МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. М.В. Ломоносова
Факультет вычислительной математики и кибернетики

На правах рукописи

ПЕТРОВ Иван Сергеевич

ОБНАРУЖЕНИЕ СКОМПРОМЕТИРОВАННЫХ КОММУТАТОРОВ В ПРОГРАММНО-КОНФИГУРИРУЕМЫХ СЕТЯХ

Специальность 05.13.11 — математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

ДИССЕРТАЦИЯ

на соискание ученой степени кандидата физико-математических наук

Научный руководитель:
доктор физ.-мат. наук,
член-корр. РАН,
профессор Р. Л. Смелянский

Москва — 2019

Оглавление

Список иллюстративного материала

Введение

> Fully secure systems don't exist today and they won't exist in the future.
>
> *Adi Shamir*

В последнее время наблюдаются высокие темпы роста применения облачных технологий и виртуальных инфраструктур. Новые технологии изменили и усложнили природу компьютерных сетей — облака и крупные центры обработки данных (ЦОД) предъявляют к сетям повышенные требования. Операторам требуются более «умные» сети, а также усовершенствования в инструментах сетевого мониторинга и управления.

Программно-конфигурируемые сети[1] (ПКС), по мнению ведущих производителей сетевого оборудования, являются одним из самых перспективных направлений сетевой индустрии на данный момент [1]. ПКС – это концепция построения сети, в которой контур управления сетью (*control-plane*) отделен от контура передачи данных (*data-plane*). Согласно концепции ПКС, вся логика управления переносится на отдельное устройство — контроллер, который способен отслеживать работу всей сети и управлять сетевыми устройствами — коммутаторами.

Наиболее важный аспект концепции ПКС — логически централизованное управление сетью, которое обеспечивает глобальное видение топологии и состояния управляемой сети как на уровне L2, так и на уровне L3. Контроллер собирает информацию о характеристиках коммутаторов, топологии сети, загрузке физических линий и об установленных соединениях между хостами (пользовательскими устройствами). В отличие от традиционных сетей, в которых каждое устройство имеет собственное представление о состоянии сети,

[1] Software-Defined Networking (SDN) – англ.

в ПКС сети контроллер имеет полную информацию о состоянии сети, и, таким образом, предоставляет централизованную платформу для реализации логики управления сетью.

В среде контроллера, подобно традиционным операционным системам, работают приложения, которые используют сервисы контроллера для управления различными ресурсами сети. Такие приложения могут выполнять множество функций, например, управлять маршрутизацией пакетов, контролировать доступ к сетевым ресурсам, балансировать нагрузку в сети и т.п. Контролер предоставляет приложениям унифицированный, открытый интерфейс и сервисы по управлению программируемой сетевой инфраструктурой. Благодаря контроллеру, сеть, состоящая из множества коммутаторов разных производителей, предстает для приложения как один логический коммутатор. Такой подход дает более удобные абстракции для разработчиков сетевых приложений по сравнению с сетями с традиционной архитектурой.

Одним из важных элементов концепции ПКС является протокол OpenFlow [2, 3] для программирования и управления коммутаторами. Схема работы протокола представлена на рисунке 1.

Протокол OpenFlow предполагает использование на коммутаторах специальных таблиц маршрутизации, в которые контроллер загружает правила для маршрутизации (рис. 2a) и/или модификации пакетов, проходящих через коммутатор. Такие таблицы называются таблицами потоков (*flow*-таблицами). Каждый коммутатор имеет несколько таблиц потоков, которые образуют конвейер обработки пакета. Конвейер обработки пакета используется для последовательной обработки пакетов несколькими правилами маршрутизации. Каждая таблица потоков имеет уникальный номер, начиная с нуля.

Протокол OpenFlow также позволяет загружать правила в специальную выделенную таблицу маршрутизации, называемую групповой таблицей. За-

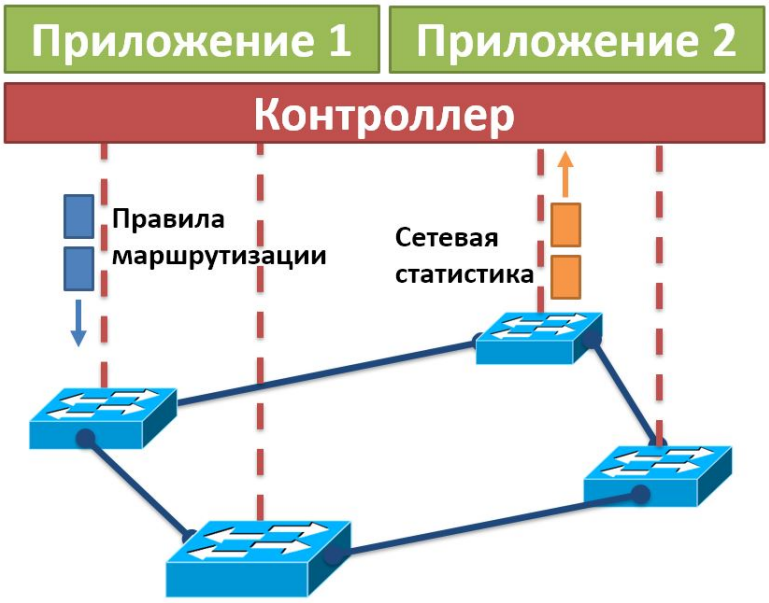
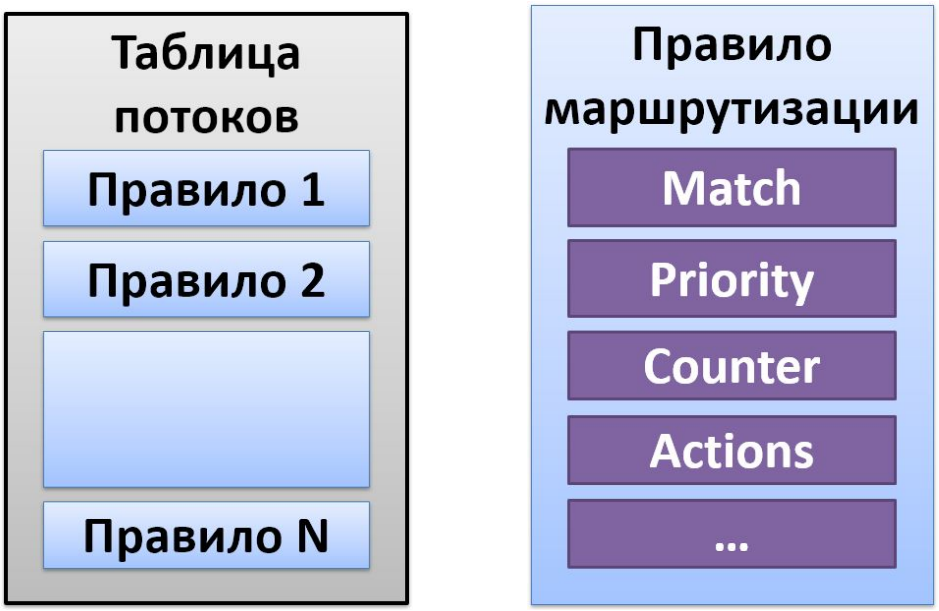


Рисунок 1: Схема работы протокола OpenFlow

писи в групповой таблице называются групповыми правилами маршрутизации. Групповые правила маршрутизации дублируют (зеркалирующее) трафик и производят различные действия с каждой копией пакета в отдельности. Действия над копиями пакета описываются записями, называемыми *bucket*-записями. Протокол OpenFlow также описывает опциональные групповые правила маршрутизации, которые не обязательно должны быть реализованы на коммутаторах. Эти опциональные групповые правила не дублируют пакеты на каждую *bucket*-запись, вместо этого они выбирают *bucket*-запись по специальному алгоритму выбора.

Пакеты, поступившие на входной буфер коммутатора, сначала проверяются на соответствие их заголовков шаблонам правил из нулевой таблицы. Заголовки пакетов сравниваются с шаблонами правил в порядке убывания приоритета правила, и если заголовок совпадает с шаблоном, то выполняется инструкция, связанная с выбранным правилом. Такими инструкциями могут быть: отправка пакета на некоторый порт, сброс пакета или отправка пакета на контроллер для дальнейшего анализа. Также пакеты могут быть



(a) Таблица потоков (b) Правило маршрутизации

Рисунок 2: Примитивы протокола OpenFlow

отправлены в таблицу потоков с большим номером, где они будут обработаны другим правилом маршрутизации.

Записями каждой таблицы потоков являются правила маршрутизации, представленные в виде кортежа *<match, priority, counter, action>* (рис. 2b). Каждая bucket-запись представляет собой правило маршрутизации без полей *<match>* и *<priority>*.

- Поле *<match>* описывает множество пакетов, которые могут обрабатываться этим правилом.

- Поле *<priority>* используется для разрешения неопределенностей обработки пакетов, то есть, если в таблице есть несколько правил, шаблонам которых соответствует пакет, то выбирается правило с наивысшим приоритетом. Спецификация протокола OpenFlow прямо указывает, что наличие правил с одинаковым приоритетом и пересекающимися полями *<match>* ведет к неопределенному поведению.

- Поле *<counter>* описывает количество пакетов, обработанных правилом маршрутизации с момента его установки на коммутатор.

- Поле *<action>* описывает набор действий с пакетами. Действиями могут быть: сброс пакета, отправка на контроллер, отправка в таблицу потоков или групповую таблицу, отправка на порт и изменение заголовка пакета.

Сети, построенные по концепции ПКС, обладают рядом свойств, которые хорошо подходят для построения безопасной и управляемой вычислительной среды:

- Разделение контура управления (*control*-plane) и контура передачи данных (*data*-plane) позволяет изолировать логику управления сетевыми устройствами.

- Парадигма потока данных более удобна для обеспечения безопасности, поскольку она предлагает сквозную (*end-to-end*) модель связей, ориентированную на сервисы, не связанную с традиционными ограничениями маршрутизации.

- Логически централизованное управление позволяет во всей сети осуществлять эффективную защиту и мониторинг появления угроз.

- Переход к программному управлению позволяет осуществлять динамичную и гибкую настройку политики безопасности.

В случае построения сети по концепции ПКС ряд угроз безопасности становятся не актуальными, часть механизмов защиты может быть реализована в коммутаторах и контроллере [4]. Например, концепция ПКС предоставляет

гибкие возможности по сегментированию и изоляции узлов сети [1]. Некоторые атаки, использующие слабости протоколов маршрутизации, могут быть исключены, так как в ПКС многие протоколы не используются, однако в случае гибридной сети они могут контролироваться средствами ПКС [5].

Несмотря на это, с появлением новой концепции компьютерных сетей появляются и новые виды угроз, которые могут привести не только к материальным потерям, но и к утрате репутации и имиджа компаний, использующих подобные сети [6]. Следовательно, во избежание негативных последствий компьютерных атак, необходимо производить анализ защищенности ПКС и разрабатывать механизмы защиты таких сетей.

Угрозы для сетей, построенных по концепции ПКС, можно описать следующим образом:

1) *Угрозы безопасности канала данных*

Этот тип угроз аналогичен угрозам безопасности канала данных в традиционных сетях. Он заключается в том, что если у злоумышленника есть доступ к каналу данных, то он может производить атаку *Man-in-the-Middle* на пользовательский трафик [7].

Разделяют два вида атаки *Man-in-the-Middle*: пассивная и активная. При пассивной атаке злоумышленник перехватывает пользовательский трафик и анализирует его на предмет наличия важных данных, таких как персональная информация, пароли и т.п.

Активная атака предполагает изменение пользовательского трафика. Например, злоумышленник может вставлять вредоносный код в Web-страницы, возвращаемые сервером пользователю.

2) *Угрозы безопасности коммутаторов*

Как и в любом программном или аппаратном обеспечении, в ПКС коммутаторах могут присутствовать уязвимости [4, 8]. Эти уязвимости могут быть проэксплуатированы злоумышленником для получения контроля над коммутатором [9].

Более того, концепция ПКС предполагает возможность использования программных коммутаторов. Такие коммутаторы могут быть установлены на серверы общего назначения, на которых одновременно с коммутатором работают другие приложения, имеющие уязвимости. И как следствие, эксплуатация уязвимости в одном из таких приложений может привести к компрометации всего сервера, и в том числе ПКС коммутатора.

Компрометация коммутатора является серьезной проблемой, которая создает угрозу компрометации всей сети, так как атакующий может использовать подконтрольные ему коммутаторы для проведения широкого спектра атак как на пользовательский трафик, так и на сетевые устройства [10].

3) *Угрозы безопасности канала управления*

Этот тип угроз является специфичным для концепции ПКС, так как в ней канал управления отделен от канала передачи данных. Если у злоумышленника есть доступ к каналу управления, то он может перехватывать и изменять управляющий трафик, посылаемый как контроллером коммутатору, так и наоборот. Таким образом, злоумышленник может нарушать взаимодействие контроллера с коммутаторами в сети, изменяя перехваченные пакеты.

Успешность таких атак на контур управления напрямую зависит от используемой защиты управляющего трафика. Спецификация протокола OpenFlow [3] предполагает защиту управляющего трафика при помощи протокола TLS [11]. Такая защита необходима для того, чтобы злоумышленник

не мог перехватывать и изменять управляющий трафик. Несмотря на то, что использование протокола TLS описано в спецификации протокола OpenFlow, этот пункт не является обязательным. Это приводит к возможности появления в сети ошибок администрирования, когда защита не была включена администратором, и злоумышленник может получить доступ к управляющему трафику.

Угроза безопасности управляющего трафика также может возникнуть из-за уязвимостей как в протоколе TLS [12, 13], так и в его реализации [14].

4) *Угрозы безопасности контроллера*

Компрометация контроллера влечет за собой компрометацию всей сети, которой он управляет [5]. Обычных методов обнаружения вторжения здесь может быть недостаточно, так как зачастую весьма сложно установить комбинацию событий, вызывающих конкретное поведение, при такой атаке, и, кроме того, доказать, что оно вредоносное [15].

5) *Угрозы безопасности приложений*

Для того, чтобы реализовывать логику управления сетью, приложения на контроллере получают доступ к разного рода операциям, в том числе и критическим. Как и любое программное обеспечение, приложения могут содержать ошибки, которые могут привести к незапланированному поведению приложения, например, к его аварийному завершению. Более того, использование сторонних приложений может привести к угрозе запуска вредоносной программы с правами управления сетью [16].

В настоящей работе рассматривается задача обнаружения скомпрометированных ПКС коммутаторов. Под скомпрометированным ПКС коммута-

тором понимается коммутатор сети, управляемой ПКС контроллером, который, в действительности, находится под контролем злоумышленника. Возможность компрометации коммутаторов обусловлена тем, что коммутаторы могут иметь уязвимости [8, 17, 18], которые могут быть проэксплуатированы злоумышленником для получения над ними контроля.

Одним из примеров уязвимости в ПКС коммутаторе, которая может привести к его компрометации некоторым злоумышленником, является *CVE-2016-2074* [9] в коммутаторах *Open vSwitch* [19] версии ниже, чем 2.4.1. Эта уязвимость заключается в том, что злоумышленник может создать специальный пакет с большим количеством MPLS [20] меток и отправить этот пакет на коммутатор. Обработка такого пакета приведет к переполнению буфера и выполнению произвольного кода на коммутаторе.

Несмотря на то, что все функции управления сетью вынесены на контроллер, компрометация коммутатора является серьезной угрозой безопасности всей сети, так как атакующий может использовать подконтрольные ему коммутаторы для проведения различных атак как на контур данных, так и на контур управления [10].

Одним из основных методов проведения атаки на *data-plane* при помощи скомпрометированного ПКС коммутатора является установка злоумышленником дополнительных правил маршрутизации [21]. Поскольку концепция ПКС дает большую свободу в управлении сетевым трафиком, злоумышленник может реализовать произвольную логику атаки на сеть: сбрасывать трафик из заранее выбранных потоков, дублировать трафик на некоторый хост для дальнейшего анализа или изменять заголовки пакета так, чтобы они были сброшены на другом легитимном коммутаторе.

Злоумышленник также может проводить атаки на *control-plane*. К таким атакам относятся атаки, влияющие на информацию о сети, хранимую

на контроллере [22]. Так как всю информацию о сети контроллер получает от коммутаторов, злоумышленник может отправлять на контроллер некорректные данные о состоянии сети. Например, атакующий может отправить контроллеру информацию о несуществующей в сети физической линии. Это может привести к тому, что некоторые потоки будут направлены по такой несуществующей линии и в итоге будут сброшены коммутатором, что приведет к отказу в обслуживании.

Сложность задачи обнаружения скомпрометированного коммутатора заключается в том, что подобный коммутатор невозможно обнаружить средствами аутентификации устройства. Это обусловлено тем, что, получая контроль над коммутатором, злоумышленник получает доступ к криптографическим ключам, находящимся в памяти коммутатора. Далее, используя эти ключи, злоумышленник сможет провести процедуру аутентификации скомпрометированного коммутатора. А любой коммутатор, прошедший процедуру аутентификации, будет считаться легитимным с точки зрения контроллера [3]. Таким образом, необходимы методы проверки различных признаков компрометации коммутаторов.

Одним их основных признаков компрометации ПКС коммутатора является наличие на коммутаторе правил маршрутизации, которые не были установлены контроллером [23] (рис. 3). Так как ПКС коммутатор не может сам устанавливать правила маршрутизации, то наличие подобных правил говорит о том, что их установил злоумышленник.

Для проверки наличия несанкционированных правил на коммутаторе, контроллер должен хранить копию правил, которые он установил на каждый коммутатор. Далее контроллер может считывать с коммутаторов информацию о всех правилах маршрутизации, которые установлены в табли-

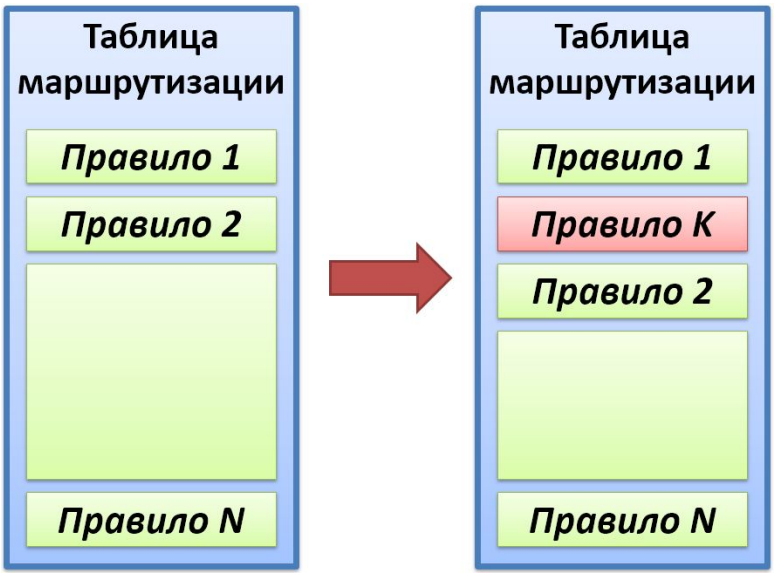


Рисунок 3: Скомпрометированный коммутатор

цах маршрутизации. Таким образом, контроллер имеет информацию о том, какие правила должны быть на каждом коммутаторе, и может сравнивать их с реальными правилами, находящимися в памяти коммутатора.

Слабостью такого подхода является то, что злоумышленник может подделывать информацию о правилах маршрутизации, отправляемую на контроллер. Например, он может создать две таблицы с правилами маршрутизации: в одну будут сохраняться все правила, устанавливаемые легитимным контроллером, а в другую, правила, устанавливаемые злоумышленником. Назовем такие таблицы *основной* и *теневой*.

Наличие двух таблиц может быть использовано злоумышленником для обхода описанного выше механизма защиты. Если контроллер запросит хранящиеся на коммутаторе правила, то коммутатор предоставит ему правила из *основной* таблицы, но для обработки реального сетевого трафика будут использоваться правила из *теневой* таблицы.

Следовательно, контроллер не сможет проверить наличие на коммутаторе лишних правил, потому что коммутатор будет предоставлять контроллеру

некорректные данные о реально используемых для обработки трафика правилах.

На основании вышесказанного следует, что необходимы методы, которые позволят обнаружить скомпрометированные коммутаторы по различным косвенным признакам, то есть по наличию атак на контур передачи данных и по влиянию этих атак на легитимные коммутаторы.

Цель диссертационной работы

Целью диссертационной работы является разработка алгоритма обнаружения скомпрометированных коммутаторов в ПКС.

Для достижения поставленной цели в рамках настоящей работы было необходимо решить следующие задачи:

1. Провести обзор и сравнительный анализ существующих средств обнаружения скомпрометированных ПКС коммутаторов.

2. Разработать математическую модель ПКС сети для описания динамики изменения счетчиков правил маршрутизации.

3. Разработать алгоритм предсказания значения счетчиков правил маршрутизации, основанный на разработанной математической модели, и доказать его корректность.

4. Разработать алгоритм обнаружения скомпрометированных ПКС коммутаторов, который будет использовать алгоритм предсказания значений счетчиков правил маршрутизации, и обосновать его корректность.

Актуальность работы

Актуальность работы определяется растущей популярностью концепции ПКС в качестве сетевой архитектуры как для сетей центров обработки данных (ЦОД) [1], так и для сетей телеком-операторов (*Metro-WAN*) [24], наличием угрозы компрометации коммутаторов в таких сетях [4, 8], а также отсутствием систем обнаружения скомпрометированных коммутаторов в ПКС, которые бы не накладывали ограничений на логику работы приложений на контроллере.

Научная новизна

В диссертации разработана новая математическая модель, описывающая динамику изменения счетчиков правил маршрутизации в ПКС, в рамках которой сделана математическая постановка задачи. На основе математической модели был разработан новый алгоритм предсказания значений счетчиков правил маршрутизации при произвольной логике работы приложений на контроллере. Также был проведен анализ известных алгоритмов обнаружения скомпрометированных коммутаторов в ПКС, на основании которого сформулированы основные ограничения существующих алгоритмов.

В диссертации предложен новый алгоритм обнаружения скомпрометированных коммутаторов в ПКС, который свободен от ограничений существующих алгоритмов.

Теоретическая и практическая значимость

Теоретическая значимость работы состоит в проведении обзора существующих систем обнаружения скомпрометированных коммутаторов в ПКС, построении математической модели, описывающей изменение счетчиков правил маршрутизации, разработке алгоритма предсказания значений счетчиков правил маршрутизации и алгоритма обнаружения скомпрометированных коммутаторов в ПКС.

Практическая значимость работы обусловлена тем, что результаты работы могут быть использованы для обеспечения безопасности в сетях телеком-операторов и центрах обработки данных.

Положения, выносимые на защиту

- Впервые разработана математическая модель, описывающая динамику изменения счетчиков правил маршрутизации в OpenFlow коммутаторах в ПКС, которая инвариантна к набору правил маршрутизации, установленных в OpenFlow коммутаторах, и логике работы приложений контроллера в ПКС.

- В рамках разработанной модели построен алгоритм предсказания значений счетчиков правил маршрутизации, корректность которого доказана.

- На основе алгоритма предсказания значений счетчиков построен алгоритм обнаружения скомпрометированных коммутаторов, для которого экспериментально получены оценки ошибок первого/второго рода и время обнаружения скомпрометированных коммутаторов на топологиях, используемых в сетях операторов связи и центров обработки данных. Показано, что представленный алгоритм обнаружения превосходит известные алгоритмы обнаружения, используемые в существующих системах обнаружения, по ряду практически важных критериев.

Апробация работы

Результаты, представленные в работе, докладывались на научных семинарах лаборатории Вычислительных комплексов кафедры Автоматизации систем вычислительных комплексов факультета ВМК МГУ под руководством чл.-корр. РАН, профессора Р.Л. Смелянского, семинаре кафедры Автоматизации систем вычислительных комплексов имени чл.-корр. РАН, профессора Л.Н. Королёва, научном семинаре кафедры Математической кибернетики фа-

культета ВМК МГУ под руководством доктора физ.-мат. наук, профессора В.А. Захарова, заседании консорциума «ПКС в научно образовательной среде», проводимом Центром Прикладных Исследований Компьютерных Сетей, а также на 5 конференциях:

1. Всероссийской научной конференции «Ломоносовские чтения — 2017»
2. Международной научной конференции «REDS-2017»
3. Международной научной конференции «ElConRus-2018»
4. Всероссийской научной конференции «Ломоносовские чтения — 2018»
5. Международной научной конференции «MoNeTec-2018»

Публикации

Основные результаты по теме диссертации изложены в 1 патенте на изобретение [25] и в 12 печатных изданиях [26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37]. 5 публикаций [32, 33, 34, 36, 37] изданы в журналах, рекомендованных ВАК, 3 из них [32, 33, 34] изданы в журналах, цитируемых Scopus / Web of Science.

В работе [28] вклад Шемякина Р.О. заключается в реализации системы обеспечения контроля доступа приложений к ресурсам контроллера и проведении экспериментального исследования. Петрову И.С. принадлежит постановка задачи и описание алгоритма обеспечения контроля доступа.

В работе [29] Шендяпин А.С. реализовал тестовые атаки Man-in-the-Middle с использованием скомпрометированного коммутатора и провел экспериментальное исследование. Вклад Петрова И.С. заключается в постановке

задачи, разработке методов проведения тестовых атак и описании методики проведения экспериментального исследования.

В работах [30, 31, 32] вклад Смелянского Р.Л. заключается в постановке задач и описании методик проведения экспериментов. Петрову И.С. принадлежит разработка алгоритмов минимизации группового трафика и обнаружения скомпрометированных коммутаторов, обзоре существующих решений, реализации алгоритмов и проведении экспериментальных исследований.

В работе [34] Моргунова О.М. реализовала алгоритм минимизации количества правил маршрутизации и провела экспериментальное исследование. Вклад Петрова И.С. заключается в разработке алгоритма минимизации количества правил маршрутизации для анализа сетевой статистики и описании методики проведения экспериментального исследования.

В работе [35] Шендяпину А.С. принадлежит экспериментальное сравнение существующих средств обеспечения анонимности в программно-конфигурируемых сетях. Петрову И.С. принадлежит описание критериев сравнения и проведение обзора существующих средств.

В патенте [25] вклад Смелянского Р.Л. и Шалимова А.В. заключается в постановке задачи и описании методики анализа алгоритма минимизации группового трафика. Петрову И.С. принадлежит проведение обзора существующих решений и разработка алгоритма минимизации группового трафика в программно-конфигурируемых сетях.

Личный вклад автора

Все представленные в диссертации результаты получены лично автором.

Структура работы

Диссертация состоит из введения, 6 глав, заключения и 1 приложения.

Во введении обоснована актуальность диссертационной работы, сформулирована цель и аргументирована научная новизна исследований, показана практическая значимость полученных результатов, представлены выносимые на защиту научные положения и дано краткое описание задачи.

В первой главе приводится постановка задачи обнаружения скомпрометированных коммутаторов в ПКС.

Во второй главе приведено описание атак, которые злоумышленник может производить при помощи скомпрометированного ПКС коммутатора.

В третьей главе проводится обзор существующих систем обнаружения скомпрометированных коммутаторов в ПКС, выделяются их основные недостатки и также определяются требования к разработке системы обнаружения скомпрометированных коммутаторов.

В четвертой главе проводится описание разработанной математической модели ПКС сети, описывающей динамику изменения счетчиков правил маршрутизации в сети с произвольной комбинацией установленных правил маршрутизации.

В пятой главе приведено описание разработанного алгоритма обнаружения скомпрометированых коммутаторов в ПКС.

В шестой главе представлено описание разработанного прототипа системы обнаружения скомпрометированных коммутаторов и результаты его экспериментального исследования.

В заключении формулируются основные результаты работы.

В приложении А подробно описаны компоненты разработанной системы обнаружения скомпрометированных коммутаторов.

Полный объём диссертации составляет 156 страниц с 17 рисунками и 2 таблицами. Список литературы содержит 83 наименования.

Глава 1

Постановка задачи

1.1. Скомпрометированный коммутатор

Скомпрометированный коммутатор, это коммутатор в сети, управляемой ПКС контроллером, который в действительности находится под контролем некоторого злоумышленника. Компрометация коммутаторов возможна из-за того, что коммутаторы могут содежать уязвимости [8, 17, 18]. Эти уязвимости могут быть проэксплуатированы некоторым злоумышленником для получения контроля над коммутатором [9].

Скомпрометированные коммутаторы — достаточно вероятное событие в ПКС [5]. Вероятность обусловлена тем, что концепция ПКС позволяет использовать коммутаторы, созданные различными производителями. Такое разнообразие коммутаторов, с одной стороны, снижает стоимость и повышает надежность сети, с другой — увеличивает вероятность того, что хотя бы один коммутатор в сети будет иметь уязвимость, которая может быть проэксплуатирована некоторым злоумышленником [8].

Более того, концепция ПКС позволяет использовать программные коммутаторы, установленные на серверы общего назначения. На таких серверах одновременно с коммутатором работают другие приложения, имеющие уязвимости. Таким образом, эксплуатация уязвимости в одном из таких приложений может привести к компрометации всего сервера, и в том числе ПКС коммутатора.

Несмотря на физическое разделение контура управления от контура передачи данных, компрометация коммутатора — это серьезная угроза безопасности сети, так как скомпрометированный коммутатор может использоваться

в качестве платформы для проведения различных атак как на клиентов, так и на контроллер [10].

Скомпрометированные коммутаторы можно разделить на 2 типа.

К **первому** типу относятся скомпрометированные коммутаторы, которые не могут выполнять правила, отличные от правил протокола OpenFlow. То есть злоумышленник может только добавлять или удалять правила маршрутизации на подконтрольный ему коммутатор. Подобная возможность предоставляется злоумышленнику в случае частично-аппаратной реализации ПКС коммутатора, например, в случаях, когда таблица потоков реализована аппаратно. Такая возможность обусловлена тем, что таблица потоков предоставляет *API* [38], который может быть использован злоумышленником для установки правил на скомпрометированный коммутатор.

Ко **второму** типу относятся скомпрометированные коммутаторы, с которыми злоумышленник способен проводить любые действия, то есть злоумышленник не ограничен действиями правил протокола OpenFlow. Таким образом, злоумышленник может не только устанавливать на коммутатор любой набор ПКС правил, но также может полностью изменить логику работы коммутатора, добавив функционал, на который неспособен ПКС коммутатор. Подобная возможность предоставляется злоумышленнику в случае, когда ПКС коммутатор реализован программно.

Одним из основных различий указанных выше типов является то, что возможности управления трафиком для злоумышленника первого типа, в отличие от второго, ограничены возможностями протокола OpenFlow.

Например, злоумышленник, получивший контроль над коммутаторами первого типа, не способен генерировать новые пакеты и изменять тело проходящих через него пакетов. В тоже время он способен модифицировать заголовки пакетов и дублировать сами пакеты.

Опишем атаки на контур передачи данных, которые может производить злоумышленник при помощи скомпрометированного коммутатора в зависимости от типа коммутатора, над которым атакующий получил контроль.

Коммутаторы первого типа:

1) *Сброс пакетов*

Скомпрометированный коммутатор может сбрасывать любое подмножество пакетов, проходящих через его порты, таким образом, нарушая процесс передачи трафика в сети.

2) *Изменение заголовков пакетов*

Скомпрометированный коммутатор может использовать возможности протокола OpenFlow по изменению заголовков проходящих через него пакетов. Изменение заголовков пакетов может повлиять на логику обработки конкретного пакета другими коммутаторами, неподконтрольными злоумышленнику. Например, некоторый набор пакетов может быть направлен по другому пути, создавая тем самым перегрузки на некоторых участках сети.

3) *Некорректная маршрутизация пакетов*

Скомпрометированный коммутатор также может перенаправлять или дублировать входящие в него пакеты на любой его порт. Одним из возможных применений данной атаки является дублирование некоторого трафика на серверы, контролируемые злоумышленником, для дальнейшего анализа.

Коммутатор второго типа позволяет производить любые действия, описанные выше для первого типа. Также к возможным атакам добавляются следующие:

1) *Модификация пакетов*

Скомпрометированный коммутатор способен модифицировать содержимое пакетов, проходящих через него. Такая возможность позволяет злоумышленнику производить *Man-in-the-Middle* атаки в сети.

2) *Нарушение порядка пакетов*

Скомпрометированный коммутатор может изменять порядок проходящих через него пакетов, что приведет к падению пропускной способности сети или к отказу в обслуживании.

3) *Искусственные задержки*

Скомпрометированный коммутатор может искусственно задерживать трафик. Также, как и нарушение порядка пакетов, искусственные задержки ведут к снижению пропускной способности сети.

4) *Подделка пакетов*

Скомпрометированный коммутатор может генерировать пакеты и внедрять их в существующий поток трафика, либо создавать новые потоки трафика. Данная атака может быть использована для проведения широкого спектра атак внутри ПКС сети, например, *DDoS* атак [39] на другие устройства ПКС сети или компрометации других коммутаторов.

1.2. Задача обнаружения

В настоящей работе рассматривается задача обнаружения скомпрометированных коммутаторов в ПКС сетях.

Важной проблемой, связанной с задачей обнаружения, является то, что скомпрометированный коммутатор невозможно обнаружить при помощи аутентификации устройства. Невозможность обусловлена тем, что злоумышленник может получить доступ к криптографическим ключам, которые находятся в памяти коммутатора, и использовать их для прохождения процедуры аутентификации коммутатора.

Таким образом, необходимы методы, которые позволят обнаруживать скомпрометированные коммутаторы по различным косвенным признакам, например по их влиянию на сеть, то есть по наличию атак на контур передачи данных, проводимых атакующим с использованием скомпрометированного коммутатора.

В данной работе рассматриваются ПКС коммутаторы, работающие по протоколу OpenFlow. Одним их основных признаков компрометации OpenFlow коммутатора является наличие на коммутаторе правил маршрутизации, которые не были установлены контроллером [23] (далее будем называть правила — лишними). Так как по протоколу OpenFlow коммутатор не может сам устанавливать правила маршрутизации [3], то наличие лишних правил говорит о том, что их установил злоумышленник.

Для проверки наличия лишних правил на коммутаторе, контроллер должен хранить копию правил, которые он установил на каждый коммутатор. По протоколу OpenFlow контроллер может считывать с коммутаторов все правила маршрутизации, которые установлены в их памяти. Таким образом,

контроллер имеет информацию о том, какие правила должны быть установлены на каждый коммутатор и способен сравнивать их с реальными правилами, находящимися в памяти коммутатора.

Однако атакующий может создать 2 таблицы с правилами маршрутизации: в одну будут сохраняться все правила, устанавливаемые легитимным контроллером, а в другую, правила, устанавливаемые атакующим. Назовем данные таблицы *основной* и *теневой*. Наличие двух таблиц может быть использовано атакующим для обхода, описанного выше механизма защиты. Если контроллер запросит хранящиеся на коммутаторе правила, то коммутатор предоставит ему правила из *основной* таблицы, но для обработки реального сетевого трафика будут использоваться правила из *теневой* таблицы.

Таким образом, контроллер не сможет проверить наличие на коммутаторе лишних правил, потому что коммутатор будет предоставлять контроллеру некорректные данные о реально используемых для обработки трафика правилах.

Одной из проблем, связанной с проверкой наличия вредоносных правил в коммутаторах, является возможность использования в сети нескольких легитимных контроллеров [40]. Протокол OpenFlow допускает возможность использования в контуре управления нескольких контроллеров для повышения надежности сети, так как контроллер является основной точкой отказа всей сети. Контроллеры могут использовать различные схемы управления сетью, такие как *active-active* и *active-standby*. Схема *active-active* предполагает разделение управления сетью между различными контроллерами. Схема *active-standby* предполагает наличие в сети «запасных» контроллеров, которые начнут управлять сетью, только когда основной контроллер перестанет функционировать.

Таким образом, необходим механизм, который позволит синхронизировать информацию об установленных правилах маршрутизации между всеми контроллерами в сети.

Еще одним возможным подходом к проблеме обнаружения скомпрометированных коммутаторов является принцип сохранения потока (*Conservation of Flow*). Под принципом сохранения потока понимается следующее: количество пакетов, отправленных на коммутатор, равно количеству пакетов, обработанных этим коммутатором. Обработанными пакетами считаются как пакеты, полученные с портов данного коммутатора, так и сброшенные пакеты.

Нарушение этого принципа может служить признаком компрометации коммутатора. Например, пусть атакующий продублировал некоторый существующий поток с измененным заголовком для того, чтобы отправить этот поток на подконтрольный ему узел в контуре передачи данных для дальнейшего анализа. Так как новый поток был создан на коммутаторе, то количество пакетов, вышедших из коммутатора не равно количеству пакетов, поданных на него. Таким образом, принцип сохранения потока нарушен и с некоторой долей вероятности можно говорить о наличии атаки.

При обнаружении отклонений от принципа сохранения потока необходимо учитывать задержки и потери пакетов, которые происходят в сети. Поэтому, устанавливается некоторый допустимый порог отклонения от принципа сохранения потока, превышение которого свидетельствует о наличии атаки. Также необходимо не учитывать потоки, для которых на анализируемом коммутаторе установлено правило сброса.

Для подсчета количества пакетов, обработанных коммутатором, можно использовать статистику, предоставляемую протоколом OpenFlow. На каждом правиле маршрутизации, установленным на ПКС коммутатор, находит-

ся набор счетчиков, которые показывают, сколько пакетов было обработано данным правилом с момента его установки на коммутатор. Таким образом, контроллер может запросить значения данных счетчиков на всех правилах коммутатора и получить информацию, необходимую для проверки выполнения принципа сохранения потока.

Однако атакующий может фальсифицировать информацию о количестве потоков, прошедших через подконтрольный ему коммутатор, что может препятствовать использованию принципа сохранения потока. То есть на запрос контроллера о значениях счетчиков на правилах маршрутизации, атакующий может ответить некорректными значениями. Данные значения могут быть специально подобраны так, чтобы либо обойти проверку, либо создать видимость, что статистика «не сошлась» из-за данных, предоставленных некоторым легитимным коммутатором.

Таким образом, проверка принципа сохранения потока должна производиться не на самом анализируемом коммутаторе, а на коммутаторах, граничащих с ним. То есть производить подсчет количества пакетов, поданных на коммутатор и полученных из него. Процедуру проверки можно описать следующим образом:

1. Собрать статистику одновременно со всех коммутаторов.
2. Для каждого коммутатора S выбрать смежные с ним коммутаторы.
3. На смежных коммутаторах выбрать правила, которые отправляют или принимают пакеты с порта, соединяющего данный коммутатор с коммутатором S.
4. Сравнить количество входящего и исходящего трафика. Если разница превышает некоторый установленный заранее порог, то коммутатор S скомпрометирован.

Стоит отметить, что статистику необходимо собирать одновременно со всех коммутаторов, так как разница между временем сбора статистики со смежных коммутаторов может привести к учету пакетов, которые были обработаны одним коммутатором, но еще не успели обработаться вторым.

Одним из возможных обходов данного механизма обнаружения является уменьшение количества легитимного потока на величину добавляемого потока для того, чтобы не нарушать принцип сохранения потока. То есть, атакующий может специально сбрасывать некоторое количество пакетов легитимного потока таким образом, чтобы сумма входящего и выходящего трафика была одинакова.

Также, если в сети есть несколько скомпрометированных коммутаторов, то они могут кооперироваться между собой для уменьшения вероятности их обнаружения. Например, если два скомпрометированных коммутатора являются смежными, то второй коммутатор может не сообщать контроллеру о наличии некоторого нового потока с первого коммутатора, и таким образом, контроллер не сможет проверить принцип сохранения потока.

Глава 2

Угрозы безопасности ПКС

В данной главе приведено описание атак, которые злоумышленник может производить при помощи скомпрометированного ПКС коммутатора. При описании используется предположение о том, что атакующий может добавлять, удалять и модифицировать сущности в таблице потоков скомпрометированного коммутатора.

2.1. Атаки на контур передачи данных

Злоумышленник может использовать подконтрольные ему коммутаторы для проведения атак как на пользовательский трафик, так и на другие коммутаторы в сети.

2.1.1. Атаки на канал передачи данных

Сброс пакетов

Скомпрометированный коммутатор может сбрасывать любое подмножество пакетов, проходящих через его порты, таким образом, нарушая процесс передачи трафика в сети.

Изменение заголовков пакетов

Скомпрометированный коммутатор может использовать возможности протокола OpenFlow по изменению заголовков проходящих через него пакетов. Изменение заголовков пакетов может повлиять на логику обработки конкретного пакета коммутаторами, которые неподконтрольны атакующему.

Например, некоторый набор пакетов может быть направлен по другому пути, создавая тем самым перегрузки на некоторых участках сети.

Некорректная маршрутизация пакетов

Скомпрометированный коммутатор может перенаправлять или дублировать входящие в него пакеты на порт, выбранный злоумышленником. Одним из возможных применений данной атаки является дублирование некоторого трафика на сервера, контролируемые атакующим, для дальнейшего анализа.

Модификация пакетов

Скомпрометированный коммутатор способен модифицировать содержимое пакетов, проходящих через него. Такая возможность позволяет атакующему производить *Man-in-the-Middle* атаки [7] в сети.

Нарушение порядка пакетов

Скомпрометированный коммутатор может изменить порядок проходящих через него пакетов, что приведет к падению пропускной способности транспортных соединений или к отказу в обслуживании.

Искусственные задержки

Скомпрометированный коммутатор может искусственно задерживать трафик. Также, как и нарушение порядка пакетов, искусственные задержки ведут к снижению пропускной способности сети.

Подделка пакетов

Скомпрометированный коммутатор может генерировать пакеты и внедрять их в существующий поток трафика, либо создавать новые потоки трафика. Данная атака может быть использована для проведения широкого спектра атак внутри ПКС сети, например, *DDoS* атак [39] на другие устройства ПКС сети или компрометацию других коммутаторов.

2.1.2. Атаки на коммутаторы

Искажение значений счетчиков

Спецификация протокола OpenFlow [3] описывает наличие счетчиков, установленных на правилах маршрутизации, на групповых правилах и на портах коммутатора. Такие счетчики описывают количество пакетов, обработанных коммутатором.

OpenFlow не предполагает наличия уведомления контроллера в случае переполнения некоторого счетчика и метода, с помощью которого контроллер может проверить факт переполнения счетчика. Следовательно, появляется возможность производить атаку на переполнение счетчика на некотором коммутаторе без обнаружения этой атаки контроллером.

Атаки подобного типа потребуют значительного времени, так как размеры счетчиков в спецификации OpenFlow равны 64 битам.

Перенаправление потока на другой порт

При использовании механизма агрегации правил, контроллер может объединять несколько пересекающихся правил в одно правило (например, одинаковые адреса назначения и источника). Если контроллер использует протоколы ARP [41], LLDP [42] или другие протоколы для определения адресов

канального уровня, уязвимых к подмене адресов, то атакующий может произвести атаку перенаправления трафика. Атака состоит в следующем:

1. Атакующий создает некоторый вредоносный сервер и подключает его к атакуемой сети.

2. Атакующий генерирует множество пакетов с поддельными адресами источника, которые направляет на вредоносный сервер.

3. Контроллер устанавливает правила для каждого потока, идущего на вредоносный сервер.

Цель атаки состоит в том, чтобы контроллер произвел объединение правил, направляющих потоки на вредоносный сервер, и установил в адрес источника целую подсеть. Тем самым трафик, который будет проходить в данной подсети, будет перенаправлен на вредоносный сервер. Периодическая генерация подобных пакетов не позволит правилам превысить период бездействия и удалиться.

Помимо атак на протоколы ARP и LLDP, атакующий может использовать MAC адрес существующего сервера системы. Например, если контроллер не хранит отображение MAC адресов [43] на порты коммутатора, то существует возможность перенаправления трафика к вредоносному серверу с тем же MAC адресом, так как контроллер будет считать, что оригинальный сервер был подключен к другому порту. Данная атака предполагает, что правила, направляющие потоки на атакуемый сервер, были удалены по истечению времени. В таком случае, атакующий начнет генерировать поддельный трафик, который установит правила, направляющие последующий трафик на вредоносный сервер.

Проверка существования правил

Атака заключается в последовательной отправке пакетов на коммутатор и замере времени их обработки. При атаке использует факт того, что пакет, который удовлетворяет шаблону установленного на коммутаторе правила, будет обрабатываться быстрее, чем, если бы он был отправлен на контроллер для анализа.

Эта атака позволяет злоумышленнику определить, установлены ли на коммутаторе правила для некоторых потоков. Целью подобных атак является проверка статуса сети — какие именно хосты активны, и какие действия они производят.

Определение реакции контроллера

Целью данной атаки является поиск пакетов, в ответ на которые контроллер создает правила на коммутаторах. Данная атака представляет собой интерес для злоумышленника, так как она позволяет понять в какой сети работает злоумышленник: традиционной или ПКС. Атакующий отправляет два пакета из одного и того же потока на коммутатор в короткий промежуток времени, позволяя контроллеру установить потоковое правило, если это необходимо. Если потоковое правило было создано, то время ответа на второй пакет будет меньше, чем на первый.

Переполнение таблицы потоков

Для того чтобы провести атаку на таблицу потоков коммутатора, необходимо заставить контроллер генерировать множество новых потоковых правил за короткий промежуток времени. Предполагается, что для каждого нового пакета контроллер устанавливает новое правило. Следует обратить

внимание на то, что успешность атаки зависит от стратегии, используемой контроллером: реактивной или проактивной. Проактивная стратегия создает правила, которые действуют на большое количество потоков, в то время как реактивная стратегия создает потоковые правила для каждого потока отдельно. Реактивная стратегия позволяет атакующему переполнить таблицу потоков.

Переполнение входного буфера

Вместо переполнения таблицы потоков, атакующий может произвести атаку на переполнение входного буфера коммутатора. Следуя семантике протокола OpenFlow [3], если буфер коммутатора переполнен, то на анализ контроллеру будут отправляться не заголовки пакетов, а пакеты целиком. Данный факт делает возможным произведение атаки отказа в обслуживании на контроллер. Предполагая, что коммутатор не сбрасывает пакеты самостоятельно, а всегда отправляет их на контроллер, имеем следующий критерий переполнения входного буфера коммутатора.

В зависимости от того, какой интерфейс атакуется, атака может быть или не быть успешной. Если атакуемый интерфейс подключен только к атакующему его хосту, то отказ в обслуживании будет происходить только для трафика атакующего. Если к порту подключены другие клиенты, то атака будет затрагивать и их трафик тоже.

В более реалистичном сценарии, порт будет являться выходом в Интернет, и, следовательно, атака приведет к невозможности использования сети Интернет.

Стоит отметить, что на успешность атаки влияет реакция самого коммутатора на переполнение буфера. Коммутатор может сбрасывать пакеты самостоятельно в силу некоторых причин (таких как QoS [44]).

Компрометация коммутатора

Злоумышленник может использовать захваченный им коммутатор для эксплуатации уязвимостей на других коммутаторах и для распространения вредоносного программного обеспечения. В статье [9] был описан вирус-червь *Rein-worm*, который способен полностью захватить сеть, состоящую из коммутаторов *Open vSwitch* [19], за 100 секунд.

2.2. Атаки на контур управления

Злоумышленник может использовать подконтрольные ему коммутаторы для проведения атак на управляющий трафик и на контроллер, управляющий ПКС сетью.

2.2.1. Атаки на канал управления

Концепция ПКС предполагает использование двух типов управляющих каналов: *out-of-band* и *in-band* [45]. При *out-of-band* каждый коммутатор имеет выделенную физическую линию для связи с контроллером. При *in-band* часть коммутаторов использует промежуточные коммутаторы для связи с контроллером. Таким образом, при *in-band* управлении, часть маршрутов управляющих каналов может проходить через скомпрометированный коммутатор.

Успешность атак на канал управления зависит от методов защиты, используемых для защиты управляющего трафика. Протокол OpenFlow предполагает защиту управляющего трафика при помощи протокола TLS [11]. Использование TLS необходимо для того, чтобы злоумышленник мог не перехватывать и изменять управляющий трафик. Несмотря на то, что использование протокола TLS описано в спецификации протокола OpenFlow [3], этот

пункт не является опциональным. Это приводит к возможности появления в сети ошибок, когда защита не была включена администратором, и злоумышленник имеет доступ к управляющему трафику.

Сброс управляющего трафика

Злоумышленник может сбрасывать управляющий трафик, идущий между другими коммутаторами и контроллером. Вследствие этой атаки коммутаторы становятся неспособны реагировать на изменения в сети и на подключение новых клиентов. Это происходит потому, что по протоколу OpenFlow коммутаторы не могут самостоятельно устанавливать правила маршрутизации. Таким образом, производится атака отказа в обслуживании.

Перехват управляющего канала

Если в сети не используется протокол TLS для защиты управляющего трафика, то злоумышленник имеет возможность произвольно изменять сообщения, пересылаемые между коммутаторами в сети и контроллером.

Например, злоумышленник может устанавливать произвольные правила маршрутизации на коммутаторы, маршруты управляющего трафика которых проходят через скомпрометированный коммутатор. В результате, злоумышленник скомпрометирует все такие коммутаторы.

Злоумышленник также может изменять информацию о сети, передаваемую коммутаторами контроллеру. Таким образом, контроллер будет хранить некорректные сведения о состоянии сети.

2.2.2. Атаки на контроллер

Атака на отказ в обслуживании

DoS атака против контроллера может быть проведена при помощи массовой отправки поддельных *packet-in* сообщений. В зависимости от реализации логики контроллера, обработка специально созданных для целей атаки пакетов может занимать значительное количество процессорного времени.

Отказ в обслуживании контроллера может привести к угрозе отказа в обслуживании всей сети, потому что контроллер не сможет устанавливать новые правила маршрутизации на коммутаторы.

Подделка состояния коммутатора

Злоумышленник может передавать контроллеру некорректные данные о состоянии захваченного коммутатора. Вся информация, которую контроллер имеет о сети, предоставляется ему коммутаторами. И если коммутатор предоставляет некорректные данные, у контроллера нет возможности их проверить.

Злоумышленник может передавать контроллеру некорректную информацию о характеристиках коммутатора, текущей загрузке физических линий связи и статистики по потокам пользовательского трафика. Из-за некорректных данных, контроллер может неправильно реализовывать логику работы сети. Например, будет производиться некорректная балансировка нагрузки, либо для коммутации будут выбираться перегруженные порты коммутатора.

Злоумышленник также может использовать такую атаку для сокрытия своих действий от контроллера.

Подделка состояния сети

Если в сети не используется протокол TLS или он используется только для аутентификации контроллера, то злоумышленник может создавать поддельные виртуальные коммутаторы в данной сети. Эта атака может быть использована для изменения сетевой топологии, которая хранится у контроллера.

Создавая поддельные коммутаторы, атакующий может влиять на процесс выбора маршрута для потоков в сети. Например, если в сети используется маршрутизация по кратчайшему пути, злоумышленник может создать набор поддельных виртуальных коммутаторов на протяжении набора маршрутов для того, чтобы заставить контроллер выбирать для трафика один маршрут. Выбор одинакового маршрута для всех пакетов в сети может привести к перегрузке на этом маршруте и к потере пользовательского трафика. Также эта атака может быть использована для создания зарезервированной полосы пропускания для других целей злоумышленника.

Злоумышленник также может передавать контроллеру некорректные данные о топологии сети. Например, он может создавать виртуальные физические линии, исходящие из скомпрометированного коммутатора. Трафик, направленный на такие линии, будет сброшен коммутаторами без уведомления контроллера. Также злоумышленник может создавать виртуальные линии между двумя скомпрометированными коммутаторами для того, чтобы получить доступ к большему количеству трафика в сети. Например, созданная виртуальная линия может создать новый кратчайший маршрут в сети, что приведет к выбору этого маршрута для различных потоков пользовательского трафика.

Компрометация контроллера

Злоумышленник может использовать компрометацию коммутатора как шаг в достижении цели захвата контроллера. Захват коммутатора необходим потому, что у злоумышленника не может быть доступа к контроллеру из произвольной точки в сети, так как контроллер соединен только с коммутаторами.

Компрометация контроллера позволяет атакующему получить полный контроль над атакуемой сетью. Так как в сети может использоваться несколько контроллеров, единственной атакой, которая может быть проведена атакующим после компрометации контроллера, является атака на повышение привилегий. Атака заключается в том, что скомпрометированный контроллер перехватывает управление коммутаторами у других контроллеров. Данная атака целиком зависит от логики приложений, реализующих переключение коммутаторов между контроллерами.

2.3. Вывод

Большой спектр атак, которые злоумышленник может проводить при помощи скомпрометированного коммутатора, показывает серьезность угрозы захвата коммутатора. Таким образом, необходимо разрабатывать средства для обнаружения скомпрометированных коммутаторов и обнаружения атак, проводимых при помощи таких коммутаторов.

Глава 3

Обзор систем обнаружения

На данный момент существует множество систем обнаружения скомпрометированных коммутаторов. Такие системы используют различные механизмы для проверки корректности работы контура передачи данных. К таким механизмам относятся анализ сетевой статистики и тестирование сети при помощи специально созданных пакетов. Также системы обнаружения используют факт того, что у ПКС контроллера имеется полная информация о топологии сети, свойствах потоков, которые проходят через эту сеть, и правилах обработки сетевого трафика, установленных на коммутаторы.

В данной главе представлен сравнительный анализ существующих систем обнаружения скомпрометированных ПКС коммутаторов, отмечены достоинства и недостатки каждой из них. Сравнение производится по выделенным критериям, которые описывают различные аспекты систем обнаружения. Также в обзоре выделены основные недостатки существующих систем.

Необходимо отметить, что в обзоре рассматриваются только системы обнаружения коммутаторов, которые уже были скомпрометированы и использованы для различных атак на контур передачи данных. Системы обнаружения процесса компрометации не рассмотрены, так как эта область информационной безопасности уже является хорошо исследованной [46].

3.1. Критерии сравнения

Ниже сформулированы требования к системе обнаружения скомпрометированных коммутаторов в ПКС, исходя из функциональности OpenFlow коммутатора. Эти требования мы будем использовать в качестве критериев сравнения систем в этом обзоре.

К1 — Обнаружение двух и более скомпрометированных коммутаторов

Злоумышленник может захватить несколько коммутаторов в сети и организовать их согласованную работу так, чтобы избежать обнаружения.

К2 — Верификация данных, поступающих от каждого коммутатора

Злоумышленник может так скоординировать действия скомпрометированных коммутаторов, что на контроллер будет поступать некорректная статистика, искаженная так чтобы скрыть факт наличия атаки, либо заставить систему обнаружения подозревать легитимный коммутатор.

К3 — Разграничение вредоносного и легитимного сброса пакетов

Важным критерием сравнения является возможность системы отличать вредоносный сброс пакета на коммутаторе, вследствие атаки, от легитимного сброса пакета в силу наличия в памяти коммутатора соответствующего правила обработки пакетов, либо из-за перегрузки легитимного коммутатора. Если система не способна различать эти случаи сброса пакетов, то обнаружение скомпрометированных коммутаторов будет сопровождаться большим

количеством ошибок первого рода, когда сброс пакетов из-за перегрузки в сети будет восприниматься как атака.

К4 — Отсутствие требования модификаций контура передачи данных

Применение системы обнаружения скомпрометированных коммутаторов не должно требовать изменений в существующих протоколах и логике работы коммутаторов в контуре передачи данных. Изменения в логике работы коммутаторов могут быть дорогостоящими и в некоторых ситуациях неприемлемым решением. Поэтому для того, чтобы система могла быть применима в реальной сети, необходимо, чтобы она не требовала внесения изменений в существующие протоколы и логику работы коммутаторов.

К5 — Обнаружение атаки вне зависимости от ее длительности

Этот критерий описывает возможность системы обнаруживать атаки, время которых незначительно по сравнению со временем жизни потока данных в сети. Например, к таким атакам могут относиться кратковременный сброс пакетов, кратковременная *DoS* атака на некоторого пользователя или атака на определенные пакеты некоторого потока в сети.

К6 — Независимость от используемых в сети алгоритмов и политик маршрутизации

Этот критерий предполагает возможность системы работать при использовании в сети разнообразных алгоритмов и политик маршрутизации. Поскольку концепция ПКС дает большую свободу в управлении сетевым трафиком, то заранее предвидеть сложность алгоритмов и политик маршрутизации невозможно. Так же система не должна зависеть от применяемых в сети

механизмов маршрутизации и оптимизации потоков таких, как агрегация потоков, балансировка нагрузки, перенаправление трафика в случае изменений в топологии из-за ошибок в сети.

K7 — Отсутствие влияния на атаки

Процедура обнаружения атаки может влиять на состояние сети, например, устанавливать новые правила маршрутизации, генерировать новые служебные пакеты. Поэтому могут возникнуть ситуации, когда процедура обнаружения может повлиять на саму атаку, проводимую в сети. Из-за такого влияния атака может прекратиться, и, таким образом, она не будет обнаружена системой.

3.2. Существующие системы обнаружения

3.2.1. ATPG

Система ATPG[1] [47] — это система предназначена для обнаружения ошибок в работе контура передачи данных. Под ошибками понимается некорректное исполнение коммутатором установленных на него правил маршрутизации. Система ATPG обнаруживает ошибки при помощи независимого и полного тестирования всех правил маршрутизации, установленных в сети.

Тестирование правил маршрутизации проводится при помощи так называемых тестовых пакетов — специальных пакетов, посылаемых в сеть системой, для которых заранее известен маршрут. Система устанавливает в сеть дополнительные правила маршрутизации, которые должны отправлять тестовые пакеты обратно на контроллер в конце их маршрута. Таким образом,

[1] Automatic Test Packet Generation

система ATPG может проверить, что пакеты действительно прошли предполагаемым маршрутом и не были изменены в процессе.

Так как большое количество тестовых пакетов может снизить производительность сети, то система ATPG формирует минимальное множество пакетов, которое будет обработано всеми правилами маршрутизации в сети. Минимальность формируемого множества доказана авторами в [47].

Для этого система формирует таблицу достижимости между всеми портами в сети. Эта таблица описывает всевозможные маршруты, которые пакеты могут пройти в сети. Также для каждого маршрута известен набор заголовков пакетов, которые могут пройти по этому маршруту.

Для каждого маршрута система случайно выбирает один пакет из множества, соответствующего этому маршруту. В результате формируется множество пакетов P такое, что для каждого правила маршрутизации найдется пакет $p \in P$, который обрабатывается этим правилом. Если разные пакеты из этого множества обрабатываются одним и тем же правилом маршрутизации, то алгоритм выбирает минимальное подмножество пакетов, которое также обрабатывается всеми правилами маршрутизации. То есть решается задача покрытия множества подмножествами. Так как известно, что эта задача NP-сложна [48, 49], то для ее решения используется аппроксимационный алгоритм со сложностью $O(n^2)$.

Одним из недостатков этой системы является то, что обнаружение ошибок работы сети при помощи тестовых пакетов может приводить к большим ошибкам первого рода, которые могут возникать вследствие сброса пакетов, происходящего из-за перегрузок в сети. Поскольку в системе не предусмотрено никаких механизмов анализа перегрузок, возникающих в сети, система не сможет отличить вредоносный сброс пакетов атакующим от легитимного сброса из-за перегрузки.

Подход тестовых пакетов также может не обнаружить атаки, время жизни которых меньше интервала между тестированием сети. Также система может не обнаружить сложные атаки, которые производятся при помощи нескольких скомпрометированных коммутаторов.

Таким образом, система ATPG не удовлетворяет критериям **K3** и **K5**

3.2.2. FADE

Система FADE[2] [21] анализирует сетевую статистику, предоставляемую счетчиками правил маршрутизации, и проверяет ее согласованность. Под согласованностью сетевой статистики понимается следующее: разность значений счетчиков правил маршрутизации, обрабатывающих один и тот же поток, должна быть в пределах заранее определенного порогового значения.

Для анализа сетевой статистики система FADE выбирает минимальный набор потоков, которые описывают поведение счетчиков всех правил маршрутизации, установленных в сети. Для нахождения таких потоков система FADE строит граф зависимостей правил маршрутизации в соответствии с топологией и установленными в сети правилами маршрутизации. Граф зависимостей правил маршрутизации — это ориентированный граф, вершинами которого являются правила маршрутизации, а ребрами — возможные переходы пакетов между правилами. Таким образом, каждый поток в сети представлен некоторым путем в графе зависимостей правил маршрутизации.

Авторы статьи [21] предполагают наличие в сети агрегирующих правил маршрутизации — правил, которые обрабатывают сразу несколько потоков. Такие правила описываются вершинами графа с несколькими входящими ребрами и одним исходящим. Из-за наличия таких вершин следует, что потоки, описываемые путями в графе зависимостей правил, объединяются в деревья,

[2] Forwarding Anomaly Detect Environment

и этот граф представляет собой лес, где потоки направлены от листьев к корням деревьев. Везде далее, если не оговорено противное, под термином граф мы будем понимать граф зависимостей правил маршрутизации.

Далее система FADE выбирает минимальный набор путей, которые покрывают все вершины в графе. Такие пути соответствуют потокам в сети, обработка которых затрагивает все правила маршрутизации. Для каждого, таким образом, выбранного потока, система FADE генерирует набор дополнительных правил маршрутизации и устанавливает его на коммутаторы, обрабатывающие этот поток. Дополнительные правила маршрутизации, установленные системой FADE, используются для сбора сетевой статистики с коммутаторов.

Для того, чтобы устанавливаемые системой FADE правила маршрутизации обрабатывали только пакеты соответствующих им потоков, система FADE помечает пакеты выделенными VLAN метками [50]. На каждом таком правиле маршрутизации установлен *hard-timeout*, после истечения которого, правило удаляется и на контроллер отправляется статистика о количестве пакетов, обработанных этим правилом.

После сбора статистики, система FADE сравнивает значения счетчиков правил маршрутизации и делает вывод о наличии аномалии маршрутизации в сети, если эти значения отличаются более чем на заранее определенную пороговую величину.

Одним из недостатков системы FADE является то, что она не учитывает возможность наличия в сети легитимных сбросов пакетов из-за перегрузок. Также система применима только когда на контроллере реализована простая логика маршрутизации, не использующая агрегацию потоков, групповую адресацию, балансировку нагрузки и т.д. Учитывается возможность наличия

агрегирующих правил, обрабатывающих сразу несколько потоков, но затем система эти потоки не различает. Не рассматривается групповая маршрутизация, изменение заголовков пакетов и деагрегация потоков, используемая для балансировки нагрузки на коммутаторы.

В статье [21] предполагается, что одна атака нацелена на одно определенное правило маршрутизации, то есть не учитывается возможность наличия в сети нескольких скомпрометированных коммутаторов.

Также существует возможность, что во время анализа сетевой статистики система FADE своими действиями может повлиять на атаку, и таким образом ее не обнаружить. Например, если дополнительные правила маршрутизации устанавливаются на скомпрометированный коммутатор, то эти правила могут начать обрабатывать пакеты вместо вредоносных правил, и таким образом атака не будет обнаружена. При этом после анализа сетевой статистики, когда дополнительные правила будут удалены с коммутаторов, атака может продолжиться.

Кроме того, в статье не описано, каким образом система реагирует на изменения в сети.

Таким образом, система FADE не удовлетворяет критериям **К1**, **К3**, **К6** и **К7**.

3.2.3. FlowMon

Система FlowMon [51] предназначена для обнаружения скомпрометированных коммутаторов, с помощью анализа сетевой статистики, получаемой с портов коммутаторов, и сообщений о новых потоках в сети. Эта система позволяет обнаружить два типа вредоносного поведения коммутатора: нелегитимный сброс пакетов и нарушение политики маршрутизации. Под сбросом пакетов подразумевается, что на скомпрометированный коммутатор устанав-

ливают вредоносное правило, которое сбрасывает пакеты всех или части потоков, обрабатываемых скомпрометированным коммутатором. Под нарушением политики маршрутизации понимается отправка пакетов на порты, не предусмотренные правилами, которые установил контроллер.

Для обнаружения вредоносного сброса пакетов, система FlowMon анализирует значения различных счетчиков на портах коммутаторов, которые соответствуют числу переданных, полученных и сброшенных пакетов. Система выбирает некоторый коммутатор для проверки, запрашивает статистику с его портов.

Коммутатор S_k считается скомпрометированным, если коэффициент сброса пакетов превышает заранее определенное пороговое значение. То есть выполняется следующая формула:

$$\frac{\sum_{\forall i \in P_k} T_k^i - \sum_{\forall i \in P_k} R_k^i}{\sum_{\forall i \in P_k} T_k^i + \sum_{\forall i \in P_k} R_k^i} > \theta, \tag{3.1}$$

где T_k^i — количество пакетов, переданных с порта i коммутатора S_k, R_k^i — количество пакетов, полученных на порту i коммутатора S_k, и P_k это множество портов коммутатора S_k.

Также система FlowMon проверяет наличие вредоносного сброса трафика на физических линиях. Система проверяет соответствие количества пакетов, отправленных и полученных на портах, соединенных одной и той же физической линией. Если некоторые пакеты были сброшены на физической линии, то статистика на портах будет отличаться и, нижеприведенное отношение не будет выполняться:

$$\Big|T_{ij} + T_{ji} - R_{ij} - R_{ji}\Big| > \alpha\Big|T_{ij} + T_{ji}\Big|, \tag{3.2}$$

где T_{ij} — количество пакетов, переданных с коммутатора S_i на коммутатор S_j, R_{ij} — количество пакетов, полученных на коммутаторе S_i с коммутатора S_j, а α это заранее определенное пороговое значение.

Для обнаружения некорректной коммутации пакетов, система FlowMon проверяет сообщения обо всех новых потоках, появившихся в сети. Проверка производится путем анализа таблицы маршрутизации предыдущего коммутатора, то есть коммутатора, с которого был получен пакет нового потока. Контроллер проверяет, на какой порт должен был быть отправлен этот пакет. Если такой порт отличается от порта, на котором был получен пакет, то делается вывод, что коммутатор некорректно коммутирует пакеты.

Серьезным недостатком системы FlowMon является то, что при ее построении используется слишком простая модель атакующего, которая не позволяет проконтролировать подделку статистики скомпрометированным коммутатором и которая не учитывает возможность наличия в сети нескольких скомпрометированных коммутаторов.

Из-за того, что система поочередно проверяет различные коммутаторы, она не способна обнаружить кратковременные атаки, которые выполняются на коммутаторе, не участвующем в проверке в данный момент.

Не учитываются также различные способы сокрытия атак. Например, атакующий может целенаправленно коммутировать пакеты на коммутаторы, на которых уже присутствуют вредоносные правила обработки подобных пакетов. Таким образом, статистика об этом пакете никогда не попадет на контроллер. Подобная ситуация может появиться, когда злоумышленник изменяет заголовки новых потоков на заголовки существующих.

Также система не учитывает возможность наличия на коммутаторе легитимных правил, которые сбрасывают трафик. Такими правилами, например, могут быть правила, установленные межсетевым экраном.

Таким образом, система FlowMon не удовлетворяет критериям **К1**, **К2**, **К3** и **К5**.

3.2.4. FDWD

Система FDWD[3] [52] — это система для обнаружения скомпрометированных коммутаторов в ПКС, которая использует тестовые пакеты для проверки того, что коммутатор не выполняет установленные на него правила маршрутизации. Применяются 2 алгоритма обнаружения: *Forwarding Detection* и *Weighting Detection*.

Алгоритм *Forwarding Detection* использует тестовые пакеты для проверки корректности выполнения правил маршрутизации на коммутаторах. Алгоритм работает следующим образом. Случайно выбирается коммутатор S и на нем случайно выбирается правило маршрутизации f с полем *match* M. Далее алгоритм генерирует новые правила маршрутизации f' с полем *match* $M' = M \cup M_t$, где M_t — поле *match*, не пересекающееся с полем M такое, что в сети нет пакетов с заголовками, удовлетворяющих этому полю. Эти правила маршрутизации устанавливаются на коммутаторы, непосредственно соединенные с коммутатором S.

После этого, алгоритм генерирует тестовый пакет p, заголовок которого удовлетворяет полю M', и отправляет его на коммутатор S. Теперь контроллер может проверить, на какой коммутатор будет отправлен пакет p. Если правило f должно отправлять пакеты на коммутатор S', то при корректном поведении коммутатора S, пакет p должен быть обработан правилом f' на коммутаторе S'. В противном случае, коммутатор S будет считаться скомпрометированным. Также некорректным поведением будет считаться изменение пакета p.

Алгоритм *Weighting Detection* проверяет специальные групповые правила протокола OpenFlow [3], которые отправляют пакеты на разные порты в заранее заданном соотношении. Проверка заключается в том, что контроллер

[3] Forwarding Detection and Weighting Detection

отправляет на тестируемый коммутатор несколько специальных пакетов для того, чтобы затем проверить, что пакеты были отправлены на порты в правильном соотношении, заданном правилом g. Проверка происходит также, как и в алгоритме *Forwarding Detection* — при помощи специальных правил, заранее установленных на соседние с тестируемым коммутаторы.

Недостатком описанной выше системы является то, что она не учитывает возможность наличия в сети нескольких скомпрометированных коммутаторов, которые могут кооперироваться для того, чтобы избежать обнаружения. Например, если несколько соседних коммутаторов скомпрометировано, то они могут передавать контроллеру неверные сведения о прохождении тестовых пакетов в сети, таким образом скрывая наличие атаки.

Так как система проверяет одно правило за раз, то она может не обнаружить кратковременные атаки.

Из-за того, что система устанавливает дополнительные правила, она может повлиять на атаку, проводимую в сети. Например, система может установить правило, имеющее больший приоритет, чем у вредоносного правила.

Также система не учитывает наличие в сети сбросов пакетов из-за перегрузок.

Таким образом, система FDWD не удовлетворяет критериям **K1**, **K3**, **K5** и **K7**.

3.2.5. MLPC

Система MLPC[4] [4] также использует тестовые пакеты и анализ сетевой статистики для обнаружения скомпрометированных коммутаторов. В статье [4] предполагается, что скомпрометированные коммутаторы могут выборочно

[4] Minimal Legal Path Cover

сбрасывать, генерировать, изменять или некорректно коммутировать пакеты. Также предполагается, что в сети может быть несколько скомпрометированных коммутаторов, которые могут кооперироваться между собой для того, чтобы избежать обнаружения.

Для проверки корректной работы сети, система вычисляет минимальный набор тестовых пакетов, обработка которых охватывает все правила на коммутаторах в сети. Для нахождения такого набора пакетов, система использует информацию об установленных в сети правилах маршрутизации и строит граф зависимостей правил маршрутизации.

Далее система использует топологическую сортировку [53] и модифицированную версию алгоритма Хопкрофта-Карпа [54] для нахождения минимального покрытия графа путями. Причем учитываются только пути, соответствующие реальным маршрутам пакетов в сети. Назовем такой путь реализуемым. Для каждого реализуемого пути генерируется тестовый пакет, который должен пройти по всем правилам маршрутизации всех коммутаторов, соответствующим этому пути. В конце каждого пути устанавливается специальное правило маршрутизации, которое должно вернуть тестовые пакеты на контроллер для анализа. Для того, чтобы процедура анализа сети не изменяла логику обработки пакетов, тестовые пакеты помечаются уникальной меткой (например, VLAN меткой, не используемой в сети).

Если тестовый пакет был сброшен или модифицирован, контроллер уменьшает величину доверия к конкретному пути, соответствующему этому пакету. Когда величина доверия уменьшается ниже допустимого порога, путь помечается как подозрительный. Для определения местонахождения скомпрометированного коммутатора, контроллер разбивает подозрительный путь на две части и повторяет процедуру тестирования рекурсивно, пока длина подозрительного пути не будет равна 1.

Для того, чтобы обнаружить скомпрометированные коммутаторы S, которые отправляют некорректную информацию, контроллер использует коммутаторы, соединенных с S физическими линиями. Система проверяет принцип сохранения потока на входе и выходе коммутатора: то есть количество пакетов, отправленных на коммутатор, должно быть равно количеству пакетов, полученных от этого коммутатора.

Для обнаружения несоответствий в сетевой статистике, контроллер устанавливает на соседние коммутаторы дополнительные правила, которые считают количество пакетов, отправленных на тестируемый коммутатор, и количество пакетов, полученных от тестируемого коммутатора.

Далее проверяется принцип сохранения потока. Если найдено несоответствие, то уровень доверия к коммутатору снижается. Если уровень доверия снизится ниже допустимого порога, то коммутатор считается скомпрометированным.

Важно отметить, что подобная проверка статистики может быть неспособна обнаружить точное местоположение скомпрометированного коммутатора. Например, скомпрометированный коммутатор может предоставлять контроллеру специально сформированную сетевую статистику так, чтобы снижать уровень доверия системы к некоторому легитимному коммутатору. Также в ситуации, когда легитимный коммутатор окружен скомпрометированными, его уровень доверия может снижаться быстрее, чем у граничащих с ним коммутаторов.

Система не учитывает случай, когда скомпрометированные коммутаторы могут быть соседними, то есть соединены физической линией. В таком случае один из них может скрывать от контроллера информацию о потоке, выходящем с другого коммутатора. Также система не отличает легитимный

сброс пакетов от вредоносного и не рассматривает агрегацию потоков, балансировку нагрузки и другие сложные механизмы маршрутизации.

Из-за того, что система проводит выборочные проверки правил маршрутизации при помощи тестовых пакетов и анализа сетевой статистики, она может пропустить кратковременные атаки.

Также из-за установки правил, система влияет на состояние сети и, как следствие, влияет на проводимые в сети атаки.

Таким образом, система MLPC не удовлетворяет критериям **K1**, **K2**, **K3**, **K5**, **K6** и **K7**.

3.2.6. PDMD

Система PDMD[5] [55] — это система обнаружения скомпрометированных коммутаторов путем проверки маршрутов, пройденных пакетами в сети.

Для выявления реальных маршрутов, пройденных пакетами, система устанавливает в сети правила маршрутизации, которые дублируют часть пакетов на контроллер. Выбор пакетов для отправки на контроллер производится при помощи алгоритма *Trajectory Sampling* [56]. Этот алгоритм выбирает пакеты на основе решающей функции от метки пакета. Под меткой понимается набор полей заголовка пакета, которые уникально идентифицируют пакет в сети. Авторы статьи предполагают, что в сети не установлены правила маршрутизации, которые изменяют заголовки пакетов. То есть значение решающей функции от одного и того же пакета одинаково на любом участке сети. Следовательно, выбранный для анализа пакет будет отправляться на контроллер на всех коммутаторах, на которых он был обработан.

Таким образом, система может получать из сети информацию о реальных маршрутах, пройденных пакетами. Далее, на основе информации об

[5] Packet Drops and Misroutings Detector

установленных в сети правилах маршрутизации, система вычисляет теоретический маршрут, который каждый такой пакет должен был пройти в сети. Теоретический маршрут вычисляют по алгоритму достижимости из модели *Header Space Analysis* [57].

Далее система может сравнить реальный маршрут, пройденный выбранным пакетом, с теоретическим маршрутом. Если маршруты не совпадают, то делается вывод о наличии в сети скомпрометированного коммутатора.

Недостатком системы является то, что при анализе маршрутов не учитывается то, что в сети может находиться несколько скомпрометированных коммутаторов, и они могут отправлять на контроллер некорректную информацию о маршрутах пакетов, влияя на процедуру обнаружения. Например, такие коммутаторы могут не отправлять пакеты на контроллер, тем самым скрывая факт атаки. Также они могут изменять заголовки тестовых пакетов для того, чтобы в дальнейшем эти пакеты не были отправлены на контроллер другими легитимными коммутаторами.

В статье [55] предполагается, что в сети не может быть перегрузок и все сброшенные пакеты, были сброшены из-за атаки, а не из-за перегрузки. Также предполагается, что задержка обработки пакетов на коммутаторах и контроллере незначительна.

Из-за того, что алгоритм *Trajectory Sampling* [56] может не выбрать для анализа пакеты, относящиеся к некоторой проводимой в сети атаке, то эта атака не будет обнаружена. Такое может произойти, например, в случае наличия в сети кратковременной атаки.

Также необходимо отметить, что система рассматривает только простую логику маршрутизации — отдельные правила для различных потоков.

Из сказанного следует, что система PDMD не удовлетворяет критериям **К1**, **К2**, **К3**, **К5** и **К6**.

3.2.7. SPHINX

Система SPHINX [23] — это система обнаружения вторжений в ПКС сетях. Она обнаруживает различные атаки, направленные на контур управления, и проверяет соблюдение политики безопасности. Также эта система позволяет обнаруживать скомпрометированные коммутаторы, которые проводят различные атаки на контур передачи данных.

Для обнаружения скомпрометированных коммутаторов система SPHINX анализирует устанавливаемые в сеть правила и строит модель сети — потоковый граф. Этот граф описывает маршруты различных потоков в сети. Потоковый граф используется для анализа сетевой статистики и обнаружения аномалий, таких как нелегитимный сброс пакетов.

Анализ сетевой статистики происходит следующим образом: для каждого потока в сети система запрашивает значения счетчиков правил, входящих в маршрут этого потока. Если счетчики правил одного и того же потока отличаются больше, чем на заранее определенное пороговое значение, то система SPHINX сообщает о наличии в сети скомпрометированного коммутатора.

Недостатком системы SPHINX является то, что она не учитывает возможность использования в сети сложных механизмов маршрутизации, таких как групповая маршрутизация, агрегация потоков и балансировка нагрузки. Все потоки представляются маршрутами в сети, для которых установлены уникальные правила маршрутизации. Также не учитывается информация о пакетах, сброшенных из-за перегрузок в сети.

Таким образом, система SPHINX не удовлетворяет критериям **К3** и **К6**.

3.2.8. RDDF

Система RDDF[6] [58] — это система обнаружения скомпрометированных коммутаторов, использующая тестовые пакеты для проверки работы правил маршрутизации в сети. В этой системе использован алгоритм оптимизации процедуры обнаружения, в котором один тестовый пакет может проверить более одного коммутатора. Оптимизация достигается за счет агрегации правил маршрутизации с совпадающими или непересекающимися полями *match*.

Как и система ATPG [47], эта система генерирует тестовые пакеты на основе информации о правилах маршрутизации. Для каждого тестового пакета эта система вычисляет маршрут, которые этот пакет должен пройти, и сравнивает его с реальным маршрутом, пройденным пакетом.

Процедура обнаружения оптимизируется за счет объединения нескольких тестовых пакетов в один. Маршруты для тестовых пакетов представляются так называемыми деревьями агрегации (*aggregation trees*). Дерево агрегации — это дерево, в котором вершины соответствуют коммутаторам, а каждый путь от корня к листу описывает путь для тестового пакета в сети.

Дерево агрегации содержит:

- Начальный коммутатор — коммутатор, который является стартовой точкой движения тестовых пакетов;
- Промежуточные коммутаторы — коммутаторы, которые дублируют тестовые пакеты на различные порты;
- Листовые коммутаторы — коммутаторы, отправляющие тестовые пакеты на контроллер.

[6] Rapid Detection of Disobedient Forwarding

Для каждого дерева агрегации в сети устанавливают правила маршрутизации из, так называемой, группы агрегации. Группой агрегации называется множество правил маршрутизации, которые удовлетворяют условиям:

- Правила маршрутизации из одной и той же группы на различных коммутаторах имеют либо совпадающие, либо непересекающиеся поля $match$;
- Тестовый пакет не должен проходить через один и тот же коммутатор дважды;
- Правило маршрутизации A может принадлежать нескольким группам агрегации только если существует правило маршрутизации B на другом коммутаторе такое, что правила A и B имеют совпадающие поля.

Для построения деревьев агрегации необходимо найти минимальное количество групп агрегации при условии, что все правила маршрутизации в сети принадлежат хотя бы одной группе агрегации. Эта задача эквивалентна задаче нахождения минимального вершинного покрытия для ориентированного графа, которая является NP-трудной [48, 49, 58].

Каждый тестовый пакет создается так, чтобы он мог быть обработан всеми правилами маршрутизации только из одной и той же группы агрегации. Такой тестовый пакет движется по дереву маршрутов, дублируется в промежуточных вершинах и отправляется на контроллер в листовых вершинах дерева. Если в сети нет скрытых правил, то все тестовые пакеты должны быть отправлены на контроллер из листовых вершин. Таким образом, контроллер может проверить, что каждый коммутатор выполняет все правила маршрутизации.

Эта система обладает тем же недостатком, что и любая система, использующая тестовые пакеты — система не способна отличить вредоносный сброс

пакетов, появившийся в следствие атаки на сеть, от легитимного сброса из-за перегрузок в сети, и не способна обнаружить кратковременные атаки.

Таким образом, система RDDF не удовлетворяет критериям **К3** и **К5**.

3.2.9. WedgeTail

Система WedgeTail [59] анализирует маршруты, пройденные пакетами в сети, для обнаружения скрытых правил маршрутизации.

Эта система перехватывает OpenFlow сообщения, передаваемые между контроллером и коммутаторами, и строит у себя виртуальную копию сети, которая затем используется для вычисления теоретических маршрутов движения пакетов в сети. Теоретические маршруты вычисляются при помощи алгоритма вычисления достижимости из модели *Header Space Analysis* [57].

Для определения реальных маршрутов, пройденных пакетами, система использует систему NetSight [60]. NetSight — это решение для обнаружения аномалий в сети, которое позволяет приложению ПКС контроллера получать информацию о коммутаторах, пройденных пакетом. NetSight требует модификации контура передачи данных для того, чтобы генерировать отчеты о пройденных пакетом коммутаторах.

После сбора информации о реальных маршрутах, пройденных пакетами, система WedgeTail сравнивает эти маршруты с теоретическими. Если реальные маршруты пакетов не являются подмножеством теоретических маршрутов, то некоторые коммутаторы на пути пакета являются скомпрометированными.

Для ускорения обнаружения скомпрометированных коммутаторов система WedgeTail упорядочивает коммутаторы на основе приоритета. Приоритет вычисляется на основании количества трафика, обрабатываемого коммутатором. Основная идея заключается в том, что проверка должна проводиться в

первую очередь на тех коммутаторах, которые обрабатывают большее количество пакетов. Для определения таких коммутаторов, WedgeTail отслеживает траектории всех пакетов на всех портах и определяет наиболее часто задействованные коммутаторы.

Одним из недостатков системы WedgeTail является то, что для ее использования необходимо изменять логику работы коммутатора, потому что эта система использует NetSight для сбора информации о маршрутах, пройденных пакетами. Таким образом, ограничивается область применения этой системы в существующих сетях.

Система WedgeTail может не обнаружить кратковременные атаки из-за того, что проверка проводится поочередно на различных коммутаторах в сети. Также система WedgeTail не разграничивает легитимный и вредоносный сброс трафика.

Таким образом, система WedgeTail не удовлетворяет критериям **K3**, **K4** и **K5**.

3.2.10. DYNAPFV

DYNAPFV [61] — это система для проверки корректности обработки пакетов на коммутаторах. Первый коммутатор, обрабатывающий пакет, называется входным для этого пакета. Выходным коммутатором для пакета, называется коммутатор, который отправил этот пакет во вне сети, например, на некоторый хост. Проверка происходит при помощи сравнения количества пакетов, полученных на входных и выходных коммутаторах.

Система устанавливает на граничные коммутаторы правила маршрутизации, которые при помощи *packet-in* сообщений [3] отправляют пакеты на контроллер для анализа. Система выбирает случайные промежутки времени

для сбора пакетов из различных потоков, то есть, на контроллер отправляется только часть пакетов, проходящих в сети.

Каждый пакет, пришедший на контроллер в *packet-in* сообщении, сохраняется до того момента, когда этот пакет не придет в другом сообщении с выходного коммутатора, либо когда истечет время, равное максимальному RTT [62] в сети. Таким образом происходит проверка того, что пакеты не были модифицированы в процессе их обработки промежуточными коммутаторами.

Если отношение количества корректно обработанных пакетов (которые пришли и со входного и с выходного коммутаторов) к количеству всех пакетов, отправленных на контроллер, превышает заранее заданное пороговое значение, то система сообщает об ошибке и начинается поиск скомпрометированных коммутаторов.

Во время поиска скомпрометированных коммутаторов выбирается набор промежуточных коммутаторов между входным и выходным коммутаторами, на которых удаляются соответствующие потоковые правила так, чтобы эти коммутаторы начинали отправлять пакеты на контроллер.

Также система проверяет корректность обработки пакетов в случае, когда таймауты потоковых правил истекают. Система извлекает соответствующие значения сетевой статистики и проверяет корректность обработки пакетов, сравнивая значения статистики с разных коммутаторов. Если отношение количества пакетов, обработанных на некотором коммутаторе S, к количеству пакетов, обработанных входным коммутатором, меньше заранее определенного порогового значения, то коммутатор, предшествующий S, считается скомпрометированным.

Для снижения нагрузки на управляющий канал, в системе используется динамическая настройка вероятности сбора пакетов для различных потоков.

Недостатком системы DYNAPFV является то, что рассматривается только одна процедура маршрутизации — динамическая установка правил на коммутатор в случае появления пакета, для которого нет правила обработки. Кроме того, не учитываются пакеты, сброшенные из-за перегрузки сети, то есть наличие легитимно сброшенного трафика будет проинтерпретировано системой как атака на сеть.

Из-за того, что система DYNAPFV запрашивает большое количество пакетов из сети, она может негативно повлиять на производительность контроллера и на загрузку управляющего канала.

Система DYNAPFV не обнаруживает атаки, когда в сети есть несколько скомпрометированных коммутаторов, которые передают на контроллер некорректные данные о пакетах, проходящих в сети.

Таким образом, система DYNAPFV не удовлетворяет критериям **К1**, **К2**, **К3** и **К6**.

3.3. Сравнительный анализ

В таблице 3.1 представлены сводные результаты анализа и сравнение систем обнаружения скомпрометированных ПКС коммутаторов, приведенные выше.

Из этой таблицы хорошо видно, что существующие системы обнаружения скомпрометированных ПКС коммутаторов используют 3 основных механизма обнаружения:

- Анализ значений счетчиков правил маршрутизации;
- Генерацию тестовых пакетов и проверку маршрутов, которые они проходят;
- Сбор существующих пакетов и анализ их маршрутов.

Одним из недостатков существующих систем обнаружения скомпрометированных коммутаторов заключается в том, что подобные системы рассматривают только самый простой механизм обработки пакетов в сети — динамическую установку отдельных правил маршрутизации для каждого потока в сети. Не учитывается групповая маршрутизация, агрегация правил и балансировка нагрузки. Таким образом, не учитывается тот факт, что ПКС позволяет реализовывать различные сложные механизмы маршрутизации и оптимизации потоков в сети. Сложные механизмы маршрутизации могут привести к очень непростым комбинациям потоковых правил, установленных на коммутаторах. Поэтому необходим алгоритм обнаружения скомпрометированных коммутаторов, который сможет работать при произвольной комбинации потоковых правил в сети.

Из таблицы 3.1 также видно, что ни одна из существующих систем не способна отличить вредоносный сброс пакетов от легитимного сброса из-за

	K1	K2	K3	K4	K5	K6	K7
ATPG	✔	✔		✔		✔	✔
FADE		✔		✔	✔		
FlowMon				✔		✔	✔
FDWD		✔		✔		✔	
MLPC				✔			
PDMD				✔			✔
SPHINX	✔	✔		✔	✔		✔
RDDF	✔	✔		✔		✔	✔
WedgeTail	✔	✔				✔	✔
DYNAPFV				✔	✔		✔

Таблица 3.1: Сравнение систем обнаружения скомпрометированных коммутаторов

перегрузок в сети. Поскольку перегрузки в сети — достаточно частое явление, то невозможность выявлять причину сброса пакета будет вызывать большое количество ошибок первого рода.

Системы, использующие тестовые пакеты или сбор существующих пакетов, не могут различать типы сброса трафика из-за того, что протокол OpenFlow не предполагает отправки коммутатором на контроллер уведомления о каждом сброшенном пакете. Даже учитывая то, что информация о количестве сброшенных пакетов записана в статистике порта, у контроллера нет информации о том, что был сброшен именно тестовый пакет.

Также существующие системы, которые используют анализ сетевой статистики, не учитывают информацию о числе сброшенных пакетов, которая хранится в счетчиках портов.

Еще одной проблемой существующих систем, которые анализируют сетевую статистику, является уязвимость к сокрытию факта атаки. То есть злоумышленник, сбрасывая некоторое количество пакетов, может при этом генерировать такое же количество новых пакетов для того, чтобы принцип сохранения потока на коммутаторе выполнялся.

Кроме того, системы, использующие анализ сетевой статистики, используют статистику только с граничных коммутаторов по отношению к проверяемому. Таким образом, если атакующий скомпрометировал несколько соседних коммутаторов, у него появляется возможность отправлять контроллеру некорректную сетевую статистику так, что атака не будет обнаружена.

3.4. Вывод

В данной главе был проведен обзор и сравнение существующих систем обнаружения скомпрометированных ПКС коммутаторов. Сравнение производилось по определенным критериям, которые охватывают различные аспекты систем обнаружения. Также в обзоре были выделены основные недостатки существующих систем. Необходимо отметить, что в обзоре рассмотрены только системы обнаружения скомпрометированных коммутаторов, которые уже были скомпрометированы и использованы для различных атак на контур передачи данных. Системы обнаружения процесса компрометации не рассмотрены, так как эта область информационной безопасности уже является хорошо исследованной [46].

Из проведенного обзора средств обнаружения скомпрометированных ПКС коммутаторов следует, что для надежного выявления скомпрометированных коммутаторов необходимо разработать систему, которая будет работать при условии использования в сети различных сложных механизмов и

оставаться прозрачной для атакующего. Для этого система должна не зависеть от внутренней логики работы контроллера. Система, удовлетворяющая этим требованиям, должна строить представление сети на основе *FlowMod* сообщений, отправленных контроллером на коммутаторы.

Важно, чтобы система обнаружения учитывала информацию о пакетах, сброшенных вследствие перегрузки в сети. Это необходимо для того, чтобы избежать ошибок первого рода во время перегрузок.

Глава 4

Модель сети

В этой главе представлено описание разработанной математической модели ПКС сети, описывающей динамику изменения счетчиков правил маршрутизации в сети с произвольной комбинацией установленных правил маршрутизации.

ПКС сети предоставляют возможность анализировать текущее состояние сети при помощи счетчиков, установленных на правилах маршрутизации [33, 34]. Счетчик каждого правила маршрутизации хранит информацию о количестве пакетов, которые были обработаны этим правилом с момента его установки на коммутатор.

Счетчики, установленные на коммутаторы, могут быть использованы для разработки алгоритма обнаружения скомпрометированных коммутаторов. Основная идея алгоритма обнаружения заключается в том, что мы можем предсказывать значения счетчиков правил маршрутизации, основываясь на информации о сети, имеющейся на контроллере, и сравнивать их с реальными значениями. Если значения отличаются, то в сети присутствует скомпрометированный коммутатор (глава 5).

Задача предсказания значений счетчиков правил маршрутизации по набору выделенных счетчиков решается в [63]. Проблема заключается в том, что в этой статье задача предсказания рассматривается в сокращенной постановке — предполагается, что для каждого сетевого потока установлены выделенные правила маршрутизации, то есть в сети отсутствуют сложные механизмы маршрутизации, такие как агрегация потоков или балансировка

нагрузки. Такая постановка не описывает случай сетей телеком операторов и центров обработки данных [1].

Таким образом, нужен алгоритм для предсказания счетчиков правил маршрутизации в сети с произвольной комбинацией установленных правил маршрутизации. Для разработки такого алгоритма необходимо разработать математическую модель ПКС сети, которая будет описывать динамику изменения счетчиков правил маршрутизации.

К настоящему моменту существует несколько математических моделей ПКС сетей [57, 64, 65, 66, 67, 68]. Однако, одни модели нацелены на проверку сетевых политик и корректности протоколов [64, 65, 66, 67], в то время как другие нацелены на обнаружение ошибок администрирования сети, таких как циклы маршрутизации или нарушение связности сети, и решение проблем с изоляцией и утечкой трафика [57, 68]. Ни одна из существующих моделей не описывают динамику изменения счетчиков правил маршрутизации.

В настоящей главе представлена математическая модель ПКС сети, работающей по протоколу OpenFlow [3], описывающая значения счетчиков правил маршрутизации как потоковую функцию [69] на графе зависимостей правил [68]. В этой главе также описан алгоритм предсказания значений счетчиков правил маршрутизации.

4.1. Граф зависимостей правил

Для разработки алгоритма предсказания значений счетчиков правил маршрутизации необходимо разработать модель ПКС сети, которая описывает динамику сетевых потоков. Разработанная модель представляет собой развитие существующих моделей ПКС, таких как *Header Space Analysis* [57] и графа зависимостей правил (*Rule Dependency Graph*) [68].

Основная идея модели *Header Space Analysis* заключается в абстракции понятия заголовка пакета от конкретных протоколов, составляющих данный заголовок. Пакеты представлены как точки некоторого пространства размерности L, где L — максимальный размер заголовка, а сетевые устройства представляются функциями, заданными на том же пространстве.

Основной идеей графа зависимостей правил является то, что в модельном графе сети вершинами являются не сетевые устройства, а конкретные правила маршрутизации, установленные на данные сетевые устройства. Ребрами в модельном графе являются возможные переходы пакетов между правилами маршрутизации.

Представление сети в виде графа зависимостей правил позволяет описывать произвольные комбинации правил маршрутизации, установленные в сети. Таким образом, возможно описание как сложных механизмов маршрутизации (агрегация потоков и балансировка нагрузки), так и ошибок маршрутизации (конечные циклы маршрутизации).

Для разработанной модели ПКС сети доказано правило прохождения потока через вершины графа зависимостей правил и показано, как эта модель может быть использована для разработки алгоритма предсказания значений счетчиков правил маршрутизации. Также в этой главе строго формализуется и расширяется модель *Rule Dependency Graph*, которая была впервые неформально описана в [68].

4.1.1. Пространство заголовков

В этой главе представлено подробное описание модели *Header Space Analysis* [57].

Пространством заголовков пакетов называется множество $\mathcal{H}$, состоящее из векторов $h \in \{0,1\}^L$, где L — верхняя граница длины заголовков пакетов.

Сетевым пространством называется множество $\mathcal{N}$, являющееся декартовым произведением $\mathcal{H} \times \mathcal{P}$, где $\mathcal{P}$ — множество портов всех коммутаторов. Это пространство представляет собой все возможные заголовки пакетов, находящиеся на всех возможных портах коммутаторов. *Сетевым пространством коммутатора* W назовем множество $\mathcal{N}_W = \mathcal{H} \times \mathcal{P}_W \subseteq \mathcal{N}$, где $\mathcal{N}$ — сетевое пространство. Это пространство представляет собой все возможные заголовки пакетов, проходящих через порты коммутатора W. Любое подмножество $D \subseteq \mathcal{N}$ будем называть *доменом*.

Передаточной функцией коммутатора W назовем функцию $T_S : \mathcal{N} \to 2^{\mathcal{N}}$, которая моделирует работу коммутатора W по обработке пакетов.

$$T_W(h,p) = \big\{(h_1,p_1),(h_2,p_2),\ldots,(h_k,p_k)\big\}, \tag{4.1}$$

где $p_1, p_2, \ldots, p_k \in \mathcal{P}_W$. Необходимо отметить, что пары (h_i, p_i) не обязательно должны быть различными, т.е. значением функции $T_W(h,p)$ является мультимножество [70].

Сужением передаточной функции T_W *на порт* $r \in \mathcal{P}_W$ назовем следующую функцию:

$$T_W^r(h,p) = \big\{(h',r) \mid (h',r) \in T_W(h,p)\big\}. \tag{4.2}$$

Таким образом, передаточную функцию можно представить следующим образом:

$$T_W(h,p) = \bigcup_{r \in \mathcal{P}_W} T_W^r(h,p). \tag{4.3}$$

Доменом D_{T_W} *передаточной функции* T_W называется множество всевозможных пар (h,p), которые принимаются этой функцией в качестве аргумен-

тов и не отображаются в пустое множество. То есть:

$$D_{T_W} = \big\{(h,p) \mid T_W(h,p) \neq \varnothing\big\}. \tag{4.4}$$

Для передаточной функции T_W каждого коммутатора определена *обратная функция* $T_W^{-1} : \mathcal{N} \to 2^{\mathcal{N}}$ следующего вида:

$$T_W^{-1}(h,p) = \big\{(h',p') \mid (h,p) \in T_W(h',p')\big\}. \tag{4.5}$$

Сетевой передаточной функцией называется функция $\Psi : \mathcal{N} \to 2^{\mathcal{N}}$, которая представляет собой композицию всех передаточных функций и моделирует работу всех коммутаторов.

$$\Psi(h,p) = \begin{cases} T_{W_1}(h,p), & \text{если } p \in W_1, \\ \dots & \\ T_{W_n}(h,p), & \text{если } p \in W_n. \end{cases} \tag{4.6}$$

Топологической передаточной функцией называется функция $\Gamma : \mathcal{N} \to \mathcal{N}$, которая моделирует поведение каналов связи, передающих пакеты с порта одного устройства на порт другого.

$$\Gamma(h,p) = \begin{cases} \big\{(h,p^*)\big\}, & \text{если } p \textit{ соединен с } p^*, \\ \big\{(h,p)\big\}, & \text{если } p \textit{ не соединен.} \end{cases} \tag{4.7}$$

Используя приведенные выше передаточные функции, возможно смоделировать поведение пакета, который проходит некоторый путь в сети, применяя на каждом сетевом устройстве, находящемся на пути следования пакета, функцию $\Phi(h,p) = \Gamma(\Psi(h,p))$. Например, если пакет с заголовком h приходит на некоторый порт p, то его заголовок после прохождения k коммутаторов будет равен

$$\Gamma\Big(\Psi\big(\dots\Gamma\big(\Psi(h,p)\dots\big)\big)\Big), \tag{4.8}$$

или более кратко $\Phi^k(h,p)$.

4.1.2. Формализация графа зависимостей правил

В этой главе представлена строгая формализация и расширение существующей модели ПКС, называемой *графом зависимостей правил* [68]. Расширение модели заключается в возможности описания множественных таблиц маршрутизации, групповых правил и набора различных действий, производимых правилами маршрутизации.

Графом зависимостей правил назовем ориентированный граф $G = G(V, A)$, где V — множество вершин, и A — множество ориентированных ребер. Множество вершин V графа G построим следующим образом.

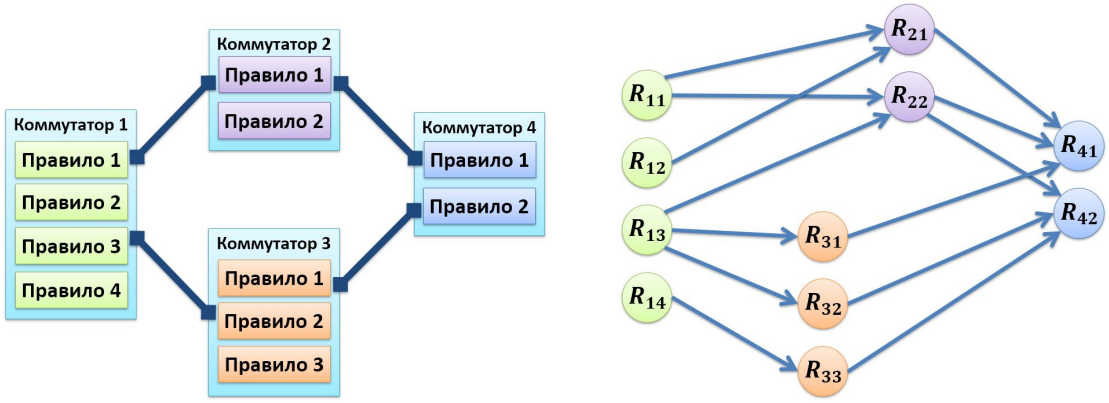
Для каждого правила маршрутизации, установленного в сети, добавим в множество V вершину v. Для каждого группового правила маршрутизации, добавим вершины $g, b_1, \ldots, b_k$, где k — количество *bucket*-записей [3] этого группового правила. *bucket*-записи описывают действия над копиями пакета, которые производят групповые правила маршрутизации.

Для каждого порта $p \in \mathcal{P}$ добавим в множество V пару вершин s и t, называемых *источником* и *стоком порта* p. Эти вершины описывают клиентов, подключенных к коммутаторам. Также, добавим в множество V два выделенных стока c и d, которые будем называть *контроллером* и *сбросом*. Множества всех источников и стоков в графе G будем обозначать как S и T соответственно. Будем считать, что $S \neq \varnothing$, $T \neq \varnothing$ и $V \setminus (S \cup T) \neq \varnothing$.

Далее, когда будем говорить "вершина v", то будем подразумевать вершину $v \in V$, а когда будем говорить "правило v", будем подразумевать правило маршрутизации, которому соответствует вершина v.

Каждой вершине $v \in V$ поставим в соответствие *передаточную функцию* $T_v : \mathcal{N} \to 2^{\mathcal{N}}$:

$$T_v(n) = \left\{n_1, n_2, \ldots, n_{\gamma(v)}\right\}, \tag{4.9}$$



(a) Топология сети (b) Граф зависимостей правил

Рисунок 4.1: Построение графа зависимостей правил

где $n, n_1, \ldots, n_{\gamma(v)} \in \mathcal{N}$. Передаточная функция T_v моделирует работу правила v по обработке пакетов. Если вершина v является истоком или стоком, то передаточная функция T_v единична, то есть $T_v(n) = \{n\}$. Число $\gamma = \gamma(v)$ называется *степенью дублирования вершины* v.

Доменом D_{T_v} *передаточной функции* T_v назовем множество всех доменов n, которые не преобразуются передаточной функцией в пустое множество:

$$D_{T_v} = \{n \mid T_v(n) \neq \varnothing\}. \tag{4.10}$$

Домен D_{T_v} описывает поле *match* правила v, то есть множество пакетов, которые обрабатываются этим правилом.

Необходимо отметить, что $\left|T_v(n)\right| = \gamma(v) \quad \forall n \in \mathcal{N}$, то есть степень дублирования фиксирована для каждой вершины, потому что значения n_i на выходе передаточной функции описывают список действий (*OpenFlow action list* [3]), который правило v выполняет при обработке пакета.

Дублирующим правилом назовем такое правило v, что существует домен $n \in \mathcal{N}$ такой, что $\gamma(v) > 1$. Истоки $s \in S$ и стоки $t \in T$ не являются дублирующими вершинами по определению.

Также для каждой вершины $v \in V$ определим *обратную передаточную функцию* $T_v^{-1} : \mathcal{N} \to 2^{\mathcal{N}}$:

$$T_v^{-1}(n) = \left\{ n' \mid n \in T_v(n') \right\}. \tag{4.11}$$

Каждой вершине v, не являющейся истоком, стоком, группой или *bucket*-записью, поставим в соответствие натуральное число pr_v, называемое *приоритетом* данной вершины. Определим на множестве вершин V отношение частичного порядка «$\succ$» следующим образом:

$$v \succ u \Leftarrow \begin{cases} pr_v > pr_u, \\ v, u \in V_\tau, \end{cases} \tag{4.12}$$

где V_τ — множество вершин, соответствующих правилам в таблице τ. То есть вершины сравнимы, если они соответствуют правилам из одной таблицы.

Приоритет вершины v описывает поле *priority* правила маршрутизации, соответствующего этой вершине. Таким образом, описанное отношение частичного порядка представляет собой очередность, в которой проверяется, какое правило должно обработать некоторый пакет. Например, пакет с заголовком h, поступивший на порт p коммутатора S, будет обработан максимально приоритетным правилом v данного коммутатора таким, что $(h, p) \in D_{T_v}$.

Протокол OpenFlow описывает, что наличие правил с одинаковым приоритетом и пересекающимися доменами ведет к неопределенному поведению [3]. Поэтому, мы можем предположить, что домены правил, имеющих одинаковый приоритет, не пересекаются.

Каждой вершине v поставим в соответствие множество $D_v \subseteq \mathcal{N}$, называемое *доменом вершины* v. Под доменом вершины v понимается множество заголовков пакетов, которые могут быть обработаны этим правилом с учетом приоритета, то есть:

$$D_v = D_{T_v} \setminus \bigcup_{w \succ v} D_{T_w}. \tag{4.13}$$

Множество заголовков пакетов может измениться после обработки правилом v, так как в своей работе правило v может изменять заголовок некоторого пакета, тем самым получая новый пакет. Например, пусть правило v обрабатывает пакеты с некоторой парой $\langle IP_{src}, IP_{dst} \rangle$ с любого порта, устанавливает для таких пакетов некоторый *VLAN ID* [50] и отправляет на порт p'. Тогда доменом D_v этой вершины будут все пары (h, p), где h — заголовки пакетов с соответствующими значениями *IP* адреса источника и получателя, а p — все порты коммутатора, на котором установлено правило v. Множество $T_v(D_v)$ будет состоять из всех пар (h', p'), где h' — заголовки из D_v с установленным *VLAN ID*.

Пусть вершина $v \in S \cup T$, то есть является либо истоком, либо стоком некоторого порта p. Поставим этой вершине в соответствие домен, равный $\mathcal{N}_p = \mathcal{H} \times p$, то есть всему сетевому пространству порта p. Контроллеру c и стоку d поставим в соответствие домен $\mathcal{N}$, описывающий все сетевое пространство.

Множество ориентированных ребер A графа G построим следующим образом.

Пусть некоторая вершина v отправляет пакеты в таблицу τ. Тогда соединим вершину v ребрами с каждой вершиной $w \in V_\tau$. Если вершина v отправляет пакеты в групповое правило маршрутизации, которое соответствует вершинам $g, b_1, \ldots, b_k$, добавим ребро vg. Также, для каждого группового правила добавим ребра gb_i для каждого $i \in [1, k]$.

Если порт p подключен к некоторому порту p^* физической линией, то соединим вершину v ребрами с каждой вершиной $w \in V_{\tau_0}$, где τ_0 — нулевая таблица коммутатора, содержащего порт p^*. Если порт p не подключен, то

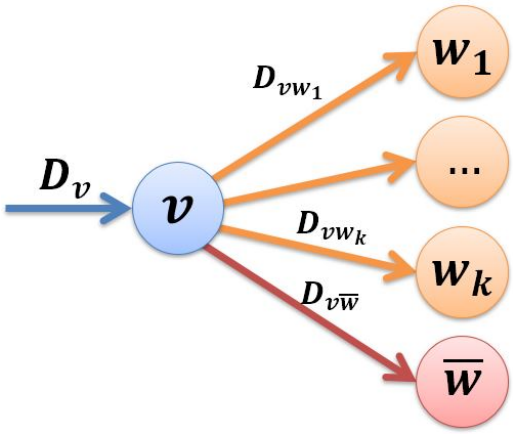


Рисунок 4.2: Ребра графа зависимостей правил

добавим ребро vt, где t — сток порта p. Если правило отправляет пакеты на контроллер или сбрасывает, то добавим ребра vc и vd соответственно.

Исток s каждого порта p, не соединенного физической линией с другим портом сети, соединим ребрами со всеми правилами $w \in V_{\tau_0}$, где τ_0 — нулевая таблица коммутатора, содержащего порт p.

Ребра графа зависимостей правил показывают возможные пути следования пакетов между правилами маршрутизации (рис. 4.2). То есть, если существует ребро $a = vw$, это значит, что существуют пакеты, которые после обработки правилом v будут обработаны правилом w.

Каждой вершине $v \in V$ поставим в соответствие *сетевую передаточную функцию* $\Phi_v : \mathcal{N} \to 2^{\mathcal{N}}$. Эта функция моделирует модификацию доменов при прохождении через вершину v в графе зависимостей правил. Если $T_v(n) = \{n_1, n_2, \ldots, n_{\gamma(v)}\}$ и $n_i = (h_i, p_i)$, где h_i — заголовок, а p_i — порт, то:

$$\Phi_v(n) = \{(h_1, p_1^*), (h_2, p_2^*), \ldots, (h_{\gamma(v)}, p_{\gamma(v)}^*)\}, \tag{4.14}$$

где p_i^* — порт, соединенный с портом p_i, если функция T_v отправляет домен n_i на порт p_i. Если же пакеты отправляются в некоторую таблицу маршрутизации, групповое правило, сбрасываются или отправляются на контроллер, то $p_i^* = p_i$.

Также каждой вершине $v \in V$ поставим в соответствие *обратную сетевую передаточную функцию* $\Phi_v^{-1} : \mathcal{N} \to 2^{\mathcal{N}}$:

$$\Phi_v^{-1}(n) = \left\{ n' \mid n \in \Phi_v(n') \right\}. \tag{4.15}$$

Каждому ребру $vw \in A$ поставим в соответствие множество $D_{vw} \subseteq \mathcal{N}$, называемое *доменом ребра* vw:

$$D_{vw} = \Phi_v(D_v) \cap D_w, \tag{4.16}$$

где $\Phi_v(D_v) = \bigcup_{n \in D_v} \Phi_v(n)$. Для простоты удалим из графа G ребра с пустыми доменами: $D_{vw} = \varnothing$.

Протокол OpenFlow специфицирует наличие в каждой таблице правила *по-умолчанию* (*table miss*) [3], то есть правила с наименьшим приоритетом, которое обрабатывает пакеты, не обработанные другими правилами таблицы. Такие правила обычно либо сбрасывают пакеты, либо отправляют их на контроллер. Таким образом, для каждой таблицы τ существует вершина $v_\tau \in V$, домен передаточной функции которой равен всему сетевому пространству: $D_{v_\tau} = \mathcal{N}$. Из наличия этого правила следует, что для любой вершины $v \in V$ выполняется:

$$\Phi_v(D_v) = \bigcup_{vw \in A} D_{vw}. \tag{4.17}$$

Таким образом, для каждого домена $n \in \Phi_v(D_v)$ существует хотя бы одно ребро vw, такое что $n \in D_{vw}$.

4.2. Потоковая модель сети

Для построения модели ПКС сети, описывающей динамику изменения счетчиков правил маршрутизации, расширим граф зависимостей правил таким образом, что в нем будут описываться значения счетчиков правил маршрутизации. Модель описывает значения счетчиков правил маршрутизации в фиксированный момент времени, когда сеть уже обработала некоторое количество пакетов.

4.2.1. Описание модели

Определение 4.1. *n-доменным потоком* в графе G назовем пару функций $f_n : V \to \mathbb{N}$ и $\hat{f}_n : A \to \mathbb{N}$, таких что $\forall v \in V \setminus S$ и $\forall vw \in A$:

$$f_n(v) = \begin{cases} \sum\limits_{uv \in A} \hat{f}_n(u, v), & \text{если } n \in D_v, \\ 0, & \text{если } n \notin D_v, \end{cases} \tag{4.18}$$

$$\hat{f}_n(v, w) = \begin{cases} \sum\limits_{m \in \Phi_v^{-1}(n)} f_m(v), & \text{если } n \in D_{vw}, \\ 0, & \text{если } n \notin D_{vw}, \end{cases} \tag{4.19}$$

и $\forall s \in S$:

$$f_n(s) = f_n^s, \tag{4.20}$$

где $f_n^s \in \mathbb{N}$ начальный поток из истока s.

Величина $f_n(v)$, где $n = (h, p)$, описывает количество пакетов с заголовком h, пришедших на порт p и обработанных правилом v. Величина $\hat{f}_n(v, w)$, где $n = (h, p)$, обозначает количество пакетов, обработанных правилом v, получивших заголовок h, отправленных на порт p и обработанных правилом w — то есть прошедших по ребру vw.

Определение 4.2. *Доменным потоком* в графе G назовем пару функций $f : V \to \mathbb{N}$ и $\hat{f} : A \to \mathbb{N}$, таких что $\forall v \in V$ и $\forall vw \in A$:

$$f(v) = \sum_{n \in D_v} f_n(v), \tag{4.21}$$

$$\hat{f}(v, w) = \sum_{n \in D_{vw}} \hat{f}_n(v, u). \tag{4.22}$$

Значения $f(v)$ и $\hat{f}(v, w)$ описывают суммарное количество пакетов, обработанных правилом v и суммарное количество пакетов, прошедших по ребру vw, соответственно. Значение $f(v)$ описывает значение счетчика, установленного на правиле маршрутизации v.

Докажем принцип прохождения доменного потока через вершину v (теорема 4.1). Этот принцип описывает, как изменяется доменный поток при прохождении вершины в графе зависимостей правил.

Теорема 4.1.

$$f(v) = \sum_{uv \in A} \hat{f}(u, v) \qquad \forall v \in V \setminus S, \tag{4.23}$$

$$f(v) = \frac{1}{\gamma(v)} \sum_{vw \in A} \hat{f}(v, w) \qquad \forall v \in V \setminus T, \tag{4.24}$$

где $\gamma(v)$ — степень дублирования вершины v.

Доказательство. Для произвольной вершины $v \in V \setminus S$ и ребра $uv \in A$ из (4.21), (4.18), (4.19) и того, что $D_{uv} \subseteq D_v$ следует что:

$$\begin{aligned} f(v) &= \sum_{n \in D_v} f_n(v) = \sum_{n \in D_v} \sum_{uv \in A} \hat{f}_n(u, v) = \sum_{uv \in A} \sum_{n \in D_v} \hat{f}_n(u, v) \\ &= \sum_{uv \in A} \left(\sum_{n \in D_v \cap D_{uv}} \hat{f}_n(u, v) + \sum_{n \in D_v \cap \overline{D}_{uv}} \hat{f}_n(u, v) \right) \\ &= \sum_{uv \in A} \sum_{n \in D_{uv}} \hat{f}_n(u, v) = \sum_{uv \in A} \hat{f}(u, v), \end{aligned} \tag{4.25}$$

и (4.23) доказано. Далее для произвольной вершины $v \in V \setminus T$ и ребра $vw \in A$ из (4.19) и (4.22) следует что:

$$\begin{aligned} \sum_{vw \in A} \hat{f}(v,w) &= \sum_{vw \in A} \sum_{n \in D_{vw}} \hat{f}_n(v,w) \\ &= \sum_{vw \in A} \sum_{n \in D_{vw}} \sum_{m \in \Phi_v^{-1}(n)} f_m(v). \end{aligned} \tag{4.26}$$

Из (4.17) следует:

$$\begin{aligned} D_v = \Phi_v^{-1}\big(\Phi_v(D_v)\big) &= \Phi_v^{-1}\Big(\bigcup_{vw \in A} \bigcup_{n \in D_{vw}} n \Big) \\ = \bigcup_{vw \in A} \bigcup_{n \in D_{vw}} \Phi_v^{-1}(n) &= \bigcup_{vw \in A} \bigcup_{n \in D_{vw}} \bigcup_{m \in \Phi_v^{-1}(n)} m. \end{aligned} \tag{4.27}$$

Таким образом, получаем, что в (4.26) суммирование идет по всем m из домена D_v. Более того, так как суммирование идет по всем ребрам $vw \in A$, то из (4.14) и (4.17) следует, что каждое m участвует в (4.26) ровно $\gamma(v)$ раз.

Следовательно, получаем:

$$\sum_{vw \in A} \sum_{n \in D_{vw}} \sum_{m \in \Phi_v^{-1}(n)} f_m(v) = \gamma(v) \sum_{m \in D_v} f_m(v) = \gamma(v) f(v), \tag{4.28}$$

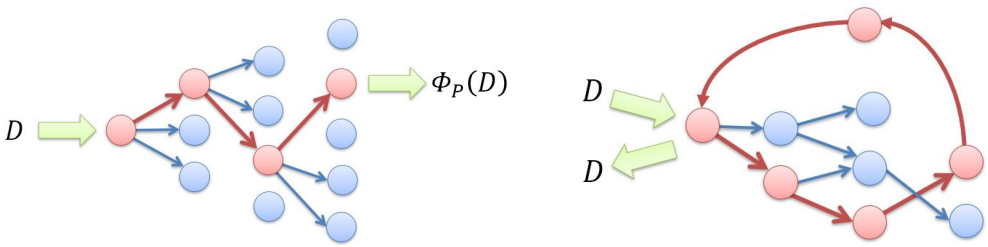
и теорема доказана. □

Доказанная теорема показывает, что счетчики правил маршрутизации могут быть описаны в виде *потока с мультипликаторами* [48]. Поток с умножителями — это потоковая функция на графе G, такая что значение потока, исходящего из вершины $v \in V \setminus S$, равно произведению значения входящего потока на величину $\gamma(v)$.

Пусть $P:\ v_0, v_1, \ldots, v_l$ — ориентированный путь в графе G. Обозначим домен, в который преобразуется домен n после прохождения пути P, как:

$$\Phi_P(n) = \Phi_{v_l}\Big(\ldots \big(\Phi_{v_0}(n)\big) \ldots \Big). \tag{4.29}$$

Если путь P пустой, то есть не содержит вершин, то $\Phi_P(n) = \{n\}$.



(a) (b)

Рисунок 4.3: Доменный путь (a) и доменный цикл (b)

Определение 4.3. *D-доменным путем* назовем ориентированный путь P : $v_0, v_1, \ldots, v_l$ в графе G, такой что $\forall n \in D$ выполняется:

- $n \in D_{v_0}$;
- $\forall i \in [1, l] \Rightarrow \Phi_{v_0,\ldots,v_{i-1}}(n) \cap D_{v_i} \neq \varnothing$.

Определение 4.4. *D-доменным циклом* назовем ориентированный цикл C : $c_0, c_1, \ldots, c_l$ в графе G, такой что:

- C — D-доменный путь;
- $\forall n \in D \Rightarrow \Phi_C(n) \cap D \neq \varnothing$.

Далее, предположим, что в графе G нет доменных циклов. Наличие доменных циклов говорит о наличии в сети *бесконечных циклов маршрутизации* [62]. То есть существует пакет, который будет следовать по одному и тому же пути в течение неопределенного промежутка времени. Бесконечные циклы маршрутизации считаются ошибками настройки сети, так как они могут привести к более серьезным проблемам, таким как *широковещательные штормы* [71]. Сетевые администраторы стремятся избавиться от подобных ошибок, поэтому существует множество методов обнаружения бесконечных циклов маршрутизации и удаления их из сети [57, 68, 72].

Необходимо отметить, что мы не предполагаем отсутствия в сети *конечных циклов маршрутизации*, в которых пакеты в конечном итоге выйдут из цикла, например, циклов на уровне L2.

Утверждение 4.2. $\forall v \in V$ и $\forall n \in D_v$ существует n-доменный путь $P_n : v = v_0, \ldots, v_l = t$, где $t \in T$.

Доказательство. Покажем существование такого пути. Зафиксируем произвольную вершину v и произвольный элемент $n \in D_v$. Опишем процедуру построения пути $P_n : v = v_0, \ldots, v_l$.

Путь P_n^0, состоящий из одной вершины v_0, является доменным путем, так как $\Phi_{v_0}(n) = \{n\}$ и $\{n\} \subseteq D_{v_0}$. Если $v_0 \in T$, то теорема доказана. Далее, предположим, что $v_0 \notin T$.

Рассмотрим произвольный шаг с номером $i \geqslant 1$. Пусть построен путь $P_n^i : v = v_0, \ldots, v_i$ такой, что $\Phi_{v_0,\ldots,v_{i-1}}(n) \subseteq D_{v_i}$. Если $v_i \in T$, то теорема доказана, если нет, то предположим, что $v_i \notin T$. Тогда из (4.17) следует, что существует ребро $v_i v_{i+1}$ такое, что $\Phi_{v_0,\ldots,v_i}(n) \cap D_{v_i v_{i+1}} \neq \varnothing$. И так как $D_{v_i v_{i+1}} \subseteq D_{v_{i+1}}$, то $\Phi_{v_0,\ldots,v_i}(n) \cap D_{v_{i+1}} \neq \varnothing$ и P_n^i — n-доменный путь.

Пусть существует номер j такой, что $v_j = v_i$ для некоторого $i < j$, то есть, вершина уже участвовала в пути P_n. Если $\Phi_{v_0,\ldots,v_i}(n) = \Phi_{v_0,\ldots,v_j}(n)$, то в графе G существует доменный цикл, что противоречит предположению об отсутствии доменных циклов. Следовательно:

$$\Phi_{v_0,\ldots,v_i}(n) \neq \Phi_{v_0,\ldots,v_j}(n) \quad \forall i, j > 0. \tag{4.30}$$

Покажем, что $\forall j : v_j \notin T$ будет существовать шаг с номером $k > j$ такой, что вершина v_k ранее не встречалась в пути P. Пусть для некоторого j такого номера не существует. Так как на каждом шаге процедуры строится путь, проходящий через вершину, не являющуюся стоком, то по (4.17) получаем,

что процедура будет иметь бесконечное число шагов. Тогда существует хотя бы одна вершина v_k, которая участвует в бесконечном числе шагов описанной процедуры. Но из конечности D_{v_k} получаем, что существуют номера $k', k'' > j$ такие, что $v_{k'} = v_{k''} = v_k$ и $\Phi_{v_0,\ldots,v_{k'}}(n) = \Phi_{v_0,\ldots,v_{k''}}(n)$. Таким образом получено противоречие с (4.30).

Следовательно, всегда существует шаг описанной процедуры такой, что вершина рассматриваемого шага ранее не участвовала в пути P. И так как множество V конечно, то из-за (4.17) описанная процедура может закончиться только в некоторой стоковой вершине $v_l \in T$. Следовательно $P_n : v = v_0, \ldots, v_l = t$ — n-доменный путь из v в t, и утверждение доказано. □

Определение 4.5. Пусть $v \in V \backslash S$. Домен $n \in D_v$ называется *достижимым*, если:

$$\exists uv \in A : n \in D_{uv}. \tag{4.31}$$

Множество достижимых доменов для вершины v обозначим через $R_v \subseteq D_v$.

Определение 4.6. *Потоком D-доменного пути* $P_D \in \mathscr{P}$ из вершины v в вершину w при $D \subseteq D_v$ назовем функцию $F : \mathscr{P} \to \mathbb{N}$, такую что:

$$F(P_D) = \sum_{n \in D} f_n(v), \tag{4.32}$$

где $\mathscr{P}$ — множество всех доменных путей в графе G.

Поток D-доменного пути описывает количество пакетов с заголовками h и входными портами p, такими что $(h, p) \in D$, которые вышли из вершины v и двигались вдоль этого пути.

Определение 4.7. *Подпространством доменных путей в вершину v* назовем

$$\mathcal{P}(v) = \{P_n : u \to v \mid n \in \mathcal{N}, u \in S\}. \tag{4.33}$$

Введем следующее обозначение:

$$F\big(\mathcal{P}(v)\big) = \sum_{P_n \in \mathcal{P}(v)} F(P_n). \tag{4.34}$$

Теорема 4.3. Для любой вершины $v \in V$ графа верно:

$$f(v) = F\big(\mathcal{P}(v)\big). \tag{4.35}$$

Доказательство. Построим последовательность множеств $\mathcal{P}^i(v)$ различных доменных путей длины i в вершину v, при $i \in [0, l]$, где l — длина максимального n-доменного пути для некоторого $n \in \mathcal{N}$. Длина путей ограничена в силу предположения об отсутствии в графе доменных циклов.

$$\mathcal{P}^i(v) = \mathcal{P}^i_R(v) \cup \mathcal{P}^i_S(v) \cup \mathcal{P}^i_U(v), \tag{4.36}$$

где:

$$\mathcal{P}^i_R(v) = \{P_n : u \to v \mid n \in D_u \cap R_v,\ u \in V \setminus S,\ |P_n| = i\}, \tag{4.37}$$

$$\mathcal{P}^i_S(v) = \{P_n : u \to v \mid n \in D_u, \qquad u \in S, \qquad |P_n| \leqslant i\}, \tag{4.38}$$

$$\mathcal{P}^i_U(v) = \{P_n : u \to v \mid n \in D_u \cap \overline{R}_v,\ u \in V \setminus S,\ |P_n| \leqslant i\}. \tag{4.39}$$

$\mathcal{P}^i_R(v)$ и $\mathcal{P}^i_U(v)$ не пересекаются с $\mathcal{P}^i_S(v)$, так как пути P_n начинаются из непересекающихся множеств вершин S и $V \setminus S$. $\mathcal{P}^i_R(v)$ не пересекается с $\mathcal{P}^i_U(v)$, так как пути P_n начинаются из непересекающихся множеств доменов $D_u \cap R_v$ и $D_u \cap \overline{R}_v$.

Покажем, что $\forall i \in [0, l]$ выполняется:

$$f(v) = F\big(\mathcal{P}^i(v)\big). \tag{4.40}$$

База индукции при $i = 0$:

Рассмотрим множество $\mathcal{P}^0(v)$. Если $v \in S$, то получаем, что $\mathcal{P}^0(v) = \mathcal{P}^0_S(v)$. Если $v \in V \setminus S$, то $\mathcal{P}^0(v) = \mathcal{P}^0_R(v) \cup \mathcal{P}^0_U(v)$. $\mathcal{P}^0(v)$ по определению описывает множество всех $P_n : v \to v$ путей длины 0, при $n \in D_v$.

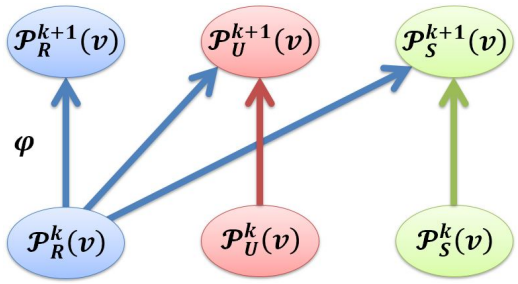


Рисунок 4.4: Инъективное отображение

По определению потока доменного пути получаем, что $F(P_n) = f_n(v)$ для любого $P_n \in \mathcal{P}^0$. Далее из определения потока через вершину и определения потока доменного пути получаем:

$$f(v) = \sum_{n \in D_v} f_n(v) = \sum_{P_n \in \mathcal{P}^0(v)} F(P_n) = F\big(\mathcal{P}^0(v)\big). \tag{4.41}$$

Шаг индукции:

Пусть утверждение (4.40) доказано для некоторого $k < l$, где l — длина максимального n-доменного пути для некоторого $n \in \mathcal{N}$. Поэтому докажем его для $k+1$. Построим инъективное отображение φ (рис. 4.4) из множества $\mathcal{P}^k(v)$ в множество $\mathcal{P}^{k+1}(v)$ такое, что:

$$F(P) = \sum_{P' \in \varphi(P)} F(P'). \tag{4.42}$$

Инъективным отображением называется такое отображение $\varphi : \mathcal{P}^k(v) \to \mathcal{P}^{k+1}(v)$, что $\varphi(P) = \varphi(P') \Rightarrow P = P'$ для любых $P \in \mathcal{P}^k(v)$ и $P' \in \mathcal{P}^{k+1}(v)$. Таким образом, если такое отображение существует, то будет доказано, что $F\big(\mathcal{P}^k(v)\big) = F\big(\mathcal{P}^{k+1}(v)\big)$.

Покажем, что каждому доменному пути $P \in \mathcal{P}_S^k(v)$ однозначно соответствует некоторый доменный путь $P' \in \mathcal{P}_S^{k+1}(v)$ такой, что $F(P) = F(P')$. Утверждение очевидно, так как в данном случае $P = P'$, то есть один и тот же путь длины k принадлежит обоим множествам. Таким же образом доказыва-

ется, что каждому доменному пути $P \in \mathcal{P}_U^k(v)$ однозначно соответствует некоторый доменный путь из $P' \in \mathcal{P}_U^{k+1}(v)$. Очевидно, что $\varphi\big(\mathcal{P}_S^k(v)\big) \neq \mathcal{P}_S^{k+1}(v)$ так как $\mathcal{P}_S^{k+1}(v)$ содержит еще и пути длины $k+1$.

Введем следующие обозначения:

$$\mathcal{P}'_S = \mathcal{P}_S^{k+1}(v) \setminus \varphi\big(\mathcal{P}_S^k(v)\big), \tag{4.43}$$

$$\mathcal{P}'_U = \mathcal{P}_U^{k+1}(v) \setminus \varphi\big(\mathcal{P}_U^k(v)\big), \tag{4.44}$$

$$\mathcal{P}'_R = \mathcal{P}_R^{k+1}(v) \cup \mathcal{P}'_S \cup \mathcal{P}'_U. \tag{4.45}$$

Теперь построим инъекцию $\varphi : \mathcal{P}_R^k(v) \to \mathcal{P}'_R$. Возьмем произвольный доменный путь $P_n \in \mathcal{P}_R^k(v)$ и рассмотрим начальную вершину u этого пути. Так как домен $n \in D_u$ достижим, то существует ребро wu такое, что $n \in D_{wu}$. Таким образом:

$$F(P_n) = f_n(u) = \sum_{wu \in A} \hat{f}_n(w,u) = \sum_{wu \in A} \sum_{m \in \Phi_w^{-1}(n)} f_m(w). \tag{4.46}$$

Рассмотрим произвольный домен $m \in \Phi_w^{-1}(n)$. Путь wP_n является m-доменным путем из w в v, так как $m_w = \Phi_u(m) = n$, и P_n — n-доменный путь из u в v. Так как wP_n — m-доменный путь из w в v, то $F(wP_n) = f_m(w)$. Таким образом, получаем:

$$F(P_n) = \sum_{wu \in A} \sum_{m \in \Phi_w^{-1}(n)} f_m(w) = \sum_{wu \in A} \sum_{m \in \Phi_w^{-1}(n)} F(wP_n) = \sum_{P_m \in \mathcal{P}'_R(v;P_n)} F(P_m), \tag{4.47}$$

где $\mathcal{P}'_R(v;P_n) = \{wP_n \mid wu \in A,\ m \in \Phi_w^{-1}(n)\}$ при $P_n : u \to v$ из множества $\mathcal{P}_R^k(v)$.

Поставим каждому доменному пути $P_n \in \mathcal{P}_R^k(v)$ набор путей $\varphi(P_n) = \mathcal{P}'_R(v;P_n)$, построенных по описанной выше процедуре. Получаем, что:

$$F(P_n) = \sum_{P_m \in \varphi(P_n)} F(P_m). \tag{4.48}$$

Покажем, что φ — инъективное отображение из $\mathcal{P}^k_R(v)$ в $\mathcal{P}'_R$. По построению следует, что $\forall P_n \in \mathcal{P}^k_R(v)$ существует некоторый $P_m = \varphi(P_n)$. Покажем, что $\forall P_m$ существует единственный доменный путь $P_n = \varphi^{-1}(P_m)$, то есть для различных доменных путей P_n и P'_n множества $\mathcal{P}'_R(v; P_n)$ и $\mathcal{P}'_R(v; P'_n)$ не пересекаются. Пусть существует путь P_m, принадлежащий обоим множествам. Но тогда $P_m = wP_n = wP'_n$, что противоречит различности P_n и P'_n.

Теперь предположим, что $\exists P_m \in \mathcal{P}'_R$ такое, что $\nexists P_n \in \mathcal{P}^k_R(v) : \ P_m = \varphi(P_n)$. Так как все пути $P_m : u_1 \dots u_k v \in \mathcal{P}'_R$ имеют длину $k+1$, то $P_n : u_2 \dots u_k v$ имеет длину k. Но тогда данный путь должен был быть выбран в описанной выше процедуре, которая бы построила непустое множество $\varphi(P_n)$.

Таким образом, доказано, что $\varphi : \mathcal{P}^k(v) \to \mathcal{P}^{k+1}(v)$ — инъекция, и $F\big(\mathcal{P}^k(v)\big) = F\big(\mathcal{P}^{k+1}(v)\big)$, и индукция завершена.

Далее, из определения следует, что $\mathcal{P}(v) = \mathcal{P}^l_S(v)$, так как l — длина максимального доменного пути. Покажем, что $\mathcal{P}^l_R(v) = \varnothing$. Предположим противное. Тогда существует путь $P_n \in \mathcal{P}^l_R(v)$ длины l такой, что n — достижимый домен. Следовательно, существует вершина w такая, что $wP_n = P_m$ — m-доменный путь длины $l+1$ для некоторого домена m, что противоречит максимальности l. Также покажем, что $F\big(\mathcal{P}^l_U(v)\big) = 0$. Это свойство следует из (4.18) и определения достижимого домена:

$$F\big(\mathcal{P}^l_U(v)\big) = \sum_{P_n \in \mathcal{P}^l_U(v)} F(P_n) = \sum_{u \in V \setminus S} \sum_{n \in D_u \cap \overline{R}_v} f_n(u) = 0. \tag{4.49}$$

Таким образом, получаем, что:

$$f(v) = F\big(\mathcal{P}^l(v)\big) = F\big(\mathcal{P}^l_S(v)\big) = F\big(\mathcal{P}(v)\big), \tag{4.50}$$

и теорема доказана. □

Рассмотрим множество $\mathcal{P}$. Пусть P — произвольный путь из S в v в графе G. Рассмотрим все доменные пути $P_n \in \mathcal{P}(v)$ такие, что P_n и P равны

как пути в графе, то есть $E(P_n) = E(P)$. Построим новый доменный путь из пути P, приписыванием к нему домена D, равного объединению всех доменов путей P_n таких, что $E(P_n) = E(P)$. Построим следующее множество:

$$\mathbb{P}(v) = \left\{ P_D \ : \ D = \bigcup_{\substack{P_n \in \mathcal{P}(v) \\ E(P_n)=E(P)}} D(P_n) \right\}. \tag{4.51}$$

Также определим поток по доменным путям $P_D \in \mathbb{P}(v)$ как:

$$F(P_D) = \sum_{n \in D} F(P_n). \tag{4.52}$$

Из построения следует, что все пути из $\mathbb{P}(v)$ различны. Из доказанной выше теоремы следует, что:

Следствие 4.4. Для любой вершины $v \in V$ выполняется следующее:

$$f(v) = F\big(\mathbb{P}(v)\big). \tag{4.53}$$

Обозначим через $\mathbb{P}$ объединение пространств путей для всех вершин графа G, то есть:

$$\mathbb{P} = \bigcup_{v \in V} \mathbb{P}(v). \tag{4.54}$$

Введем на множестве $\mathbb{P}$ отношение частичного порядка «$\preceq$».

Определение 4.8. Пусть $P, P' \in \mathbb{P}$, тогда $P \preceq P'$ если:

1. P и P' имеют общую начальную вершину;
2. $E(P) \subseteq E(P')$.

Таким образом, $\mathbb{P}$ есть решетка путей в графе G. Обозначим через $\mathbb{P}|_P$ подрешетку, образованную из всех путей $P' \in \mathbb{P}$ таких, что $P \preceq P'$. Для путей $P \preceq P'$ определена операция «$-$» такая, что путь $P' - P$ получен из $E(P') \setminus E(P)$ удалением вершин с нулевой степенью.

Определение 4.9. Мультипликатором пути $P = v_0, \ldots, v_k$ назовем следующую величину:

$$\hat{\gamma}(P) = \prod_{i \in [0,k-1]} \gamma(v_i), \tag{4.55}$$

где $\gamma(v)$ - степень дублирования вершины v. Если $E(P) = \varnothing$, то $\hat{\gamma}(P) = 1$.

Теорема 4.5.

$$F(P) = \sum_{P' \in AC(P)} \frac{F(P')}{\hat{\gamma}(P' \setminus P)}, \tag{4.56}$$

где $AC(P)$ есть произвольная максимальная по включению антицепь из $\mathbb{P}|_P$.

Доказательство. Зафиксируем некоторую антицепь $AC(P)$ и проведем доказательство по индукции. Построим последовательность множеств $\mathbb{P}^i = \mathbb{P}^i_{AC} \cup \mathbb{P}^i_{=}$ при $i \in [0, l]$, где l — длина максимального по длине доменного пути.

$$\mathbb{P}^i_{AC} = \{P' : |P'| \leqslant i, P' \in AC(P)\}, \tag{4.57}$$

$$\mathbb{P}^i_{=} = \{P' : |P'| = i, P' \in \mathbb{P}|_P, P' \prec AC(P)\}, \tag{4.58}$$

где под $P' \prec AC(P)$ понимается, что существует элемент из антицепи $AC(P)$, который больше, чем P'. Докажем, что:

$$F(P) = \sum_{P' \in \mathbb{P}^l} \frac{F(P')}{\hat{\gamma}(P' - P)}. \tag{4.59}$$

База индукции при $i = 0$:

База индукции выполняется, так как P — единственный элемент из $\mathbb{P}^0$ и $\hat{\gamma}(P - P) = 1$, то есть:

$$F(P) = \frac{F(P)}{\hat{\gamma}(P - P)}. \tag{4.60}$$

Шаг индукции:

Пусть (4.59) выполнено для некоторого $i > 0$. Для перехода к $i + 1$ поочередно рассмотрим все пути $P' \in \mathbb{P}^i_{=}$ и их концевые вершины v. Если $\mathbb{P}^i_{=}$

непусто, то $v \notin T$, так как $\exists P'' \in AC(P) : \ P'' \succ P'$, в то время как пути, оканчивающиеся в T, максимальные.

По определению путь P' из $\mathbb{P}$ с концевой вершиной v принадлежит множеству $\mathbb{P}(v)$ и раскладывается на доменные пути P'_n такие, что $n \in D(P')$, где $D(P')$ — домен пути P'.

Для любого пути P'_n существует $\gamma(v)$ ребер vw таких, что:

$$\Phi_v\big(D_v(P'_n)\big) \cap D_{vw} \neq \varnothing, \tag{4.61}$$

где $D_v(P'_n) = n_{P'}$. Следовательно, существует $\gamma(v)$ доменных путей $P'_n w$ длины $i+1$ таких, что $F(P'_n) = F(P'_n w)$. Это равенство выполняется, потому что потоки доменных путей равны $f_n(s)$, где s — общее начало обоих путей.

Следовательно, слагаемое $\frac{F(P')}{\hat{\gamma}(P'-P)}$ из (4.59) для i разлагается в сумму:

$$\frac{F(P')}{\hat{\gamma}(P'-P)} = \frac{1}{\hat{\gamma}(P'-P)} \sum_{n \in D(P')} \sum_{vw \in A} I_n(vw) \frac{1}{\gamma(v)} F(P'_n w), \tag{4.62}$$

$$\text{где } I_n(vw) = \begin{cases} 1, & \text{if } \Phi_v\big(D_v(P'_n)\big) \cap D_{vw} \neq \varnothing, \\ 0, & \text{иначе.} \end{cases} \tag{4.63}$$

Изменим порядок слагаемых в (4.62):

$$\sum_{vw \in A} \frac{1}{\hat{\gamma}(P'-P)} \frac{1}{\gamma(v)} \sum_{n \in D(P')} I_n(vw) F(P'_n w). \tag{4.64}$$

Рассмотрим путь $P'' = P'w$ с доменом $D(P'')$:

$$D(P'') = \big\{ n \in D(P') \ : \ I_n(vw) = 1 \big\}. \tag{4.65}$$

Путь $P'' \in \mathbb{P}(w)$, так как P'' является объединением путей $P'_n w$ и не существует такого пути $P''_m \in \mathcal{P}(w)$, что $E(P'') = E(P''_m)$ и $m \notin D(P'')$.

Также из определения мультипликатора пути следует, что:

$$\frac{1}{\hat{\gamma}(P'-P)} \frac{1}{\gamma(v)} = \frac{1}{\hat{\gamma}(P''-P)}. \tag{4.66}$$

Остается показать, что каждый P'' принадлежит $\mathbb{P}^{i+1}$. Это утверждение очевидно выполняется, если $P'' \in AC(P)$, так как длина пути P'' есть $i+1$.

Пусть теперь $P'' \notin AC(P)$. Предположим, что $P'' \notin \mathbb{P}^{i+1}_{=}$. Так как $P'' \in \mathbb{P}|_P$, то не существует $\hat{P} \in AC(P)$ такого, что $\hat{P} \succ P''$. По построению множества $\mathbb{P}^{i+1}$ видно, что мы не могли построить путь такой, что $P'' \succ AC(P)$, так как процедура увеличения происходила только для путей из $\mathbb{P}^{i}_{=}$. Таким образом, мы получили путь P'', не сравнимый ни с каким элементом антицепи $AC(P)$, что противоречит ее максимальности по включению.

Таким образом, каждое слагаемое $\frac{F(P')}{\hat{\gamma}(P'-P)}$ из (4.59) для i раскладывается в сумму $\sum \frac{F(P'')}{\hat{\gamma}(P''-P)}$, где каждый $P'' \in \mathbb{P}^{i+1}$.

Для доказательства теоремы остается показать, что $\mathbb{P}^{l} = AC(P)$. Для этого нужно показать, что $\mathbb{P}^{l}_{=} = \varnothing$. Пусть существует $\overline{P} \in \mathbb{P}^{l}_{=}$. Так как из утверждения 4.2 следует, что для каждой вершины v и каждого $n \in D_v$ существует доменный путь в T, то любой путь $\overline{P}$ максимальной длины должен заканчиваться в T. Что приводит к противоречию, так как $\nexists \overline{P}' \succ \overline{P}$, потому что пути, заканчивающиеся в T — максимальные элементы решетки $\mathbb{P}$. $\square$

Определение 4.10. Антицепью называется такое подмножество U частично-упорядоченного множества, что любая пара элементов $u, v \in U$ несравнима.

Далее докажем следующее утверждение:

Утверждение 4.6. $\mathbb{P}(T)$ есть максимальная по включению антицепь в $\mathbb{P}$.

Доказательство. Так как пути, заканчивающиеся в T — максимальные элементы решетки $\mathbb{P}$, то очевидно, что элементы из $\mathbb{P}(T)$ несравнимы, и $\mathbb{P}(T)$ — антицепь. Далее, предположим, что антицепь $\mathbb{P}(T)$ не максимальна. Тогда существует путь $P \in \mathbb{P} \setminus \mathbb{P}(T)$ несравнимый с $\mathbb{P}(T)$. Что невозможно, потому что из утверждения 4.2 следует, что для любой вершины и любого

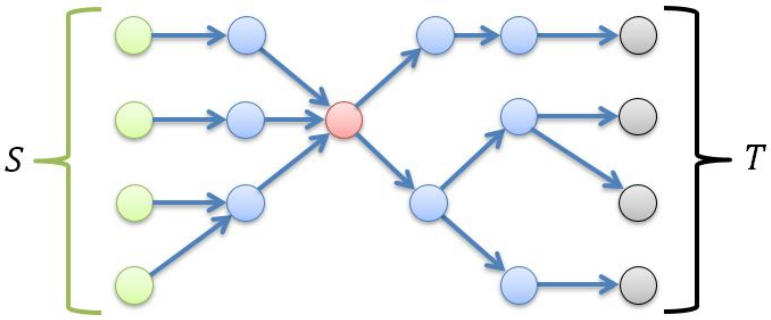


Рисунок 4.5: Предсказание потока через вершину

домена существует путь $\overline{P}$ из данной вершины в T, для которого очевидно, что $\overline{P} \succ P$. □

Необходимо отметить, что любое подмножество антицепи также является антицепью.

Из доказанных ранее утверждений следует основная теорема:

Теорема 4.7.

$$f(v) = \sum_{P \in \mathbb{P}(v)} \sum_{P' \in \mathbb{P}(T) \cap \mathbb{P}|_P} \frac{F(P')}{\hat{\gamma}(P' - P)}. \tag{4.67}$$

Из теоремы 4.7 следует, что поток через каждую вершину можно однозначно предсказать (рис. 4.5), используя значения потоков по путям в $P_D \in \mathbb{P}(T)$, которые в свою очередь равны значениям потоков $f_D(s)$.

Используем данный факт для построения алгоритма, который предсказывает значения потоков в вершинах по значениям потока в S. Важно отметить, что в сумме (4.67) один и тот же путь может встречаться несколько раз из-за возможности наличия в сети конечных циклов.

Докажем следующее утверждение:

Утверждение 4.8. Пусть $P \in \mathbb{P}(T)$, тогда, если $\hat{\gamma}(P) = 1$, то:

$$\forall P' \in \mathbb{P}(T) : E(P') \neq E(P) \Rightarrow D(P) \cap D(P') = \varnothing. \tag{4.68}$$

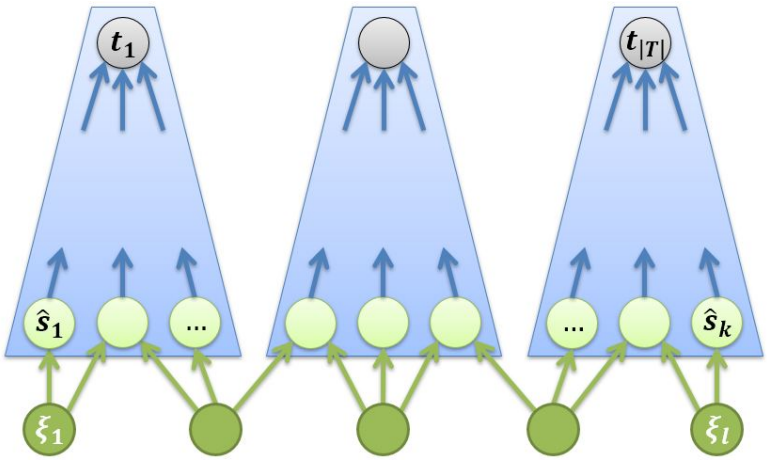


Рисунок 4.6: Путевая развертка

Доказательство. Предположим, что утверждение не выполняется. Тогда существует $n \in D(P) \cap D(P')$ при $E(P) \neq E(P')$. Так как $\hat{\gamma}(P) = 1$, то на каждой очередной вершине v пути P, для любого $m \in D_v$ существует единственное ребро vw такое, что $\Phi_v(m) \in D_{vw}$. Таким образом, если s — начальная вершина пути P, то для любого $n \in D_s$ однозначно определен доменный путь P_n такой, что $E(P) = E(P_n) = E(P')$, следовательно $n \in D(P)$. Получено противоречие с тем, что $E(P) \neq E(P')$, и утверждение доказано. □

4.2.2. Алгоритм предсказания потока

Для построения алгоритма предсказания значений $f(v)$ определим граф $\mathcal{G}$, который назовем *путевой разверткой графа* G (рис. 4.6). Опишем алгоритм построения путевой развертки, представленный в алгоритме 4.1.

Каждой вершине $v \in V$ соответствует множество вершин $\rho(v) \in \widetilde{V}$ в графе $\mathcal{G}$. Каждой вершине $t \in T$ соответствует единственная вершина $\tilde{t} = \rho(t)$ с доменом $D_{\tilde{t}} = D_t$. Множество таких вершин $\tilde{t}$, что $\rho^{-1}(\tilde{t}) \in T$, будем обозначать как $\widetilde{T}$.

Зафиксируем некоторую вершину $\tilde{t} \in \widetilde{T}$ и опишем алгоритм построения дерева с корнем в $\tilde{t}$. Занесем вершину $\tilde{t}$ в очередь Q. Далее будем выбирать

вершины из этой очереди до того, как она станет пустой.

Пусть на некотором шаге алгоритма из очереди была выбрана вершина $\tilde{v}$. Для каждого ребра $uv \in A$, где $v = \rho^{-1}(\tilde{v})$, добавим в дерево вершину $\tilde{u}$ и ориентированное ребро $\tilde{u}\tilde{v}$. Домены ребра $\tilde{u}\tilde{v}$ и вершины $\tilde{u}$ определим следующим образом:

$$D_{\tilde{u}\tilde{v}} = D_{uv} \cap D_{\tilde{v}}, \tag{4.69}$$

$$D_{\tilde{u}} \ = \Phi_u^{-1}(D_{\tilde{u}\tilde{v}}). \tag{4.70}$$

Определим для вершины дерева $\tilde{u}$ значение $\hat{\gamma}(\tilde{u}) = \gamma(u)\hat{\gamma}(\tilde{v})$, где $\gamma(u)$ — степень дублирования вершины u. Значение $\hat{\gamma}(\tilde{u})$ назовем *мультипликатором* вершины $\tilde{u}$. Для вершин $\tilde{t} \in \widetilde{T}$ будем считать, что $\tilde{\gamma}(\tilde{t}) = 1$.

Далее занесем вершину $\tilde{u}$ в очередь Q и продолжим выполнение алгоритма. Алгоритм построения дерева закончится тогда, когда на некотором шаге из очереди будет удалена последняя вершина и не будет добавлена новая.

Удалим такие листовые вершины $\tilde{s}$ полученного дерева, что $\rho(\tilde{s}) \notin S$ и перейдем к построению дерева с корнем $\tilde{t}' \in T \setminus \tilde{t}$. Таким образом, мы построим дерево, соответствующее каждой вершине t из множества T. Множество всех вершин $\tilde{s} \in \rho^{-1}(s)$, где $s \in S$, обозначим как $\widetilde{S}$.

Выберем из построенных деревьев вершины $\tilde{s} \in \rho^{-1}(s)$, где $s \in S$, и построим разбиение $\mathscr{D}_s$ доменов D_s такое, что:

$$D_s = \bigsqcup_{D \in \mathscr{D}_s} D, \tag{4.71}$$

и для любого $\tilde{s} \in \rho^{-1}(s)$ существует $\widetilde{\mathscr{D}}_s \subseteq \mathscr{D}_s$ такое, что:

$$D_{\tilde{s}} = \bigsqcup_{\widetilde{D} \in \widetilde{\mathscr{D}}_s} \widetilde{D}. \tag{4.72}$$

Это разбиение можно построить, производя всевозможные пересечения доменов вершин $\tilde{s} \in \rho^{-1}(s)$.

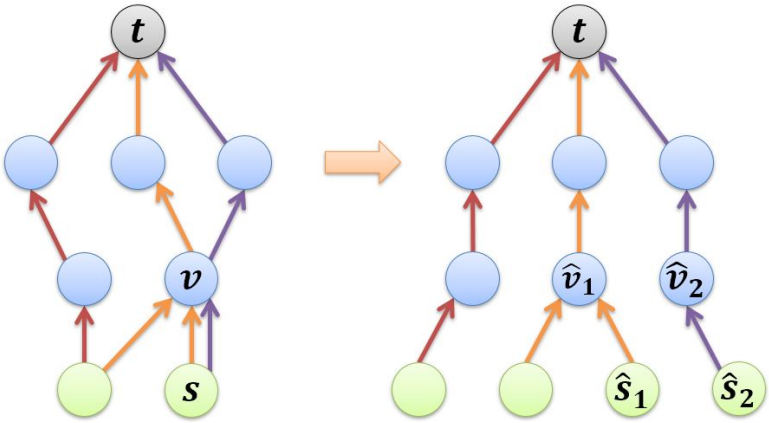


Рисунок 4.7: Построение путевой развертки

Для каждого домена $D \in \mathscr{D}_s$ добавим в $\widetilde{V}$ вершину ξ с доменом $D_\xi = D$. Из вершины ξ проведем ребро в вершину $\tilde{s}$ такую, что $\rho(\tilde{s}) \in S$, и поставим ему в соответствие домен, равный D_ξ. Множество всех таких ξ обозначим за $\mathscr{S}$ и назовем *множеством дополнительных вершин*. На этом построение путевой развертки графа G завершено.

Алгоритм 4.1 также использует следующие дополнительные функции:

- $create\_node(v, D, \gamma)$;
- $prune\_branch(\tilde{v})$;
- $create\_domain\_partition\big(\widetilde{S}\big)$.

Функция $create\_node(v, D, \gamma)$ создает новую вершину $\tilde{v}$ путевой развертки, соответствующую вершине v графа зависимостей правил. В качестве домена вершины $\tilde{v}$ выбирается D, а в качестве мультипликатора — γ, то есть:

$$D_{\tilde{v}} = D, \tag{4.73}$$

$$\hat{\gamma}(\tilde{v}) = \gamma. \tag{4.74}$$

Функция $prune\_branch(\tilde{v})$ удаляет ветвь дерева путевой развертки, оканчивающуюся в вершине $\tilde{v}$. Функция $create\_domain\_partition(\widetilde{S})$ создает набор

новых вершин $\mathscr{S}$, доменами которых являются все попарные пересечения доменов вершин из $\widetilde{S}$. То есть, создаются вершины с множеством доменов $\mathscr{D}_s$.

Теперь опишем алгоритм предсказания значений $f(v)$, использующий путевую развертку. Формальное описание представлено в алгоритме 4.2. В этом алгоритме используются следующие обозначения: $\delta^{out}(\tilde{v})$ обозначает исходящие ребра из вершины $\tilde{v}$, а $\delta^{in}(\tilde{v})$ — ребра, входящие в вершину $\tilde{v}$.

Предположим, что для любой вершины $\xi \in \mathscr{S}$ известны значения потоков $f(\xi)$. Утверждается, что используя эту информацию можно предсказать значения потока $f(v)$ в каждой вершине $v \in V$.

Добавим в очередь Q все вершины $\xi \in \mathscr{S}$ и припишем им значения потока $f(\xi)$. Далее будем выбирать вершину $\tilde{v}$ из очереди Q и добавлять значение ее потока $f(\tilde{v})$, деленное на $\hat{\gamma}(\tilde{w})$, в вершину $\tilde{w}$ такую, что она является родительской вершиной для $\tilde{v}$ в путевой развертке. Ребро $\tilde{v}\tilde{w}$ отметим как просмотренное.

Если на очередном шаге все исходящие ребра некоторой вершины $\tilde{w}$ просмотрены, то добавляем вершину $\tilde{w}$ в очередь. Алгоритм заканчивает свою работу, когда очередь Q пуста.

Утверждается, что сумма значений $f(\tilde{v})$ по всем $\tilde{v} \in \rho^{-1}(v)$ есть поток $f(v)$. Этот факт был доказан в теореме 4.11.

Введем следующее обозначение:

$$\widetilde{\mathbb{P}}(v) = \left\{ P_{\widetilde{D}} : s \to v \mid \widetilde{D} = D_{\tilde{s}} : \ \tilde{s} \in \widetilde{S};\ \exists \tilde{P} : \tilde{s} \to \tilde{v},\ \tilde{v} \in \rho^{-1}(v) \right\}. \tag{4.75}$$

Докажем следующее утверждение:

Утверждение 4.9. Для любого $t \in T$ следует, что:

$$\widetilde{\mathbb{P}}(t) = \mathbb{P}(t). \tag{4.76}$$

Algorithm 4.1 Построение путевой развертки

Input: Граф зависимостей правил G

Output: Путевая развертка $\mathcal{G}(\widetilde{V}, \widetilde{A}, \mathscr{S})$

1: **procedure** CREATE_PATH_SCAN(G)
2: **for** $t \in T$ **do**
3: $\tilde{t} \leftarrow$ **create_node**$(t, \mathcal{N}, 1)$
4: $\widetilde{V} \leftarrow \widetilde{V} \cup \{\tilde{t}\}$
5: $Q \leftarrow \tilde{t}$
6: **while** $Q \neq \varnothing$ **do**
7: $\tilde{v} \leftarrow Q$
8: $v \leftarrow \rho(\tilde{v})$
9: **for** $u \in \delta^{in}(v)$ **and** $D_{uv} \cap D_{\tilde{v}} \neq \varnothing$ **do**
10: **if** $\delta^{in}(u) = \varnothing$ **and** $u \notin S$ **then**
11: **prune_branch**$(\tilde{v})$
12: **else**
13: $D \leftarrow \Phi_u^{-1}(D_{uv} \cap D_{\tilde{v}})$
14: $\gamma \leftarrow \gamma(u)\hat{\gamma}(\tilde{v})$
15: $\tilde{u} \leftarrow$ **create_node**(u, D, γ)
16: $\widetilde{V} \leftarrow \widetilde{V} \cup \{\tilde{u}\}$
17: $\widetilde{A} \leftarrow \widetilde{A} \cup \{\tilde{u}\tilde{v}\}$
18: **if** $u \in S$ **then**
19: $\widetilde{S} \leftarrow \tilde{u}$
20: **else**
21: $Q \leftarrow \tilde{u}$
22: $\mathscr{S} \leftarrow$ **create_domain_partition**$(\widetilde{S})$
23: **return** $\mathcal{G}(\widetilde{V}, \widetilde{A}, \mathscr{S})$

Algorithm 4.2 Предсказание потока

Input: Путевая развертка $\mathcal{G}(\widetilde{V}, \widetilde{A}, \mathscr{S})$

Output: Значение $f(v)$ для всех вершин $v \in V$

1: **procedure** PREDICT_FLOW($\mathcal{G}$)
2: **for** $\xi \in \mathscr{S}$ **do**
3: $Q \leftarrow \xi$
4: **while** $Q \neq \varnothing$ **do**
5: $\tilde{v} \leftarrow Q$
6: $v \leftarrow \rho^{-1}(\tilde{v})$
7: $f(v) \leftarrow f(v) + f(\tilde{v})/\hat{\gamma}(\tilde{v})$
8: **for** $\tilde{w} \in \delta^{out}(\tilde{v})$ **do**
9: $f(\tilde{w}) \leftarrow f(\tilde{w}) + f(\tilde{v})$
10: $visited \leftarrow visited \cup \{\tilde{w}\}$
11: **if** $\delta^{in}(\tilde{w}) \subseteq visited$ **then**
12: $Q \leftarrow \tilde{w}$
13: **return** f

Доказательство. Докажем, что $\widetilde{\mathbb{P}}(t) \subseteq \mathbb{P}(t)$. Пусть $P_{\widetilde{D}} = v_0, \ldots, v_l \in \widetilde{\mathbb{P}}(t)$, где $v_0 = s$ и $v_l = t$. Покажем, что для любого $n \in \widetilde{D}$ P_n является доменным путем для n. Для этого проверим выполнение свойств доменного пути. Из определения $\widetilde{\mathbb{P}}(t)$ следует, что $n \in D_{v_0}$. Из построения путевой развертки следует, что:

$$D_{\tilde{v}_{i-1}} = \Phi^{-1}_{v_{i-1}}(D_{v_{i-1}v_i} \cap D_{\tilde{v}_i}), \tag{4.77}$$

и что $D_{\tilde{v}_i} \neq \varnothing$. Таким образом, $n_{v_0,\ldots,v_i} \neq \varnothing$.

Более того, из (4.77) следует, что не существует $m \notin \widetilde{D}$ такого, что P_m является доменным путем, так как для вершины v_{i+1} в каждом шаге алгоритма выбираются все возможные домены $m' \in D_{v_i}$, такие что $m'_{v_i} \cap D_{v_{i+1}} \neq \varnothing$. Следовательно, $P_{\widetilde{D}}$ является объединением всех доменных путей из s в t, то есть $P_{\widetilde{D}} \in \mathbb{P}(t)$. Из чего следует, что $\widetilde{\mathbb{P}}(t) \subseteq \mathbb{P}(t)$.

Теперь докажем, что $\widetilde{\mathbb{P}}(t) \supseteq \mathbb{P}(t)$. Возьмем произвольный путь $P = v_0, \ldots, v_l \in \mathbb{P}(t)$. По построению путевой развертки (алгоритм 4.1) существует путь $\widetilde{P} \in \widetilde{\mathbb{P}}(t)$ такой, что $E(P) = E(\widetilde{P})$, то есть множество ребер путей совпадает. Равенство доменов этих путей будет следовать из (4.77).

Таким образом, доказано, что $\widetilde{\mathbb{P}}(t) = \mathbb{P}(t)$. □

Введем следующее обозначение: пусть $P \in \mathbb{P}(v)$ тогда:

$$\widetilde{\mathbb{P}}|_P = \left\{P' \in \widetilde{\mathbb{P}}(T) \mid E(P) \subseteq E(P')\right\}. \tag{4.78}$$

Из утверждения 4.9 следует:

Следствие 4.10.

$$\widetilde{\mathbb{P}}|_P = \mathbb{P}|_P \cap \mathbb{P}(T). \tag{4.79}$$

Докажем основное утверждение:

Теорема 4.11. Для любой вершины $v \in V$ графа выполняется:

$$\tilde{f}(v) = f(v), \tag{4.80}$$

где $\tilde{f}(v)$ — значение полученное алгоритмом 4.2.

Доказательство. Зафиксируем некоторую вершину $v \in V$. Пусть $\tilde{f}(v)$ — значение потока, которое приписывает алгоритм вершине v. Докажем, что $\tilde{f}(v) = f(v)$. По построению алгоритма видно, что:

$$\tilde{f}(v) = \sum_{\tilde{v} \in \rho^{-1}(v)} \sum_{P' \in \mathcal{I}(\tilde{v})} \frac{F(P')}{\gamma(\tilde{v})}, \tag{4.81}$$

где:

$$\mathcal{I}(\tilde{v}) = \left\{ P' \in \widetilde{\mathbb{P}}(T) \mid \rho(\tilde{v}) \in P' \right\}. \tag{4.82}$$

То есть, значение потока в вершине v складывается из значений потоков всех путей в путевой развертке, проходящих через вершины $\tilde{v}$ такие, что $\rho(\tilde{v}) = v$.

Каждый путь $P' : s \to t$ в некотором слагаемом суммы (4.81) образует путь $P : s \to \rho(\tilde{v})$ такой, что $E(P) \subseteq E(P')$. Стоит отметить, что, если вершина v встречается в пути P' несколько раз, то столько же путей будет образовано после перегруппироваки. Также стоит отметить, что по построению $\gamma(\tilde{v}) = \hat{\gamma}(P' - P)$. Таким образом, слагаемые в сумме (4.81) возможно перегруппировать следующим образом:

$$\tilde{f}(v) = \sum_{P \in \mathcal{J}(v)} \sum_{P' \in \widetilde{\mathbb{P}}|_P} \frac{F(P')}{\hat{\gamma}(P' - P)}, \tag{4.83}$$

где:

$$\mathcal{J}(v) = \left\{ P : S \to v \mid \exists P'' \in \widetilde{\mathbb{P}} \ : \ E(P'') = E(P) \right\}. \tag{4.84}$$

Из построения видно, что в сумме (4.83) не участвуют пути P такие, что не существует n таких, что P_n не является доменным путем. Поэтому пути

в первой сумме принадлежат $\mathbb{P}(v)$, то есть $\mathcal{J}(v) = \mathbb{P}(v)$. Таким образом, из следствия 4.10 и теоремы 4.7 следует:

$$\tilde{f}(v) = \sum_{P \in \mathbb{P}(v)} \sum_{P' \in \mathbb{P}(T) \cap \mathbb{P}|_P} \frac{F(P')}{\hat{\gamma}(P' - P)} = f(v), \tag{4.85}$$

и теорема доказана. □

Из теоремы 4.11 следует, что с помощью описанных выше алгоритмов возможно предсказать значения потока во всем графе зависимостей правил, зная значения потока на множестве $\mathscr{S}$, где $\mathscr{S}$ — множество дополнительных вершин. Это множество далее будет использоваться для создания дополнительных правил маршрутизации в сети.

Утверждение 4.12. Сложность алгоритма 4.1 составляет $O(|V| * |\mathbb{P}| + |\mathbb{P}|^2)$, где V — множество вершин графа, $\mathbb{P}$ — множество доменных путей в графе.

Доказательство. Строки 6—21 описывают основной цикл алгоритма, который проходит по всем вершинам графа. Из утверждения 4.9 следует, что алгоритм проходит по всем доменным путям графа. Так как длина каждого доменного пути может быть ограничена сверху размером множества вершин, мы получаем сложность цикла, равную $O(|V| * |\mathbb{P}|)$.

Строка 22 описывает вызов подпроцедуры, которая находит всевозможные пересечения пар доменов путей. Так как в подпроцеуре производятся попарные пересечения доменов, то сложность подпроцедуры равна $O(|\mathbb{P}|^2)$.

Таким образом, получаем сложность алгоритма 4.1, равную $O(|V| * |\mathbb{P}| + |\mathbb{P}|^2)$. □

Утверждение 4.13. Сложность алгоритма 4.2 составляет $O(|V| * |\mathbb{P}|)$, где V — множество вершин графа, $\mathbb{P}$ — множество доменных путей в графе.

Доказательство. Строки 4—12 описывают основной цикл алгоритма, который также, как и основной цикл алгоритма 4.9, проходит по всем вершинам графа. Таким образом, аналогично доказательству утверждения 4.12 получаем сложность алгоритма 4.2, равную $O(|V| * |\mathbb{P}|)$. □

Стоит отметить, что, если любой путь в графе зависимостей правил является доменным путем, то величина $|\mathbb{P}|$ может экспоненциально зависеть от количества вершин графа [73]. Но такая ситуация не описывает реальных механизмов маршрутизации, которые используются в ПКС [1]. Таким образом, реальное время работы алгоритма должно быть исследовано экспериментальным путем.

Глава 5

Алгоритм обнаружения

В этой главе подробно описывается разработанный алгоритм обнаружения скомпрометированых коммутаторов в ПКС.

5.1. Общее описание предложенного алгоритма

Алгоритм обнаружения скомпрометированных коммутаторов основан на анализе сетевой статистики, предоставляемой счетчиками, установленными на правилах маршрутизации.

Идея алгоритма заключается в том, что, используя информацию об установленных правилах маршрутизации в сети, имеющуюся на контроллере, возможно проверить корректность сетевой статистики, получаемой от коммутаторов. А именно, проверить корректность значений счетчиков правил маршрутизации.

Некорректные значения счетчиков правил маршрутизации, передаваемых коммутатором контроллеру, могут быть получены в следствие атак на *data-plane*, проводимых при помощи скомпрометированных коммутаторов. Из классификации атак на *data-plane*, приведенной в главе 1, следует, что атаки производятся при помощи добавления новых или удаления существующих правил маршрутизации. При этом контроллер не получает информацию о таких изменениях, если злоумышленник пытается скрыть факт наличия атаки.

Таким образом, представление контроллера о состоянии сети отличается от её реального состояния. А значит и модельные графы зависимостей правил, описывающие реальную сеть и представление сети, имеющееся на контроллере, будут отличаться.

Такие атаки, как сброс пакетов, дублирование трафика или некорректная маршрутизация, изменяют множества доменных путей модельного графа, так как они создают/удаляют маршруты в сети.

Если множества доменных путей различаются, то по теореме 4.7 следует, что значения потоковых функций на вершинах двух модельных графов будут отличны. Такие потоковые функции на вершинах графа описывают значение счетчиков правил маршрутизации, установленных в сети. Следовательно, атаки, проводимые при помощи скомпрометированных коммутаторов, будут приводить к некорректным значениям счетчиков правил маршрутизации. Под некорректностью понимается наличие значений, которые не совпадают со значениями потоковых функций в графе, описывающем представление контроллера о сети.

Таким образом, необходим алгоритм проверки корректности счетчиков правил маршрутизации. Такой алгоритм может быть реализован с использованием алгоритма предсказания значений потока через вершину графа зависимостей правил (алгоритм 4.2). Алгоритм проверки выглядит следующим образом:

- Предсказываются значения счетчиков правил маршрутишации.
- Предсказанные значения сравниваются с реальными.

Если значения счетчиков отличаются больше чем на заданный допустимый порог, то фиксируется наличие атаки. Алгоритм также учитывает легитимные потери пакетов, связанные с перегрузками в сети.

Использование алгоритма предсказания значений счетчиков предполагает наличие путевой развертки, описывающей доменные пути. Путевая развертка строится при помощи алгоритма 4.1 на основе графа зависимостей правил, который в свою очередь может быть построен при помощи анализа

OpenFlow сообщений, проходящих между контроллером и коммутаторами.

Для предсказания значений счетчиков правил маршрутизации также необходимо необходимо проанализировать сетевую статистику – то есть узнать значения потоковой функции на доменных путях. Для этого нужно установить в сеть дополнительные правила маршрутизации, поля *match* которых определяются доменными путями. Эти правила предназначены для подсчета количества пакетов с заголовками, определенными доменными путями.

Таким образом алгоритм состоит из двух основных фаз:

1. Установка дополнительных правил;
    - Построение графа зависимостей правил;
    - Построение путевой развертки;
    - Создание дополнительных правил;
    - Установка приоритетов;
    - Установка правил.
2. Анализ сетевой статистики.
    - Предсказание значений счетчиков;
    - Обнаружение скомпрометированных коммутаторов.

5.2. Установка дополнительных правил

Первая фаза алгоритма обнаружения скомпрометированных коммутаторов предполагает установку в сеть дополнительных правил маршрутизации.

Каждое такое правило маршрутизации ставится в соответствие некоторому доменному пути из путевой развертки, который определяет поля *match* этих правил маршрутизации. Поля *match* устанавливаются таким образом, чтобы дополнительные правила обрабатывали только пакеты из соответствующих им доменных путей. Это необходимо для того, чтобы счетчики дополнительных правил маршрутизации отражали значения потоковых функций на доменных путях.

Для того, чтобы определить вид дополнительных правил, необходимо построить путевую развертку, которая описывает доменные пути. Путевая развертка сроится по алгоритму 4.1 на основе графа зависимостей правил, описывающего доменные пути.

5.2.1. Построение графа зависимостей правил

Граф зависимостей правил строится инкрементально на основе событий изменения состояния сети. События изменения сети описываются OpenFlow сообщениями, передаваемыми между контроллером и коммутаторами. Такие события включают:

1. Подключение нового коммутатора;
2. Отключение существующего коммутатора;
3. Обнаружение физической линии между коммутаторами;
4. Обрыв физической линии между коммутаторами;

5. Установку нового правила маршрутизации;

6. Удаление существуюшего правила маршрутизации.

Каждое описанное выше событие добавляет или удаляет некоторый набор вершин графа зависимостей правил. При добавлении/удалении вершины также изменяется набор ребер графа. Опишем более подробно процесс изменения ребер графа.

Подключение/отключение коммутатора

При появлении нового коммутатора создается следующий набор вершин:

- Стоковые вершины;
- Истоковые вершины;
- Вершины, соответствующие правилам по-умолчанию.

Пара стоковых и истоковых вершин создается для каждого порта коммутатора. Этими правилами описываются получатели и генераторы трафика. Также добавляются вершины, соответствующие правилам по-умолчанию для каждой таблицы коммутатора. Такие правила описывают действия над пакетами, не обработанными никакими другими правилами в таблице маршрутизации.

Для правила по-умолчанию в нулевой таблице коммутатора создаются ребра из каждого истокового правила, соответствующего портам этого коммутатора. Домены этих ребер устанавливаются в $<p, \mathcal{H}>$, где p — это номер соответствующего порта, а $\mathcal{H}$ — все пространство заголовков. То есть истоки могут генерировать пакеты с произвольными заголовками.

При отключении коммутатора удаляются все вершины, соответствующие правилам из таблиц маршрутизации этого коммутатора.

Добавление/удаление правила маршрутизации

При добавлении нового правила маршрутизации создается вершина v графа зависимостей правил. Если новое правило не является правилом по-умолчанию, то обновляются домены ребер, входящих в вершины с меньшим приоритетом. То есть, если добавляется вершина v с доменом D_v, то из доменов всех ребер, входящих в вершины с меньшим приоритетом, вычитается домен D_v.

Также для новой вершины v создаются входящие и исходящие ребра. Входящие ребра создаются следующим образом.

1. Для каждого ребра, входящего в вершину с приоритетом, меньшим чем у вершины v, берется начальная вершина u;

2. Исходящий домен вершины u пересекается с доменом вершины v;

3. Полученное пересечение устанавливается в качестве домена D_{uv} нового ребра uv.

Исходящие ребра создаются на основе действий, производимых новой вершиной v:

1. *Отправка пакетов на некоторый порт p.*
   Создаются ребра ко всем вершинам, описывающим правила из нулевой таблицы коммутатора, находящегося на порту p.

2. *Отправка пакетов в некоторую таблицу τ.*
   Создаются ребра ко всем вершинам таблицы τ. При создании ребер также учитывается приоритет правил маршрутизации.

3. *Отправка пакетов в некоторое групповое правило.*
   Создается ребро в вершину g, соответствующую групповому правилу.

4. *Отправка пакетов на контроллер.*

   Создается ребро в вершину c.

5. *Сброс пакетов.*

   Создается ребро в вершину d.

Также, необходимо учитывать, что действия могут изменять заголовок обрабатываемого пакета. Такое изменение описывает передаточную функцию T_v, описываемую добавляемой вершиной графа.

Добавление/удаление физической линии

При обнаружении физической линии между двумя портами p_1 и p_2 выполняются следующие действия:

1. Удаляются ребра исходящие из стоковых и истоковых вершин портов p_1 и p_2.

2. Создаются ребра из вершин, отправляющих пакеты на порт p_1 или p_2, к вершинам из нулевых таблиц обоих коммутаторов. Эти ребра описывают пересылку пакетов по физической линии.

При удалении физической линии между двумя портами p_1 и p_2 выполняются следующие действия:

1. Удаляются ребра между вершинами из нулевой таблицы обоих коммутаторов и вершинами, отправляющими пакеты на порт p_1 или p_2.

2. Создаются ребра из истоков портов p_1 или p_2 ко всем правилам из нулевых таблиц обоих коммутаторов.

5.2.2. Построение путевой развертки

По построенному графу зависимостей правил можно построить путевую развертку при помощи алгоритма 4.1.

При построении путевой развертки необходимо учитывать, что состояние сети постоянно изменяется, то есть устанавливаются/удаляются правила маршрутизации, добавляются/удаляются коммутаторы и т.д. Таким образом, необходимо поддерживать динамическое добавление/удаление вершин в путевой развертке для того, чтобы не перестраивать всю путевую развертку по алгоритму 4.1 заново при каждом изменении состояния сети.

Динамическое изменение путевой развертки заключается в добавлении и удалении поддеревьев. Такие поддеревья могут быть найдены при помощи списка изменений графа зависимостей правил. Под изменениями понимается добавление новых ребер и удаление существующих. Эти ребра влияют на путевую развертку, так как каждому ребру графа зависимостей правил соответствует набор ребер из путевой развертки.

- Добавление нового ребра в графе зависимостей правил приводит к добавлению набора поддеревьев в путевой развертке;
- Удаление ребра приводит к удалению соответствующих поддеревьев в путевой развертке.

При изменении путевой развертки учитываются только ребра, исходящие из новой/удаленной вершины, так как ребра, входящие в эти вершины, уже будут учтены при перестроении поддерева из-за того, что дерево строится по путям из ребер в обратную сторону.

5.2.3. Создание дополнительных правил

Доменные пути, полученные после построения путевой развертки, могут быть использованы для предсказания значений счетчиков правил маршрутизации, установленных в сети. Для предсказания значений счетчиков необходима информация о количестве пакетов, прошедших по каждому доменному пути. Количество таких пакетов описывается значением потока по домену такого пути.

Для определения количества пакетов, проходящих по некоторому доменному пути, необходимо установить в сеть дополнительные правила маршрутизации. Основная цель таких правил маршрутизации — считать количество пакетов с определенными заголовками.

При создании дополнительных правил необходимо учитывать, что домены путей могут пересекаться, то есть один и тот же пакет может проходить по разным доменным путям. Такая ситуация возникает в случае *multicast* маршрутизации, которая предполагает наличие в сети правил маршрутизации, дублирующих пакеты и отправляющих их на разные порты.

Спецификация протокола OpenFlow [3] описывает, что наличие правил с пересекающимися полями *match*, приводит к неопределенному поведению коммутатора. То есть, выбор правила для обработки пакетов в данном случае будет целиком зависеть от реализации коммутатора.

Поэтому необходимо создать набор дополнительных правил с непересекающимися полями *match*. Для этого требуется найти домены, являющиеся пересечениями доменов путей. Эти домены буду описывать новые правила, которые считают количество пакетов сразу для нескольких доменных путей. Стоит отметить, что пересечения доменов производятся только для тех путей, мультипликатор которых не равен 1, то есть для путей проходящих через дублирующие правила.

5.2.4. Установка приоритетов

Домены путей состоят из номеров портов и множеств пространства заголовков. Необходимо отметить, что каждое множество H в пространстве заголовков может быть представлено в виде *wildcard* выражения:

$$H = \sum_{i \in [1,n]} \left(x_i - \sum_{j \in [1,m_i]} y_{ij} \right), \tag{5.1}$$

где x_i и y_{ij} — *wildcard* маски, и $y_{ij} \subseteq x_i$ для любого $i \in [1, n]$ и для любого $j \in [1, m_i]$. Под операцией включения $\subseteq$ понимается включение множеств, описываемых *wildcard* масками. Маски x_i далее будем называть *положительными*, а маски y_{ij} — *отрицательными*.

Каждое OpenFlow правило содержит поле *match*, которое соответствует одной *wildcard* маске, описывающей множество загловков. Следовательно, для *wildcard* выражений, содержащих более одной маски, необходимо устанавливать более одного правила.

При подсчете количества пакетов из множества H необходимо учитывать счетчики правил с положительными масками и не учитывать счетчики правил с отрицательными, так как заголовки этих пакетов, описываемых отрицательной маской, не принадлежат доменам путей. Таким образом, необходимо устанавливать разные приоритеты правилам с положительными и отрицательными масками, а именно, правила с отрицательными масками должны иметь больший приоритет.

Благодаря семантике протокола OpenFlow, счетчики на правилах с положительными масками будут описывать количество пакетов из домена пути, так как в данном случае не будут учитываться счетчики правил с отрицательными масками.

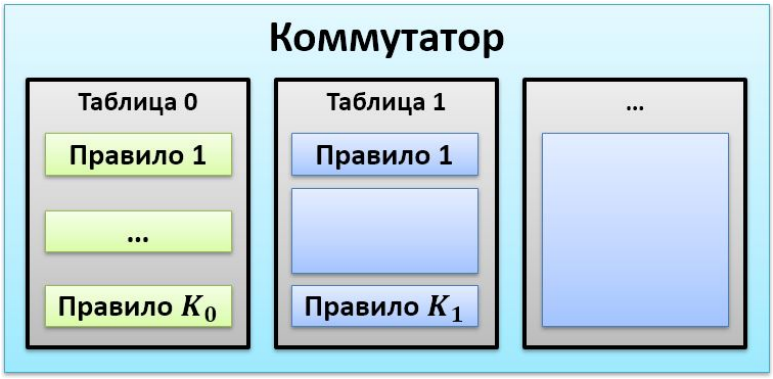


Рисунок 5.1: Дополнительные правила маршрутизации

5.2.5. Установка правил

После того, как построена путевая развертка, в сеть установливаются дополнительные правила маршрутизации. Эти правила установливаются в нулевые таблицы граничных коммутаторов, для того, чтобы они могли обрабатывать все пакеты, проходящие через коммутатор (рис. 5.1).

Приложения на контроллере также могут использовать нулевую таблицу. Для того, чтобы не было пересечений с дополнительными правилами, во всех сообщениях, передаваемых между контроллером и коммутаторами, изменяются номера таблиц. То есть, все устанавливаемые контроллером правила смещаются на одну таблицу. Например, если правило устанавливается контроллером в нулевую таблицу, то это правило будет установлено в первую.

При отправке контроллеру ответов от коммутаторов, в ответах восстанавливается нулевая таблица для того, чтобы не нарушить работу приложений контроллера.

5.3. Анализ сетевой статистики

Вторая фаза алгоритма обнаружения скомпрометированных коммутаторов предполагает предсказание значений счетчиков правил маршрутизации и сравнение этих значений с реальными, предоставляемыми коммутаторами в сети.

5.3.1. Предсказание значений счетчиков

Для предсказания значения счетчика правила, соответствующего некоторой вершине v графа зависимостей правил, выполняется следующий набор действий:

1. Определяется набор $\mathcal{V}$ вершин путевой развертки, соответствующих вершине v графа зависимостей правил;

2. Следуя по ветвям деревьев путевой развертки с корнями в множестве $\mathcal{V}$, выбираются вершины, соответствующие дополнительным правилам;

3. В сеть отправляются запросы статистики для набора выбранных дополнительных правил маршрутизации;

4. После получения статистики по дополнительным правилам маршрутизации выполняется алгоритм предсказания значений счетчиков (алгоритм 4.2);

5. В качестве предсказанного значения счетчика правила, соответствующего вершине v, возвращается сумма значений потоков вершин из множества $\mathcal{V}$;

6. Значения потока в вершинах путевой развертки сохраняются для того,

чтобы при новом запросе учитывать только изменение счетчика правила маршрутизации по сравнению с предыдущим значением.

Необходимо отметить, что отличие в значениях счетчиков может появиться из-за легитимных потерь пакетов на портах коммутаторов. Следовательно, нужно проверять, что пакеты были потеряны вследствие перегрузки, а не из-за компрометации коммутатора.

Все пакеты, потерянные из-за перегрузки, учитываются в счетчиках на портах коммутаторов. Для проверки легитимности потерь трафика необходимо сравнить значение нехватки пакетов, полученное при предсказании счетчиков, и количество пакетов, потерянных на портах. При сравнении необходимо учитывать только те порты, через которые проходят доменные пути, соответствующие проверяемому в данный момент правилу маршрутизации.

5.3.2. Обнаружение скомпрометированных коммутаторов

Для обнаружения скомпрометированного коммутатора из сети запрашивается реальное значение счетчика правила машрутизации и сравнивается с предсказанным значением. Основная идея заключается в том, что при наличии в сети атаки на контур передачи данных эти значения будут отличаться.

Сравнение значений счетчиков

Каждому коммутатору присвоено значение $T \in [0, 1]$, называемое *уровнем доверия*. Если уровень доверия некоторого коммутатора понижается ниже заданного порога, то регистрируется наличие в сети скомпрометированного коммутатора.

При несоответствии значений счетчиков уровни доверия изменяются следующим образом:

1. Выбираются коммутаторы, которые влияют на значение анализируемого счетчика правила маршрутизации;

    - Для нахождения таких коммутаторов, алгоритм проходится по всем доменным путям, содержащим вершину, соответствующую анализируемому правилу. Вершины на этих путях влияют на значение анализируемого счетчика по теореме 4.7.

2. Уровни доверия выбранных коммутаторов понижаются в δ раз.

Если проверка правила прошла успешно, а именно, значения предсказанного и реального значения счетчика равны с учетом легитимных потерь пакетов, то уровни доверия коммутаторов, влияющих на этот счетчик, увеличиваются. В данном случае увеличение уровня доверия будет аддитивное, то есть уровень увеличится на константу ε.

Таким образом, при успешной проверке уровни доверия будут увеличиваться аддитивно на константу ε, в то время как при неуспешной проверке, они будут уменьшаться мультипликативно в δ раз.

Выбор правила для анализа

В реальной сети может быть установлено большое количество правил маршрутизации, и проверка всех правил маршрутизации при каждом изменении состояния сети может создать дополнительную нагрузку на сеть. Нагрузка на сеть появляется из-за того, что каждое предсказание значения счетчика требует набора запросов сетевой статистики к дополнительным правилам маршрутизации.

Для минимизации нагрузки на сеть используется рандомизированный механизм проверки правил. Механизм заключается в следующем.

- Выбор правила для проверки значений счетчиков происходит случайно после фиксированного интервала времени;
- Правило выбирается в зависимости от уровня доверия коммутатора, на котором оно установлено.

Вероятность P_i выбора правила r_i для проверки равна:

$$P_i = \frac{1/T(v_i)}{\sum_{j \in [1,n]} 1/T(v_j)}, \tag{5.2}$$

где $T(v_i)$ — уровень доверия коммутатора, на котором установлено правило v_i, а n — количество правил маршрутизации, установленных в сети.

Из (5.2) следует, что при уменьшении уровней доверия некоторого коммутатора увеличивается вероятность проверки правила маршрутизации, установленного на этом коммутаторе, увеличивается.

Глава 6

Реализация

В данной главе представлено описание прототипа системы обнаружения скомпрометированных коммутаторов и результаты его экспериментального исследования. Прототип реализует алгоритм обнаружения скомпрометированных коммутаторов, предложенный в главе 5.

При реализации использовано два языка программирования: C++ [74] и Python [75]. На момент написания текста диссертационной работы прототип содержал 11475 строк кода.

6.1. Архитектура

Система обнаружения скомпрометированных ПКС коммутаторов представляет собой промежуточный слой между контроллером и коммутаторами.

Эта система перехватывает OpenFlow сообщения и на их основе строит граф зависимостей правил. Используя граф зависимостей правил, система обнаружения строит путевую развертку и устанавливает в сеть дополнительные правила маршрутизации. При помощи статистики, полученной с этих правил, система обнаружения предсказывает значения счетчиков произвольных правил маршрутизации в сети и сравнивает их с реальными значениями, полученными из сети.

Если значения счетчиков отличаются, то система обнаружения понижает уровень доверия всех коммутаторов, влияющих на значения этого счетчика. Такие коммутаторы находятся при помощи обхода всех путей в графе зависимостей правил, которые проходят через тестируемое правило маршрутизации. Если уровень доверия некоторого коммутатора понижается ниже

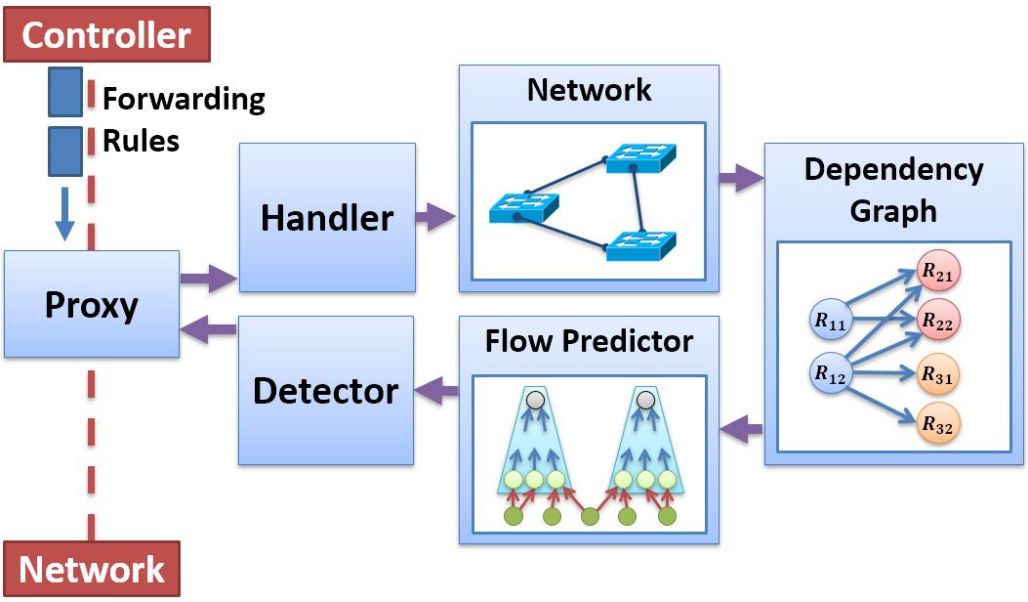


Рисунок 6.1: Архитектура системы обнаружения

заданного порога, то система сообщает о наличии в сети скомпрометированного коммутатора.

На рисунке 6.1 приведена схема архитектуры системы обнаружения скомпрометированных коммутаторов. Прямоугольниками обозначены компоненты системы, стрелками — потоки данных между компонентами.

Модуль ***Proxy*** отвечает за получение информации о топологии сети и правилах маршрутизации, установленных на коммутаторы. Получение информации производится при помощи перехвата OpenFlow сообщений между контроллером и коммутаторами. Перехват сообщений необходим для построения графа зависимостей правил, описывающего состояние сети.

Модуль ***Handler*** отвечает за обработку OpenFlow сообщений, перехваченных модулем *Proxy*. Этот модуль также преобразует OpenFlow сообщения во внутреннее представление программы, а именно представление полей *match* в виде *wildcard* масок модели *Header Space*.

Модуль ***Network*** представляет собой базу данных, хранящую информацию о текущем состоянии сети. Под этой информацией понимается: список коммутаторов с их свойствами, список портов и таблиц маршрутизации, топология сети и правила маршрутизации установленные на коммутаторах.

Модуль ***Dependency Graph*** строит граф зависимостей правил на основе событий, изменяющих состояние сети. Этот модуль также предоставляет интерфейс к графу зависимостей правил для других модулей системы.

Модуль ***Flow Predictor*** отвечает за предсказание значений счетчиков правил маршрутизации. Модуль строит путевую развертку на основе графа зависимостей правил и использует ее для создания и установки в сеть дополнительных правил маршрутизации. Модуль также запрашивает статистику с этих правил маршрутизации для предсказания значений счетчиков произвольных правил маршрутизации.

Модуль ***Detector*** отвечает за обнаружение скомпрометированных коммутаторов, основываясь на данных, предоставляемых модулями *Dependency Graph* и *Flow Predictor*.

6.2. Экспериментальная проверка предложенного решения

В этой главе описаны результаты экспериментальной проверки предложенного решения для обнаружения скомпрометированных коммутаторов.

6.2.1. Методика испытаний

Экспериментальное исследование нацелено на определение следующих свойств предложенного решения:

1. Погрешность предсказания значений счетчиков;
2. Время работы алгоритма;
3. Ошибки первого и второго рода при обнаружении.

Под ошибками первого рода понимаются ситуации, когда легитимный коммутатор был классифицирован как скомпрометированный. Под ошибками второго рода понимаются ситуации, когда скомпрометированный коммутатор был классифицирован системой обнаружения как легитимный.

Погрешность предсказания значений счетчиков

Под погрешностью предсказания счетчиков маршрутизации понимается величина, равная разности реального знаечния счетчика и предсказанного.

Погрешность предсказания влияет на ошибки первого рода при обнаружении скомпрометированных коммутаторов, так как наличие отклонения предсказанных значений счетчиков от реальных влияет на уровни доверия коммутаторов.

На погрешность предсказания значений счетчиков влияют следующие факторы:

1. Задержки обработки трафика;

2. Объемы потоков данных.

Задержки обработки трафика

Под задержками обработки трафика понимается сумма следующих величин:

1. Задержек распространения — времени прохождения сигнала по каналам данных;

2. Задержек пакетизации — времени, за которое биты пакета с первого до последнего переданы в канал;

3. Задержек в буфере коммутатора — времени, которое пакеты проводят в ожидании обработки коммутатором.

Задержки влияют на погрешность предсказания, потому что предсказание счетчиков основывается на запросах статистики с дополнительных правил маршрутизации. Во время запроса статистики из сети, часть пакетов, обработанных дополнительными правилами маршрутизации, может не успеть дойти по сети до анализируемого правила. Разница в этих пакетах будет отражаться в разнице между предсказанным и реальным значением счетчика анализируемого правила.

Объемы потоков данных

Объемы потоков данных, проходящих через коммутаторы в сети, влияют на погрешность предсказания значений счетчиков в совокупности с задержками обработки пакетов. Чем больше объем трафика в сети, тем больше пакетов не успеет дойти до анализируемого правила, и тем больше будет различие между реальными и предсказанными значениями счетчика анализируемого правила.

Время работы алгоритма

Система обнаружения перехватывает и ставит в очередь OpenFlow сообщения от контроллера, которые описывают установку правил маршрутизации. Очередь необходима для того, чтобы создавать дополнительные правила маршрутизации и устанавливать их одновременно с оригинальными правилами, созданными контроллером.

Наличие этой очереди замедляет реакцию контроллера на события в сети, потому что правила маршрутизации устанавливаются с задержкой. Эта задержка зависит от времени работы алгоритма, а именно времени построения графа зависимостей правил и путевой развертки, которое в свою очередь зависит от размера сети и количества правил маршрутизации, установленных в сеть.

Таким образом, необходимо оценить время работы алгоритма в зависимости от:

1. Размера сети;
2. Количества правил маршрутизации.

Ошибки первого и второго рода при обнаружении

Основная задача алгоритма — обнаружение скомпрометированных коммутаторов. Поэтому необходимо, во-первых, экспериментально показать корректность работы алгоритма, и во-вторых, оценить ошибки первого и второго рода.

Была исследована зависимость ошибок первого и второго рода от следующих факторов:

1. Размера сети;

2. Количества правил;

3. Объемов потоков данных;

4. Типа атаки;

5. Длительности атаки;

6. Количества скомпрометированных коммутаторов.

Необходимо отметить, что величина ошибок первого и второго рода зависит от всех факторов, влияющих на точность предсказания счетчиков.

Тестовый стенд

Для реализации экспериментального исследования был разработан тестовый стенд на основе библиотеки *Mininet* [76]. Эта библиотека позволяет создавать виртуальные сети из программных коммутаторов и также создавать искусственные задержки в сети. В качестве коммутаторов использовались программные коммутаторы *Open vSwitch* [19].

В качестве тестовых топологий сети использовались топологии из библиотеки *TopologyZoo* [77]. Эта библиотека содержит 522 топологии, описывающих реальные топологии сетей телеком операторов. Также для экспериментов использовались линейные топологии и топологии типа *Tree* и *FatTree* [78], которые часто используются в центрах обработки данных [79]. Размеры топологий варьировались от 5 до 300 коммутаторов. В качестве тестового ПКС контроллера использовался контроллер RunOS [80].

Виртуальная сеть с программными коммутаторами была запущена на сервере со следующими характеристиками:

1. ЦПУ (Центральное процессорное устройство) — Intel® Xeon® E5-2620 v4, 32 ядра по 2.10GHz;

2. ОЗУ (Оперативное запоминающее устройство) — 32 Gb DDR3 1333 MHz;

3. Сетевой интерфейс — 1 Gb Ethernet.

Контроллер был запущен на виртуальной машине, находящейся на отдельном сервере со следующими характеристиками:

1. ЦПУ (Центральное процессорное устройство) — Intel® Core® i7 9xx (Nehalem Class Core i7), 2 ядра по 2.4 GHz;

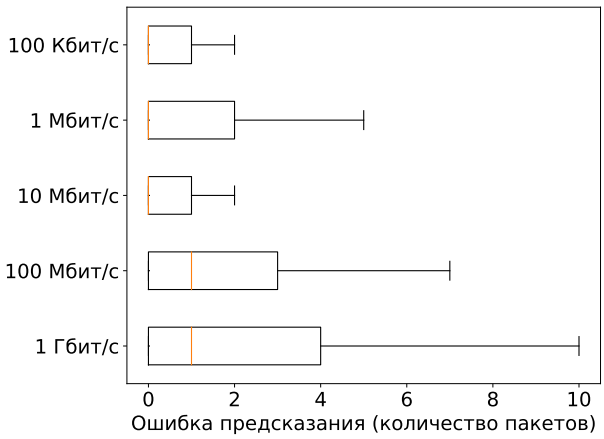
2. ОЗУ (Оперативное запоминающее устройство) — 2 Gb DDR3 1333 MHz;

3. Сетевой интерфейс — 1 Gb Ethernet.

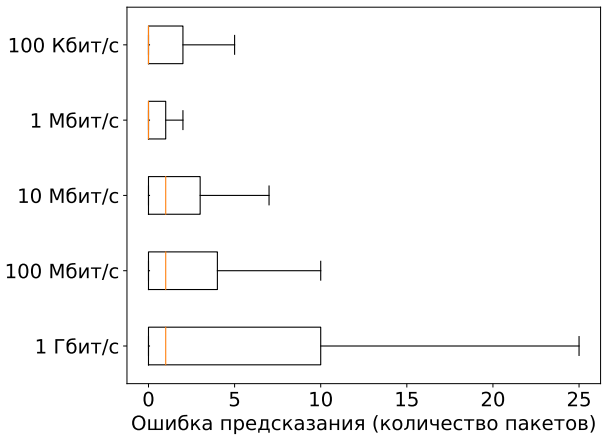
Среднее время работы алгоритма определялось следующим образом. Эксперименты проводились по 10 раз на каждой топологии. После каждого запуска измерялась задержка установки правил маршрутизации. Все времена суммировались и усреднялись по числу запусков. Полученная величина бралась за среднее время выполнения программы.

Для определения погрешности предсказания значений счетчиков в сети создавалась искуственная задержка и измерялась разность в значениях предсказанных и реальных счетчиков правил маршрутизации.

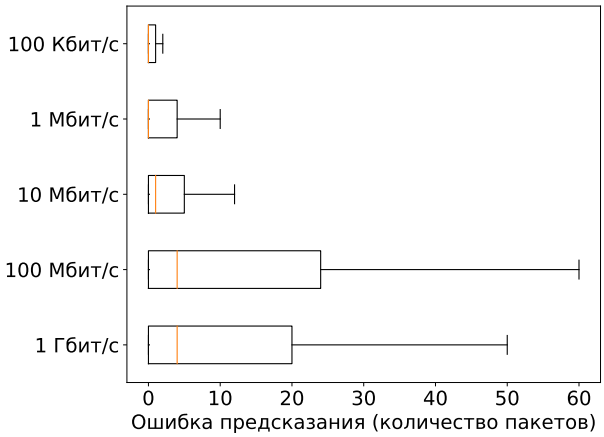
Для проверки корректности работы алгоритма в сеть случайно добавлялись правила сбрасывающие, дублирующие и некорректно маршрутизирующие трафик. Далее проверялось, что алгоритм корректно обнаружил скомпрометированные коммутаторы, и вычислялись величины ошибки первого и второго рода.



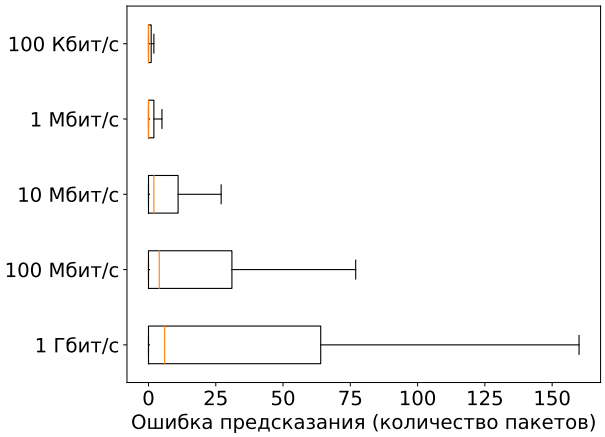
(a) Задержка < 1 мс



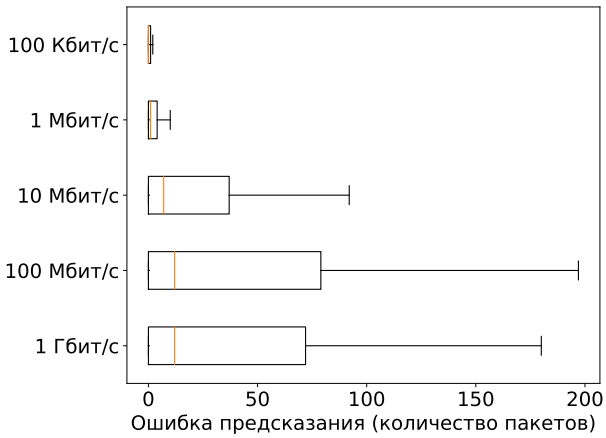
(b) Задержка 5 мс



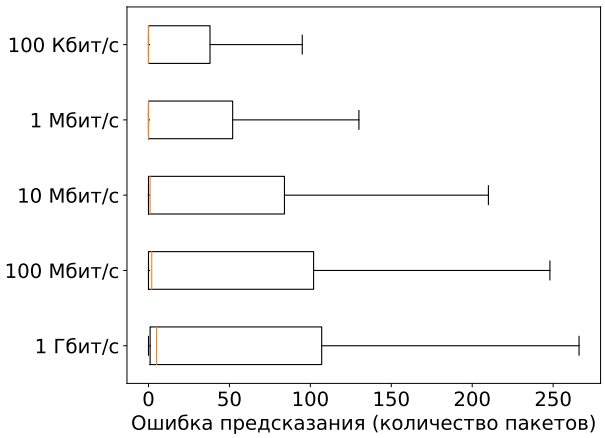
(c) Задержка 10 мс



(d) Задержка 50 мс



(e) Задержка 100 мс



(f) Задержка 300 мс

Рисунок 6.2: Зависимость ошибки предсказания счетчика от объема потоков данных

6.2.2. Результаты испытаний

Погрешность предсказания значений счетчиков

На рисунке 6.2 показана зависимость погрешности предсказания счетчика (разница между реальными и предсказанными значениями счетчиков) от объема потоков данных.

На рисунке также показана зависимость погрешности от задержек в сети. Под задержкой понимается средняя сквозная (*end-to-end*) задержка для всех пар подключенных клиентов.

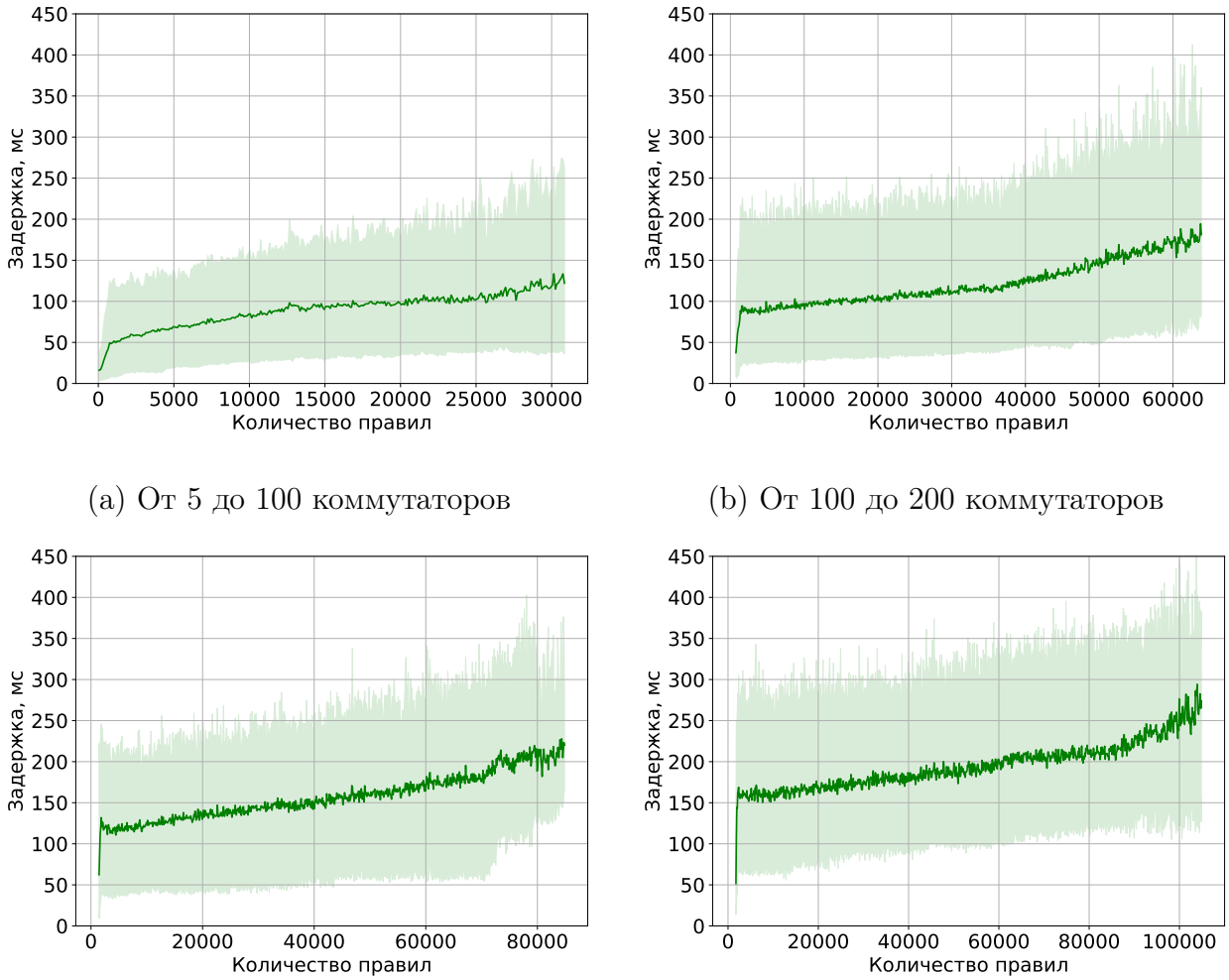
График представляет собой *box-plot*, описываемый следующим образом:

1. Оранжевой линией показана медиана значений погрешности;
2. Прямоугольником показаны первая и третья квартили, отделяющие 25% и 75% значений погрешности соответственно;
3. Штрихами показан 95%-ый доверительный интервал.

Под 95%-ым доверительный интервал показывает, что 95%-ов значений выборки принадлежат этому интервалу.

Согласно результатам тестирования погрешность практически одинакова для объемов потоков данных 100 Кбит/с до 10 Мбит/с и начинает возрастать для потоков объемом более 100 Мбит/с. Также при сквозной задержке в 300 мс погрешность начинает возрастать до 100 пакетов даже для потоков объемом 100 Кбит/с. Это объясняется тем, что из-за больших значений задержки, большое количество пакетов не успевает пройти от дополнительных правил на границе сети до анализируемого правила.

Тестирование также показало, что максимальный порог легитимных отклонений, превышение которого может свидетельствовать о наличии в сети атаки, равен 270 пакетам.



(a) От 5 до 100 коммутаторов

(b) От 100 до 200 коммутаторов

(c) От 200 до 250 коммутаторов

(d) От 250 до 300 коммутаторов

Рисунок 6.3: Зависимость задержки от количества коммутаторов и правил в сети и размера топологии

Время работы алгоритма

На рисунке 6.3 показана зависимость задержки от количества правил в сети и размера топологии. Линией на графиках показано среднее значение времени работы алгоритма а закрашенными областями — 95%-ый доверительный интервал.

На рисунке 6.4 показана зависимость задержки от количества коммутаторов в сети усредненная по различным топологиям с одинаковым количе-

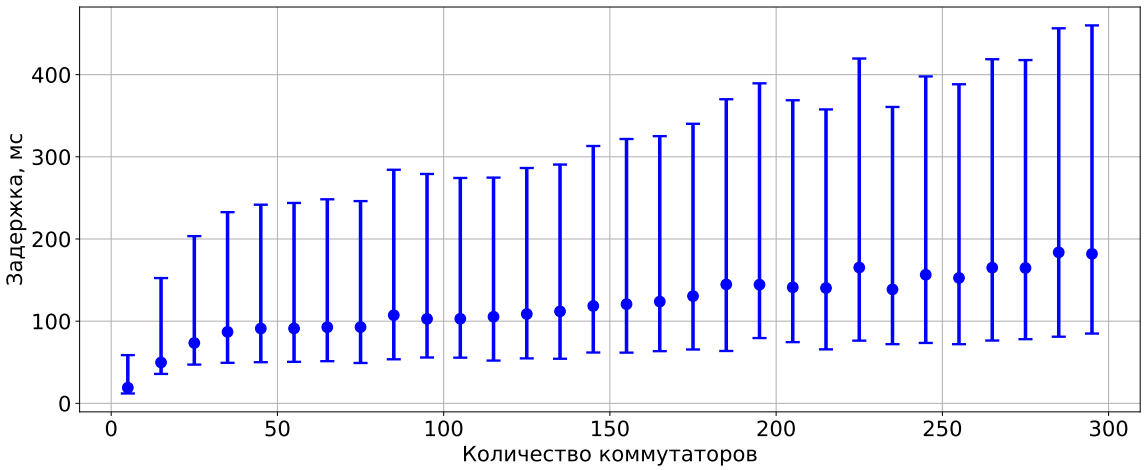


Рисунок 6.4: Зависимость задержки от количества коммутаторов в сети

ством коммутаторов. Точкой на графике показано среднее значение а интервалами — 95%-е доверительные интервалы.

Результаты экспериментов показывают, что время работы алгоритма, увеличивается с увеличением количества правил, установленных в сеть. Задержки могут быть ограничены сверху 420 миллисекундами для больших топологий (300 коммутаторов) с большим количеством правил (100.000). Также стоит отметить, что время работы алгоритма линейно возрастает с увеличением количества правил маршрутизации.

Точность обнаружения скомпрометированных коммутаторов

Тип атаки	**FP**	**FN**
Отсутствие атаки	0.002	n/a
Некорректная маршрутизация	0.008	0.019
Дублирование трафика	0.023	0.049
Сброс трафика	0.037	0.066

Таблица 6.1: Ошибки первого (FP) и второго рода (FN) при обнаружении

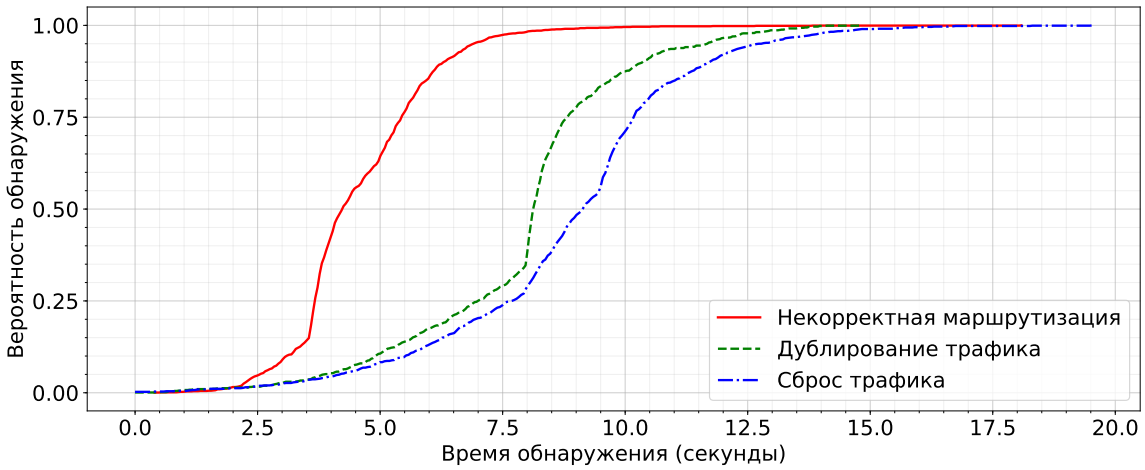


Рисунок 6.5: Функция распределения вероятности обнаружения

В таблице 6.1 показаны ошибки первого и второго рода при обнаружении скомпрометированных коммутаторов. Ошибки второго рода при отсутствии атаки не имеют смысла ($\mathbf{n/a}$), так как ошибка второго рода описывает ситуации, когда скомпрометированный коммутатор был классифицирован как легитимный. Результаты экспериментов показывают, что ошибки первого рода происходят в 3.7% случаев, а ошибки второго рода — в 6.6% случаев.

Меньшее количество ошибок при некорректной маршрутизации объясняется тем, что эта атака затрагивает как минимум 2 доменных пути — новый маршрут трафика и старый маршрут, по которому должен был идти трафик, таким образом, повышая вероятность нахождения некорректного значения счетчика.

Также необходимо отметить, что более 90% от всех ошибок возникало на больших топологиях — более 240 коммутаторов.

На рисунке 6.5 показано время обнаружения скомпрометированных коммутаторов в зависимости от типа атаки и объемов потоков данных. Графики представляют собой функции распределения, где случайной величиной является время обнаружения скомпрометированного коммутатора. Необходимо

отметить, что в построении графика участвовали только данные, на которых скомпрометированный коммутатор был обнаружен корректно. Интервал проверки в экспериментах был выбран равным 10 мс — так как меньший интервал может привести к перегрузкам управляющего канала [81].

Результаты экспериментов показывают, что с вероятностью 90% некорректная маршрутизация будет обнаружена менее чем за 6.5 секунд после начала атаки, дублирование трафика менее чем за 10.5 секунд и сброс трафика менее чем за 11.9 секунд.

6.3. Выводы

В главе описана реализация алгоритма обнаружения скомпрометированных коммутаторов в ПКС и описаны результаты экспериментального исследования.

Эксперименты на топологиях размером до 300 коммутаторов и интенсивности потоков до 1 Гбит/с показывают, что предложенный алгоритм способен обнаружить скомпрометированный коммутатор менее, чем через 15 секунд после начала атаки, при этом ошибки первого и второго рода составляют 3.7% и 6.6% соответственно. Также негативное влияние на сеть, представляемое задержками реакции контроллера на изменение состояния сети, не превышает 420 мс.

Заключение

В диссертационной работе получены следующие основные результаты:

1. Впервые разработана математическая модель, описывающая динамику изменения счетчиков правил маршрутизации в OpenFlow коммутаторах в ПКС, которая инвариантна к набору правил маршрутизации, установленных в OpenFlow коммутаторах, и логике работы приложений контроллера в ПКС.

2. В рамках разработанной модели построен алгоритм предсказания значений счетчиков правил маршрутизации, корректность которого доказана.

3. На основе алгоритма предсказания значений счетчиков построен алгоритм обнаружения скомпрометированных коммутаторов, для которого экспериментально получены оценки ошибок первого/второго рода и время обнаружения скомпрометированных коммутаторов на топологиях, используемых в сетях операторов связи и центров обработки данных. Показано, что представленный алгоритм обнаружения превосходит известные алгоритмы обнаружения, используемые в существующих системах обнаружения, по ряду практически важных критериев.

Список литературы

1. Software-defined networking: A comprehensive survey / Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo et al. // Proceedings of the IEEE. — 2015. — Vol. 103, no. 1. — P. 14–76.
2. Openflow: enabling innovation in campus networks / Nick McKeown, Tom Anderson, Hari Balakrishnan et al. // ACM SIGCOMM Computer Communication Review. — 2008. — Vol. 38, no. 2. — P. 69–74.
3. Foundation O. N. Openflow switch specification v1.5.1. — 2015.
4. Securing data planes in software-defined networks / Tzu-Wei Chao, Yu-Ming Ke, Bo-Han Chen et al. // NetSoft Conference and Workshops (NetSoft), 2016 IEEE / IEEE. — 2016. — P. 465–470.
5. Scott-Hayward S., O'Callaghan G., Sezer S. Sdn security: A survey // Future Networks and Services (SDN4FNS), 2013 IEEE SDN For / IEEE. — 2013. — P. 1–7.
6. Kloti R., Kotronis V., Smith P. Openflow: A security analysis // Network Protocols (ICNP), 2013 21st IEEE International Conference on / IEEE. — 2013. — P. 1–6.
7. Desmedt Y. Man-in-the-middle attack // Encyclopedia of cryptography and security. — Springer, 2011. — P. 759–759.
8. Analysis of operating system diversity for intrusion tolerance / Miguel Garcia, Alysson Bessani, Ilir Gashi et al. // Software: Practice and Experience. — 2014. — Vol. 44, no. 6. — P. 735–770.
9. Reins to the cloud: Compromising cloud systems via the data plane / Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig et al. // arXiv preprint arXiv:1610.08717. — 2016.
10. Neti S., Somayaji A., Locasto M. E. Software diversity: Security, entropy

and game theory. // HotSec. — 2012.

11. Turner S. Transport layer security // IEEE Internet Computing. — 2014. — Vol. 18, no. 6. — P. 60–63.
12. Summarizing known attacks on transport layer security (tls) and datagram tls (dtls) : Rep. ; Executor: Yaron Sheffer, Ralph Holz, Peter Saint-Andre : 2015.
13. Meyer C., Schwenk J. Lessons learned from previous ssl/tls attacks-a brief chronology of attacks and weaknesses. // IACR Cryptology ePrint Archive. — 2013. — Vol. 2013. — P. 49.
14. The matter of heartbleed / Zakir Durumeric, Frank Li, James Kasten et al. // Proceedings of the 2014 conference on internet measurement conference / ACM. — 2014. — P. 475–488.
15. Towards fine-grained network security forensics and diagnosis in the sdn era / Haopei Wang, Guangliang Yang, Phakpoom Chinprutthiwong et al. // Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security / ACM. — 2018. — P. 3–16.
16. Rosemary: A robust, secure, and high-performance network operating system / Seungwon Shin, Yongjoo Song, Taekyung Lee et al. // Proceedings of the 2014 ACM SIGSAC conference on computer and communications security / ACM. — 2014. — P. 78–89.
17. Myerson J. M. Identifying enterprise network vulnerabilities // International Journal of Network Management. — 2002. — Vol. 12, no. 3. — P. 135–144.
18. Network vulnerability analysis / Blackburn Skaggs, B Blackburn, G Manes, S Shenoi // Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium on / IEEE. — Vol. 3. — 2002. — P. III–493.
19. The design and implementation of open vswitch. / Ben Pfaff, Justin Pettit,

Teemu Koponen et al. // NSDI. — Vol. 15. — 2015. — P. 117–130.

20. Davie B. S., Rekhter Y. MPLS: technology and applications. — Morgan Kaufmann Publishers Inc., 2000.
21. Pang C., Jiang Y., Li Q. Fade: Detecting forwarding anomaly in software-defined networks // Communications (ICC), 2016 IEEE International Conference on / IEEE. — 2016. — P. 1–6.
22. Poisoning network visibility in software-defined networks: New attacks and countermeasures. / Sungmin Hong, Lei Xu, Haopei Wang, Guofei Gu // NDSS. — Vol. 15. — 2015. — P. 8–11.
23. Sphinx: Detecting security attacks in software-defined networks. / Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, Vijay Mann // NDSS. — Vol. 15. — 2015. — P. 8–11.
24. B4: Experience with a globally-deployed software defined wan / Sushant Jain, Alok Kumar, Subhasree Mandal et al. // ACM SIGCOMM Computer Communication Review / ACM. — Vol. 43. — 2013. — P. 3–14.
25. Способ минимизации многоадресного трафика и обеспечение его отказоустойчивости в ПКС сетях : Патент 2676239 МПК H04L 12/869 (2013.01) / Петров И.С., Шалимов А.В., Смелянский Р.Л. (RU) ; патентообладатель Некоммерческое партнерство «Центр прикладных исследований компьютерных сетей» ; — № 2017122409 ; опубл. 26.12.2018, Бюл. № 36 ; приоритет 11.09.2017.
26. Петров И.С. Задача обнаружения скомпрометированных коммутаторов в SDN сетях // REDS: Телекоммуникационные устройства и системы. — 2017. — Vol. 7, no. 4. — P. 515–518.
27. Петров И.С., Смелянский Р.Л. Обнаружение скомпрометированных коммутаторов в SDN сетях // Ломоносовские чтения. Тезисы докладов научной конференции. — 2017. — P. 82–83.

28. Шемякин Р.О., Петров И.С. Обеспечение контроля доступа приложений к ресурсам контроллера программно конфигурируемых сетей // Программные системы и инструменты. Тематический сборник. Под общей редакцией Р.Л. Смелянского. — 2017. — Vol. 17. — P. 61–72.
29. Шендяпин А.С., Петров И.С. Исследование методов проведения атаки Man-in-the-Middle в программно-конфигурируемых сетях // Программные системы и инструменты. Тематический сборник. Под общей редакцией Р.Л. Смелянского. — 2017. — Vol. 17. — P. 73–82.
30. Петров И.С., Смелянский Р.Л. Алгоритм обнаружения скомпрометированных коммутаторов в SDN // Ломоносовские чтения 2018 ф-т ВМК МГУ. — 2018. — P. 98–99.
31. Петров И.С., Смелянский Р.Л. Минимизация группового трафика и обеспечение его отказоустойчивости в программно-конфигурируемых сетях // Известия Российской академии наук. Теория и системы управления. — 2018. — no. 3. — P. 64–75.
32. Petrov I., Smeliansky R. Minimization of multicast traffic and ensuring its fault tolerance in software-defined networks // Journal of Computer and Systems Sciences International. — 2018. — Vol. 57, no. 3. — P. 407–419.
33. Petrov I. Mathematical model for predicting forwarding rule counter values in sdn // Young Researchers in Electrical and Electronic Engineering (EIConRus), 2018 IEEE Conference of Russian / IEEE. — 2018. — P. 1313–1317.
34. Petrov I., Morgunova O. Forwarding rule minimization for network statistics analysis in sdn // 2018 International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTeC) / IEEE. — 2018. — P. 1–6.
35. Петров И.С., Шендяпин А.С. Обзор средств обеспечения анонимности в

SDN // Программные системы и инструменты. Тематический сборник. Под ред. В. В. Балашов, Е. И. Большакова, Д. Ю. Волканов и др. — 2018. — Vol. 18. — P. 78–88.

36. Петров И.С. Системы обнаружения скомпрометированных коммутаторов в программно-конфигурируемых сетях // Информационные технологии. — 2019. — Vol. 25, no. 3. — P. 131–142.
37. Петров И.С. Алгоритм минимизации количества правил маршрутизации в ПКС // Моделирование и анализ информационных систем. — 2019. — Vol. 26, no. 1. — P. 122–133.
38. Implementing an openflow switch on the netfpga platform / Jad Naous, David Erickson, G Adam Covington et al. // Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems / ACM. — 2008. — P. 1–9.
39. Mirkovic J., Reiher P. A taxonomy of ddos attack and ddos defense mechanisms // ACM SIGCOMM Computer Communication Review. — 2004. — Vol. 34, no. 2. — P. 39–53.
40. Towards an elastic distributed sdn controller / Advait Dixit, Fang Hao, Sarit Mukherjee et al. // ACM SIGCOMM computer communication review / ACM. — Vol. 43. — 2013. — P. 7–12.
41. Ethernet address resolution protocol: Or converting network protocol addresses to 48. bit ethernet address for transmission on ethernet hardware : Rep. ; Executor: David Plummer : 1982.
42. Congdon P. Link layer discovery protocol // RFC. — 2002. — Vol. 2002.
43. A standard for the transmission of ip datagrams over ethernet networks : Rep. ; Executor: Charles Hornig : 1984.
44. Xiao X., Ni L. M. Internet qos: a big picture // IEEE network. — 1999. — Vol. 13, no. 2. — P. 8–18.

45. In-band network telemetry via programmable dataplanes / Changhoon Kim, Anirudh Sivaraman, Naga Katta et al. // ACM SIGCOMM. — 2015.
46. Intrusion detection systems: A survey and taxonomy : Rep. / Technical report ; Executor: Stefan Axelsson : 2000.
47. Automatic test packet generation / Hongyi Zeng, Peyman Kazemian, George Varghese, Nick McKeown // Proceedings of the 8th international conference on Emerging networking experiments and technologies / ACM. — 2012. — P. 241–252.
48. Garey M. R., Johnson D. S. Computers and intractability. — wh freeman New York, 2002. — Vol. 29.
49. Introduction to algorithms / Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein. — MIT press, 2009.
50. Ieee 802.1 q / Patricia Thaler, Norman Finn, Don Fedyk et al. — 2013.
51. Kamisiński A., Fung C. Flowmon: Detecting malicious switches in software-defined networks // Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense / ACM. — 2015. — P. 39–45.
52. How to detect a compromised sdn switch / Po-Wen Chi, Chien-Ting Kuo, Jing-Wei Guo, Chin-Laung Lei // Network Softwarization (NetSoft), 2015 1st IEEE Conference on / IEEE. — 2015. — P. 1–6.
53. Dilworth R. P. A decomposition theorem for partially ordered sets // Annals of Mathematics. — 1950. — P. 161–166.
54. Hopcroft J. E., Karp R. M. An n^5/2 algorithm for maximum matchings in bipartite graphs // SIAM Journal on computing. — 1973. — Vol. 2, no. 4. — P. 225–231.
55. Mohammadi A. A., Kazemian P., Pakravan M. R. Detecting malicious packet drops and misroutings using header space analysis // Telecommu-

nications (IST), 2016 8th International Symposium on / IEEE. — 2016. — P. 521–526.

56. Duffield N. G., Grossglauser M. Trajectory sampling for direct traffic observation // IEEE/ACM Transactions on Networking (ToN). — 2001. — Vol. 9, no. 3. — P. 280–292.
57. Kazemian P., Varghese G., McKeown N. Header space analysis: Static checking for networks. // NSDI. — Vol. 12. — 2012. — P. 113–126.
58. Chiu Y.-C., Lin P.-C. Rapid detection of disobedient forwarding on compromised openflow switches // Computing, Networking and Communications (ICNC), 2017 International Conference on / IEEE. — 2017. — P. 672–677.
59. Shaghaghi A., Kaafar M. A., Jha S. Wedgetail: An intrusion prevention system for the data plane of software defined networks // Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security / ACM. — 2017. — P. 849–861.
60. I know what your packet did last hop: Using packet histories to troubleshoot networks. / Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar et al. // NSDI. — Vol. 14. — 2014. — P. 71–85.
61. Dynamic packet forwarding verification in sdn / Qi Li, Xiaoyue Zou, Qun Huang et al. // IEEE Transactions on Dependable and Secure Computing. — 2018.
62. Tanenbaum A. S., Wetherall D. Computer networks. — Prentice hall, 1996.
63. Tootoonchian A., Ghobadi M., Ganjali Y. Opentm: traffic matrix estimator for openflow networks // International Conference on Passive and Active Network Measurement / Springer. — 2010. — P. 201–210.
64. Splendid isolation: A slice abstraction for software-defined networks / Stephen Gutz, Alec Story, Cole Schlesinger, Nate Foster // Proceedings of the first workshop on Hot topics in software defined networks / ACM. —

2012. — P. 79–84.

65. Veriflow: Verifying network-wide invariants in real time / Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, P Godfrey // Proceedings of the first workshop on Hot topics in software defined networks / ACM. — 2012. — P. 49–54.
66. Zakharov V. A., Smelyansky R., Chemeritsky E. V. A formal model and verification problems for software defined networks // Automatic Control and Computer Sciences. — 2014. — Vol. 48, no. 7. — P. 398–406.
67. Yang H., Lam S. S. Real-time verification of network properties using atomic predicates // IEEE/ACM Transactions on Networking. — 2016. — Vol. 24, no. 2. — P. 887–900.
68. Real time network policy checking using header space analysis. / Peyman Kazemian, Michael Chan, Hongyi Zeng et al. // NSDI. — 2013. — P. 99–111.
69. Ford Jr L. R., Fulkerson D. R. Flows in networks. — Princeton university press, 2015.
70. Jech T. Set theory. — Springer Science & Business Media, 2013.
71. The broadcast storm problem in a mobile ad hoc network / Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, Jang-Ping Sheu // Wireless networks. — 2002. — Vol. 8, no. 2-3. — P. 153–167.
72. Detection and analysis of routing loops in packet traces / Urs Hengartner, Sue Moon, Richard Mortier, Christophe Diot // Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment / ACM. — 2002. — P. 107–112.
73. Diestel R. Graph theory. — Springer Publishing Company, Incorporated, 2018.
74. Stroustrup B. The C++ programming language. — Pearson Education In-

dia, 2000.

75. Van Rossum G., Drake F. L. The python language reference manual. — Network Theory Ltd., 2011.
76. Lantz B., Heller B., McKeown N. A network in a laptop: rapid prototyping for software-defined networks // Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks / ACM. — 2010. — P. 19.
77. The internet topology zoo / Simon Knight, Hung X Nguyen, Nick Falkner et al. // IEEE Journal on Selected Areas in Communications. — 2011. — Vol. 29, no. 9. — P. 1765–1775.
78. Al-Fares M., Loukissas A., Vahdat A. A scalable, commodity data center network architecture // ACM SIGCOMM Computer Communication Review / ACM. — Vol. 38. — 2008. — P. 63–74.
79. 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 / ICST (Institute for Computer Sciences, Social-Informatics and .... — 2013. — P. 82–91.
80. The runos openflow controller / Alexander Shalimov, Sergey Nizovtsev, Danila Morkovnik, Ruslan Smeliansky // Software Defined Networks (EWSDN), 2015 Fourth European Workshop on / IEEE. — 2015. — P. 103–104.
81. Payless: A low cost network monitoring framework for software defined networks / Shihabur Rahman Chowdhury, Md Faizul Bari, Reaz Ahmed, Raouf Boutaba // Network Operations and Management Symposium (NOMS), 2014 IEEE / IEEE. — 2014. — P. 1–9.
82. Vidal A., Rothenberg C. E., Verdi F. L. The libfluid openflow driver implementation // Proceedings of SBRC. — 2014.
83. Sarnak N. I. Persistent data structures. — 1986.

Приложение А

Компоненты системы

Данный раздел подробно описывает компоненты разработанной системы обнаружения.

А.1. Proxy

Основные функции модуля *Proxy* включают:

1. Перехват OpenFlow сообщений.
2. Установку дополнительных правил маршрутизации.
3. Опрос статистики с коммутаторов.

Proxy реализован с использованием библиотеки *libfluid* [82]. Эта библиотека предоставляет базовые функции для работы с протоколом OpenFlow. В базовые функции входит обработка сообщений протокола и установка соединений с коммутаторами. В эту библиотеку также была добавлена логика установки соединений с контроллером.

Вся информация о сети и правилах маршрутизации определяется системой из OpenFlow сообщений, перехватываемых модулем *Proxy*. Сбор информации о сети реализован на основе анализа сообщений протокола OpenFlow и, следовательно, не зависит от конкретной реализации контроллера и его приложений.

A.2. Handler

Модуль *Handler* отвечает за обработку OpenFlow сообщений, перехваченных модулем *Proxy*. Под обработкой понимается следующий набор действий:

1. Преобразование полей *match*.
2. Модификация устанавливаемых правил
3. Очередизация сообщений от контроллера

Преобразование полей *match*

Модуль *Handler* преобразует информацию, полученную из OpenFlow сообщений, в объекты модели *Header Space*. Поля *match* правил маршрутизации преобразуются в *wildcard* маски, а поля *action* в трансферные функции.

Преобразования также производятся в обратную сторону для создания дополнительных правил маршрутизации.

Модификация устанавливаемых правил

Модуль *Handler* сдвигает все правила, устанавливаемые контроллером в сеть, на одну таблицу вперед для того, чтобы использовать нулевые таблицы коммутаторов для установки дополнительных правил маршрутизации.

В сообщениях, идущих от коммутаторов к контроллеру, восстанавливаются оригинальные номера таблиц маршрутизации для того, чтобы не нарушать работу приложений контроллера.

Очередизация сообщений от контроллера

OpenFlow сообщения, которые описывают установку правил маршрутизации, изменяют набор ребер в графе зависимостей правил. При изменении графа зависимостей создаются новые ветви деревьев путевой развертки, и, следовательно, новые дополнительные правила маршрутизации. Таким образом, при обработке правила R создается список дополнительных правил $\overline{R}_1, \ldots, \overline{R}_n$. Эти дополнительные правила затем будут использоваться системой обнаружения для предсказания значений счетчиков правила маршрутизации R.

Если правило R будет установлено в сеть раньше дополнительных правил, то оно успеет обработать некоторое количество пакетов до того, как установятся дополнительные правила. Следовательно, правила $\overline{R}_1, \ldots, \overline{R}_n$ обработают меньшее количество пакетов, чем правило R, и во время предсказания значений счетчика правила R, произойдет расхождение реального значения счетчика и предсказанного. Для предотвращения подобной ситуации, дополнительные правила маршрутизации $\overline{R}_1, \ldots, \overline{R}_n$ устанавливаются одновременно с правилом R.

Для решения проблемы расхождения значений счетчиков модулем *Handler* поддерживается очередь сообщений от контроллера. В этой очереди накапливаются правила, устанавливаемые контроллером. Сообщения из очереди отправляются в сеть только тогда, когда будут расчитаны соответствующие им дополнительные правила маршрутизации. Таким образом, дополнительные правила будут установлены на коммутаторы одновременно с оригинальными.

Необходимо отметить, что из-за асинхронности установки правил маршрутизации на разные коммутаторы, эти правила могут установиться не одновременно. В данном случае расхождение значений счетчиков будет меньше,

так как оно зависит от скорости установки правил на коммутаторы, а не от времени построения путевой развертки.

Для учета таких расхождений используется интервал легитимных отклонений, полученный экспериментальным путем.

A.3. Network

Модуль *Network* хранит текущее состояние сети и предоставляет другим модулям *API* к этому состоянию. Состояние сети описывается при помощи:

1. Списка коммутаторов.
2. Списка портов.
3. Списка таблиц маршрутизации.
4. Списка правил маршрутизации.
5. Топологии сети.

A.4. Dependency Graph

Основные функции модуля *Dependency Graph* включают:

1. Построение графа зависимостей правил.
2. Предоставление интерфейса к графу для других приложений.

Построение графа

Модуль *Dependency Graph* строит граф зависимостей правил на основе OpenFlow сообщений, перехваченных модулем Proxy. Граф зависимостей правил изменяется при получении следующих OpenFlow сообщений:

1. FeaturesReply
2. FlowMod
3. GroupMod

Сообщение ***FeaturesReply*** отправляется коммутатором на контроллер и содержит информацию о количестве портов и таблиц коммутатора. Эта информация необходима для создания стоковых и истоковых вершин графа, соответствующих портам коммутатора, а также вершин, соответствующих правилам по-умолчанию в таблицах маршрутизации.

Стоковые вершины описывают входы в сеть, истоковые вершины — выходы. Правила по-умолчанию, это правила с наименьшим приоритетом, которые могут обрабатывать пакеты с любыми заголовками, не обработанные другими правилами маршрутизации. Согласно протоколу OpenFlow [3] такие правила находятся в каждой таблице маршрутизации.

Сообщение ***FlowMod*** предназначено для установки и удаления правил маршрутизации. При установке правила, в сообщении указано единственное правило, которое необходимо установить в таблицу маршрутизации коммутатора. При удалении правила, в сообщении указывается набор параметров, включающий поле *match*. Коммутатор удаляет все правила маршрутизации, которые описываются этими параметрами, причем удаляются правила, поля *match* которых являются подмножеством поля, указанного в сообщении.

Таким образом, каждое сообщение *FlowMod* приводит либо к созданию единственной вершины графа, либо к удалению набора вершин.

Сообщение ***GroupMod*** предназначено для установки и удаления групповых правил. Групповые правила протокола OpenFlow отвечают за групповую

маршрутизацию и балансировку нагрузки. Сообщение *GroupMod* приводит к установке или удалению вершин графа зависимостей правил, соответствующих групповому правилу и набору его *buckets*-записей.

Также на изменение графа влияет обнаружение физических линий, проложенных между коммутаторами. Обнаружение таких линий происходит при помощи сообщений протокола LLDP, отправляемых коммутаторами на контроллер при помощи *Packet-In* сообщений.

Интерфейс к графу

Модуль *Dependency Graph* также предоставляет другим модулям интерфейс к графу зависимостей правил. Под интерфейсом понимется доступ к вершинам и ребрам графа и возможности прохода по путям графа, с определением заголовков пакетов, которые могут пройти по этому пути.

Интерфейс к графу используется при получении сообщения *Packet-Out* от контроллера. Эти OpenFlow сообщения указывают коммутатору, что ему нужно сгенерировать пакет на некотором порту или в некоторой потоковой таблице. Так как сгенерированные пакеты повлияют на значения счетчиков правил маршрутизации, они учитываются при анализе сетевой статистики (рис. A.1).

Учет сгенерированных пакетов происходит следующим образом:

1. Используя заголовок пакета, находится путь, состоящий из правил маршрутизации, который пройдет этот пакет.

2. Значения потоков всех вершин в найденом пути увеличиваются на 1.

Также, при изменении графа зависимостей правил, модуль *Dependency Graph* накапливает список изменений. Список изменений включает набор добавленных и удаленных ребер графа. Этот список необходим для того, чтобы

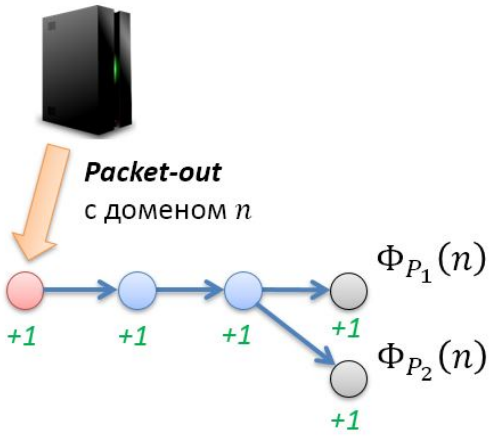


Рис. А.1: Учет сообщений *Packet-Out*

изменять путевую развертку только по накопленному набору изменений графа. Накопление изменений графа позволяет избежать ситуаций, когда удаление и добавление одних и тех же ребер графа приводят к удалению и добавлению одинаковых поддеревьев в путевой развертке.

A.5. Flow Predictor

Основные функции модуля *Flow Predictor* включают:

- Построение путевой развертки.
- Предсказание значений счетчиков правил маршрутизации.
- Установку дополнительных правил маршрутизации.
- Создание запросов статистики с коммутаторов.

Путевая развертка строится на основе списка изменений графа зависимостей правил, полученного из модуля *Dependency Graph*. После каждого обновления путевой развертки, создается список изменений для дополнительных правил маршрутизации, установленных в сети. Список изменений пред-

ставляет собой список с удаляемыми и устанавливаемыми правилами маршрутизации.

Удаление правил

При удалении старых правил учитываются ситуации, когда эти правила могли успеть обработать некоторое количество пакетов. То есть значения счетчиков этих правил маршрутизации не равны нулю. В данном случае значения счетчиков сохраняются, так как после удаления правил эти значения станут недоступны.

Между отправкой запроса статистики и получением ответа может пройти время, за которое путевая развертка может измениться. До того момента, как придут ответы статистики по удаляемым правилам, путевая развертка хранит удаленные ветви и, таким образом, представляет собой *персистентную структуру данных* [83]. Персистентная структура данных — это структура, которая хранит предыдущие состояния и предоставляет к ним доступ. В данном случае, предыдущими состояниями являются ветви деревьев путевой развертки, соответствующие удаленным доменным путям.

Старые ветви деревьев путевой развертки помечаются флагом *deleted*, но при этом остаются в дереве. Они не учитываются при построении новых поддеревьев и при вычислении полей *match* новых правил маршрутизации. Эти ветви используются только для обработки ответов статистики. Старые ветви удаляются из путевой развертки только после того, как придет последний ответ статистики по удаляемым правилам.

Обновление путевой развертки

Также необходимо отметить, что если путевая развертка будет перестраиваться после каждого изменения графа зависимостей правил, может возник-

нуть ситуация, когда в сети будут устанавливаться и удаляться одни и те же правила маршрутизации.

Такая ситуация может возникнуть, когда контроллер устанавливает путь между двумя хостами в сети. При установке путей контроллер одновременно устанавливает набор правил, отправляющих пакеты с коммутатора i на коммутатор $i+1$ при $i \in [1, N-1]$. Эти правила устанавливаются асинхронно, поэтому может возникнуть ситуация, когда они установятся в порядке от первого коммутатора к последнему.

В этой ситуации, в ответ на каждое изменение графа зависимостей правил будет строиться новый доменный путь от клиента к коммутатору $i+1$ и удаляться старый путь к коммутатору i. При этом домены путей к коммутатору i и к коммутатору $i+1$ будут одинаковыми. А значит и правила, соответствующие этим доменам будут одинаковые. Таким образом, одни и те же дополнительные правила маршрутизации будут устанавливаться и удаляться N раз.

Так как заранее не известно, какие правила будут устанавливаться в сеть, разработанная система в течение некоторого времени накапливает изменения графа зависимостей правил и только потом перестраивает путевую развертку. Для этого вводится интервал времени, во время которого накапливаются добавленные и удаленные ребра графа. По истечении этого интервала вызывается функция, которая использует накопленные изменения для построения путевой развертки и устанавливает в сеть дополнительные правила маршрутизации. Размер интервала может настраиваться в зависимости от размера сети.

A.6. Detector

Модуль *Detector* производит обнаружение скомпрометированных коммутаторов при помощи информации, предоставляемой модулями *Dependency Graph* и *Flow Predictor*.

Модуль *Detector* использует предсказания значений счетчиков правил маршрутизации, полученные из модуля *Flow Predictor*, и сравнивает эти значения с реальными значениями, запрошенными у коммутаторов.

Если эти значения отличаются, то модуль *Detector* начинает процедуру изменения уровней доверия к коммутаторам. Процедура заключается в следующем.

1. Модуль *Detector* использует модуль *Dependency Graph* для того, чтобы найти все пути в графе, проходящие через вершину, соответствующую анализируемому в данный момент правилу маршрутизации.

2. Эти пути используются для обхода всех коммутаторов, которые могли повлиять на потоки, проходящие через анализируемое правило.

3. При обходе модуль *Detector* понижает уровень доверия всех пройденных коммутаторов.

4. Если уровень доверия некоторого коммутатора опускается ниже заданного порога, то модуль сообщает о том, что этот коммутатор скомпрометирован.

Модуль *Detector* также учитывает легитимные потери пакетов, происходящие вследствие перегрузок в сети. По протоколу OpenFlow, количество всех потерянных пакетов записано в счетчиках на портах коммутаторов. Модуль *Detector* использует эти счетчики для того, чтобы проверить, что сумма

нехватки пакетов по путям, проходящим через некоторый порт, равна количеству пакетов, потерянных на этом порту. В данном случае, различие счетчиков будет считаться легитимным.