \rho_2} - t$$

$$\tau \geq T_1 + T_2 + \frac{t\,(\rho_1 + \rho_2 - \min(R_1, R_2)) + \rho_2(T_1 + T_2) + \sigma_1 + \sigma_2}{\min(R_1, R_2) - \rho_2}$$

$$d^{PMOO} = T_1 + T_2 + \frac{\rho_2(T_1 + T_2) + \sigma_1 + \sigma_2}{\min(R_1, R_2) - \rho_2}$$

Сравним полученное значение с оценкой метода SFA при $\sigma_2 = 0$ и $T_1 = 0$:

$$d^{SFA} = T_2 + \frac{\sigma_1}{\min(R_1, R_2) - \rho_2} + \frac{\rho_2 T_2}{R_2 - \rho_2}$$

$$d^{PMOO} = T_2 + \frac{\rho_2 T_2 + \sigma_1}{\min(R_1, R_2) - \rho_2}$$

$$d^{PMOO} - d^{SFA} = \rho_2 T_2 \left(\frac{1}{\min(R_1, R_2) - \rho_2} - \frac{1}{R_2 - \rho_2} \geq 0 \right)$$

Несмотря на то, что анализ пересечения потоков PMOO на практике обычно даёт лучшие оценки, чем анализ отдельного потока SFA, в некоторых случаях результат последнего может быть более точным. Таким образом, метод PMOO не может давать наилучшую оценку в общем случае.

Из анализа последней формулы для оценки задержки методом PMOO следует, что данный метод учитывает время отклика второго обработчика $\rho_2 T_2$ на обработчике с минимальной скоростью $\min(R_1, R_2)$. В то же время, данные потока могут испытывать такую задержку лишь на втором обработчике. Поэтому при скоростях обслуживания $R_1 < R_2$ указанная оценка заведомо не оптимальна.

2.5 Обобщение алгоритмов вычисления задержки

Представленные в статье [99] и кратко описанные в предыдущем разделе работы алгоритмы построения верхних оценок задержки для сетей с прямой связью, реализующих агрегированное планирование: TFA, SFA и PMOO – предполагают, что все кривые нагрузки и сервиса представляют собой кусочно-линейные функции, состоящие из не более, чем двух сегментов. Несмотря на то, что подобные предположения могут быть справедливы для достаточно широкого класса сетей, существуют такие конфигурации оборудования, для которых они не выполняются. Например, продвинутые средства шейпинга трафика

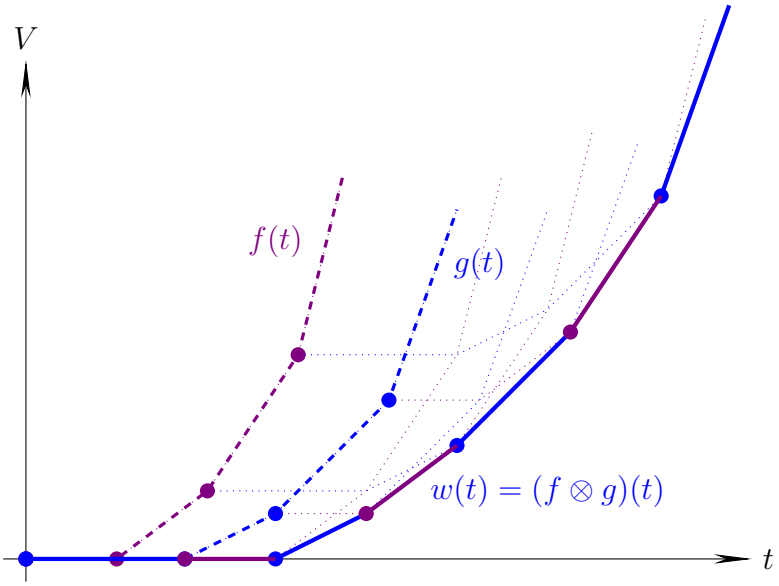

Рисунок 2.6: Построение графика функции Min-Plus свёртки $w(t) = (f \otimes g)(t)$ для пары выпуклых кусочно-линейных функций $w(t)$ и $g(t)$.

работают по двойному алгоритму текущего ведра (dual leaky bucket), который позволяет не только ограничить всплески активности потоков, но и предельные скорости поступления данных в течение указанных всплесков. Наиболее точное моделирование поведение такого средства требует использования кусочно-линейных функций, состоящих из трёх сегментов.

Основным препятствием на пути к расширению перечисленных алгоритмов на функции с большим числом сегментов является сложность вычисления на множестве таких функций операций прямой и обратной min-plus свёртки. В данном разделе приводится описание алгоритмов, позволяющих решить указанную проблему. Алгоритм вычисления обратной свёртки от вогнутой и выпуклой кусочно-линейных функций – оригинальный. Алгоритм вычисления свёртки двух выпуклых функций не является новым и приведён в работе потому, что доказательство оригинального алгоритма обратной свёртки осуществляется по аналогии.

2.5.1 Схемы вычисления функции свёртки

Рассмотренный ранее пример построения композиции двух RL-обработчиков является частным случаем следующей более общей теоремы.

Теорема (Свёртка выпуклых кусочно-линейных функций)**.** График функции свёртки для двух выпуклых кусочно-линейных функции $f \in \mathcal{F}$ и $g \in \mathcal{F}$ может быть построен путём

последовательной композиции составляющих эти функции отрезков в порядке увеличения значений их угловых коэффициентов.

Доказательство. Иллюстрация для предложенного условием теоремы метода приведена на рисунке 2.6. Обозначим множества точек плоскости, соответствующих графикам функций f и g через $S(f)$ и $S(g)$. Пусть графики функций f и g образуют $n = |f| + |g|$ линейных отрезков вида $L_i = \{(x, k_i x + b_i) | x^i_{\min} \leq x \leq x^i_{\max}\}$, пронумерованных в порядке возрастания их угловых коэффициентов $0 \leq k_1 \leq k_2 \leq \cdots \leq k_n \leq +\infty$. Пусть T_i обозначает проекцию отрезка с номером i на ось абсцисс. Тогда проекция этого отрезка на ось ординат может быть выражена произведением $k_i T_i$.

Рассмотрим функцию $w(t)$, график которой получен путём поочерёдного соединения сегментов последовательности $\{L_i\}_1^n$. Пусть точка $(t, w(t))$ этого графика лежит на отрезке с номером j, который является компонентом графика $S(f)$. Если отрезок с указанным номером является компонентом $S(g)$, то поменяем обозначения f и g местами. Тогда для выбранной точки справедливо равенство:

$$w(t) = k_j \left(t - \sum_{i=1}^{j-1} T_i \right) + \sum_{i=1}^{j-1} k_i T_i$$

Пусть s_0 обозначает сумму длин T_i всех горизонтальных проекций сегментов L_i функции g, номера которых меньше j. Тогда для разницы $t - s_0$ выполнено выражение:

$$t - s_0 = t - \sum_{\substack{L_i \in S(g) \\ 1 \leq i \leq j-1}} T_i = t - \sum_{i=1}^{j-1} T_i + \sum_{i=1}^{j-1} T_i - \sum_{\substack{L_i \in S(g) \\ 1 \leq i \leq j-1}} T_i = t - \sum_{i=1}^{j-1} T_i + \sum_{\substack{L_i \in S(f) \\ 1 \leq i \leq j-1}} T_i$$

Так как на каждом из сегментов функций f и g их приращения получаются из приращений аргумента умножением на соответствующий угловой коэффициент, то:

$$f(t - s_0) = k_j (t - \sum_{i=1}^{j-1} T_i + \sum_{\substack{L_i \in S(f) \\ 1 \leq i \leq j-1}} k_i T_i$$

$$g(s_0) = \sum_{\substack{L_i \in S(g) \\ 1 \leq i \leq j-1}} k_i T_i$$

Таким образом, для произвольного значения аргумента t и соответствующего ему значения s_0 выполнено условие $w(t) = f(t - s_0) + g(s_0)$. Рассмотрим зависимость значения этой же суммы от параметра s_0 при его изменении в диапазоне $[0; t]$. Использование свойства выпуклости функций f и g позволяет получить следующие неравенства:

$$f(t - (s_0 + \Delta)) \geq f(t - s_0) - k_j \Delta$$

$$g(s_0 + \Delta) \geq g(s_0) + k_j \Delta$$

Таким образом, точка s_0 является точкой минимума суммы функций f и g при каждом фиксированном значении t, поэтому построенная кусочно-линейная функция $w(t)$ является свёрткой этих функций:

$$w(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\} = (f \otimes g)(t)$$

□

Следствие 1: об обратной свёртке кусочно-линейных функций. Пусть свёртка двух функций $w = f \otimes g$ является кусочно-линейной функцией, график которой состоит из последовательности соединённых между собой сегментов, упорядоченных по возрастанию их угловых коэффициентов. Пусть функция g является выпуклой кусочно-линейной функцией, причём для каждого сегмента функции g функция w имеет сегмент не меньшей протяженности с таким же угловым коэффициентом. Тогда график функции f может быть построен из графика функции w путём укорачивания части её сегментов. Если для очередного сегмента функции w функция g имеет сегмент с таким же угловым коэффициентом, то длина соответствующего сегмента графика для функции f равна разнице длин указанных сегментов w и g. Таким образом, если длины сегментов w и g равны, то новых сегментов в график f не добавляется.

2.5.2 Схема вычисления функции обратной свёртки

В данном разделе приводится доказанная в ходе проведённых исследований теорема, позволяющая осуществлять расчёт обратной свёртки для пары вогнутой и выпуклой кусочно-линейных функций произвольного вида.

Лемма 1. Пусть задана пара вогнутых кусочно-линейных функций $f \in \mathcal{F}$ и $g \in \mathcal{F}$. Тогда график функции $w \in \mathcal{F}$ их свёртки $w(t) = (f \bar{\otimes} g)(t) = \sup_{0 \leq s \leq t} \{f(t-s) + g(s)\}$ в алгебре Max-Plus может быть построен путём последовательной композиции составляющих эти функции отрезков в порядке уменьшения их угловых коэффициентов.

Доказательство. Выполняется в полной аналогии с доказательством теоремы 2.5.1 о Min-Plus свёртке пары выпуклых кусочно-линейных функций. □

Теорема 2 (Обратная свёртка вогнутой и выпуклой кусочно-линейных функций)**.** Пусть задана пара кусочно-линейных функций: вогнутая функция $w \in \mathcal{F}$ и выпуклая функция

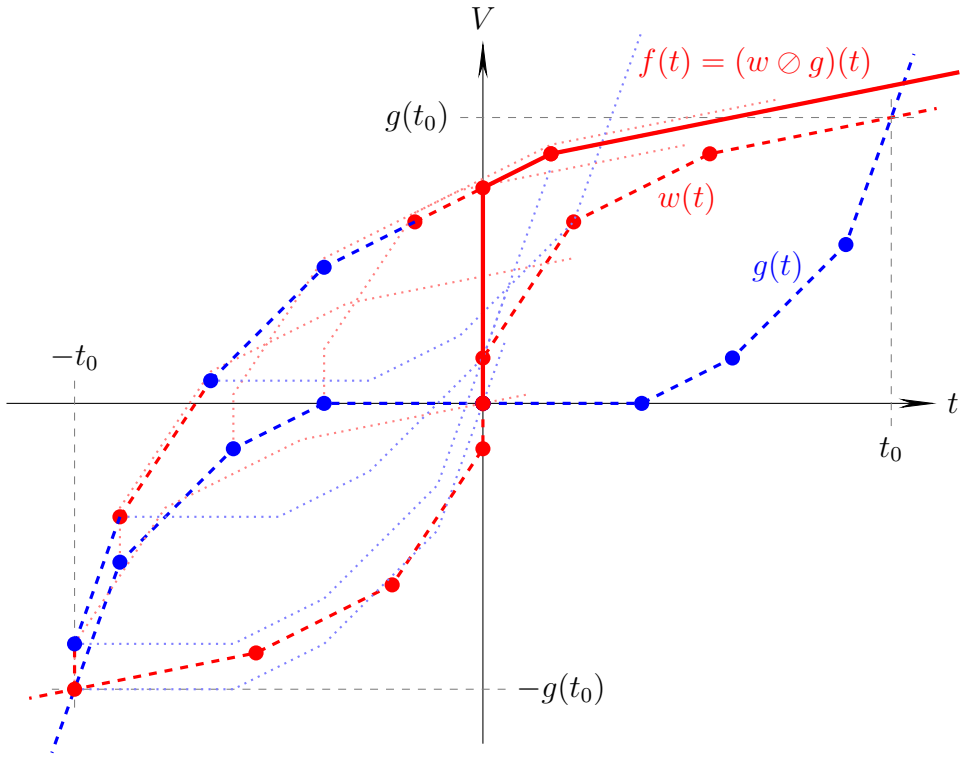

Рисунок 2.7: Построение графика функции обратной свёртки $f(t) = (w \oslash g)(t)$ для пары вогнутой и выпуклой кусочно-линейных функций $w(t)$ и $g(t)$.

$g \in \mathcal{F}$ – пересекающихся ровно в двух точках: 0 и $t_0 = \sup\{w(t) = g(t)\} > 0$. Тогда график их обратной свёртки $f = w \oslash g$ может быть построен из точки $(t_0, -g(t_0))$ последовательным соединением отрезков, составляющих графики функций $w(t)$ и $\hat{g}(t) = \min(g(t), g(t_0))$, в порядке уменьшения их угловых коэффициентов.

Доказательство. Иллюстрация для предложенного условием теоремы метода приведена на рисунке 2.7. Доопределим функцию g на отрицательной полуплоскости нулевым значением и вычислим в алгебре Max-Plus свёртку $\bar{w}(t)$ для вогнутых функций $w(t)$ и $\bar{g}(t) = [g(t_0) - g(t_0 - t)]^+$ в точке $t + t_0$:

$$\bar{w}(t) = (w \bar{\otimes} \bar{g})(t + t_0) = \sup_{0 \le s \le t + t_0} \{w(t + t_0 - s) + \bar{g}(s)\}$$

Обозначение $u = t_0 - s$ позволяет переписать выражение следующим образом:

$$\bar{w}(t) = \sup_{-t \le u \le t_0} \{w(t + u) + \bar{g}(t_0 - u)\}$$

При $u < 0$ значение неубывающей функции w не возрастает $w(t+u) \le w(t)$, а функция $\bar{g}$ тождественно равна нулю. Таким образом, область изменения переменной u в последнем выражении может быть сокращена до отрезка $0 \le u \le t_0$.

$$w(t + u) + \bar{g}(t_0 - u) = w(t + u) + g(t_0) - g(u)$$

В силу вогнутости кривой w и выпуклости кривой g при каждом значении аргумента $t \geq t_0$ приращение функции w будет меньше приращения функции g. Таким образом, при $u > t_0$ выполнено выражение:

$$w(t+u) + \bar{g}(t_0 - u) = w(t+u) + g(t_0) - g(u) \leq w(t+t_0) + g(t_0) - g(0) = w(t+t_0) + \bar{g}(0)$$

Поэтому область определения u может быть расширена на $u \geq 0$:

$$\bar{w}(t) = \sup_{u \geq 0} \{w(t+u) + \bar{g}(t_0 - u)\} = \sup_{u \geq 0} \left\{w(t+u) + [g(t_0) - g(t_0 - t)]^+\right\}$$

$$\forall t \geq 0 : \ \bar{w}(t) = \sup_{u \geq 0} \{w(t+u) - g(u)\} + g(t_0) = (w \oslash g)(t) + g(t_0)$$

Таким образом, график обратной свёртки функций w и g может быть построен из графика свёртки функций w и $\bar{g}$ путём его переноса на t_0 единиц вправо и на $g(t_0)$ единиц вниз, а так же отбрасывания части той полученного графика, которая оказалась в отрицательной полуплоскости.

$$\forall t \geq 0 : \ f(t) = (w \oslash g)(t) = (w \bar{\otimes} \bar{g})(t + t_0) - g(t_0)$$

Для завершения доказательства стоит заметить, что функции w и $\bar{g}$ – вогнутые, для построения графика Max-Plus свёртки двух вогнутых функций достаточно знать множество линейных отрезков, из которых они состоят, а множества отрезков, составляющих функции $\bar{g}$ и $\hat{g}$ совпадают.

□

Замечание. Вместо абсциссы t_0 правой точки пересечения кривых w и g можно в условии теоремы достаточно выбрать самую левую точку $t_1 \leq t_0$, в которой угловой коэффициент функции g будет превышать угловой коэффициент первого отрезка функции w. Отрезки графика функции g, соответствующие изменению аргумента в диапазоне $[t_1; t_0]$ будут самыми крутыми среди всех отрезков функций w и $\hat{g}$. Поэтому значения графика свёртки max-plus на интервале $[-t_0; -t_1]$ будут компенсированы значениями функции g на интервале $[t_1; t_0]$, и они не будет оказывать влияния на результаты вычислений.

Представленный в данном разделе алгоритма для вычисления операции обратной свёртки позволяет расширить область применения предложенных ранее методов построения верхних оценок для задержки на более широкий класс функций – кусочно-линейные функции с произвольным количеством сегментов. Однако разработанный алгоритм не позволяет повысить точность указанных методов. В следующем разделе рассматривается альтернативный подход к вычислению задержек, способный решить указанную проблему и предоставить наиболее точные оценки задержки, которые могут быть вычислены в рамках используемых теорией сетевого исчисления предположений.

2.6 Оценка задержки с помощью линейного программирования

Причина сложности вычисления точных верхних оценок задержки и отставания с помощью стандартных приёмов теории сетевого исчисления заключается в свойствах оператора свёртки в алгебре Min-Plus. В ряде случаев свёртка позволяет объединить группу обработчиков в единый абстрактный обработчик, который описывал бы свойства сети достаточно точно и не приводил бы к потерям точности при вычислениях. Построение указанной абстракции тривиально в модели интегрированных сервисов, когда для каждого обработчика известна доля производительности, которую он выделяет на обслуживание каждого из проходящих через него потоков. Однако если распределение ресурса обработчиков динамически зависит от интенсивности поступающих на них потоков, что соответствует предположениям модели дифференцированных сервисов DiffServ, то свёртка не в состоянии достаточно точно описать взаимное влияние потоков друг на друга.

Одним из способов решения данной проблемы является отказ от использования свёртки, и описание всей сети в виде системы неравенств, отражающей зависимость объёма переданных через сеть данных от количества данных, поступивших на её входы, во времени. В настоящем разделе рассматривается один из способов построения указанной системы.

2.6.1 Модель сети

Топология сети обработчиков задана ориентированным ациклическим графом $G_{\mathcal{N}} = \langle V, E \rangle$ без кратных дуг, вершины которого пронумерованы натуральными числами $V = \{1, \ldots, m\}$. Вершине с номером v сопоставлен обработчик $S_v \in S_{\mathcal{N}}$, который обслуживает совокупность проходящих через него потоков данных в соответствии со строгой кривой сервиса $\beta_v \in \mathcal{F}$. Обработчики обслуживают каждый отдельный поток по дисциплине FIFO, однако правила мультиплексирования этих потоков между собой, вообще говоря, произвольные – модель построена в ограничениях «слепого» мультиплексирования.

Каждый путь π длины $l = |\pi|$ через граф $G_{\mathcal{N}}$ однозначно описывается неповторяющейся последовательностью $\pi = \{v_k\}_{k=1}^{l}$ его вершин.

Для обозначения k-й вершины маршрута π будем использовать запись $\pi[k]$. При этом вне зависимости от маршрута π при $k = 0$ значение выражения $\pi[k]$ условимся считать тождественно равным нулю: $\pi[0] \equiv 0$.

Через граф $G_{\mathcal{N}}$ пролегают маршруты $\pi_1, \ldots, \pi_n$ передачи данных множества пронумерованных натуральными числами потоков $F_{\mathcal{N}} = \{F_1, \ldots, F_n\}$. При этом потоку с номером i сопоставлена собственная кривая нагрузки $\alpha_i \in \mathcal{F}$.

Кривые нагрузки потоков и кривые сервиса обработчиков согласованы между собой так, что суммарный объём данных, пребывающих на каждый из обработчиков сети, асимптотически не превышает того объёма, который этот обработчик в состоянии обслужить:

$$\forall v \in V, t \to \infty : \sum_{i:\ v \in \pi_i} \alpha_i(t) \leq \beta_v(t)$$

Функцию поступления данных потока $F_i \in F_{\mathcal{N}}$ в сеть будем обозначать записью F_i^0. Для обозначения функции передачи данных, которая для каждого момента времени $t \geq 0$ возвращает суммарное количество данных потока $F_i \in F_{\mathcal{N}}$, сумевших преодолеть обработчик с номером $v \in \pi_i$ за время $[0; t]$, будем использовать запись F_i^v. Таким образом, интенсивность передачи данных потока $F_i \in F_{\mathcal{N}}$ через сеть характеризуется множеством функций $\bar{F}_i = \{F_i^{\pi_i[k]} \,|\, 0 \leq k \leq |\pi_i|\}$.

Сценарием работы сети $\mathcal{N}$ называется всякое множество $\bigcup \bar{F}_i$ из $\sum |\pi_i| + n$ функций поступления данных, которое удовлетворяет следующим ограничениям:

1. Каждая из указанных функций должна принадлежать классу $\mathcal{F}$;

2. В каждый момент времени объём данных, преодолевших произвольный обработчик сети, не превышает объёма данных, которые на него поступили:

$$\forall F_i \in F_{\mathcal{N}},\ \forall t \geq 0,\ \forall k, 1 \leq k \leq |\pi_i| :\ F_i^{\pi_i[k-1]}(t) \geq F_i^{\pi_i[k]}(t) \qquad (\Psi_{con})$$

3. Функции, непосредственно описывающие поступление потоков в сеть, удовлетворяют ограничениям соответствующих кривых нагрузки $\alpha_1, \ldots, \alpha_n$:

$$\forall F_i \in F_{\mathcal{N}} :\ F_i^0 \in \mathcal{A}(\alpha_i) \qquad (\Psi_{arv})$$

4. Функции, проходящие через один и тот же обработчик, удовлетворяют соответствующим ему кривым сервиса $\beta_1, \ldots, \beta_m$:

$$\forall S_v \in S_{\mathcal{N}} :\ \left\langle \sum_{\pi_i[k]=v} F_i^{\pi_i[k-1]}, \sum_{\pi_i[k]=v} F_i^{\pi_i[k]} \right\rangle \in \mathcal{S}_{strict}(\beta_v) \qquad (\Psi_{srv})$$

Поскольку ограничения (Ψ_{con}), (Ψ_{arv}), (Ψ_{srv}) определяют функции поступления данных неоднозначно, то обслуживание потоков может происходить по разным сценариям. Совокупность сценариев работы сети $\mathcal{N}$ обозначается записью $\mathcal{H}(\mathcal{N})$.

Таким образом, для построения достижимой верхней оценки задержки передачи заданного потока $F \in F_{\mathcal{N}}$ данных через сеть $\mathcal{N}$ достаточно рассмотреть множество допустимых сценариев её работы и выбрать среди них один из тех сценариев, при котором некоторая порция данных этого потока передаётся наибольшее время.

Настоящая работа предлагает разбивать множество допустимых сценариев на несколько семейств и описывать каждое из этих семейств соответствующей ему системой линейных неравенств. Преимущество указанного подхода заключается в том, что выбрать такой сценарий работы сети, на котором достигается максимальная задержка передачи данных заданного потока, внутри каждого семейства можно с помощью линейного программирования.

Последующие разделы подробно описывают один из способов разделения множества сценариев работы сети на семейства и построения соответствующих им систем линейных неравенств.

2.6.2 Упорядочивание значений функции поступления

Не ограничивая общности, построим семейство сценариев работы сети, необходимое для вычисления максимальной задержки передачи данных потока с номером i – будем называть его *целевым потоком*.

Поскольку обработчики сети обслуживают каждый поток по дисциплине FIFO и не изменяют относительного порядка данных внутри потоков, то одна и та же порция данных потока может быть уникально идентифицирована в произвольных точках сети суммарным объёмом данных этого потока, которые были переданы через эти точки до неё. Например, порция данных потока $F_i \in F_{\mathcal{N}}$, поступившая в сеть в момент t может быть уникально идентифицирована значением $F_i^0(t)$.

Для определённости будем считать, что максимальная задержка целевого потока $F_i \in F_{\mathcal{N}}$ достигается при обслуживании его порции $U = F_i^{\pi_i[l]}(t_\varnothing)$, преодолевшей заключительный обработчик $S_{\pi_i[l]} \in S_{\mathcal{N}}$ маршрута π_i длины $l = |\pi_i|$ в момент времени $t_\varnothing \geq 0$.

Поскольку дисциплина мультиплексирования указанного обработчика неизвестна, но под нагрузкой он не простаивает, то порция U могла поступить на него обработчик в произвольный момент времени, начиная с его текущего периода отставания и вплоть до момента передачи этой порции за пределы сети:

$$BPS_{\pi_i[l]}(t_\varnothing) \leq \inf\left\{t \geq 0 \middle| U \leq F_i^{\pi_i[l-1]}(t)\right\} \leq t_\varnothing$$

Следовательно, выполняется следующее соотношение для функций передачи данных:

$$F_i^{\pi_i[l-1]}(BPS_{\pi_i[l]}(t_\varnothing)) \le U \le F_i^{\pi_i[l-1]}(t_\varnothing)$$

Предыдущий обработчик $S_{\pi_i[l-1]}$ пути π_i, передавший порцию U в один из моментов в течение интервала $\left[BPS_{\pi_i[l]}(t_\varnothing); t_\varnothing\right]$, мог получить её лишь после начала соответствующего периода отставания $BPS_{\pi_i[l-1]}(BPS_{\pi_i[l]}(t_\varnothing))$, но, в то же время, лишь до момента $t_\varnothing$ завершения её обслуживания, и так далее.

Итеративное применение описанных рассуждений по упорядочиванию моментов прохождения порции U через сеть к каждому обработчику вдоль маршрута π_i передачи целевого потока F_i, позволяет получить систему интервальных ограничений на функции передачи данных из $\bar{F}_i$. Пусть ϖ_k соответствует пути, который остаётся преодолеть данным целевого потока для завершения своего обслуживания после того, как они преодолели обработчик с номером $\pi_i[l-k]$. Совокупность переменных образует множество всевозможных суффиксов пути π_i, начиная от пустого пути $\varpi_0 = \varnothing$ и заканчивая полным путём $\varpi_l = \pi_i$. Тогда выполняются следующие соотношения:

$$\forall k, 0 \le k < l :$$

$$\begin{gathered}
\varpi_{k+1} = \langle \pi_i[l-k] \rangle \varpi_k \quad t_{\varpi_{k+1}} = BPS_{\pi_i[l-k]}(t_{\varpi_k}) \\
t_{\varpi_{k+1}} \le \inf\left\{ t \ge 0 \middle| U \le F_i^{\pi_i[l-(k+1)]}(t) \right\} \le t_{\varpi_k} \\
F_i^{\pi_i[l-(k+1)]}(t_{\varpi_{k+1}}) \le U \le F_i^{\pi_i[l-(k+1)]}(t_{\varpi_0}) \qquad (\Phi_{int})
\end{gathered}$$

Используемые полученной системой неравенств значения функций связаны дополнительными ограничениями. В силу (Ψ_{con}) значения функций передачи из системы (Φ_{int}) в каждый момент времени могут быть упорядоченны по невозрастанию при их перечислении вдоль соответствующих им вершин маршрута π_i:

$$\forall k, 0 \le k < l : \; F_i^{\pi_i[0]}(t_{\varpi_k}) \ge F_i^{\pi_i[1]}(t_{\varpi_k}) \ge \cdots \ge F_i^{\pi_i[l]}(t_{\varpi_k}) \qquad (\Phi_{hor})$$

Указанные соотношения можно уточнить, заметив, что в момент начала периода отставания значение функции передачи для потока, поступающего на обработчик, равно значению функции прибытия для потока, полученного после их обслуживания:

$$\forall k, 0 \le k \le l : \; F_i^{\pi_i[l-(k+1)]}(t_{\varpi_k}) = F_i^{\pi_i[l-k]}(t_{\varpi_k}) \qquad (\Phi_{equ})$$

Поскольку последовательность моментов, соответствующих началам периодов отставания, не возрастает, а функции передачи из $\bar{F}_i$ не убывают, то значения функций в левых

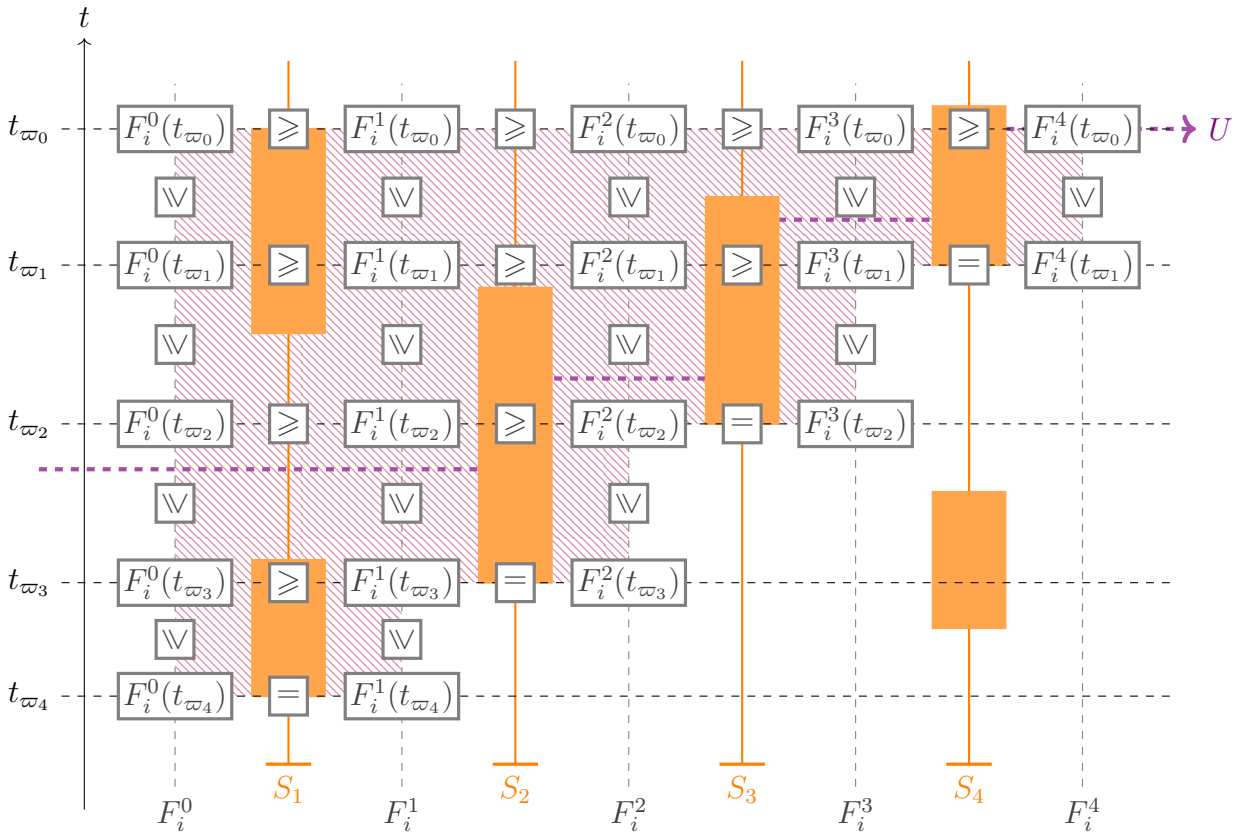

Рисунок 2.8: Сетка ограничений для потока F_i, проходящего через последовательность из четырёх обработчиков $S_1, \ldots, S_4$.

частях неравеств системы (Φ_{int}) могут быть упорядочены по неубыванию:

$$\forall k, 1 \le k \le l: \; t_{\varpi_l} \le t_{\varpi_{l-1}} \cdots \le t_{\varpi_0}$$

$$F_i^{\pi_i[k]}(t_{\varpi_l}) \le F_i^{\pi_i[k]}(t_{\varpi_{l-1}}) \le \cdots \le F_i^{\pi_i[k]}(t_{\varpi_0}) \qquad (\Phi_{ver})$$

Совмещая системы (Φ_{equ}) и (Φ_{ver}) можно получить неравенства, связывающие между собой нижние оценки прохождения порции U через каждый из обработчиков вдоль пути передачи целевого потока:

$$F_i^{\pi_i[0]}(t_{\varpi_l}) = F_i^{\pi_i[1]}(t_{\varpi_l}) \le F_i^{\pi_i[1]}(t_{\varpi_{l-1}}) = F_i^{\pi_i[2]}(t_{\varpi_{l-1}}) \le \cdots \le F_i^{\pi_i[l]}(t_{\varpi_0})$$

$$F_i^{\pi_i[0]}(t_l) \le F_i^{\pi_i[1]}(t_{\varpi_{l-1}}) \le \cdots \le F_i^{\pi_i[l]}(t_{\varpi_0}) \qquad (\Phi_{low})$$

Графически множество полученных соотношений для функций из $\bar{F}_i$ может быть представлено в виде *сетки ограничений*, образованной их значениями. Пример такой сетки для потока данных $F_i \in F_{\mathcal{N}}$, последовательно проходящего через четыре сеть $\mathcal{N}$ из четырёх обработчиков $S_{\mathcal{N}} = S_1, \ldots, S_4$, приведён на рисунке 2.8. Функции $F_0, \ldots, F_4$ передачи данных образуют вертикальные оси сетки, а моменты начала рассмотренных периодов отста-

вания $t_{\varpi_0}, \ldots, t_{\varpi_4}$ – горизонтальные. Полные временные интервалы периодов отставания изображены в виде прямоугольников на осях обработчиков $S_1, \ldots, S_4$.

В соответствии с соотношениями (Φ_{int}) сетка ограничивает множество допустимых траекторий для порции данных U, которая покинула сеть в момент t_{ϖ_0}, штрихованной зоной. Одна из таких траекторий передачи порции U показана жирной пунктирной линией.

Конкретная траектория передачи порции данных U зависит от дисциплин обслуживания потоков, которые реализуют обработчики вдоль пути её передачи. Если бы данные целевого потока обрабатывались по остаточному принципу, после обслуживания всех прочих потоков, то порции U соответствовала бы нижняя граница штрихованной зоны. Если бы обработчики, наоборот, обслуживали поток F_i с наивысшим приоритетом, откладывая обслуживание прочих потоков, и были бы способны начать передавать данные без задержки, то порция U могла бы пройти по верхней границе окрашенной зоны.

2.6.3 Стабилизация узлов ограничительной сетки

Рассмотренные ограничения вводят на множестве значений функций из $\bar{F}_i$ отношение порядка, которое определяет относительное расположение этих значений внутри построенной сетки. Однако, для оценки расстояния между отдельными значениями сетки необходимы дополнительные ограничения.

Условие (Ψ_{arv}) позволяет построить систему неравенств, каждое из которых ограничивает сверху длительность временного промежутка между наименьшим моментом, когда порция данных U могла бы поступить на один из обработчиков сети, и наименьшим моментом, в который она могла бы преодолеть этот или один из последующих обработчиков вдоль маршрута её передачи:

$$\forall k', k'',\ 0 \le k' < k'' \le |\pi_i| :$$

$$F_i^{\pi_i[k'']}(t_{\varpi_{k''}}) - F^{\pi_i[k']}(t_{\varpi_{k'}}) \le F_i^0(t_{\varpi_{k''}}) - F_i^0(t_{\varpi_{k'}})$$

$$F_i^0(t_{\varpi_{k''}}) - F_i^0(t_{\varpi_{k'}}) \le \alpha_i(t_{\varpi_{k''}} - t_{\varpi_{k'}}) \qquad (\Phi_{shrink})$$

Поскольку для пути длины $l = |\pi_i|$ количество неравенств указанного вида равно количеству различных способов выбрать пару из тех $l+1$ узлов, с которыми ассоциированы значения функции поступления F_i^0, то оно может быть найдено путём вычисления соответствующего биномиального коэффициента:

$$|\Phi_{shrink}| = \binom{l+1}{2} = \frac{(l+1)!}{2!\,(l-1)!} = \frac{l(l+1)}{2}$$

Условия (Φ_{shrink}) стягивают узлы ограничительной сетки, не давая ей чрезмерно расширяться. Однако, эти условия не устанавливают каких-либо пределов сжатия, допуская «схлопывание» сетки в точку. Для предотвращения коллапса можно использовать незадействованное ранее условие (Ψ_{srv}). Оно позволяет ограничить снизу расстояние между моментами $t_{\varpi_{k-1}}$ и t_{ϖ_k}, которые неравенствами (Φ_{shrink}), для каждого обработчика вдоль маршрута π_i:

$$\forall k, 1 \le k \le |\pi_i| : \sum_{\substack{\forall s,k': \\ \pi_s[k']=\pi_i[k]}} \left(F_s^{\pi_s[k']}(t_{\varpi_{k-1}}) - F_s^{\pi_s[k']}(t_{\varpi_k}) \right) \ge \beta_{\pi_i[k]}(t_{\varpi_{k-1}} - t_{\varpi_k}) \qquad (\Phi_{expand})$$

Каждое из неравенств системы (Φ_{expand}) затрагивает не только целевой поток F_i, но и другие потоки данных, проходящие через соответствующий обработчик. Таким образом, вводятся новые неизвестные величины, которые не позволяют сразу же вычислить пределы «сжатия» сетки ограничений. Однако, система неравенств, полученная путём объединения ограничительных сеток, построенных для каждого из потоков, которые были задействованы в неравенствах (Φ_{expand}), устойчива – её характеристики «растяжения» и «сжатия» однозначно определяются модельным представлением исследуемой сети. Для построения упомянутой устойчивой системы неравенств может использоваться алгоритм 1.

2.6.4 Задание целевой функции

Пусть порция данных, покинувшая сеть в момент t_l, поступила на неё в момент u. В силу обслуживания потока по принципу FIFO, суммарное количество его данных, поступивших в сеть к моменту u, не уступает суммарному количеству U его данных, покинувших сеть к моменту t_l:

$$F_i^0(u) \ge U$$

Неравенство в указанном выражении возможно лишь в том случае, если функция F_i^0 имеет разрыв в точке u. В то же время, значение $t_l - u$ задержки потока для такой функции совпадает со значением для функции $\hat{F}_i^0$, полученной из F_i^0 изменением её значения в точке u. Таким образом, полученное неравенство может быть заменено равенством. Чтобы соответствовать построенной системе ограничений, значение $A_i^0(u)$ должно удовлетворять требованиям монотонности функции прибытия и требованиям согласованности кривой нагрузки:

$$\exists k, k \le l :$$

$$F_i^0(t_{\varpi_k}) \le F_i^0(u) \le F_i^0(t_{\varpi b+1})$$

Алгоритм 1: Построение устойчивой системы неравенств.

$\mathcal{X} = \emptyset$; /* Множество переменных, использованных при описании системы. */

$\mathcal{R} = \emptyset$; /* Множество линейных ограничений, связывающих эти переменные. */

/* Рекурсивная функция для построения устойчивой системы неравенств. */

Function *traceback*(v, t_π)**:**

- $\mathcal{X} \leftarrow \mathcal{X} \cup \{t_{v\pi}, t_\pi\}$; /* Регистрируем моменты t_π и $t_{v\pi} = BPS_v(t_\pi)$. */
- $\mathcal{R} \leftarrow \mathcal{R} \cup \{t_{v\pi} \le t_\pi\}$; /* Добавляем условие их упорядочивания. */
- /* Дополняем сетки ограничений каждого из потоков, */
- /* которые пересекают рассматриваемый обработчик. */
- **foreach** $s, k : \pi_s[k] = v$ **do**
  - $\mathcal{X} \leftarrow \mathcal{X} \cup \{F_s^{\pi_s[k-1]}(t_{v\pi}), F_s^{\pi_s[k]}(t_\pi)\}$; /* Регистрируем и упорядочиваем */
  - $\mathcal{R} \leftarrow \mathcal{R} \cup \{F_s^{\pi_s[k-1]}(t_{v\pi}) \le F_s^{\pi_s[k]}(t_\pi)\}$; /* переменные из (Φ_{low}). */
  - $\mathcal{X} \leftarrow \mathcal{X} \cup \{F_s^0(t_{v\pi}), F_s^0(t_\pi)\}$; /* Регистрируем и упорядочиваем */
  - $\mathcal{R} \leftarrow \mathcal{R} \cup \{F_s^0(t_{v\pi}) \le F_s^0(t_\pi)\}$; /* переменные из (Φ_{int}). */
  - $\mathcal{R} \leftarrow \mathcal{R} \cup \{F_s^0(t_{v\pi}) \le F_s^{\pi_s[k-1]}(t_{v\pi})\}$; /* Упорядочиваем переменные из */
  - $\mathcal{R} \leftarrow \mathcal{R} \cup \{F_s^0(t_\pi) \le F_s^{\pi_s[k]}(t_{v\pi})\}$; /* (Φ_{int}) и (Φ_{low}) между собой. */
  - **if** $k \ne 1$ **then**
    - /* Переход к предыдущему обработчику по пути передачи потока. */
    - `traceback`($\pi_s[k-1]$, $t_{v\pi}$);
  - **else** /* Если такого обработчика нет, мы добрались до отправителя. */
    - /* Все переменные добавлены, и сетка ограничений построена. */
    - /* Осталось стянуть её узлы неравенствами из (Φ_{shrink}). */
    - **foreach** $k', k'' : 0 \le k' < k'' \le |\pi|$ **do**
      - $\mathcal{R} \leftarrow \mathcal{R} \cup \{F_s^0(t_\pi) - F_s^0(t_{v\pi}) \le \alpha_s(t_\pi - t_{v\pi})\}$;
- /* Не позволяем сетке ограничений «схлопнуться» в точку. */
- /* Добавляем условия отталкивания узлов из (Φ_{expand}). */
- $\mathcal{R} \leftarrow \mathcal{R} \cup \left\{ \sum\limits_{s,k:\ \pi_s[k]=v} \left(F_s^{\pi_s[k]}(t_\pi) - F_s^{\pi_s[k-1]}(t_{v\pi}) \right) \ge \beta_v(t_\pi - t_{v\pi}) \right\}$;
- **return**;

$$F_i^0(t_{\varpi_{b+1}}) - F_i^0(u) \le \alpha_i(t_{\varpi k+1} - u)$$

$$F_i^0(u) - F_i^0(t_b) \le \alpha_i(u - t_b)$$

Поочерёдно располагая момент u на одном из отрезков вида $[t_{\varpi_k}; t_{\varpi_{k+1}}]$, можно получить множество согласованных систем ограничений, каждая из которых позволяет вычислить максимальную задержку при заданных предположениях. Тогда наибольшая задержка в общем случае вычисляется как максимальное значение множества полученных частных решений.

Проведённые рассуждения показывают, что для вычисления задержки передачи данных потока, проходящего по сети обработчиков без неориентированных циклов по маршруту из n вершин, достаточно решить $n-1$ задачу линейного программирования. При этом количество переменных и неравенств, составляющих каждую из этих задач, обладает полиномиальной зависимостью от количества потоков и числа вершин, через которые пролегают их маршруты. Таким образом, справедлива следующая

Теорема 3. Если топология сети обработчиков представлена неориентированным ациклическим графом, то задача построения достижимой верхней оценки передачи данных потока по маршруту из n обработчиков сводится к решению $n-1$ системы линейных неравенств и может быть решена за полиномиальное время.

Полученный теоретический результат является новым. В статье [104] было показано существование полиномиального алгоритма вычисления достижимой верхней оценки для задержки передачи данных через сеть обработчиков с линейной топологией. Так же продемонстрировано, что сетей с прямой связью (feed-forward networks) общего вида задача построения достижимой оценки задержки принадлежит классу NP. Указанная оценка сложности объясняется тем фактом, что если одна и так же пара вершин связана между собой сразу несколькими маршрутами, то в процессе построения стабильной системы неравенств возникает необходимость согласования сеток ограничений между собой. Однако условия для согласования разных ограничительных сеток имеют нелинейную природу, поэтому для их учёта приходится решать несколько задач линейного программирования – число таких зада зависит от количества альтернативных маршрутов экспоненциально.

2.7 Реализация и экспериментальное исследование

2.7.1 Модуль оценки задержки

Для автоматизации процесса построения верхней оценки задержки при передаче потока данных через сеть с заданной конфигурацией была разработана экспериментальная версия соответствующего программного средства. Средство реализовано на языке C++.

На вход средства подаётся файл конфигурации сети, описывающий её топологию, внутреннее устройство и настройки маршрутизации её коммутационных устройств, а так же расположение и характеристики генерирующих потоки данных сетевых приложений. Разработанная экспериментальная версия работает в следующих предположениях:

1. Коммутаторы сети реализуют буферизацию на выходе (OQ). Таким образом, маршрут обработки каждого пакета на коммутационном конвейере состоит из анализатора пакетов, коммутационной матрицы и буферного блока;

2. Установленные в коммутаторы сети правила обработки потоков данных не образуют циклических зависимостей между ними;

3. Пропускные способности каждого из поступающих в сеть потоков данных ограничиваются на периферийных коммутаторах сети с помощью шейпинга по алгоритму текущего ведра.

Описанная указанным образом конфигурация сети транслируется в граф обработчиков сетевого исчисления с использованием следующего алгоритма. Сначала для каждого потока данных строится его траектория – последовательность из всех анализаторов, матрицы и буферных блоков, которые пересекает этот поток при прохождении через сеть. Далее, траектории отдельных потоков связываются в единый граф потоков путём объединения общих буферных блоков, через которые они проходят. Наконец, для каждой вершины полученного графа потоков по соответствующему ей элементу коммутационного конвейера строится обработчик сетевого исчисления и заданной кривой сервиса и правилами мультиплексирования.

Для построения верхней оценки задержки потока при его передаче через полученную сеть обработчиков было реализовано два алгоритма: Separated Flow Analysis (SFA) [99] и алгоритм, приведённый в разделе 2.6 – по аналогии назовём его Linear Programming Analysis (LPA).

2.7.2 Имитационная модель сети

Для оценки точности предложенного метода и валидации результатов разработанного средства построения верхних оценок задержки была создана имитационная модель сети, позволяющая вычислить время прохождения через сеть для каждого пересылаемого через неё пакета. Разработанная модель основана на библиотеке имитационного моделирования компьютерных сетей Network Simulator 3 (NS-3 [105]) и использует готовые компоненты

этой библиотеки для моделирования работы каналов передачи данных, подключённых к ним сетевых карт и сетевого стека операционной системы. Для генерации сетевого трафика используются следующие модели приложений:

1. *On-Off Application* – двухфазное приложение, которое пересылает UDP пакеты с постоянной скоростью в течение первой фазы и не активно в течение второй. Длительность каждой фазы определяется случайно по экспоненциальному распределению. Такой алгоритм генерации приложения хорошо подходит для моделирования телефонного трафика VOIP [106];

2. *UDP Trace Client* генерирует UDP пакеты по заданной трассе фреймов MPEG кодека. Примеры таких трасс для некоторых известных фильмов доступны онлайн;

3. *TCP Bulk Sender* генерирует TCP пакеты, пытаясь передать данные отправителя получателю как можно быстрее. Построенные им трассы передачи пакетов соответствуют, например, копированию файлов из сетевого хранилища данных;

4. Packet Sink моделирует приложение-адресат, и поддерживает и UDP, и TCP.

Помимо встроенных в NS-3, предложенная имитационная модель использует несколько программных модулей собственной разработки:

1. *QoS Switch Device* реализует модель коммутатора с буферизацией на выходе и поддержкой дифференцированной обработки потоков, относящихся к разным классам обслуживания;

2. *Flow Handler* обеспечивает передачу пакета по заданному маршруту, а так же осуществляет шейпинг потоков по алгоритму текущего ведра;

3. *Queue Block* моделирует буферный блок коммутатора. Реализованная версия поддерживает несколько очередей пакетов и их обработку по дисциплине WRR.

На вход программе имитационного моделирования подаётся тот же файл с конфигурацией сети, что и разработанному модулю оценки задержки. На основе информации из данного файла строится соответствующая ему модель NS-3. Затем запускается прогон модели. Прогон завершается по достижении заданного модельного времени. Результатом прогона имитационной модели на заданной конфигурации сети является трасса эксперимента, событиями которой являются моменты прохождения пакетов через элементы модели. Постобработка сгенерированной трассы позволяет получить близкие к реальным

значениям задержки передачи пакетов и сравнить их с величинами, полученными с помощью предложенного теоретического метода.

2.7.3 Модели потоков данных

Для моделирования нагрузки при экспериментальном исследовании построенных программных средств было разработано три модели потоков данных: голосовой трафик (VoIP), потоковое видео и копирование файлов через сеть.

Голосовой трафик VOIP

Голосовой трафик моделировался с использованием методики, предложенной в работе [106]. Указанная модель различает фазу активности, в течение которой аудио кодек генерирует поток данных с фиксированной интенсивностью, соответствующей его чувствительности, и фазу простоя, когда кодек не может зафиксировать звуков в заданном частотном диапазоне и не генерирует каких-либо данных. В соответствии с представленными результатами исследований средние длительности активной и пассивной фазы работы кодека составляют $\alpha = 352$ и $\beta = 650$ миллисекунд соответственно.

Генерируемые в течение активной фазы работы кодека данные формируют последовательность пакетов, передающихся через равные промежутки времени. Чем больше значение этого промежутка, тем большего размера пакеты будут передаваться, тем большее количество полезной информации будет приходиться на единицу переданных через сеть данных, но и тем большую задержку передачи будет испытывать голосовой поток.

В работе рассматривался аудио кодек G.711, который генерирует аудио данные со скоростью $r = 64$ Kbps. Время пакетизации было установлено равным $t = 20$ миллисекундам. При этом полезный объём передаваемых пакетов данных составлял $rt = 160$ байтов. Сгенерированные кодеком данные инкапсулировались в заголовки следующих протоколов:

1. RTP (уровень приложений) размер заголовков 12 байтов;
2. UDP (транспортный уровень) – 8 байтов;
3. IP (сетевой уровень) – 20 байтов;
4. Ethernet 802.3 (канальный уровень) – 38 байтов (с учётом 8-байтовой преамбулы и 12-байтовым промежутком между фреймами, необходимыми для корректной передачи).

В результате, передача каждого пакета через физическую среду расходовала 1904 бита его пропускной способности, а необходимая для обработки активной фазы работы кодека пропускная способность канала передачи данных составляла $95,2$ Kbps. Поскольку в течение фазы простоя аудио кодек данных не генерировал, то асимптотическая пропускная способность необходимая для передачи трафика без потерь была меньше в $\alpha/(\alpha+\beta)$ раз и составляла около $33,5$ Kbps. Графики зависимости количества переданных аудио потоком данных от времени представлены на рисунке 2.9.

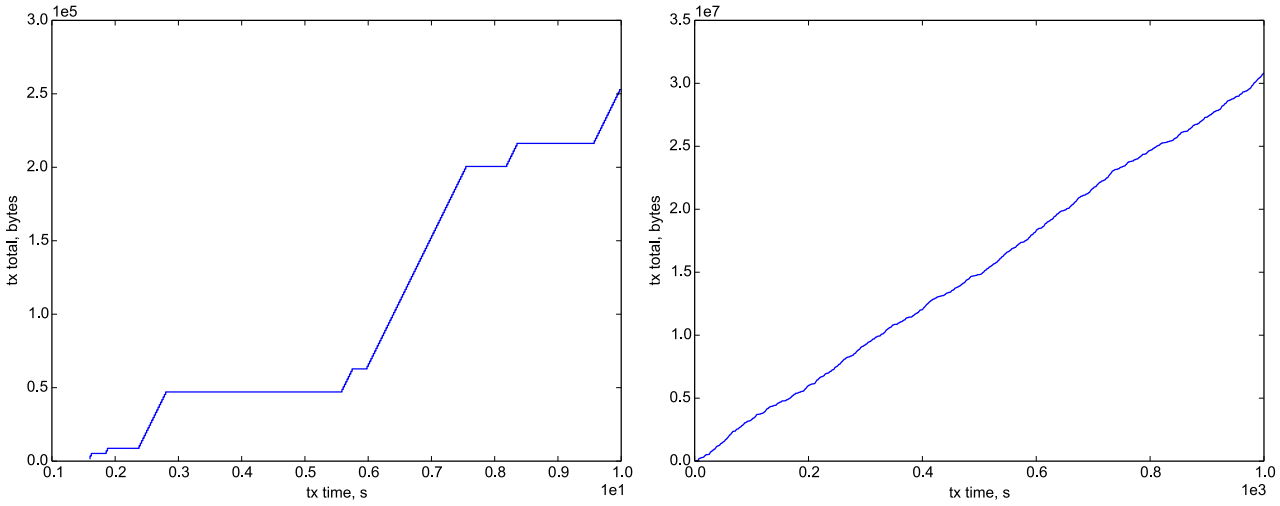

Рисунок 2.9: Зависимость количества переданных битов потока аудио трафика от времени (сек) на интервале длины 10 секунд (слева) и 1000 секунд (справа).

Шейпер с достаточным объёмом буфера для временного хранения пакетов, вообще говоря, способен сгладить всплески потока аудио данных и превратить его в равномерный поток пакетов, для передачи которого требуется указанная пропускная способность. При этом часть пакетов, соответствующих всплескам активности аудио кодека, задерживалась бы в буфере шейпера и передавалась бы в периоды простоя кодека. Однако подобное сдерживание трафика мало подходит для аудио потока, накладывающего жёсткие ограничения на сквозную задержку. Поэтому в проводимых экспериментах помимо пропускной способности шейперу аудио потока задавался ненулевой всплеск, а объём буфера хранения пакетов, наоборот, ограничивался. Графики зависимости сквозной задержки для передачи пакетов аудио потока от времени их передачи для неограниченного буфера и разных параметров всплеска изображены на рисунке 2.10.

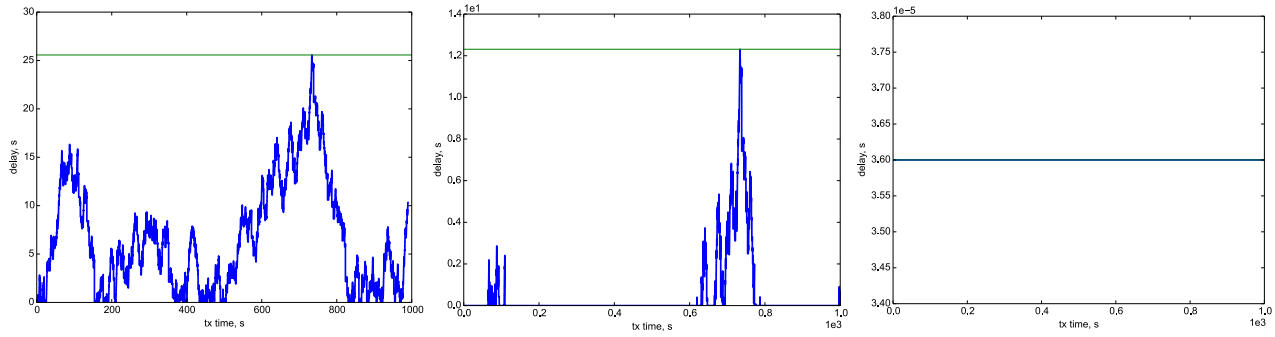

Рисунок 2.10: Зависимость задержки передачи пакета (сек) аудио потока от времени (сек) при размере всплеска равном нулю (слева), 250 пакетам или около 50 Кб (в центре) и 500 пакетам или около 100 кб (справа). Зелёной линией обозначена максимальная задержка, которая была зафиксирована в ходе эксперимента.

Потоковое видео

При моделировании передачи потокового видео рассматривался кодек MPEG4. Указанный кодек различает три типа видео фреймов: I, P и B. I- фреймы самодостаточны в том смысле, что для сжатия и распаковки каждого такого фрейма достаточно лишь содержащейся в нём информации. Для сжатия P- фрейма необходима информация из предыдущего I- или P- фрейма, а для B- фрейма нужна информация как из предыдущего, так и из последующего I- или P- фрейма. Таким образом, множество сформированных B- фреймов не может быть обработано до тех пор, пока не построен следующий за ними I- или P- фрейм. В результате, порядок передачи видео фреймов через сеть отличается от порядка их формирования, а видео поток имеет характерные всплески активности в моменты генерации I- и P- фреймов. Чем большее количество B- фреймов формирует кодек, тем лучшую степень сжатия он обеспечивает, однако, тем большую задержку передачи видео через сеть он подразумевает.

Размер фреймов видео кодека обычно превышает размеры MTU, поэтому передача пакетов соответствующего потока происходит сразу же после сжатия очередного видео фрейма, а его пакеты имеют максимально допустимые для данной сети размеры.

В сети Интернет доступны примеры трасс генерации MPEG-4 фреймов для множества известных видеороликов с различными настройками качества картинки и разнообразными параметрами сжатия [107]. В разработанной модели рассматривалась трасса 10 секундной сцены из кинофильма “Скорость”. Разрешение указанного ролика составляло 1920×1080

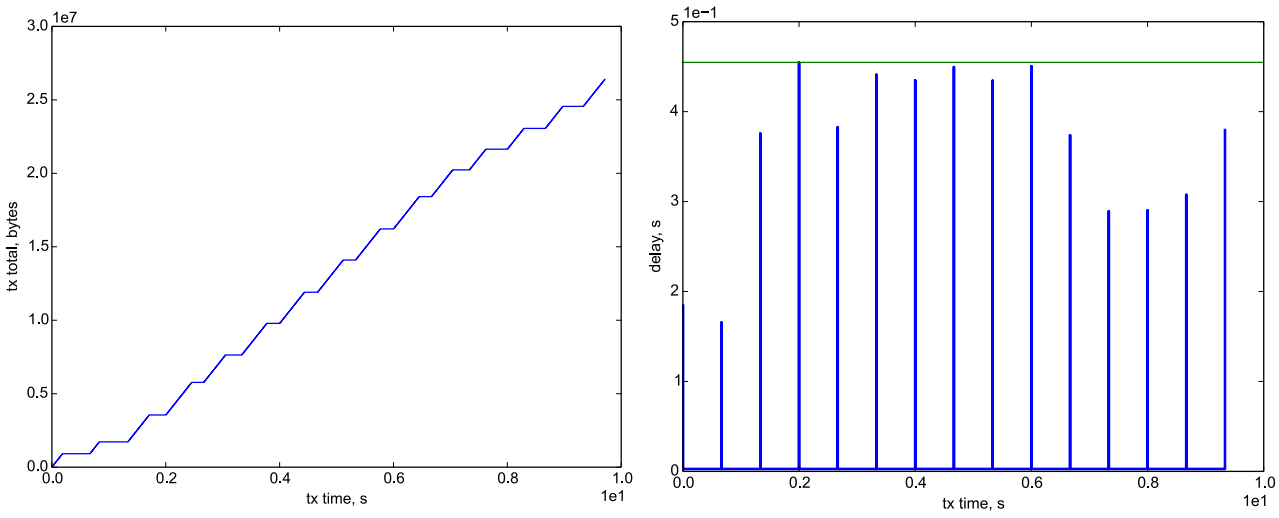

Рисунок 2.11: Зависимость количества переданных битов потокового видео (слева) и задержки передачи данных между устройствами (справа) от времени (сек).

пикселов, его средняя скорость передачи данных – 319521 байтов или около 2,56 Mbps, а максимальный размер всплеска при кодировании фреймов – 262373 байта (рисунок 2.11).

В предположении о применении для передачи видео такого же стека протоколов, что и при передаче аудио: RTP, UDP, IP, Ethernet – размер полезной нагрузки для каждого пакета равна 1460 байтов. При этом каждый пакет затрачивает 1538 байтов или 12304 битов пропускной способности физического канала, а общая пропускная способность канала, необходимого для передачи видео, составляет порядка 2,7 Mbps. На практике значение данной величины ещё больше из-за неучтённой в расчётах фрагментации видео фреймов по пакетам. На рисунке 2.12 изображены графики зависимости сквозной задержки и количества сброшенной информации при шейпинге с асимптотической скоростью 2,7 Mbps, минимальным размером всплеска и объёмом буфера шейпера равным в 500 MTU. Задержка нарастает вплоть до полного заполнения буфера шейпера. Затем наступает стабилизация задержки, но часть передаваемых пакетов начинает сбрасываться.

Полоса пропускания канала, необходимая для передачи всех пакетов видеопотока, составляла порядка 2,9 Mbps. При этом размер необходимый для хранения данных потока во время всплесков активности видеокодека составлял около 455 Кбайт, а максимальное время сквозной задержки было зафиксировано на 1,25 секунды. Как это показано на рисунке 2.13, шейпер с достаточным размером буфера способен сгладить всплески потока и перераспределить генерируемый видеокодеком трафик практически равномерно.

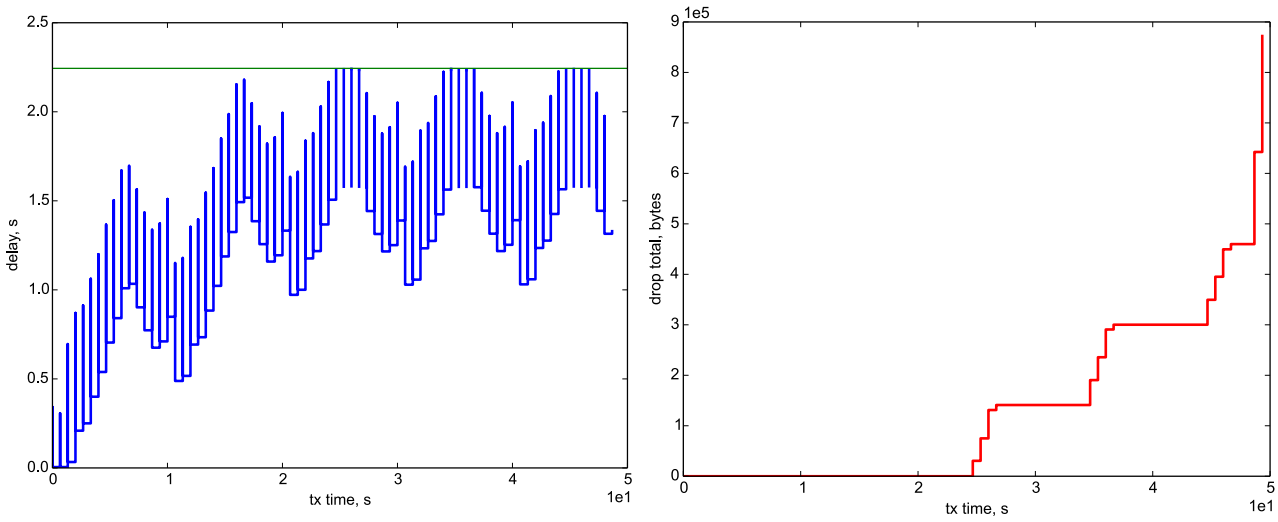

Рисунок 2.12: Зависимость задержки (сек) передачи данных видео потока (слева) и количества сброшенных при этом байтов информации (справа) от времени (сек). Асимптотическая скорость передачи равна $2,7$ Mbps, допустимый размер всплеска – 0 байт, размер буфера хранения пакетов – 750 Кбайт.

Задержка при вещании потокового видео не так значительна, как при передаче голосового трафика, и для него допустимы достаточно большие значения сквозной задержки. Тем не менее, при его шейпинге по алгоритму текущего ведра обычно используются более мягкие параметры, и размер всплеска, как правило, превышает минимальное значение и составляет несколько MTU, а буфер для временного хранения данных имеет меньший размер, вплоть до его отсутствия.

Копирование файлов

При копировании файлов необходимо передать заданный массив данных с одного конечного узла сети на другой её конечный узел. Поскольку передача данных не должна допускать потерь и ошибок, то разработанная модель потока предполагает использование протокола TCP. Так как требований к задержке отдельных порций передаваемого массива данных не предъявляется, то пакеты потока имеют фиксированный размер равный MTU. Для шейпинга трафика указанного типа использовался алгоритм текущего ведра с максимальной асимптотической скоростью передачи 25 Mbps. В отличие от остальных модельных потоков, поток передачи массива данных двухсторонний: на каждый полученный пакет данных конечная машина назначения генерирует пакет подтверждения минимального размера – 64 байта. Отправка каждого такого пакета затрачивает $64 + 20$ байтов

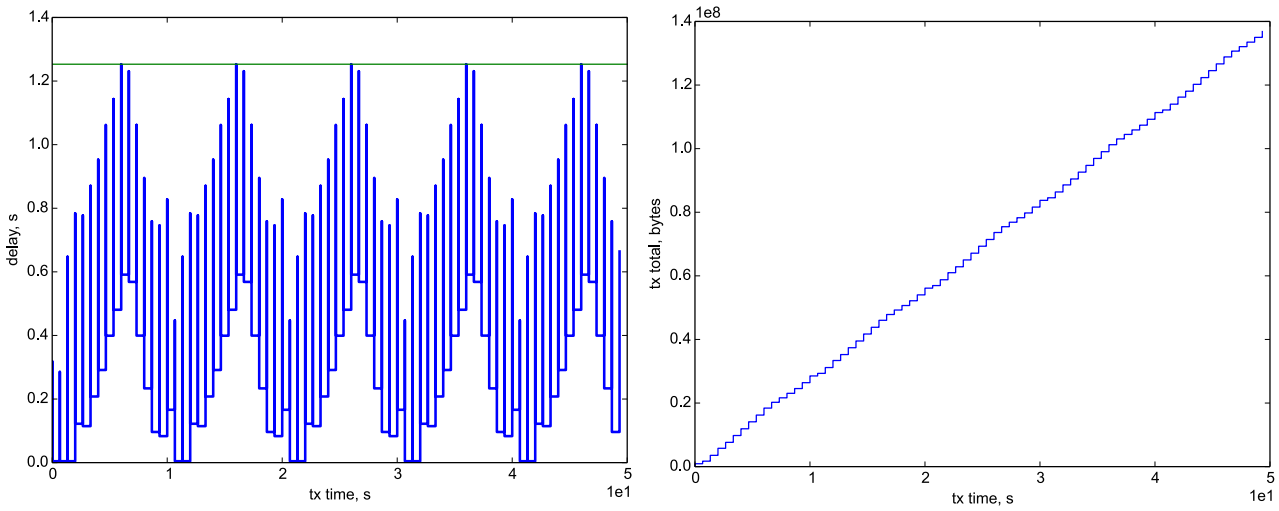

Рисунок 2.13: Зависимости сквозной задержки (сек) передачи пакетов (слева) и количества переданных видеопотоком байтов информации (справа) от времени (сек). Асимптотическая скорость передачи равна $2,9$ Mbps, допустимый размер всплеска – 0 байт, размер буфера хранения пакетов – 455 Кбайт.

пропускной способности канала. Поэтому при выбранной скорости передачи пакетов данных интенсивность потока подтверждений может быть рассчитана по формуле $25/1538 \times 84$ и составляет порядка $1,365$ Mbps. В качестве параметров алгоритма текущего ведра для шейпинга потока подтверждений использовались буфер нулевой длины, размер всплеска 1 MTU и максимальная скорость $1,4$ Mbps. В качестве протокола перегрузки использовался алгоритм NewReno [108]. Характерные для него изменения окна перегрузки можно наблюдать на рисунке 2.14.

Попытки (рисунки 2.15 и 2.16) заменить буфер ненулевым размером всплеска хоть и приводят значительному снижению задержки, но возникающее при этом постоянное дрожание пропускной способности предоставленного потоку канала снижает объём успешно переданных и увеличивает объём сброшенных данных.

2.7.4 Тестовые сценарии

Для тестирования функциональности разработанных программных средств было построено два параметризованных семейства тестовых сценариев, которые используют линейную топологию и топология с горловиной соответственно.

Каждая топология строится из одинаковых элементов. Линии связи работают на скорость 100 Mbps. Интерфейсы коммутаторов способны обслуживать передаваемые через

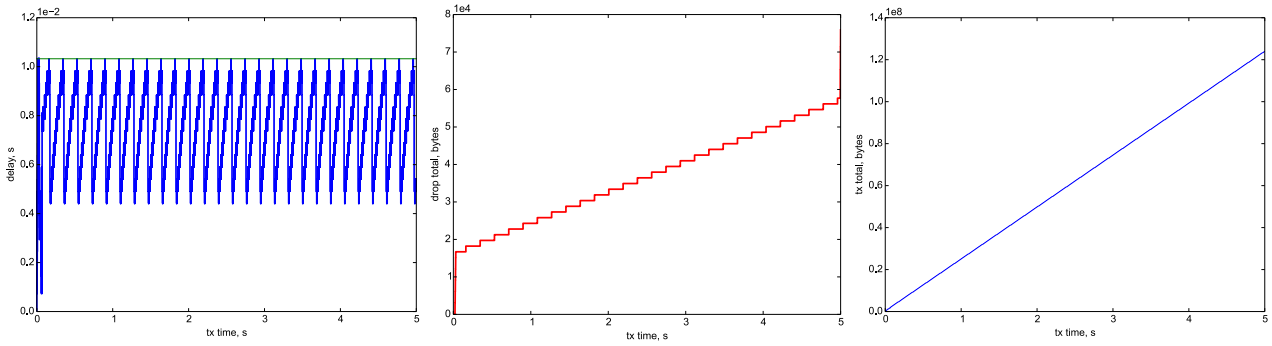

Рисунок 2.14: Зависимость задержки (сек) передачи данных при копировании файлов (слева), количества сброшенных (в центре) и успешно переданных (справа) при этом байтов информации от времени (сек). Асимптотическая скорость передачи равна 25 Mbps, допустимый размер всплеска – 0 байт, размер буфера хранения пакетов – 20 MTU.

них потоки данных с различным качеством сервиса. На каждом из интерфейсов расположено по четыре очереди для исходящих пакетов, выборка из которых осуществляется по дисциплине Weighted Round Robin (WRR). Размер каждой очереди составляет 100 пакетов, а их весовые коэффициенты изменяются от 4 до 1 в порядке возрастания их номеров. Таким образом, множество потоков, передающихся через первую очередь интерфейса, получает лучшее качество обслуживания, чем каждая из групп потоков, передающихся через остальные его очереди.

Для каждого из моделируемых потоков в топологию сети добавляется два хоста: один передатчик и один приёмник. При этом один хост может работать лишь с одним соединением. Передаваемый хостом поток данных подвергается шейпингу при поступлении на подключённый к нему периферийный коммутатор сети. Маршруты передачи потоков от коммутатора к коммутатору генерируются и устанавливаются в них заранее на этапе генерации конфигурации сети.

Первое семейство тестовых сценариев (рисунок 2.17) рассматривает последовательность из заданного числа коммутаторов, через которую передаётся по одному потоку каждого из рассмотренных типов: аудио поток VOIP, потоковое видео и поток копирования данных. При передаче через интерфейсы коммутаторов каждый из указанных потоков помещается в собственную очередь: аудио поток – в очередь с весом 4, видео поток – в очередь с весом 3, копирование данных – в очередь с весом 1.

Второе семейство тестовых сценариев (рисунок 2.18) использует топологию из двух деревьев с соединёнными вершинами. В качестве параметров топологии используются

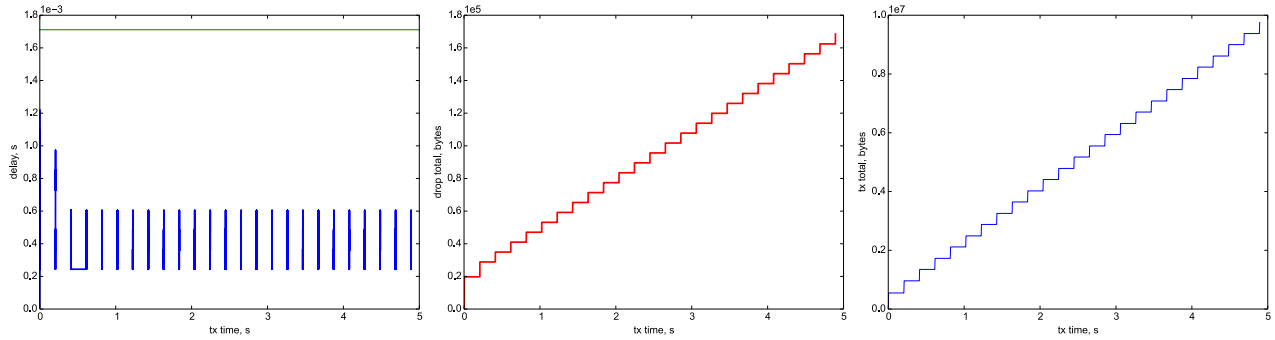

Рисунок 2.15: Задержка передачи данных, количество сброшенных и количество успешно доставленных пакетов при шейпинге потока алгоритмом текущего ведра с асимптотической скоростью 25 Mbps, размером всплеска 20 MTU и отсутствием буфера для временного хранения данных.

количество коммутаторов каждого уровня присоединяющихся к одному коммутатору следующего уровня (кратность вершины дерева) и глубина дерева. Хосты присоединяются к коммутаторам в основании указанных деревьев, сохраняя при этом кратность вершины. На каждом хосте одного дерева инициируется поток передачи данных, а на каждом соответствующем ему узле второго дерева – поток для его приёма. При проведении экспериментов суммарная интенсивность передаваемых через каждый канал данных сети данных не должна превышать пропускной способности этого канала.

В ходе проведённых экспериментов скорость потоков изменялась в зависимости от конфигурации исходной топологии так, чтобы суммарная скорость передаваемых через топологию потоков была равна пропускной способности канала, соединяющего вершины составляющих её деревьев.

2.7.5 Результаты экспериментов

Результаты проведённых экспериментов содержатся в таблицах 2.1 и 2.2.

Результаты 2.1 показывают, что вместе с увеличением количества коммутаторов указанной топологии оценки задержки передачи данных, полученные методом SFA увеличиваются линейно. Такой рост обусловлен взаимных влиянием потоков друг на друга – при построении оценки алгоритм SFA рассчитывает на максимальную активность каждого побочного потока, однако такое допущение здесь не оправдано. Прирост оценок задержки, полученных с помощью алгоритма LPA, так же линейный, но значительно более медленный. Значения реальных задержек, измеренных с помощью подготовленной имитационной

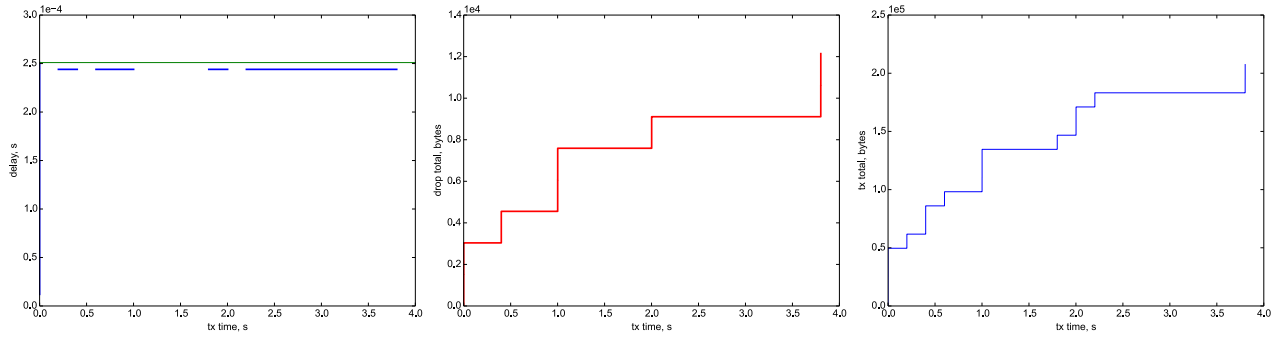

Рисунок 2.16: Задержка передачи данных, количество сброшенных и количество успешно доставленных пакетов при шейпинге потока алгоритмом текущего ведра с асимптотической скоростью 25 Mbps, размером всплеска 1 MTU и отсутствием буфера для временного хранения данных.

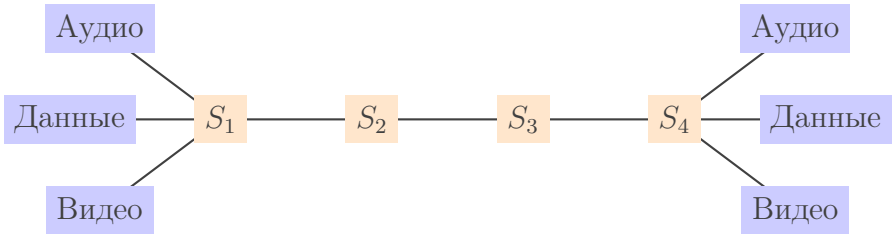

Рисунок 2.17: Пример топологии сети для последовательного тестового сценария.

модели, на несколько порядков меньше оценок, данных теоретическими методами – причины такого отставания указаны в конце раздела.

Результаты тестов на топологии с горловиной 2.2 демонстрируют достаточно высокую точность методов теоретической оценки при использовании небольшого количества потоков, однако вместе с увеличением количества потоков их оценки начинают расти быстрее результатов имитационной модели. Причина такого разрыва кроется в том, что чем боль-

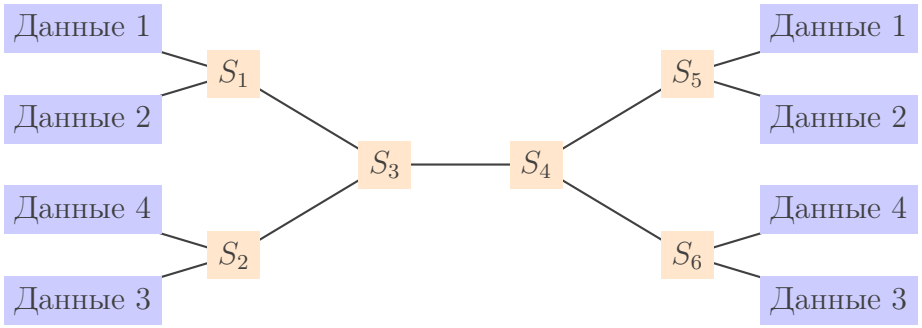

Рисунок 2.18: Пример топологии сети для тестового сценария с горловиной.

Таблица 2.1: Верхние оценки для задержки, вычисленные теоретически, и максимальные значения задержки, полученные с помощью имитационного моделирования, для линейной топологии, (мкс)

	Аудио			Видео			Данные		
Длина	NS3	SFA	LPA	NS3	SFA	LPA	NS3	SFA	LPA
2	159	12348	12348	244	11881	11881	244	9204	9204
3	281	16053	12519	459	23681	12045	366	18167	9334
4	403	23123	12689	488	35640	12209	515	27231	9457
5	525	33558	12860	692	47877	12373	639	36486	9584
6	647	47359	13031	733	60510	12537	733	46022	9711
7	769	64527	13201	891	73662	12701	962	55934	9838
8	891	85063	13372	998	87459	12866	977	66316	9964
9	1013	108972	13543	1099	102031	13030	1099	77270	10091

ше потоков передаётся через сеть, тем больше факторов должно совпасть, чтобы имитационная модель показала наибольшее время передачи пакета.

Приведённые в этом разделе значения задержки, полученные путём имитационного моделирования, нельзя напрямую сравнивать с оценками, данными теоретическими методами: имитационная модель даёт худшее время передачи пакета, которое было зафиксировано в течение одного из прогонов заданной конфигурации сети, в то время как теоретические методы дают верхнюю оценку указанного времени для наиболее «неудачного» прогона. Тем не менее, необходимо перечислить причины возникновения погрешности в приведённых теоретических методах.

Точность теоретических результатов заведомо снижена из-за аппроксимации потоков трафика. Предложенная математическая модель опирается в расчётах на параметры шейпинга поступающего в сеть трафика: считается, что всплеск и скорость потока могут достигать всплеска и скорости установленных шейпером. Однако в реальных сетях эти параметры выбираются с некоторым запасом. При описании моделей потоков было наглядно продемонстрировано, что голосовой трафик может быть ужат до $33,5$ Kbps, однако из-за требований к задержке предложенному алгоритму оценки задержки приходится допускать возникновение всплеска в 450 пакетов в каждый момент времени. При использовании, например, потоков TCP, которые в силу своей природы пытаются точно соответствовать заданной при шейпинге форме, точность метода теоретической оценки заметно возраста-

Таблица 2.2: Верхние оценки для задержки, вычисленные теоретически, и максимальные значения задержки, полученные с помощью имитационного моделирования, для топологии с горловиной, (мкс)

Глубина дерева	Кратность вершины	Скорость (Мб/с)	Задержка		
			NS3	SFA	LP
1	4	25	593	2584	2584
1	8	12.5	1058	8982	8982
1	16	6.25	1175	33590	33590
2	2	25	837	4771	3240
2	4	6.25	1535	39407	35710
3	3	12.5	1547	19186	11288
4	2	6.25	2721	63479	39998

ет. Поэтому перспективным направлением для исследований является уточнение профиля передаваемого трафика.

Кроме того, использованная в расчётах модель сети предполагала, что каждый пакет равен 1 MTU (1538 байт), в то время как пакеты телефонного трафика равны 238 байтам. Таким образом, результирующая теоретическая оценка для телефонного трафика проигрывает вплоть до 6 раз в задержке пакетизации на каждом исходящем интерфейсе. В то же время необходимо отметить, что абсолютное значение проигрыша зависит от скорости линий связи и будет снижаться при увеличении их производительности.

Заключение

Основные результаты работы заключаются в следующем:

1. Разработаны язык спецификации политик маршрутизации и реляционная модель поведения ПКС, в рамках которой строго описана семантика этого языка. Впервые было показано, что для задания статических свойств ПКС достаточно логики первого порядка с оператором транзитивного замыкания;

2. Построен алгоритм проверки соответствия конфигурации ПКС заданному множеству политик маршрутизации, для которого определена структура данных, позволяющая эффективно представлять отношения, моделирующие поведение ПКС;

3. Построена математическая модель функционирования ПКС с временем, в рамках которой поставлена задача и построены алгоритмы вычисления верхней оценки задержки передачи пакетов через последовательность коммутаторов с учётом текущей нагрузки на сеть;

4. Все разработанные алгоритмы были реализованы и прошли экспериментальную апробацию.

Представленные методы и средства могут служить базой для улучшения механизмов управления ПКС. В частности, они могут дать развитие по следующим направлениям:

Улучшение контроля безопасности. Средства верификации могут перехватывать команды управления оборудованием, генерируемые контроллером, или же отдельными управляющими приложениями, проверять их корректность и пропускать к коммутаторам лишь те из них, которые не нарушают требований действующих в сети политик маршрутизации. Таким образом, ошибки, возникшие внутри контура управления ПКС не достигнут её контура передачи данных, и инженеры смогут локализовать и исправить ошибку до того, как она приведёт к недовольству пользователей или нарушению работы сети;

Интеллектуальная адаптация контура управления. Разработанные средства верификации могут выполнять роль оракула, который способен проверять совместимость переданных ему команд со спецификой сети, к которой они применяются. Тем самым управляющим приложениям контроллера предоставляется возможность изменять своё поведение так, чтобы оно было не только наиболее эффективным для заданной конфигурации оборудования, но и удовлетворяло предъявляемым к нему требованиям;

Корректная композиция приложений. Построенные методы верификации можно использовать для обеспечения безопасной композиции независимых управляющих приложений, предназначенных для решения различных задач администрирования. Если каждое приложение контроллера специфицирует свои предположения о сетевом окружении в виде соответствующих формальных спецификаций, и одно из приложений попытается выполнить команду, которая приведёт к нарушению указанных требования, то будет выявлена ошибка совместимости приложений.

Повышение абстракции управления. Построенные средства могут использоваться в качестве интеллектуального ядра, способного синтезировать корректные последовательности низкоуровневых команд обновления конфигурации коммутаторов, соответствующие более высокоуровневым командам приложений. Например, если приложению необходимо проложить новый маршрут, в ряде случаев верификатор сможет вычислить такие правила и такую последовательность команд изменения конфигурации, при использовании которой действующие в сети политики маршрутизации не будут нарушены.

Приложение А

Описание синтаксиса и семантики логических формул для перспективных языков спецификации политик маршрутизации

А.1 Темпоральные логики

Формулы темпоральной логики CTL* описывают свойства размеченных систем переходов. Пусть задано некоторое множество атомарных высказываний (элементарных свойств состояний данных) AP. Размеченной системой переходов (моделью Крипке, LTS) M над множеством AP называется четверка $M = \langle S, S_0, R, L \rangle$, в которой:

- S – конечное множество состояний;
- $S_0 \subseteq S$ – множество начальных состояний;
- $R \subseteq S \times S$ – отношение переходов, которое обязано быть тотальным, то есть, для каждого состояния $s \in S$ должно существовать такое состояние $s' \in S$, для которого выполняется отношение $R(s, s')$;
- $L : S \to AP$ – функция, которая помечает каждое состояние множеством атомарных высказываний, истинных в этом состоянии.

Формулы темпоральной логики CTL* описывают свойства размеченной системы переходов (модели Крипке) $M = \langle S, S_0, R, L \rangle$ в терминах содержащихся в ней путей.

Путь в модели M из состояния s – это такая бесконечная последовательность состояний $\pi = s_0, s_1, s_2, \ldots$, что $s_0 = s$ и для всех $i \geq 0$ выполняется отношение $R(s_{i-1}, s_i)$.

Формулы CTL* состоят из кванторов состояния и темпоральных операторов. Кванторы состояния используются для указания того, что все пути или некоторые из путей, исходящие из некоторого заданного состояния, обладают предписанным свойством. Выделяют два квантора состояния:

- A (“для всех путей”) и
- E (“для некоторого пути”).

Темпоральные операторы описывают свойства путей из модели M. Выделяют пять основных темпоральных операторов:

- Оператор сдвига по времени X (neXttime, «в следующий момент») требует, чтобы свойство соблюдалось во втором состоянии пути;
- Оператор происшествия F (Future, «рано или поздно», «когда-то в будущем») применяется, если нужно объявить, что свойство будет соблюдаться в каком-то последующем состоянии пути;
- Оператор инвариантности G (Globally, «всегда», «повсюду»), позволяет заявить, что некоторое свойство соблюдается в каждом состоянии пути;
- Оператор условного ожидания U (Until, «до тех пор пока»), чуть более изощренный, поскольку в нем содержится комбинация двух свойств. Оператор выполняется, если на пути имеется состояние, в котором соблюдается второе свойство, и при этом каждое предшествующее на этом пути состояние обладает первым свойством;
- Оператор разблокировки R (Release, «высвободить»), логически двойственен оператору U. Он требует, чтобы второе свойство выполнялось на пути вплоть до такого состояния, в котором соблюдается первое свойство. Однако первое свойство не обязательно должно когда-нибудь быть выполнено.

В заключение этого раздела приведем строгое описание синтаксиса и семантики CTL*. В логике CTL* существуют формулы двух типов: *формулы состояния* (способны обращаться в истину в некотором состоянии) и *формулы пути* (способны быть истинными на протяжении некоторого пути).

Синтаксис формул состояния определяется следующими правилами:

- если $p \in AP$, то p – формула состояния;
- если f и g – формулы состояния, то $\neg f$, $f \wedge g$ и $f \vee g$ – формулы состояния;
- если f – формула пути, то Af и Ef – формулы состояния.

Синтаксис формул пути – правилами:

- если f – формула состояния, то f – формула пути;
- если f и g – формулы пути, то $\neg f$, $f \wedge g$, $f \vee g$, Xf, Ff, gf, fUg, fRg – формулы пути.

Семантику CTL* определим на моделях Крипке. Для пути $\pi = s_0, s_1, \ldots$ в модели M будем использовать запись π^i для обозначения суффикса π, начинающегося состоянием s_i:

- Если f – формула состояния, то запись $M, s \models f$ означает, что f выполняется в состоянии s в модели M.
- Если f – формула пути, то запись $M, \pi \models f$ означает, что f выполняется на протяжении пути π в модели M.

Отношение $\vDash$ определяется индуктивно следующим образом:

1. $M, s \vDash p \iff p \in L(s)$;
2. $M, s \vDash \neg f_1 \iff M, s \nvDash f_1$;
3. $M, s \vDash f_1 \vee f_2 \iff M, s \vDash f_1$ или $M, s \vDash f_2$;
4. $M, s \vDash f_1 \wedge f_2 \iff M, s \vDash f_1$ и $M, s \vDash f_2$;
5. $M, s \vDash Eg_1 \iff$ в есть такой путь π из состояния s, что $M, \pi \vDash g_1$;
6. $M, s \vDash Eg_1 \iff$ для любого пути π из состояния s в модели M верно соотношение $M, \pi \vDash g_1$;
7. $M, \pi \vDash f_1 \iff$ для первого состояния s на пути π в модели M верно соотношение $M, s \vDash f_1$;
8. $M, \pi \vDash \neg g_1 \iff M, \pi \nvDash g_1$;
9. $M, \pi \vDash g_1 \vee g_2 \iff M, \pi \vDash g_1$ или $M, \pi \vDash g_2$;

10. $M, \pi \vDash g_1 \wedge g_2 \iff M, \pi \vDash g_1$ и $M, \pi \vDash g_2$;

11. $M, \pi \vDash g_1 \iff M, \pi^1 \vDash g_1$;

12. $M, \pi \vDash Fg_1 \iff$ существует такое $k \geq 0$, что $M, \pi^k \vDash g_1$;

13. $M, \pi \vDash Gg_1 \iff$ для любого $k \geq 0$ верно соотношение $M, \pi^k \vDash g_1$;

14. $M, \pi \vDash g_1 U g_2 \iff$ существует такое $k \geq 0$, что $M, \pi^k \vDash g_2$ и для каждого $0 \leq j \leq k$ верно соотношение $M, \pi^j \vDash g_1$;

15. $M, \pi \vDash g_1 R g_2 \iff$ каково бы ни было $j \geq 0$, если для каждого $i < j$ верно соотношение $M, \pi^i \nvDash g_1$, то $M, \pi^j \vDash g_2$.

В логике CTL* можно выделить два подмножества, которые чаще всего используются в качестве языков спецификации поведения вычислительных систем: одна из этих логик называется *логикой ветвящегося времени, CTL*, а другая – *логикой линейного времени, LTL*. Различие между ними проявляется в том, как они относятся к альтернативным переходам из состояний системы переходов. В логике ветвящегося времени темпоральные операторы находятся непосредственно под действием кванторов пути. В логике линейного времени операторы предназначены для описания событий на протяжении единственного пути вычисления.

Логика ветвящегося времени (CTL) представляет собой фрагмент CTL*, в котором каждый темпоральный оператор X, F, G, U и R должен следовать непосредственно за квантором пути. Говоря более строго, CTL – это подмножество CTL*, в котором структура формул пути ограничена следующим правилом:

- Если f и g – формулы состояния, то Xf, Ff, gf, fUg, fRg – формулы пути.

Линейная темпоральная логика (LTL), в свою очередь, состоит из всех формул Af, где f – формула пути, в которой все формулы состояния – это атомарные высказывания. Более строгое определение формул пути LTL таково:

- Если $p \in AP$, то f – формула пути;
- Если f и g – формулы пути, то $\neg f$, $f \wedge g$, $f \vee g$, Xf, Ff, gGf, fUg, fRg – формулы пути.

A.2 Пропозициональное μ-исчисление

Как и темпоральные логики, формулы пропозиционального μ-исчисления интерпретируются на системах переходов, состояния которых размечены атомарными высказываниями из множества AP. Отличие заключается в том, что переходы этих систем имеют собственную разметку. Каждому переходу сопоставлено некоторое количество *действий* из множества $B = \{a_1, a_2, \ldots, a_n\}$. Поэтому вместо одного отношения переходов R удобно иметь дело с множеством отношений переходов T. Формально система переходов представляет собой тройку $M = \langle S, T, L \rangle$, где:

- S – непустое множество состояний;
- T – множество переходов, в которых для каждого действия $a \in B$, определено отношение $R_a \subseteq S \times S$;
- $L : S \to 2^{AP}$ – отображение, устанавливающие для каждого состояния множество истинных в нем атомарных высказываний.

В μ-исчислении вводится множество VAR *реляционных переменных*. Значениями реляционной переменной $Q \in VAR$ являются наборы состояний из S. Формулы μ-исчисления строятся по следующим правилам:

1. если $p \in AP$, то p – формула;
2. реляционная переменная является формулой;
3. если f и g – формулы, то $\neg f$, $f \wedge g$ и $f \vee g$ – формулы;
4. если f – формула и $a \in B$, то $[a]f$ и $\langle a \rangle f$ – формулы;
5. если $Q \in VAR$ и f – формула, то выражения $\mu Q.f$ и $\nu Q.f$ являются формулами при условии, что f *синтаксически монотонна* относительно Q, то есть все вхождения Q в f располагаются в областях действия четного числа связок отрицания.

Переменные μ-исчисления могут быть как *свободными*, так и *связанными* операторами неподвижной точки. *Замкнутой формулой* называется формула, не имеющая свободных переменных. Для обозначения того, что формула f содержит свободные реляционные переменные $Q_1, \ldots, Q_n$, используется запись $f(Q_1, \ldots, Q_n)$.

Содержательный смысл формулы $\langle a \rangle f$ таков: «может быть совершен a-переход в состояние, в котором выполняется f». Аналогично формула $[a]f$ гласит: «f выполняется во

всех состояниях, достижимых (за один шаг) a-переходом». Операторы μ и ν используются для обозначения наименьшей и наибольшей неподвижной точки. Пустое множество состояний будем обозначать $False$, а все множество состояний S будем обозначать $True$.

Формула f интерпретируется как множество состояний, на которых f обращается в истину. Такое множество будем обозначать $[\![f]\!]_{Me}$, где M – система переходов, а $e : VAR \to 2^S$ – *окружение*. Запись $e[Q \leftarrow W]$ служит обозначением нового окружения, отличающегося от e только тем, что $e[Q \leftarrow W](Q) = W$. Множество $[\![f]\!]_{Me}$ имеет следующее рекурсивное определение:

- $[\![p]\!]_{Me} = \{s \mid p \in L(s)\}$;
- $[\![Q]\!]_{Me} = e(Q)$;
- $[\![\neg f]\!]_{Me} = S \setminus [\![f]\!]_{Me}$;
- $[\![f \wedge g]\!]_{Me} = [\![f]\!]_{Me} \cap [\![g]\!]_{Me}$;
- $[\![f \vee g]\!]_{Me} = [\![f]\!]_{Me} \cup [\![g]\!]_{Me}$;
- $[\![\langle a \rangle f]\!]_{Me} = \{s \mid \exists t : \ (s,t) \in R_a \wedge t \in [\![f]\!]_{Me}\}$;
- $[\![[a] f]\!]_{Me} = \{s \mid \forall t : \ (s,t) \in R_a \Rightarrow t \in [\![f]\!]_{Me}\}$;
- $[\![\mu Q.f]\!]_{Me}$ – наименьшая неподвижная точка преобразователя предикатов $\tau : \ 2^S \to 2^S$, которая определяется соотношением: $\tau(W) = [\![f]\!]_{Me[Q \leftarrow W]}$;
- $[\![\nu Q.f]\!]_{Me}$ – наибольшая неподвижная точка преобразователя предикатов $\tau : \ 2^S \to 2^S$, которая определяется соотношением: $\tau(W) = [\![f]\!]_{Me[Q \leftarrow W]}$.

Формула μ-исчисления f считается выполнимой в состоянии s системы переходов M, если $s \in [\![f]\!]_{Me}$. Множество формул считается *противоречивым*, если не существует такой модели, в одном из состояний которой выполнялись бы все формулы этой системы.

A.3 Логика первого порядка с оператором транзитивного замыкания

Для формального описания статических свойств поведения ПКС могут быть также пригодны разрешимые фрагменты логики предикатов первого порядка, обогащенные дополнительными операторами (например, оператором транзитивного замыкания [64] или

оператором вычисления неподвижной точки [109]). Логика первого порядка с оператором транзитивного замыкания ($FO[TC]$) используется для формального определения свойств достижимости в структурах с одним или несколькими бинарными отношениями. Синтаксис $FO[TC]$ определяется следующим образом. Пусть задано множество предметных переменных X и два множества предикатных символов AP и B. Элементы множества AP – атомарные высказывания, а элементы множества $B = \{a_1, a_2, \ldots, a_n\}$ – двухместные отношения (атомарные действия). Формулы $FO[TC]$ определяются по следующим правилам:

- если p – атомарное высказывание, и x – предметная переменная, то выражение $p(x)$ является формулой $FO[TC]$ с множеством свободных переменных $\{x\}$;
- если a – атомарное действие, и x, y – предметные переменные, то выражения (x, y) и $x = y$ являются формулами $FO[TC]$ с множеством свободных переменных $\{x, y\}$;
- если f и g – формулы $FO[TC]$ с множествами свободных переменных X_1 и X_2, соответственно, то $\neg f$, $f \wedge g$, $f \vee g$ – формулы $FO[TC]$ с множествами свободных переменных X_1, $X_1 \cap X_2$ и $X_1 \cup X_2$ соответственно,
- если f – формула $FO[TC]$ множеством свободных переменных X, и x –предметная переменная, то $\forall x.f$ и $\exists x.f$ – формулы $FO[TC]$ множеством свободных переменных $X \setminus \{x\}$,
- если f – формула $FO[TC]$ с двумя свободными переменными, то $TC[f]$ – формула $FO[TC]$ с двумя свободными переменными (транзитивное замыкание отношения f).

Семантика логики $FO[TC]$ задается на размеченных системах переходов (моделях Крипке). Наибольший интерес представляет фрагмент логики $FO[TC]$, в котором каждая формула содержит не более двух свободных переменных x, y. Такой фрагмент обозначим записью $FO_2[TC]$. Как и в случае μ-исчисления, формула $f(x)$ фрагмента $FO_2[TC]$ интерпретируется как множество состояний, на которых f обращается в истину, а формула $g(x, y)$ фрагмента $FO_2[TC]$ – как множество упорядоченных пар состояний, которые удовлетворяют бинарному отношению, выражаемому этой формулой. Такие множества будем обозначать $[\![f]\!]_{Me}$ и $\langle\langle g \rangle\rangle_{Me}$, где M – система переходов. Множества $[\![f]\!]_{Me}$ и $\langle\langle g \rangle\rangle_{Me}$ имеют следующее рекурсивное определение на основании отношения выполнимости $M \vDash \phi[s', s'']$, где s', s'' – состояния системы переходов M.

- Если p – атомарное высказывание, то

- $M \vDash p(x)[s', s''] \iff s' \in L(p)$,
- $M \vDash p(y)[s', s''] \iff s'' \in L(p)$;

- Если a – базовое действие, то $M \vDash a[s', s''] \iff \langle s', s'' \rangle \in a$;
- $M \vDash (x = y)[s', s'] \iff$ состояния s' и s'' совпадают;
- $M \vDash (\neg\phi)[s', s''] \iff$ неверно, что $M \vDash \neg\phi[s', s'']$;
- $M \vDash (\phi \wedge \psi)[s', s''] \iff M \vDash \phi[s', s'']$ и $M \vDash \psi[s', s'']$;
- $M \vDash (\phi \vee \psi)[s', s''] \iff M \vDash \phi[s', s'']$ или $M \vDash \psi[s', s'']$;
- $M \vDash \forall x.\phi[s', s''] \iff$ для любого состояния s верно $M \vDash \phi[s, s'']$;
- $M \vDash \forall y.\phi[s', s''] \iff$ для любого состояния s верно $M \vDash \phi[s', s]$;
- $M \vDash \exists x.\phi[s', s''] \iff$ для некоторого состояния s верно $M \vDash \phi[s, s'']$;
- $M \vDash \exists y.\phi[s', s''] \iff$ для некоторого состояния s верно $M \vDash \phi[s', s]$;
- $M \vDash TC[\phi][s', s''] \iff$ существует последовательность состояний $s_1, s_2, \ldots, s_n$, удовлетворяющая условиям 1) $s_1 = s'$, 2) $s_n = s''$, 3) для любого i, $1 \leq i < n$ выполняется отношение $M \vDash \phi[s_i, s_{i+1}]$.

Приложение Б

Описание некоторых перспективных формальных моделей ПКС

Б.1 Модели Крипке

С практической точки зрения размеченная система переходов или модель Крипке – конечный автомат, расширенный функцией разметки, которую удобно использовать при описании реагирующих систем. Более формально, моделью Крипке M над множеством атомарных высказываний (булевых переменных) AP и множеством элементарных действий EA называется четверка $M = (S, S_0, R, L)$, в которой:

S – конечное множество состояний;

$S_0 \subseteq S$ – подмножество начальных состояний;

$R \subseteq S \times S$ – отношение переходов;

$L: S \to 2^{AP}$ – функция, которая помечает каждое состояние множеством атомарных высказываний, истинных в этом состоянии.

Вычислением (маршрутом, трассой) в модели M из состояния s называется такая (возможно, бесконечная) последовательность состояний $\pi = s_0 s_1 s_2 \ldots$, что $s_0 = s$ и для каждого $i \geq 0$ выполняется отношение $R(s_i, s_{i+1})$.

Размеченные системы переходов – это простейшая и в то же время наиболее часто используемая модель вычислений. Данная модель пригодна для описания поведения как последовательных, так и параллельных вычислительных систем. Если вычислительная система состоит из нескольких параллельно работающих процессов, для описания ее по-

ведения посредством размеченных систем переходов вычисление представляется множеством последовательностей чередующихся действий, выполняемых отдельными процессами. Размеченные системы переходов допускают символьное представление, в котором отношение переходов R и функция разметки L описываются булевыми формулами и OBDD.

Размеченные системы переходов выступают в роли интерпретаций (моделей Крипке) для многих темпоральных логик (CTL, LTL, $CTL*$, $TCTL$, μ-исчисления). Для этих логик разработаны разнообразные средства проверки выполнимости формул на конечных моделях Крипке (например NuSMV [33]).

Посредством размеченных систем переходов можно определять семантику многих моделей вычислений, включая другие формальные модели (алгебры процессов, автоматы, сети Петри). Для моделей Крипке разработаны различные отношения симуляции, которые позволяют корректно определять отношения абстракции и конкретизации как для самих размеченных систем переходов, так и для тех моделей вычислений, семантика которых основывается на размеченных системах переходов.

Б.2 Конечные автоматы реального времени

Формальное описание модели конечных временных автоматов таково: пусть заданы множество таймеров $Clocks$, множество целочисленных переменных Var и множество каналов связи Ch. Над этими множествами определены множества инвариантов $Invariants$, предусловий $Guards$ и действий $Acts$, включающее действия отправления синхронизирующего сигнала $c!$, приема сигнала $c?$ сброса таймера $t := 0$ и присваивания $x := E$.

Временной автомат – это система $A = (L, l_0, Clocks, Vars, Ch, T, I)$, где:

L – конечное множество (локальных) состояний управления;

l_0 – начальное состояния управления;

$Clocks$, Var, Ch – множества таймеров, переменных и каналов связи;

$T, T \subseteq L \times Guards \times Acts \times L$ – множество переходов;

$I: L \to Invariants$ – назначения инвариантов.

Переход $l' \xrightarrow{g,a} l''$ считается активным при заданной оценке таймеров τ, если предусловие g перехода выполняется для этой оценки. При срабатывании активного перехода автомат изменяет своё текущее состояние управления l' на состояние l'' и выполняется

действие a. Автомат может пребывать в состоянии управления l при оценке таймеров τ только в том случае, когда инвариант $I(l)$, приписанный состоянию управления l, выполняется для оценки τ.

Несколько конечных автоматов могут вступать во взаимодействие друг с другом посредством действий синхронизации, образуя параллельную композицию (или иначе – сеть) конечных временных автоматов N:

$$N = \left\{ A_i = \langle L_i, l_0^i, Clocks, Vars, Ch, T_i, I_i \rangle,\ 1 \leq i \leq n \right\}$$

Отдельные автоматы $A_1, \ldots, A_n$ сети имеет непересекающиеся множества локальных состояний управления, но общие множества таймеров, переменных и каналов связи. Наборы $\bar{l} = (l_1, l_2, \ldots, l_n)$ локальных состояний автоматов сети образуют глобальные состояния управления сети. Набор $\bar{l_0} = (l_0^1, l_0^2, \ldots, l_0^n)$ является начальным состоянием управления сети.

Поведение сети временных автоматов N определяется системой переходов $M_N = \langle S, \bar{s}_0, \rightarrow \rangle$, которая образована множеством состояний вычисления сети автоматов $S = (L_1 \times L_2 \times \cdots \times L_n) \times \mathbb{Z}_{Vars} \times \mathbb{R}_{Clocks}$ с выделенным начальным состоянием вычисления $\bar{s}_0 = (\bar{l}_0, \mu_0, \tau_0)$, а также отношением вычислительных переходов $\rightarrow \subseteq S \times S$. Каждое состояние вычисления определяется: глобальным состоянием управления сети автоматов, значением переменных и показанием таймеров. Отношение $(l_1', l_2', \ldots, l_n', \mu', \tau') \rightarrow (l_1'', l_2'', \ldots, l_n'', \mu'', \tau'')$ определяется следующими тремя правилами:

Продвижение времени. Показания всех таймеров увеличиваются на одну и ту же положительную величину d так, чтобы инварианты $I_i(l_i'), 1 \leq i \leq n$, выполнялись для новой оценки таймеров $\tau'' = \tau' + d$.

Внутренние действия. Выполняется активный переход $l' \xrightarrow{g,a} l''$ с внутренним действием a одного из автоматов A_j. Оценки переменных и таймеров изменяются в соответствии с действием . Оценка таймеров и локальные состояния остальных автоматов остаются неизменными. То есть $\tau'' = \tau'$ и $l_i'' = l_i'$ для всех $i, i \neq j$. При выполнении этого перехода должны соблюдаться: инвариант $I_j(l_j'')$, и все инварианты $I_i(l_i'), i \neq j$.

Синхронизация. Выполняются два перехода $l' \xrightarrow{g,c!} l''$ и $l' \xrightarrow{g,c?} l''$ с действиями отправления и приема сигнала по каналу c. Оценки переменных и таймеров, а также локальные состояния управления остальных автоматов остаются неизменными. При этом должны соблюдаться: инварианты $I_j(l_j'')$, $I_k(l_k'')$, а также все инварианты $I_i(l_i'), i \neq j, i \neq k$.

Вычислением сети временных автоматов N называется всякая максимальная последовательность $run = s_0 \to s_1 \to s_2 \to \cdots \to s_i \to s_{i+1} \to \cdots$ состояний вычисления, являющаяся маршрутом в графе, соответствующем системе переходов M_N.

Модель конечных автоматов реального времени (временных автоматов), предложена Р. Алуром и Д. Диллом [110—113]. Свойства поведения автоматов реального времени описываются формулами темпоральной логики TCTL (Timed Computation Tree Logic) [114]). Эффективный математический метод решения задач анализа поведения конечных автоматов реального времени был разработан Р. Алуром и Т. Хензингером [115; 116]. Эта модель вычислений нашла широкое применение при решении задач верификации информационных систем, в которых длительность и сроки выполнения имеют ключевое значение. На основе этих алгоритмов было разработано множество программно-инструментальных средств верификации программ, наиболее завершенными из которых оказались системы UPPAAL [34], HyTech [117] и Kronos [118]. Во всех трех системах программы моделируются сетями взаимодействующих временных автоматов, для которых строятся конечные системы переходов. Отличия указанных систем заключаются в форме описания систем переходов, допустимых множеств формул логики TCTL и алгоритмах верификация моделей.

Б.3 Алгебры процессов

Исчисление взаимодействующих систем (CCS), а также родственное ему исчисление последовательных процессов (CSP) и алгебра взаимодействующих процессов (ACP) были предложены в статьях [119; 120] в качестве простой и универсальной алгебраической системы для описания эффектов параллельного выполнения и коммуникационного взаимодействия процессов в распределенных системах вычислений.

Процессы в этих алгебрах – это выражения, которые образуются из базовых процессов (действий) множества $A = \{a_1, a_2, \ldots\}$ и пустого процесса $\emptyset$ посредством операций последовательной композиции $P \cdot Q$, параллельной композиции $P \mid Q$ и недетерминированного выбора $P + Q$. Обычно на множестве базовых действий вводится некоторое бинарное отношение взаимодействия T: если $(a, b) \in T$, то эта пара действий обеспечивает синхронизацию процессов при их параллельном выполнении.

Семантика алгебры процессов сопоставляет каждому процессу размеченную систему переходов, состояниями которой служат процессы (исходный процесс является начальным

состоянием). На множестве всех процессов вводится размеченное отношение переходов, которое определяется следующими правилами:

- $a \xrightarrow{a} \emptyset$;
- если $P \xrightarrow{x} P'$ то:
  - $P \cdot Q \xrightarrow{x} P' \cdot Q$;
  - $P \mid Q \xrightarrow{x} P' \mid Q$;
  - $P + Q \xrightarrow{x} P'$;
- если $(a, b) \in T$, то $a \cdot P \mid b \cdot Q \xrightarrow{\tau} P' \cdot Q$, где τ – особо выделенное невидимое действие.

На основе алгебр взаимодействующих процессов был разработан формальный язык спецификаций LOTOS (Language Of Temporal Ordering Specification). Для анализа поведения моделей параллельных вычислений, представленных на основе алгебр процессов, были также разработаны некоторые программно-инструментальные средства, такие как PAT [121] и TAPAS [122].

Б.4 Сети Петри

Сети Петри – это наиболее известный класс математических моделей систем с параллельно функционирующими асинхронно взаимодействующими компонентами.

Обыкновенная сеть Петри – это набор $N = (P, T, F, M_0)$, где:

P – непустое множество элементов сети, называемых местами;

T – непустое множество элементов сети, называемых переходами;

$F: P \times T \cup T \times P \to \mathbb{N}$ – функция инцидентности;

$W_0: P \to \mathbb{N}$ – начальная разметка.

Элементы множества P называются *местами*, а элементы множества T – *переходами*. *Разметкой сети* M называется всякое отображение $P \to \mathbb{N}$. Пометки, которые функция M сопоставляет местам сети, обычно называют «*фишками*». Для перехода t, $t \in T$, запись $\bullet t$ обозначает функцию $\bullet t\ P \to \mathbb{N}$, удовлетворяющую равенству $\bullet t(p) = F(p, t)$ для каждого места p, $p \in P$. Запись $t\bullet$ обозначает функцию $t\bullet: P \to \mathbb{N}$, удовлетворяющую равенству $t\bullet(p) = F(t, p)$ для каждого места p, $p \in P$.

Для заданной разметки M сети N переход t считается активным, если для каждого места p выполняется неравенство $\bullet t(p) \leq M(p)$. Каждый переход t сети N задает на множестве разметок сети отношение *срабатывания* $\xrightarrow{t}$, которое определяется следующими требованиями: для любой пары разметок M и M' сети N соотношение $M \xrightarrow{t} M'$ выполняется в том и только том случае, если:

- переход t активен для разметки M;
- для любого места p, $p \in P$ верно равенство $M'(p) = M(p) - \bullet t(p) + t\bullet(p)$.

Вычислением сети $N = (P, T, F, M_0)$ называется всякая последовательность срабатываний переходов: $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} \cdots \xrightarrow{t_n} M_n$. Разметка M_n называется *достижимой* из начальной разметки M_0, если существует вычисление сети N, оканчивающееся разметкой M_n. Проблема достижимости для обыкновенных сетей Петри разрешима [123; 124] и является EXPSPACE-трудной [125]: сложность всех известных алгоритмов ее решения оценивается непримитивно рекурсивной функцией. Для некоторых ограниченных классов обыкновенных сетей Петри проблема достижимости имеет меньшую сложность, являясь PSPACE-полной, EXP-полной, или даже полиномиально разрешимой задачей [126]. Применение метода абстракции позволяет конструировать полиномиальные по времени полуразрешающие процедуры проверки достижимости разметки для сетей Петри [127].

Для построения моделей систем взаимодействующих процессов наиболее часто используются расширенные варианты сетей Петри. Одним из них являются раскрашенные сети Петри [128]. В отличие от обыкновенных сетей Петри в качестве средств разметки (фишек) в раскрашенных сетях Петри используются мультимножества элементов конечного числа типов. Раскрашенные сети Петри более удобны для описания систем параллельных вычислений, хотя их выразительные возможности такие же, как и у обыкновенных сетей Петри.

Еще одной разновидностью расширенных сетей Петри являются вложенные сети Петри [129]. В качестве фишек во вложенных сетях Петри также выступают сети Петри. Объемлющая сеть Петри называется *системной сетью*, а вложенные в нее сети, используемые в качестве фишек – *сетевыми фишками*. Вложенность сетей Петри может быть многоуровневой – сетевые фишки также могут быть вложенными сетями. Функционирование вложенных сетей Петри определяется на основе тех же принципов, что и вычисления обыкновенных сетей Петри. Основное отличие, увеличивающее выразительные возможности этой модели, является наличие синхронизации переходов различных сетей. Возможны

два типа синхронизации – *вертикальная синхронизация* и *горизонтальная синхронизация*.

При вертикальной синхронизации между некоторыми переходами системной сети и сетевых фишек устанавливается связь, в силу которой переходы в них обязаны срабатывать одновременно. Подобным же образом определяется и горизонтальная синхронизация. Однако на этот раз связь устанавливается между некоторыми переходами в сетевых фишках, а переходы в них не только обязаны срабатывать одновременно, но и срабатывают лишь в том случае, если обе сетевые фишки располагаются в одном и том же месте системной сети.

Подобно тому, как это было сделано для конечных автоматов, в аппарат сетей Петри можно добавить средства хронометрирования (таймеры) и операции над ними (сравнения и сбрасывания показаний таймеров). Сети Петри такого рода называются временными сетями Петри. Они используются в качестве математических моделей встроенных систем вычислений реального времени.

Для сетей Петри разработаны алгоритмы и программно-инструментальные средства верификации. Обычно сети Петри N сопоставляется система переходов – граф, вершинами которого являются возможные разметки, а дугами – отношение срабатывания переходов сети Петри. Спецификация поведения сетей Петри обычно описывается некоторой темпоральной логикой. Поскольку множество достижимых разметок сети N может быть бесконечным, то обычно проводится лишь ограниченная верификация. Наиболее известные из средств анализа поведения сетей Петри – TAPAAL [130] и Romeo [131].

Приложение B

Некоторые теоремы сетевого исчисления

Теорема (Оценка отставания)**.** Пусть поток данных с функцией прибытия $A \in \mathcal{F}$, ограниченный кривой нагрузки $\alpha \in \mathcal{F}$, обслуживается обработчиком с кривой сервиса β. Тогда значение отставания обработчика не превышает вертикального отклонения между кривыми прибытия α и сервиса β:

$$\forall t \in \mathbb{R} : \; b(t) \leq v(\alpha, \beta) = \sup_{t \geq s \geq 0} \{\alpha(s) - \beta(s)\}$$

Доказательство. В соответствии с определениями кривой прибытия и кривой сервиса справедливы выражения:

$$b(t) = A(t) - D(t) \leq A(t) - \inf_{0 \leq s \leq t} \{A(s) + \beta(t-s)\}$$

$$b(t) \leq \sup_{0 \leq s \leq t} \{A(t) - A(s) - \beta(t-s)\} \leq \sup_{0 \leq s \leq t} \{\alpha(t-s) - \beta(t-s)\}$$

$$b(t) \leq \sup_{0 \leq s \leq t} \{\alpha(s) - \beta(s)\}$$

□

Теорема (Оценка задержки)**.** Пусть поток данных с функцией прибытия $A \in \mathcal{F}$, ограниченный кривой нагрузки $\alpha \in \mathcal{F}$, обслуживается обработчиком с кривой сервиса $\beta \in \mathcal{F}$ по дисциплине FIFO. Тогда значение задержки обслуживания этого потока не превышает горизонтального отклонения между кривыми прибытия α и сервиса β:

$$\forall t \in \mathbb{R} : d(t) \leq h(\alpha, \beta) = \sup_{t} \{\inf \{\tau \geq 0 | \alpha(t) \leq \beta(t+\tau)\}\}$$

Если дисциплина обслуживания неизвестна, то для задержки d(t) справедлива оценка:

$$d(t) \leq \inf \{\tau \geq 0 | \alpha(\tau) \leq \beta(\tau)\}$$

Доказательство. Рассмотрим случай обработчика с дисциплиной обслуживания FIFO. Выберем произвольный момент времени $t \geq 0$ и любое $\tau_0 < d(t)$. Тогда по определению задержки выполняется неравенство $A(t) > D(t + \tau_0)$. При этом согласно определению кривой сервиса β существует такой момент времени $s_0 \in \mathbb{R}$, что:

$$D(t + \tau_0) \geq \inf_{s \leq t+\tau} \{A(t + \tau_0 - s) + \beta(s)\} \geq A(t + \tau_0 - s_0) + \beta(s_0)$$

В результате для функции прибытия A справедливо следующее условие:

$$A(t) - A(t + \tau_0 - s_0) > \beta(s_0)$$

В то же время из определения кривой прибытия следует соотношение:

$$\alpha(s_0 - \tau_0) \geq A(t) - A(t - (s_0 - \tau_0)) > \beta(s_0)$$

$$\alpha(s_0 - \tau_0) > \beta(s_0)$$

Теперь рассмотрим функцию $delta(t)$ в точке $s_0 - \tau_0$:

$$delta(t) = \inf \{\tau \geq 0 | \alpha(t) \leq \beta(t + \tau)\}$$

$$delta(s_0 - t_0) = \inf \{\tau \geq 0 | \alpha(s_0 - \tau_0) \leq \beta(s_0 - \tau_0 + \tau)\}$$

В силу построенного ранее соотношения между кривой прибытия α и кривой сервиса β характеристическое свойство множества в правой части указанного выражения может выполняться лишь при $\tau > \tau_0$. Таким образом, для любых t и $\tau_0 < d(t)$ имеет место неравенство $\tau_0 < \sup delta(t) = h(\alpha, \beta)$. Отсюда следует, что для значения задержки $d(t)$, в свою очередь, справедлива искомая оценка: $d(t) \leq h(\alpha, \beta)$.

Теперь рассмотрим случай неизвестной дисциплины обслуживания. По определению задержки, в этом случае, справедлива формула:

$$d(t) \leq \inf \{\tau \geq 0 | A(t + \tau) = D(t + \tau)\}$$

Получим из неё константную верхнюю оценку задержки, не зависящую от времени, заметив, что параметр τ принимает максимальное значение в начале периода отставания $SBP(t)$:

$$d(t) \leq \sup_t \inf \{\tau \geq 0 | A(SBP(t) + \tau) = D(SBP(t) + \tau)\}$$

$$d(t) \leq \sup_t \inf \{\tau \geq 0 | A(SBP(t) + \tau) - A(SBP(t)) = D(SBP(t) + \tau) - D(SBP(t))\}$$

Поскольку характеристическое свойство множества заведомо выполняется при более строгом условии $\alpha(\tau) \leq \beta(\tau)$, то для задержки справедлива следующая оценка:

$$d(t) \leq \inf \{\tau \geq 0 | \alpha(\tau) \leq \beta(\tau)\}$$

□

Теорема (Оценка выходного потока)**.** Если поток функцией прибытия $A \in \mathcal{F}$, ограниченный кривой нагрузки $\alpha \in \mathcal{F}$, поступает на обработчик с кривой сервиса $\beta \in \mathcal{F}$, то полученный в результате выходной поток ограничен кривой нагрузки $\alpha' \in \mathcal{F}$:

$$\alpha'(s) = \sup_{\tau \geq 0} \{\alpha(s+\tau) - \beta(\tau)\}$$

$$D(t) - D(s) \leq \alpha'(t-s)$$

Доказательство. По определению кривой сервиса в каждый момент времени t, где $t - s \geq 0$, существует такой интервал τ, где $\tau \leq t - s$, что:

$$D(t-s) \geq D(t-s-\tau) + \beta(\tau)$$

$$D(t) - D(t-s) \leq D(t) - D(t-s-\tau) - \beta(\tau)$$

В каждый момент времени объём переданных данных не превышает объёма полученных, поэтому выполняется:

$$D(t) - D(t-s) \leq A(t) - A(t-s-\tau) - \beta(\tau) \leq \alpha(s+\tau) - \beta(\tau)$$

$$D(t) - D(t-s) \leq \sup_{\tau \geq 0} \{\alpha(s+\tau) - \beta(\tau)\}$$

Таким образом, записанная в правой части последнего выражения функция является кривой нагрузки для потока передаваемых обработчиком данных. □

Список рисунков

Список таблиц

Литература

1. *Gilder G.* Telecosm: How Infinite Bandwidth Will Revolutionize Our World. — Free Press, 2000.
2. *Metcalfe B.* Metcalfe's Law after 40 Years of Ethernet // Computer. — 2013. — Dec. — Vol. 46, no. 12. — Pp. 26–31.
3. *Clark D.* The Design Philosophy of the DARPA Internet Protocols // Symposium Proceedings on Communications Architectures and Protocols, (SIGCOMM '88). — ACM, 1988. — Pp. 106–114.
4. *Vojdak W.* Rapid Spanning Tree Protocol: A new solution from an old technology // Compact PCI systems. — 2008.
5. *Devaney R.* An Introduction to Chaotic Dynamical Systems, Addison-Wesley studies in nonlinearity. — Westview Press, 2003.
6. *Al-Shaer E.*, *Al-Haj S.* FlowChecker: Configuration Analysis and Verification of Federated Openflow Infrastructures // Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration, (SafeConfig '10). — Chicago, Illinois, USA, 2010. — Pp. 37–44.
7. *Dougherty D. J.*, *Fisler K.*, *Krishnamurthi S.* Specifying and reasoning about dynamic access-control policies // of Lecture Notes in Computer Science. — Springer, 2006. — Pp. 632–646.
8. *Qie X.*, *Narain S.* Using Service Grammar to Diagnose BGP Configuration Errors // Proceedings of the 17th USENIX Conference on System Administration, (LISA '03). — San Diego, CA : USENIX Association, 2003. — Pp. 237–246.
9. *Open Networking Foundation* Software-Defined Networking: The New Norm for Networks: White paper. — 2012.

10. *Bloem R.*, *Chatterjee K.*, *Henzinger T. A.*, *Jobstmann B.* Better Quality in Synthesis through Quantitative Objectives // Computer Aided Verification, Lecture Notes in Computer Science. — 2009. — Vol. 5643. — Pp. 140–156.

11. *Lamport L.* Time, Clocks, and the Ordering of Events in a Distributed System // Commun. ACM. — 1978. — July. — Vol. 21, no. 7. — Pp. 558–565.

12. *Clarke E. M.*, *Emerson E. A.*, *Sistla A. P.* Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications // ACM Trans. Program. Lang. Syst. — 1986. — Apr. — Vol. 8, no. 2. — Pp. 244–263.

13. *Henzinger T. A.*, *Nicollin X.*, *Sifakis J.*, *Yovine S.* Symbolic Model Checking for Real-time Systems // Inf. Comput. — 1994. — June. — Vol. 111, no. 2. — Pp. 193–244.

14. *Cruz R.* A Calculus for network delay. II. Network analysis // IEEE Transactions on Information Theory. — 1991. — Jan. — Vol. 37, no. 1. — Pp. 132–141.

15. *Le Boudec J.-Y.*, *Thiran P.* Network Calculus: A Theory of Deterministic Queuing Systems for the Internet. — Springer-Verlag, 2001. — URL: http://ica1www.epfl.ch/PS_files/netCalBookv4.pdf.

16. *Chemeritskiy E.*, *Smelyansky R.*, *Zakharov V.* A Formal Model and Verification Problems for Software Defined Networks // Proceedings of the 4-th International Workshop “Program Semantics, Specification and Verification: Theory and Applications”. — Yekaterinburg, Russia, 2013. — Pp. 21–30.

17. *Захаров В.*, *Смелянский Р.*, *Чемерицкий Е.* Формальная модель и задачи верификации программно-конфигурируемых сетей // Моделирование и анализ информационных систем. — 2013. — Т. 20, № 6. — С. 33—48.

18. *Chemeritsky E.*, *Smeliansky R.*, *Zakharov V.* A formal model and verification problems for Software Defined Networks // Automatic Control and Computer Sciences. — 2014. — Vol. 48, no. 7.

19. *Chemeritskiy E.*, *Zakharov V.* On the Network Update Problem for Software Defined Networks // Proceedings of the 5-th Workshop “Program Semantics, Specification and Verification: Theory and Applications”. — Moscow, Russia, June 4, 2014. — Pp. 26–37.

20. *Захаров В.*, *Чемерицкий Е.* О некоторых задачах реконфигурирования программно-конфигурируемых сетей // Моделирование и анализ информационных систем. — 2014. — Т. 21, № 6.

21. *Chemeritskiy E.*, *Smelansky R.* On QoS management in SDN by multipath routing // Proceedings 2014 international science and technologe conference “Modern Networking Technologies (MoNeTec)”. — Москва, Россия : Макс Пресс Москва, 27—29 окт. 2014. — С. 41—46.

22. *Altukhov V.*, *Chemeritskiy E.* On real-time delay monitoring in Software-Defined Networks // Porceedings 2014 international science and technologe conference “Modern Networking Technologies (MoNeTec)”. — Москва, Россия : Макс Пресс Москва, 27—29 окт. 2014. — С. 1—6.

23. *Altukhov V.*, *Chemeritskiy E.*, *Podymov V.*, *Zakharov V.* VERMONT - a toolset for checking SDN packet forwarding policies on-line // Proceedings of 2014 International Science and Technology Conference “Modern Networking Technologies (MoNeTec)”. — Москва, Россия : Макс Пресс Москва, 27—29 окт. 2014. — С. 7—12.

24. *Chemeritskiy E.*, *Zakharov V.* Consistent network update without tagging // Proceedings of 2014 International Science and Technology Conference “Modern Networking Technologies (MoNeTec)”. — Москва, Россия : МАКС Пресс Москва, 27—29 окт. 2014. — С. 47—52.

25. *Altukhov V.*, *Chemeritskiy E.*, *Podymov V.*, *Zakharov V.* A runtime verification system for Software Defined Networks // Материалы Международной научно-практической конференции “Инструменты и методы анализа программ - Tools & Methods of Program Analysis (TMPA-2014)”. — Кострома, Россия : Костромской государственный технологический университет Кострома, 14—15 нояб. 2014. — С. 19—28.

26. *Алтухов В.*, *Захаров В.*, *Подымов В.*, *Чемерицкий Е.* VERMONT - средство верификации программно-конфигурируемых сетей // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. — 2015. — Т. 212, № 1. — С. 74—87.

27. *Atlas A.*, *Halpern J.*, *Hares S.*, *Ward D.*, *Nadeau T.* An Architecture for the Interface to the Routing System: IETF Network Working Group Internet-Draft. — Mar. 6, 2015. — URL: https://datatracker.ietf.org/doc/draft-ietf-i2rs-architecture/.

28. *Pfaff B.*, *Davie B.* The Open vSwitch Database Management Protocol: IETF Request for Comments: 7047. — 2013. — URL: https://tools.ietf.org/html/rfc7047.

29. *Foundation O. N.* OpenFlow Switch Specification: tech. rep. — Oct. 14, 2013. — URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

30. *Царьков Д.* Система формальной верификации распределённых программ в среде имитационного моделирования DYANA // Труды международной конференции «Параллельные вычисления и задачи управления», (PACO '01). — Институт проблем управления им. В.А. Трапезникова РАН, 2—4 окт. 2001. — С. 161—182.

31. *Clarke E. M.*, *Emerson E. A.*, *Sistla A. P.* Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications // ACM Trans. Program. Lang. Syst. — 1986. — Apr. — Vol. 8, no. 2. — Pp. 244–263.

32. *Holzmann G. J.* The Model Checker SPIN // IEEE Trans. Softw. Eng. — 1997. — May. — Vol. 23, no. 5. — Pp. 279–295.

33. *Cimatti A.*, *Clarke E.*, *Giunchiglia E.*, *Giunchiglia F.*, *Pistore M.*, *Roveri M.*, *Sebastiani R.*, *Tacchella A.* NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking // Proceedings of International Conference on Computer-Aided Verification, (CAV 2002). Vol. 2404. — Copenhagen, Denmark : Springer, July 2002.

34. *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.

35. *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. — 7:1–7:6.

36. *Reitblatt M.*, *Foster N.*, *Rexford J.*, *Schlesinger C.*, *Walker D.* Abstractions for Network Update // Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, (SIGCOMM '12). — Helsinki, Finland, 2012. — Pp. 323–334.

37. *Gutz S.*, *Story A.*, *Schlesinger C.*, *Foster N.* Splendid Isolation: A Slice Abstraction for Software-defined Networks // Proceedings of the First Workshop on Hot Topics in Software Defined Networks, (HotSDN '12). — Helsinki, Finland, 2012. — Pp. 79–84.

38. *McClurg J.*, *Hojjat H.*, *Cerny P.*, *Foster N.* Efficient Synthesis of Network Updates // ACM SIGPLAN Conference on Programming Language Design and Implementation, (PLDI'15). — Portland, USA, June 2015.

39. *Kozen D.* Results on the propositional Mu-calculus // Automata, Languages and Programming, Lecture Notes in Computer Science. Vol. 140. — 1982. — Pp. 348–359.

40. *Alechina N.*, *Immerman N.* Reachability Logic: An Efficient Fragment of Transitive Closure Logic // Logic Journal of IGPL. — 2000. — Vol. 8. — Pp. 325–337.

41. *Immerman N.*, *Vardi M. Y.* Model Checking and Transitive-Closure Logic // Proceedings of the 9th International Conference on Computer Aided Verification, (CAV '97). — 1997. — Pp. 291–302.

42. *Kim H.*, *Feamster N.* Improving network management with software defined networking // Communications Magazine, IEEE. — 2013. — Feb. — Vol. 51, no. 2. — Pp. 114–119.

43. *Qadir J.*, *Hasan O.* Applying Formal Methods to Networking: Theory, Techniques, and Applications // IEEE Communications Surveys and Tutorials. — 2015. — Т. 17, № 1. — С. 256—291.

44. *Kazemian P.*, *Chang M.*, *Zeng H.*, *Varghese G.*, *McKeown N.*, *Whyte S.* Real Time Network Policy Checking Using Header Space Analysis // Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation, (NSDI'13). — Lombard, IL, USA, 2013. — Pp. 99–111.

45. *Mai H.*, *Khurshid A.*, *Agarwal R.*, *Caesar M.*, *Godfrey P. B.*, *King S. T.* Debugging the Data Plane with Anteater // Proceedings of the ACM SIGCOMM 2011 Conference, (SIGCOMM '11). — Toronto, Ontario, Canada, 2011. — Pp. 290–301.

46. *Al-Shaer E.*, *Marrero W.*, *El-Atawy A.*, *Elbadawi K.* Network configuration in a box: towards end-to-end verification of network reachability and security // Proceedings of the 17th IEEE International Conference on Network Protocols, (ICNP '09). — Oct. 2009. — Pp. 123–132.

47. *Yang H.*, *Lam S.* Real-Time Verification of Network Properties Using Atomic Predicates // IEEE/ACM Transactions on Networking. — 2015. — Vol. PP, no. 99. — Pp. 1–14.

48. *McGeer R.* New results on BDD sizes and implications for verification // Proceedings of the International Workshop on Logic Synthesis, (IWLS '12). — June 1–3, 2012.

49. *Lopes N.*, *Bjorner N.*, *Godefroid P.*, *Varghese G.* Network Verification in the Light of Program Verification: tech. rep. — Sept. 2013. — URL: http://research.microsoft.com/apps/pubs/default.aspx?id=201589.

50. *Gutnik G.*, *Kaminka G.* A Scalable Petri Net Representation of Interaction Protocols for Overhearing // Agent Communication, Lecture Notes in Computer Science. Vol. 3396. — 2005. — Pp. 50–64.

51. *Waez M. T. B.*, *Dingel J.*, *Rudie K.* A survey of timed automata for the development of real-time systems // Computer Science Review. — 2013. — Vol. 9. — Pp. 1–26.

52. *Волканов Д.*, *Захаров В.*, *Зорин Д.*, *Коннов И.*, *Подымов В.* Как разработать простое средство верификации систем реального времени // Моделирование и анализ информационных систем. — 2012. — Т. 19, № 6. — С. 45—56.

53. *Bryant R. E.* Graph-Based Algorithms for Boolean Function Manipulation // IEEE Transactions on Computers. — 1986. — Vol. 35. — Pp. 677–691.

54. *Knuth D. E.* The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams. — 12th. — Addison-Wesley Professional, 2009.

55. *Bryant R. E.* Symbolic Boolean Manipulation with Ordered Binary-decision Diagrams // ACM Comput. Surv. — 1992. — Sept. — Vol. 24, no. 3. — Pp. 293–318.

56. *Bryant R. E.*, *Seger C.-J. H.* Formal Verification of Digital Circuits Using Symbolic Ternary System Models // Proceedings of the 2Nd International Workshop on Computer Aided Verification, (CAV '90). — 1991. — Pp. 33–43.

57. Buddy OBDD package. — Mar. 20, 2015. — URL: http://sourceforge.net/projects/buddy/.

58. *Milvang-Jensen K.*, *Hu A. J.* BDDNOW: A Parallel BDD Package // Proceedings of the Second International Conference on Formal Methods in Computer-Aided Design, (FMCAD '98). — 1998. — Pp. 501–507.

59. *Ossowski J.* JINC: a multi-threaded library for higher-order weighted decision diagram manipulation: PhD thesis. — 2010. — Pp. 1–124.

60. *Foster N.*, *Harrison R.*, *Freedman M. J.*, *Monsanto C.*, *Rexford J.*, *Story A.*, *Walker D.* Frenetic: A Network Programming Language // Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming, (ICFP '11). — Tokyo, Japan, 2011. — Pp. 279–291.

61. *Khurshid A.*, *Zhou W.*, *Caesar M.*, *Godfrey P. B.* VeriFlow: Verifying Network-wide Invariants in Real Time // Proceedings of the First Workshop on Hot Topics in Software Defined Networks, (HotSDN '12). — Helsinki, Finland, 2012. — Pp. 49–54.

62. *Kang N.*, *Reich J.*, *Rexford J.*, *Walker D.* Policy Transformation in Software Defined Networks // Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, (SIGCOMM '12). — Helsinki, Finland, 2012. — Pp. 309–310.

63. *Canini M.*, *Venzano D.*, *Perešíni P.*, *Kostić D.*, *Rexford J.* A NICE Way to Test Openflow Applications // Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, (NSDI'12). — San Jose, CA : USENIX Association, 2012. — Pp. 127–140.

64. *Immerman N.* Languages Which Capture Complexity Classes // Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, (STOC '83). — 1983. — Pp. 347–354.

65. *Galil Z.* Hierarchies of complete problems // Acta Informatica. — 1976. — Vol. 6, no. 1. — Pp. 77–88.

66. *Kang N.*, *Liu Z.*, *Rexford J.*, *Walker D.* Optimizing the "One Big Switch" Abstraction in Software-defined Networks // Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, (CoNEXT '13). — Santa Barbara, California, USA, 2013. — Pp. 13–24.

67. *Ахо А. В.*, *Лам М. С.*, *Сети Р.*, *Ульман Д. Д.* Компиляторы: принципы, технологии и инструментарий. — 2 изд. — М.: Вильямс, 2008.

68. *Palmer C.*, *Steffan J.* Generating network topologies that obey power laws // Proceedings of the Global Telecommunications Conference, (IEEE GLOBECOM '00). Vol. 1. — 2000. — Pp. 434–438.

69. *Al-Fares M.*, *Loukissas A.*, *Vahdat A.* A Scalable, Commodity Data Center Network Architecture // Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, (SIGCOMM '08). — Seattle, WA, USA, 2008. — Pp. 63–74.

70. *Cisco Systems I.* The Zettabyte Era: Trends and Analysis: white paper. — June 10, 2014.

71. *Pana F.*, *Put F.* A Survey on the Evolution of RSVP // Communications Surveys & Tutorials. — 2013. — Vol. 15, no. 4. — Pp. 1859–1887.

72. *Yeh R. T.* Requirements Analysis – A Management Perspective // Proceedings of COMPSAC'82. — Nov. 1982. — Pp. 410–416.

73. *Fantinato M.*, *Jino M.* Applying Extended Finite State Machines in Software Testing of Interactive Systems // Interactive Systems. Design, Specification, and Verification, Lecture Notes in Computer Science. Vol. 2844. — Springer Berlin Heidelberg, 2003. — Pp. 34–45.

74. *Chung L.*, *Prado Leite J. do* On Non-Functional Requirements in Software Engineering // Conceptual Modeling: Foundations and Applications, Lecture Notes in Computer Science. Vol. 5600. — Springer Berlin Heidelberg, 2009. — Pp. 363–379.

75. *Katchabaw M.*, *Lutfiyya H.*, *Bauer M.* Administrative Policies to Regulate Quality of Service Management in Distributed Multimedia Applications // Management of Multimedia Networks and Services, Lecture Notes in Computer Science. Vol. 2839. — Springer Berlin Heidelberg, 2003. — Pp. 341–354.

76. *Braden R.*, *Clark D.*, *Shenker S.* RFC1633: Integrated Services in the Internet Architecture: An Overview: Requests for Comments. — 1994. — Pp. 1–33. — URL: https://tools.ietf.org/html/rfc1633.

77. *Heller B.*, *Seetharaman S.*, *Mahadevan P.*, *Yiakoumis Y.*, *Sharma P.*, *Banerjee S.*, *Mckeown N.* ElasticTree: Saving Energy in Data Center Networks // Proceedings of the 7th USENIX conference on Networked Systems Design and Implementation (NSDI'12). — San Jose, CA, USA : USENIX Association, Apr. 25–27, 2012. — Pp. 17–17.

78. *Ohara Y.*, *Imahori S.*, *Van Meter R.* MARA: Maximum Alternative Routing Algorithm // Proceedings of the 28th Conference on Computer Communications (INFOCOM 2009). — Rio de Janeiro, Brazil : IEEE Communications Society, Apr. 19–25, 2009. — Pp. 298–306.

79. *Garroppo R. G.*, *Giordano S.*, *Tavanti L.* A Survey on Multi-constrained Optimal Path Computation: Exact and Approximate Algorithms // Computer Networks: The International Journal of Computer and Telecommunications Networking. — 2010. — Dec. — Vol. 54, no. 17. — Pp. 3081–3107.

80. *Hancock R.*, *Karagiannis G.*, *Loughney J.*, *Van den Bosch S.* RFC4080: Next Steps in Signaling (NSIS): Framework: Requests for Comments. — 2005. — Pp. 1–49. — URL: http://www.ietf.org/rfc/rfc4080.txt.

81. *Kazemian P.*, *Chang M.*, *Zeng H.*, *Varghese G.*, *McKeown N.*, *Whyte S.* Real Time Network Policy Checking Using Header Space Analysis // Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI'13). — Lombard, IL, USA : USENIX Association, 2013. — Pp. 99–112.

82. Cut-Through and Store-and-Forward Ethernet Switching for Low-Latency Environments: tech. rep. — 2008. — URL: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-5020-switch/white_paper_c11-465436.html.

83. *Ganjali Y.*, *Keshavarzian A.*, *Shah D.* Cell Switching Versus Packet Switching in Input-Queued Switches // The IEEE/ACM Transactions on Networking. Vol. 13. — Aug. 2005. — Pp. 782–789.

84. *Nichols K.*, *Blake S.*, *Baker F.*, *Black D.* RFC2474: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers: Requests for Comments. — 1998. — Pp. 1–20. — URL: https://www.ietf.org/rfc/rfc2474.txt.

85. *Sawashima H.*, *Hori Y.*, *Sunahara H.*, *Yuji O.* Characteristics of UDP Packet Loss: Effect of TCP Traffic // Proceedings of The Seventh Annual Conference of the Internet Society (INET'97). — Kuala Lumpur, Malaysia, June 24–27, 1997.

86. *Iyer S.*, *McKeown N. W.* Analysis of the Parallel Packet Switch Architecture // The IEEE/ACM Transactions on Networking. — 2003. — Apr. — Vol. 11, no. 2. — Pp. 314–324.

87. *McKeown N.*, *Anantharam V.*, *Walrand J.* Achieving 100% Throughput in an Input-queued Switch // Proceedings of the 15th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'96). Vol. 1. — San Francisco, California, USA : IEEE Computer Society, 1996. — Pp. 296–302.

88. *Prabhakar B.*, *McKeown N.* On the speedup required for combined input and output queued switching // Proceedings of the IEEE International Symposium on Information Theory. — Aug. 1998. — Pp. 165–176.

89. *Karol M.*, *Hluchyj M.*, *Morgan S.* Input Versus Output Queueing on a Space-Division Packet Switch // IEEE Transactions on Communications. — 1987. — Dec. — Vol. 35, no. 12. — Pp. 1347–1356.

90. *Danilewicz G.*, *Glabowski M.*, *Kabacinski W.*, *Kleban J.* Packet switch architecture with multiple output queueing // Global Telecommunications Conference, GLOBECOM'04. Vol. 2. — IEEE Communications Society, Nov. 2004. — Pp. 1192–1196.

91. *Kumar A.*, *Manjunath D.*, *Kuri J.* Communication Networking: An Analytical Approach. — Morgan Kaufmann Publishers Inc., 2004.

92. *Mezger K.*, *Petr D.* Bounded Delay for Weighted Round Robin: tech. rep. — 1995. — Technical Report TISL-10230-07. — URL: https://www.ittc.ku.edu/publications/documents/Mezger1995_tr-tisl-10230-07.pdf.

93. *Misra V.*, *Gong W.-b.*, *Towsley D.* Stochastic Differential Equation Modeling and Analysis of TCP-Windowsize Behavior // Proceedings of PERFORMANCE'99. — Istanbul, Turkey, Nov. 1999.

94. *Ciucu F.*, *Schmitt J.* Perspectives on Network Calculus: No Free Lunch, but Still Good Value // ACM SIGCOMM Computer Communication Review (SIGCOMM'12). — 2012. — Aug. — Vol. 42, no. 4. — Pp. 311–322.

95. *Bouillard A.*, *Jouhet L.*, *Thierry E.* Service curves in Network Calculus: dos and don'ts: tech. rep. — 2009. — URL: https://hal.inria.fr/file/index/docid/431674/filename/RR-7094.pdf.

96. *Bouillard A.*, *Steay G.* Exact worst-case delay for FIFO-multiplexing tandems // Proceedings of the 6th International Conference on Evaluation Methodologies and Tools (VALUETOOLS'12). — Oct. 2012. — Pp. 158–167. — URL: http://www.di.ens.fr/~bouillar/Publis/Valuetools-fifo.pdf.

97. *Bouillard A.*, *Junier A.* Worst-case Delay Bounds with Fixed Priorities Using Network Calculus // Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools, (VALUETOOLS'11). — Paris, France : ICST (Institute for Computer Sciences, Social-Informatics, Telecommunications Engineering), 2011. — Pp. 381–390.

98. *Boyer M.*, *Steay G.*, *Sofack W.* Deficit Round Robin with network calculus // Proceedings of the 6th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS). — Oct. 2012. — Pp. 138–147.

99. *Schmitt J.*, *Zdarsky F.*, *Fidler M.* Delay Bounds under Arbitrary Multiplexing: When Network Calculus Leaves You in the Lurch... // Proceedings of the 27th Conference on Computer Communications (INFOCOM). — IEEE Computer Society, Apr. 2008. — Pp. 1669–1677.

100. *Jiang Y.* Relationship between guaranteed rate server and latency rate server // IEEE Global Telecommunications Conference (GLOBECOM '02). Vol. 3. — Nov. 2002. — Pp. 2415–2419.

101. *Starobinski D.*, *Karpovsky M.*, *Zakrevski L. A.* Application of Network Calculus to General Topologies Using Turn-prohibition // IEEE/ACM Transitions on Computer Networks. — 2003. — June. — Vol. 11, no. 3. — Pp. 411–421.

102. *Charny A.*, *Boudec J.-Y.* Delay Bounds in a Network with Aggregate Scheduling // Quality of Future Internet Services, Lecture Notes in Computer Science. Vol. 1922. — Springer Berlin Heidelberg, 2000. — Pp. 1–13.

103. *Jiang Y.* Delay Bounds for a Network of Guaranteed Rate Servers with FIFO Aggregation // Computer Networks: The International Journal of Computer and Telecommunications Networking. — 2002. — Dec. — Vol. 40, no. 6. — Pp. 683–694.

104. *Bouillard A.*, *Jouhet L.*, *Thierry E.* Tight performance bounds in the worst-case analysis of feed-forward networks // Proceedinga of INFOCOM 2010. — San Diego, USA, Mar. 14–19, 2010. — Pp. 1–9.

105. *Saino L.*, *Cocora C.*, *Pavlou G.* A Toolchain for Simplifying Network Simulation Setup // Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques, (SimuTools '13). — Cannes, France : ICST (Institute for Computer Sciences, Social-Informatics, Telecommunications Engineering), 2013. — Pp. 82–91.

106. *Hassan H.*, *Garcia J.-M.*, *Brun O.* Generic modeling of multimedia traffic sources: tech. rep. — 2005. — URL: http://researchwebshelf.com/uploads/248_P14.pdf.

107. *Seeling P.*, *Reisslein M.* Video Transport Evaluation With H.264 Video Traces // IEEE Communications Surveys & Tutorials. — 2012. — Vol. 14, no. 4. — Pp. 1142–1165.

108. *Henderson T.*, *Floyd S.*, *Gurtov A.*, *Nishida Y.* RFC6582: The NewReno Modification to TCP's Fast Recovery Algorithm. Requests for Comments. — 2012. — Pp. 1–16. — URL: http://www.rfc-editor.org/rfc/rfc6582.txt.

109. *Grädel E.*, *Walukiewicz I.* Guarded Fixed Point Logic // Proceedings of the 14th Symposium on Logic in Computer Science. — Trento, Italy, July 2–5, 1999. — Pp. 45–54.

110. *Alur R.*, *Dill D.* Automata-theoretic Verification of Real Time Systems // Formal Methods for Real-Time Computing, Trends in Software Series. — 1995. — Pp. 55–82.

111. *Кларк Э. М.*, *Грамберг О.*, *Пелед Д.* Верификация моделей программ: Model Checking. — МЦНМО, 2002.

112. *Alur R.*, *Dill D. L.* A Theory of Timed Automata // Theor. Comput. Sci. — 1994. — Apr. — Vol. 126, no. 2. — Pp. 183–235.

113. *Alur R.*, *Madhusudan P.* Decision Problems for Timed Automata: A Survey // Formal Methods for the Design of Real-Time Systems, Lecture Notes in Computer Science. Vol. 3185. — Springer Berlin Heidelberg, 2004. — Pp. 1–24.

114. *Alur R.*, *Courcoubetis C.*, *Dill D.* Model-checking in Dense Real-time // Inf. Comput. — 1993. — May. — Vol. 104, no. 1. — Pp. 2–34.

115. *Alur R.*, *A. T.* Real-time system = discrete system + clock variables // International Journal on Software Tools for Technology Transfer. — 1997. — Vol. 1, 1-2. — Pp. 86–109.

116. *Henzinger T. A.*, *Nicollin X.*, *Sifakis J.*, *Yovine S.* Symbolic Model Checking for Real-time Systems // Inf. Comput. — 1994. — June. — Vol. 111, no. 2. — Pp. 193–244.

117. *Henzinger T.*, *Ho P.-H.*, *Wong-Toi H.* HyTech: A model checker for hybrid systems // Computer Aided Verification, Lecture Notes in Computer Science. Vol. 1254. — Springer Berlin Heidelberg, 1997. — Pp. 460–463.

118. *Bozga M.*, *Daws C.*, *Maler O.*, *Olivero A.*, *Tripakis S.*, *Yovine S.* Kronos: A model-checking tool for real-time systems // Formal Techniques in Real-Time and Fault-Tolerant Systems, Lecture Notes in Computer Science. Vol. 1486. — Springer Berlin Heidelberg, 1998. — Pp. 298–302.

119. *Milner R.* Algebras for communicating systems // Proceedings of AFCET/SMF Joint Colloquim in Applied Mathamatics. — 1978.

120. *Hoare C. A. R.* Communicating Sequential Processes // Commun. ACM. — 1978. — Aug. — Vol. 21, no. 8. — Pp. 666–677.

121. *Sun J.*, *Liu Y.*, *Roychoudhury A.*, *Liu S.*, *Dong J.* Fair Model Checking with Process Counter Abstraction // FM 2009: Formal Methods, Lecture Notes in Computer Science. Vol. 5850. — Springer Berlin Heidelberg, 2009. — Pp. 123–139.

122. *Calzolai F.*, *De Nicola R.*, *Loreti M.*, *Tiezzi F.* TAPAs: A Tool for the Analysis of Process Algebras // Transactions on Petri Nets and Other Models of Concurrency I, Lecture Notes in Computer Science. Vol. 5100. — 2008. — Pp. 54–70.

123. *Mayr E.* Persistence of vector replacement systems is decidable // Acta Informatica. — 1981. — Vol. 15, no. 3. — Pp. 309–318.

124. *Lambert J. L.* A Structure to Decide Reachability in Petri Nets // Theor. Comput. Sci. — 1992. — June. — Vol. 99, no. 1. — Pp. 79–104.

125. *Lipton R. J.* Reachability problem requires exponential space: Research Report. — 1976. — No. 62.

126. *Esparza J.*, *Nielsen M.* Decidability Issues for Petri Nets - a Survey // Bulletin of the European Association for Theoretical Computer Science. — 1994. — Vol. 52. — Pp. 245–262.

127. *Küngas P.* Petri Net Reachability Checking is Polynomial with Optimal Abstraction Hierarchies // Proceedings of the 6th International Conference on Abstraction, Reformulation and Approximation, (SARA'05). — Airth Castle, UK, 2005. — Pp. 149–164.

128. *Jensen K.* Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 2. — London, UK, UK : Springer-Verlag, 1995.

129. *Ломазова И. А.* Моделирование мультиагентных динамических систем вложенными сетями Петри // Программные системы: Теоретические основы и приложения. — 1999. — С. 143—156.

130. *Byg J.*, *Jørgensen K.*, *Srba J.* TAPAAL: Editor, Simulator and Verifier of Timed-Arc Petri Nets // Automated Technology for Verification and Analysis, Lecture Notes in Computer Science. Vol. 5799. — Springer Berlin Heidelberg, 2009. — Pp. 84–89.

131. *Gardey G.*, *Lime D.*, *Magnin M.*, *Roux O.* Romeo: A Tool for Analyzing Time Petri Nets // Computer Aided Verification, Lecture Notes in Computer Science. Vol. 3576. — Springer Berlin Heidelberg, 2005. — Pp. 418–423.