Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой

_____ О.В. Непомнящий
подпись инициалы, фамилия
«___» _____20___ г.

МАГИСТЕРСЕКАЯ ДИССЕРТАЦИЯ

Методы управления качеством обслуживания корпоративных сетей
тема

09.04.01 Информатика и вычислительная техника
код и наименование направления

09.04.01.05 Сети ЭВМ и телекоммуникации
код и наименование магистерской программы

Научный руководитель	_____ подпись, дата	доцент, канд. физ.-мат. наук должность, ученая степень	К.В. Коршун инициалы, фамилия
Выпускник	_____ подпись, дата		А.О. Табаков инициалы, фамилия
Рецензент	_____ подпись, дата	генеральный директор ООО «Интертакс» должность, ученая степень	М.В. Алексеев инициалы, фамилия
Нормоконтролер	_____ подпись, дата	доцент, канд. физ.-мат. наук должность, ученая степень	К.В. Коршун инициалы, фамилия

Красноярск 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Корпоративные сети все чаще становятся сдерживающим фактором развития вычислительной инфраструктуры предприятия. Причинами являются возрастающий объем трафика и его разнообразие. В сети передается разнородный трафик, включающий как обычные текстовые данные, так и голосовые, видео данные, требующие особого обслуживания. Проблему усугубляют переход пользователей на дистанционную форму занятости и использование мобильных устройств в своей работе.

В этой ситуации актуальной задачей является поиск возможных способов повышения качества работы сети за счет внутренних резервов. Одним из таких резервов является оптимизация управления трафиком, которая может обеспечить эффективную работу разнородных систем. Эффективность использования ресурсов канала пакетной передачи данных всегда была актуальной задачей, но ее важность возросла в последние годы в связи с появлением все более жестких требований к качеству обслуживания разнородного трафика, в особенности IP-телефонии и видеосвязи. Негативные последствия неудовлетворительного качества обслуживания могут привести к финансовым проблемам, некачественной коммуникации и увеличению задержек в производстве. Поэтому возникла необходимость гарантировать время реакции, пропускную способность сети и другие параметры. Такая технология была разработана и получила название «Качество обслуживания» (Quality of Service, QoS). Качество обслуживания использует распределение по категориям и назначение приоритетов трафикам, что позволяет гарантировать данным с большим приоритетом лучшие условия передачи через сетевую магистраль, вне зависимости от требований к пропускной способности потоков менее важных приложений [1].

Существуют различные методы управления технологией QoS. В данной работе будут рассмотрены такие методы, как ручная и автоматизированная

настройки, программно-определяемые сети (англ. software-defined networking, SDN).

Целью выпускной квалификационной работы является создание нового метода управления качеством обслуживания корпоративных сетей. В результате его применения будут решены проблемы, возникающие при использовании существующих методов, такие как трудоемкость и ошибки конфигурации из-за человеческого фактора, значительные финансовые расходы по внедрению SDN и автоматизированных систем, невозможность перенастройки QoS при изменении топологии сети у автоматизированных систем. Данные проблемы будут решены за счет использования созданного в ходе работы программного обеспечения, функционал которого позволит автоматизировать и оптимизировать процесс настройки сетевого оборудования.

Для достижения указанной цели были поставлены следующие задачи:

1. раскрыть актуальность исследуемой темы;
2. рассмотреть существующие методы и модели управления качеством обслуживания сети и выявить их недостатки;
3. предложить и описать новый метод управления качеством обслуживания сети;
4. предложить возможные направления развития работы в дальнейшем.

После проверок на работоспособность и анализа результатов разработанного ПО планируется выявить достоинства и недостатки предложенного метода по автоматизации настройки качества обслуживания корпоративной сети.

Глава 1. Обзор предметной области

1.1 Метрики качества обслуживания

Многие современные приложения требуют высокое качество сети, которое представляет собой совокупность показателей, называемых, метриками сети. К основным метрикам качества сети относятся:

1. потери пакетов;
2. задержки;
3. джиттер.

Потери пакетов говорят о том, сколько из отправленных источником пакетов дошло до адресата [2]. Как правило, как справляется с потерями зависит от приложения. В случае с веб-приложениями, использующих протокол TCP, пакет отправляется заново, а в случае телефонного разговора (протокол UDP) — пакет отбрасывается.

Задержка — это время, которое необходимо данным, чтобы добраться от источника до получателя [2]. Задержка создает неудобство при ведении диалога, приводит к перекрытию разговоров и возникновению эхо [3].

Чаще всего для оценки занятости канала используют круговую задержку RTT (Round Trip Time). Это интервал времени между отправкой пакета и окончанием его обработки на принимающей стороне. В основном данную метрику отслеживают с помощью ICMP протокола (Internet Control Message Protocol).

Колебание задержек при передаче последовательных пакетов называется джиттером. Опять же зависит от приложения существенность данной метрики. Большинству приложений достаточно, чтобы пакет был доставлен и неважно была ли там задержка. Но для той же IP-телефонии это играет важную роль. Джиттер приводит к специфическим нарушениям передачи речи, слышимым как трески и щелчки [3].

1.2 Качество обслуживания

Для повышения производительности сети без роста пропускной способности транспортной сети была создана технология «Качество обслуживания» (Quality of Service, QoS). Качество обслуживания использует распределение по категориям и назначение приоритетов трафикам, что позволяет гарантировать данным с большим приоритетом лучшие условия передачи через сеть, вне зависимости от требований к пропускной способности потоков менее важных приложений [1].

Существует три модели обеспечения QoS:

1. Best Effort: не гарантирует качества передачи трафика, все потоки равны;
2. IntServ: гарантирует качество для каждого потока, заблаговременно резервирует ресурсы от источника до получателя;
3. DiffServ: за определение качества отвечает каждый узел, по которым идёт передача, отсутствует резервирование.

В настоящее время в сетях чаще всего используются механизмы DiffServ. так как механизмы IntServ требуют огромных ресурсов от вычислительных процессоров предающих устройств, в связи с постоянным заблаговременным резервированием ресурсов.

Модель дифференцированного обслуживания определяет обеспечение QoS на основе четко определенных компонентов, комбинируемых с целью предоставления требуемых услуг. Архитектура DiffServ предполагает наличие классификаторов и формирователей трафика на границе сети, а также поддержку функции распределения ресурсов в ядре сети в целях обеспечения требуемой политики. Разделяет трафик на классы, вводя несколько уровней QoS [4].

Данная модель управления трафиком гарантирует, что трафик большую часть времени будет получать адекватное обслуживание, возможно, с некоторой степенью дифференциации. В случае перегрузки потоки будут

адаптировать свой трафик к имеющимся ресурсам и продолжать обслуживаться, хотя, может быть, с более низким качеством. Преимуществом данного решения является более высокая общая эффективность, так как оно позволяет передавать большее число потоков более простым образом, с минимальной поддержкой сигнализации и с помощью простых механизмов организации путей передачи данных [4].

1.3 Методы реализации качества обслуживания

1.3.1 Ручная настройка

Метод ручной настройки качества обслуживания заключается в поэтапной монотонной ручной настройке администратором сети правил обслуживания трафика на каждом сетевом устройстве.

Обычно сетевые администраторы составляют в электронных таблицах правила QoS на основе анализа требований качества обслуживания клиентов. Затем на основе этих правил создают типовые конфигурации для сетевого оборудования. И наконец, данные конфигурации вручную вносятся на требующее настройки сетевое оборудование.

Это длительный и неэффективный метод, сопряженный с ошибками из-за человеческого фактора, который может привести к деградации сервисов и дополнительным финансовым затратам. Но зато он относительно дешев, так как не предполагает покупку, внедрение и эксплуатацию нового дорогостоящего оборудования с поддержкой SDN, либо ПО для автоматизации процесса настройки сетевого оборудования.

1.3.2 SDN

Главная идея программно-определяемой сети (SDN) заключается в отделении функций передачи трафика от функций управления (включая контроль как самого трафика, так и осуществляющих его передачу устройств) [5].

Необходимый анализ пакетов, изменения информации в пакетах и правила пересылки возлагаются на контроллер. Сами устройства следуют инструкциям контроллера, нагрузка на вычислительные ресурсы значительно снижается. Достигается централизация логики управления сетью, что позволяет конфигурировать сеть как единое целое и существенно упрощает эксплуатацию сети.

Открытая архитектура SDN обеспечивает совместимость с несколькими поставщиками. API-интерфейсы поддерживают широкий спектр приложений, включая облачную оркестровку и важные для бизнеса сетевые приложения. Кроме того, интеллектуальное программное обеспечение может управлять оборудованием различных производителей с помощью открытых программных интерфейсов, таких как OpenFlow. Наконец, из SDK интеллектуальные сетевые службы и приложения могут работать в общей программной среде [6].

Архитектура сетей SDN подразделяется на три уровня:

1. уровень приложений;
2. уровень контроля;
3. уровень передачи данных.

Ресурсы уровня передачи данных выполняют сетевые функции, такие как передача и обработка данных, но их поведение управляется уровнем контроля. Взаимодействие контроллера с уровнем передачи данных возможно с помощью интерфейсов и специализированных протоколов, например, Open Flow, обеспечивающих взаимодействие с сетевыми устройствами. На другой стороне контроллер предоставляет стандартизированные программные

интерфейсы API, наличие которых позволяет создавать приложения для управления сетью [5]. Такие приложения могут, например, управлять пропускной способностью, динамически назначать приоритеты различным видам трафика, контролировать доступ к сетевым ресурсам.

Архитектура сети SDN представлена на рисунке 1.

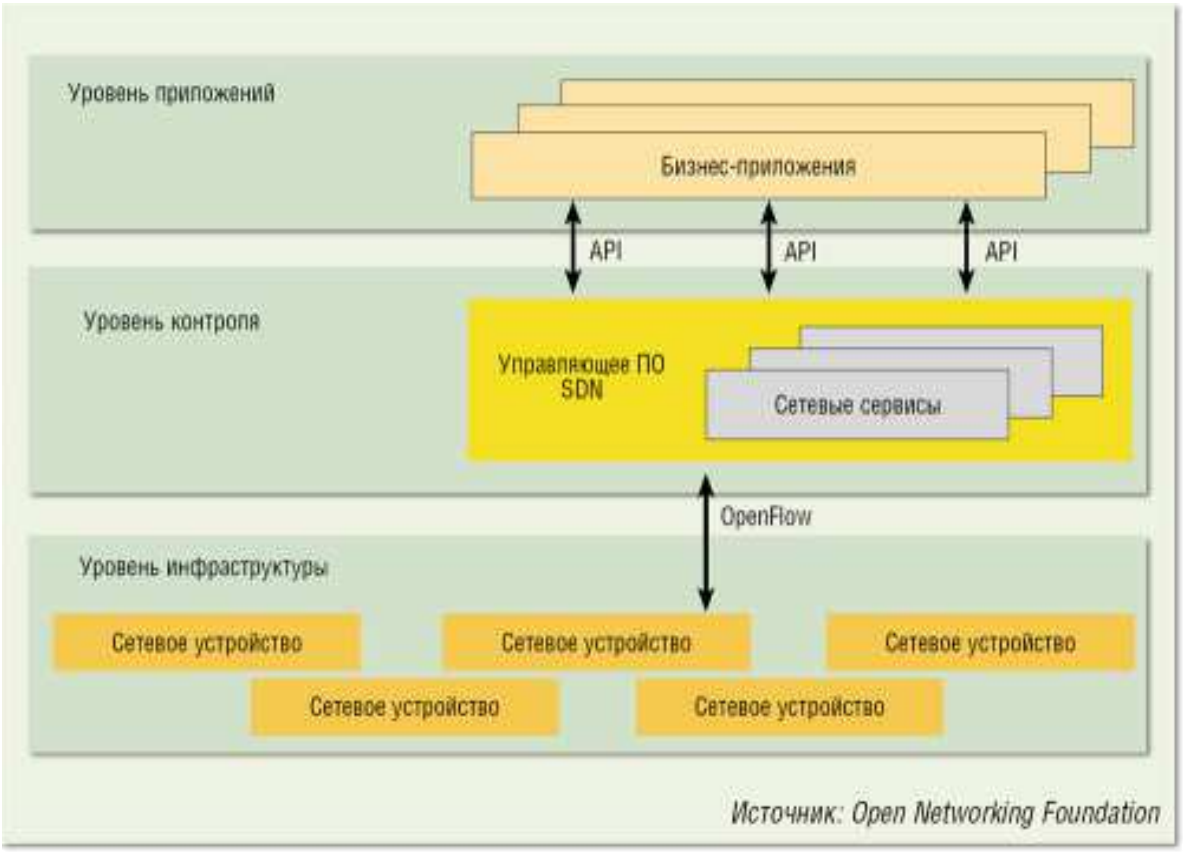

Рисунок 1. Архитектура SDN.

К недостатку концепции относится высокая стоимость реализации. Оборудование с поддержкой данной технологии, полностью управляемое из центрального контроллера, может позволить не каждая компания. К тому же, в настоящее время существует лишь несколько производителей подобного оборудования.

1.3.3 Автоматизированные системы управления качеством обслуживания

Автоматизированные системы управления качеством обслуживания — это программно-аппаратные комплексы, которые централизованно рассчитывают топологию сети, а затем автоматически подключаются к сетевому оборудованию и настраивают его. Основными компонентами являются:

1. управляющая система — представляет собой ЭВМ со специальным программным обеспечением, который автоматизировано настраивает сетевое оборудование;
2. база данных, хранящая информацию о сети (топология, политики и т.д.);
3. управляемое сетевое оборудование, на котором необходимо настроить качество обслуживания.

Рассмотрим автоматизированные системы управления качеством обслуживания на примере AutoQoS от компании Cisco Systems.

AutoQoS позволяет автоматизировать настройку функции QoS, т.е. приоритизации при доставке пакетов в сети. AutoQoS ввело данную функциональность в операционные системы IOS и Catalyst OS и упростило внедрение IP QoS в локальные и глобальные сети. Это дало возможность автоматизировать настройку инфраструктуры IP при внедрении технологии «голос поверх IP» (VoIP) по всей сети — от распределительного шкафа до корпоративной IP-магистрали, а также упростить настройку услуг, предоставляемых сервис-провайдером [8].

Преимуществами автоматизированных систем управления качеством обслуживания являются:

1. автоматизация и оптимизация процесса конфигурирования вычислительной инфраструктуры предприятия;

2. автоматизация процессов классификации трафика, создания политик обработки трафика, настройка конфигурации и других;

3. повышение надежности сети путем уменьшения (в некоторых случаях даже исключения) операторских и конфигурационных ошибок;

4. возможность централизованного управления для мониторинга трафика и получения более подробной информации о состоянии сетевых операций;

5. возможность экономично управлять функциями QoS в крупных IP-сетях, благодаря углубленному анализу, интеллектуальному дизайну QoS и масштабируемому внедрению.

Недостатками автоматизированных систем управления качеством обслуживания являются невозможность перенастройки QoS при изменении топологии сети и дополнительные финансовые расходы для компании при покупке, внедрении и эксплуатации данных систем.

1.4 Выводы по главе 1

По результатам сравнительного анализа и обзора существующих методов управления качеством обслуживания корпоративных сетей можно сделать следующие выводы:

1. ручная настройка качества обслуживания — это трудоемкий и неэффективный метод, сопряженный с ошибками из-за человеческого фактора, который может привести к деградации сервисов и дополнительным финансовым расходам.

2. недостатком SDN является необходимость замены существующего парка сетевого оборудования на новое с поддержкой SDN, что приводит к значительным финансовым расходам предприятия. Также при выходе из строя контроллера сети, перестает функционировать вся сеть

полностью, что вынуждает резервировать контроллер, а это приводит к дополнительным финансовым расходам.

3. недостатками автоматизированных систем управления качеством обслуживания являются невозможность перенастройки QoS при изменении топологии сети и дополнительные финансовые расходы для компании при покупке, внедрении и эксплуатации данных систем.

По результату обзора также можно сделать вывод, что на сегодняшний день в сетях применяются все три модели QoS — Best Effort Service, Integrated Service и DiffServ. Поскольку в Best Effort задействованы все свободные на текущий момент времени ресурсы сети без специального конфигурирования сетевого оборудования, то смысл модели заключается в своевременном расширении пропускной способности каналов связи. Для Integrated Service характерна необходимость планирования и прогнозирования потребности ресурсов, что часто проблематично, либо вообще невозможно. В большинстве сетей используется DiffServ, поэтому далее в работе будет рассматриваться только модель DiffServ.

В рассмотренных методах управления качеством обслуживания корпоративных сетей (ручном, автоматизированных системах, SDN) выявлены и описаны выше значительные недостатки. Поэтому существуют такие ситуации, когда применение ни одного из данных методов нецелесообразно. Например, необходимо автоматизировать управление качеством обслуживания при недостаточном финансировании или замена сетевого оборудования невозможна из-за недоступности (удаленность местоположения, отсутствие допуска на узлы связи). Поэтому необходимо предложить новый метод.

Дальнейшая работа посвящена исследованию создания нового метода управления качеством обслуживания корпоративных сетей. Будет разработан новый алгоритм управления качеством обслуживания. Также планируется создать программное обеспечение, которое этот подход позволит реализовать и протестировать на жизнеспособность.

Глава 2. Аналитический обзор нового метода управления качеством обслуживания корпоративных сетей

2.1 Структура сети

На примере сети, схема которой представлена на рисунке 2, опишем механизм работы нового метода.

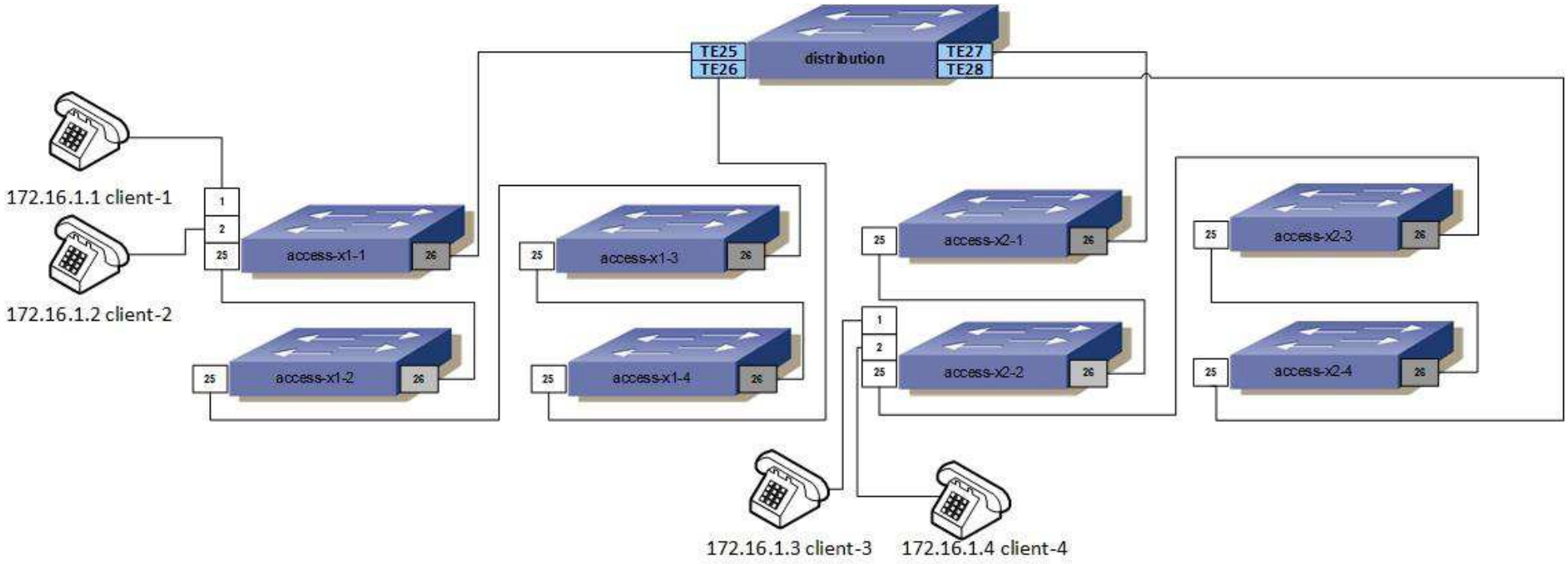

Рисунок 2. Структурная схема сети для демонстрации работы метода

Перед администратором сети стоит задача предоставить определенный класс качества обслуживания для ip-телефонов, обозначенных на схеме, как client-1, client-2, client-3, client-4.

Для выполнения поставленной задачи необходимо пройти следующие этапы:

1. обеспечить механизм настройки коммутатора доступа. На данном этапе необходимо определить какие именно настройки нужно указать на конечном сетевом оборудовании и осуществить его конфигурирование;
2. коммутатор доступа должен произвести поиск соседних коммутаторов и совместно с ними построить топологию сети;
3. коммутаторы должны осуществить построение кратчайшего маршрута между ip-телефонами в разных концах сети;
4. коммутаторы вычисляют и автоматически выставляют политику обслуживания к транзитным портам, по которым будет передаваться трафик, а также к портам доступа для ip-телефонов.

2.2 Настройка конечного сетевого оборудования

Рассмотрим пример настройки QoS в IOS Cisco на коммутаторах серии Catalyst 6500/6000.

Настройка политики в IOS включает следующие шаги:

1. Определение диспетчеров политик — нормальная скорость трафика, максимальная скорость (если используется), блок и действие политики.
2. Создание QoS ACL для выбора контролируемого трафика и присоединение диспетчера политик к этому ACL.
3. Применение QoS ACL ко всем нужным портам или VLAN [9].

В нашем случае ip-телефон client-1 подключен к коммутатору access-x1-1 в порт ge1 и посылает трафик UDP со скоростью 1 Мбит/сек. Необходимо ограничить этот поток трафика UDP до 50 кбит/сек и отбросить лишний трафик.

Настроим функции применения политик на конечном коммутаторе access-x1-1. Диспетчер политик присоединяется к VLAN. Порт ge1 принадлежит VLAN 10.

Нужно настроить QoS на порту ge1 в режиме vlan-based. Используем команду `set port qos ge1 vlan-based` для нашей задачи.

Конфигурация конечного коммутатора access-x1-1:

```
set qos enable #Включение QoS [9]

set qos policer aggregate udp_50kbps rate 50 burst 10
drop #Определение диспетчера политик [9]

set qos acl ip udp_qos_vid10 dscp 0 aggregate udp_50kbps
udp any any eq 5060 #Создание QoS ACL для выбора трафика
и присоединение диспетчера политик к QoS ACL [9]

commit qos acl all #Компиляция этого QoS ACL [9]

set port qos ge1 vlan-based #Настройка порта для режима
QoS на основе виртуальной локальной сети [9]

set qos acl map udp_qos_vid10 10 #Сопоставление QoS ACL
с виртуальной локальной сетью 10 [9]
```

2.3 Описание работы протокола

После включения протокола коммутаторы делают рассылку hello-сообщений на групповой адрес 01:80:C2:00:00:60 и ожидает в ответ такие же hello-сообщения. Обменявшись hello-сообщениями, коммутаторы устанавливают соседские отношения, добавляя каждый друг друга в свою локальную таблицу соседей.

Аналогично работе Русина М.С. по теме «Построение полносвязанных сетей на базе коммутаторов Ethernet» [10] для вставки заголовка, который содержит в себе метрику и поле Type of Service (ToS), были выделены 4 байта между тегом 802.1Q и PayLoad. Первые 2 байта занимает EtherType, который позволяет отличать Ethernet кадры проектируемого протокола от всех остальных. Для проверки работоспособности алгоритма и моделирования был выбран EtherType 8999, т. к. он является свободным. Вторые 2 байта содержат в себе непосредственно метрику, которая влияет на то, как будут построены маршруты, а также поле ToS, которое характеризует конкретный класс сервиса в данном DiffServ домене. Тем самым получается сохранить и использовать все исходные EtherType и заголовки, которые были вставлены ранее.

Итоговый вид кадра представлен на рисунке 3.

* — может быть, как просто EtherType, так и 802.1Q Header + EtherType

Рисунок 3. Формат Ethernet кадра для работы протокола

Для разработки алгоритма был введен ряд терминов, описывающий некоторые новые функции и состояния коммутаторов и его интерфейсов:

- Edge интерфейс – интерфейсы, к которым подключаются конечные устройства, не входящие в работу протокола;

- Core интерфейс – интерфейсы, которые подключаются к другим коммутаторам из сети протокола;
- Metric – числовое значение в таблице MAC-адресов, влияющее на выбор маршрута для отправляемого трафика;
- заголовок кадра – набор байтов, содержащих какую-либо информацию. В данном случае, это информация о стоимости маршрута до устройства, с которого пришел кадр [10].

В ходе работы протокол анализирует кадры и хранит кратчайшие маршруты между сетевыми устройствами. Для этого создана таблица, похожая на стандартную таблицу MAC-адресов, но имеющая дополнительное поле Metrics, а также возможность хранить для одного адреса несколько интерфейсов. Хранение нескольких интерфейсов обусловлено возможностью балансировать трафик, если позволяет топология.

Пример таблицы представлен на рисунке 4.

MAC	Interface	Age	Metrics
A	e0/0	20	10
B	e0/1	20	40
C	e0/2	20	20
...	...	...	....

Рисунок 4. Пример таблицы проектируемого протокола

После обмена сообщениями, каждый коммутатор знает про все соединения, на основании пар строится полная топология сети, включающая все коммутаторы и все связи между ними.

Затем опираясь на информацию о полной топологии сети, коммутаторы выстраивают кратчайший маршрут между сетевым оборудованием, находящимся в разных концах сети.

Коммутаторы обмениваются информацией из таблиц, содержащей связки мак-адресов и соответствующих им значений Metrics. Зная за каким интерфейсом находится сетевое оборудование с данным мак-адресом,

коммутаторы отправляют данные через нужные интерфейсы соседям. Именно на этих интерфейсах алгоритм настраивает необходимое качество обслуживания. Изначально настройки QoS задает администратор на конечном коммутаторе доступа, к которому непосредственно подключен клиент, требующий определенного качества обслуживания. После этого коммутатор конвертирует настройки качества обслуживания в бинарный код и помещает его в поле QoS сообщения Update. Сформированное сообщение Update отправляется соседним коммутаторам. Соседи принимают сообщение Update. Изымают бинарный код из поля QoS и конвертируют его в понятную для данного коммутатора конфигурацию. Заменив интерфейсы на нужные (интерфейс, с которого получено сообщение Update, и интерфейс, за которым находится следующий сосед), протокол применяет полученную настройку качества обслуживания на данном коммутаторе. Затем формирует новое сообщение Update с бинарным кодом настроек в поле QoS и передает своим соседям далее по маршруту следования трафика.

Если у одного из коммутаторов пропадает связь с соседом:

1. он рассылает всем новые сообщения;
2. все коммутаторы заново строят топологию сети;
3. заново считают кратчайшие маршруты между сетевым оборудованием;
4. обновляют свою таблицу данных всех состояний каналов домена.

2.4 Типы сообщений протокола

1. Hello:

отправляются регулярно через фиксированные интервалы времени (по умолчанию, 10 секунд) на групповой адрес 01:80:C2:00:00:60, который входит в диапазон свободных групповых MAC-адресов 01:80:C2:00:00:50 — 01:80:C2:00:00:FF.

Сообщения Hello выполняют следующие задачи:

1. поиск соседей и установление соседских отношений;
2. осуществляют функцию keepalive для поддержания установленных соседских отношений.

EtherType = 8999		Type	ID Коммутатора						Hello interval		Dead interval		ID соседнего коммутатора					
1	2	1	1	2	3	4	5	6	1	2	1	2	1	2	3	4	5	6

Рисунок 5. Формат сообщения Hello

Сообщение Hello содержит следующие поля:

- Type — тип сообщения (для сообщения Hello значение равно 1);
- ID Коммутатора — идентифицирует коммутатор в пределах сети. В роли идентификатора используется MAC-адрес коммутатора;
- Hello interval — задает временной период рассылки hello-сообщений (по умолчанию, 10 секунд);
- Dead interval — время ожидания ответа соседей;
- ID соседнего коммутатора — идентификатор соседнего коммутатора.

Список соседей формируется из идентификаторов соседей, от которых коммутатор получил сообщение Hello в течение времени, заданного в поле Dead interval [11].

2. Update:

сообщение Update используется для обмена конфигурацией, управляющей качеством облуживания, между коммутаторами с соседскими отношениями. Данная конфигурация находится в двоичном формате в поле QoS. Значение поля Type равно 2.

EtherType = 8999		Type	ID Коммутатора						MAC-адрес						QoS
1	2	1	1	2	3	4	5	6	1	2	3	4	5	6	300

Рисунок 6. Формат сообщения Update

На коммутаторе доступа, к которому подключен конечный клиент, администратор выполняет настройку, управляющую качеством обслуживания для данного клиента. Необходимую часть созданной конфигурации коммутатор, используя сериализацию, кодирует в двоичном формате, помещает полученный код в поле «QoS» сообщения Update и передает соседнему коммутатору. Соседний коммутатор принимает сообщение Update. Изымает двоичный код из поля «QoS» и десериализует его, приводя в текстовый вид. Затем коммутатор применяет полученную конфигурацию на восходящий порт, с которого получено сообщение Update. По таблице соседей коммутатор определяет нисходящий порт и также применяет данную конфигурацию. Таким образом коммутатор меняет номера портов в полученной от соседнего коммутатора конфигурации и выполняет автоматизированную настройку транзитных Core портов. Измененную конфигурацию аналогичным способом коммутатор сериализует в двоичный код, помещает его в поле «QoS» сообщения Update и передает следующему соседу по маршруту. Такой алгоритм повторяется до тех пор, пока сообщение Update не достигает коммутатора доступа, к которому подключен второй конечный клиент. Данный коммутатор доступа также десериализует двоичный код из сообщения Update в текстовый вид, применяет полученную конфигурацию на восходящий порт. Затем по таблице MAC-адресов

определяет порт, к которому подключен адресат сообщения, и применяет на него необходимую конфигурацию.

В результате, все порты на маршруте передачи данных между конечными клиентами настроены на определенный уровень качества обслуживания с помощью автоматизированной настройки.

3. Notification:

информационное сообщение. Используются, когда возникают ошибки и/или меняется топология сети. Отправляются соседним коммутаторам. Значение поля Type равно 3.

EtherType = 8999		Type	ID Коммутатора						Error code
1	2	1	1	2	3	4	5	6	1

Рисунок 7. Формат сообщения Notification

Коды ошибок:

1 — Ошибка обработки сообщения;

2 — Ошибка в соседских отношениях;

3 — Ошибка обмена информацией о состоянии каналов, синхронизации баз данных;

4 — Изменение топологии сети.

2.5 Машина состояний протокола

Коммутатор с протоколом для принятия решений в операциях с другими коммутаторами использует конечный автомат (finite-state machine (FSM)), который включает четыре состояния: Idle, HelloSent, HelloConfirm, Established. Для каждого сеанса протокол поддерживает переменную состояния, которая отслеживает, в каком из этих четырех состояний находится сеанс. Протокол определяет сообщения, которыми каждый коммутатор должен обмениваться, чтобы переключить сеанс из одного состояния в другое.

Первое состояние (Idle) — это состояние ожидания. В состоянии ожидания протокол инициализирует все ресурсы, отклоняет все попытки входящих соединений. Второе состояние — HelloSent. В состоянии HelloSent коммутатор отправляет сообщение Hello по всем Core интерфейсам и ожидает такое же сообщение в ответ, чтобы перейти в состояние HelloConfirm. Обмен сообщениями проверки активности выполняется, и после успешного получения коммутатор переводится в состояние Established. Это означает, что запущена правильная версия протокола и все настройки верны. Коммутаторы-соседи переходят к обмену информацией из своих баз данных, содержащих бинарные значения, определяющих качество обслуживания для сетевого оборудования с конкретным MAC-адресом. Для это используются сообщения Update. В состоянии Established коммутатор может отправлять и получать: Hello, Update, Notification сообщения к своему соседу и от него.

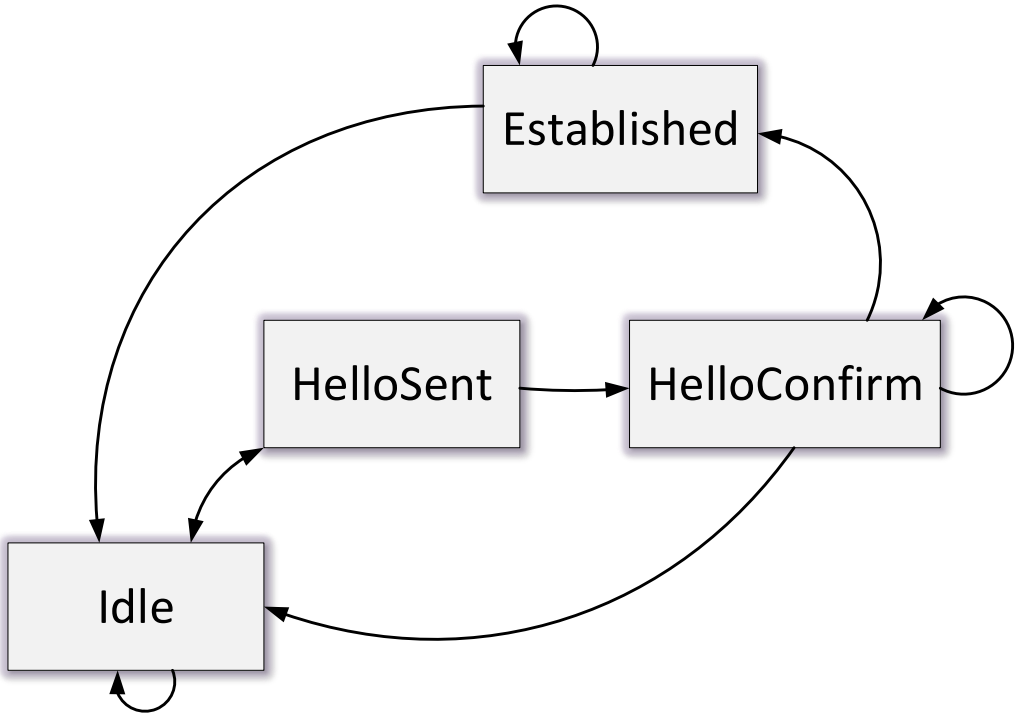

Рисунок 8. Машина состояний протокола

2.6 Выводы по главе 2

По результатам анализа предметной области и предложений на рынке сетевого оборудования и программного обеспечения, было принято решение разработать ПО, реализующее возможности и функции исследуемого протокола для управления качеством обслуживания корпоративных сетей. Данный протокол позволит автоматизировать и оптимизировать настройку качества обслуживания для конкретных клиентов. Следующим этапом в разработке протокола будет его реализация и тестирование на специально подготовленном стенде, которое должно подтвердить заявленные функции.

Глава 3. Реализация и тестирование протокола

3.1 Структура кода протокола

При написании протокола за основу были взяты блоки кода изучения кадра и управления пересылкой кадров из протокола, разработанного в ВКР Русина М.С. «Построение полносвязанных сетей на базе коммутаторов Ethernet». Для добавления заявленного функционала были созданы блоки управления служебными сообщениями и обработки конфигурации качества обслуживания.

Таким образом, код реализованного протокола можно разделить на четыре части:

- блок изучения кадра;
- блок управления служебными сообщениями;
- блок управления пересылкой кадров;
- блок обработки конфигурации качества обслуживания.

Блок изучения кадра можно разделить на следующие составные:

- блок изучения метрик и заполнения таблиц;
- блок балансировки трафика.

Основной задачей блока изучения кадров является заполнение таблиц маршрутизации. Каждый кадр, который проходит через коммутатор, изучается и информация помещается в специальную таблицу, содержащую MAC-адрес источника, интерфейс, с которого пришел кадр, «возраст» записи (по умолчанию 30 секунд) и метрику. Записи с одинаковым MAC-адресом источника попадают в таблицу с заменой, которая отрабатывает по алгоритму «меньше метрика – лучше маршрут». Если у таких записей метрика одинакова, но отличаются входные интерфейсы, то в таблицу добавляются обе записи. Это позволяет в дальнейшем использовать балансировку трафика.

Блок балансировки трафика необходим для разделения трафика по доступным каналам передачи данных. Для определения порядкового номера интерфейса, в который можно отравить кадр, используется формула:

$$X = (\text{SIP} + \text{DIP} + \text{Proto} + \text{SPort} + \text{DPort}) \% N,$$

где X – номер полученного канала;

SIP – IP адрес источника;

DIP – IP адрес назначения;

Proto – идентификатор протокола;

SPort – TCP\UDP порт источника;

DPort – TCP\UDP порт назначения;

N – количество доступных каналов.

Хэш-сумма, полученная путем суммирования полей заголовков кадра, делится на количество доступных интерфейсов, за которыми находится устройство с MAC-адресом назначения. Тем самым получается добиться распределения потока трафика на все доступные каналы.

После изучения кадра и получения номера интерфейса срабатывает блок пересылки кадра. Если кадр был получен от устройств, которые не входят в сеть коммутаторов, где работает реализуемый протокол, то на интерфейсе, откуда пришел данный кадр в таблице ставится метрика 0, а в сам кадр вставляется EtherType 8999. Данный EtherType необходим для упрощения поиска метрики в кадре и для возможности отличить пакеты протокола от других пакетов в сети. Если пакет пришел с другого коммутатора сети протокола, то происходит проверка записи в таблице на возможность ее изменения и увеличив метрику на заданную в настройках единицу. При выходе пакета из сети работы протокола EtherType и метрика с кадра снимаются [10].

Благодаря блоку управления служебными сообщениями осуществляется поиск и установление соседских отношений с сетевым оборудованием с помощью сообщений Hello, передача настроек качества обслуживания

(сообщения Update), уведомление соседей об изменениях и ошибках в сети (сообщения Notification).

Блок обработки конфигурации качества обслуживания отвечает за прием настроек из поля QoS в сообщении Update, полученном от соседнего сетевого оборудования или от администратора сети, в случае ручной настройки. Данный блок интерпретирует полученные настройки в понятный для используемого оборудования лексикон, меняет наименование интерфейсов для настройки на них политик QoS. Затем передает эти настройки дальше на формирование нового сообщения Update, который будет отправлен следующему соседу по топологии сети.

3.2 Описание стенда для тестирования

Для проведения тестов на проверку функционала разработанного протокола был собран виртуальный стенд, который представляет собой тринадцать виртуальных машин (ВМ) с установленной ОС GNU/Linux Ubuntu 12.04.5. ВМ имеют по два процессора Intel Xeon E5320 с тактовой частотой 1.86 ГГц и 1 Гбайт оперативной памяти каждая. Подробная информация о технических характеристиках ВМ изображена на рисунках 9 и 10. Девять ВМ выполняют функции коммутаторов — на них запущен разработанный протокол. Оставшиеся четыре ВМ — клиенты. Они работают в упрощенном режиме и запускают приложения, которые диагностируют сеть (ping, tcpdump). Все ВМ соединены виртуальными каналами передачи данных.

```
client-1@switch:~$ free
             total       used       free     shared    buffers     cached
Mem:       1025644     272764     752880          0      43896     162740
-/+ buffers/cache:      66128     959516
Swap:      1046524          0    1046524
```

Рисунок 9. Информация об оперативной памяти (вывод команды free)

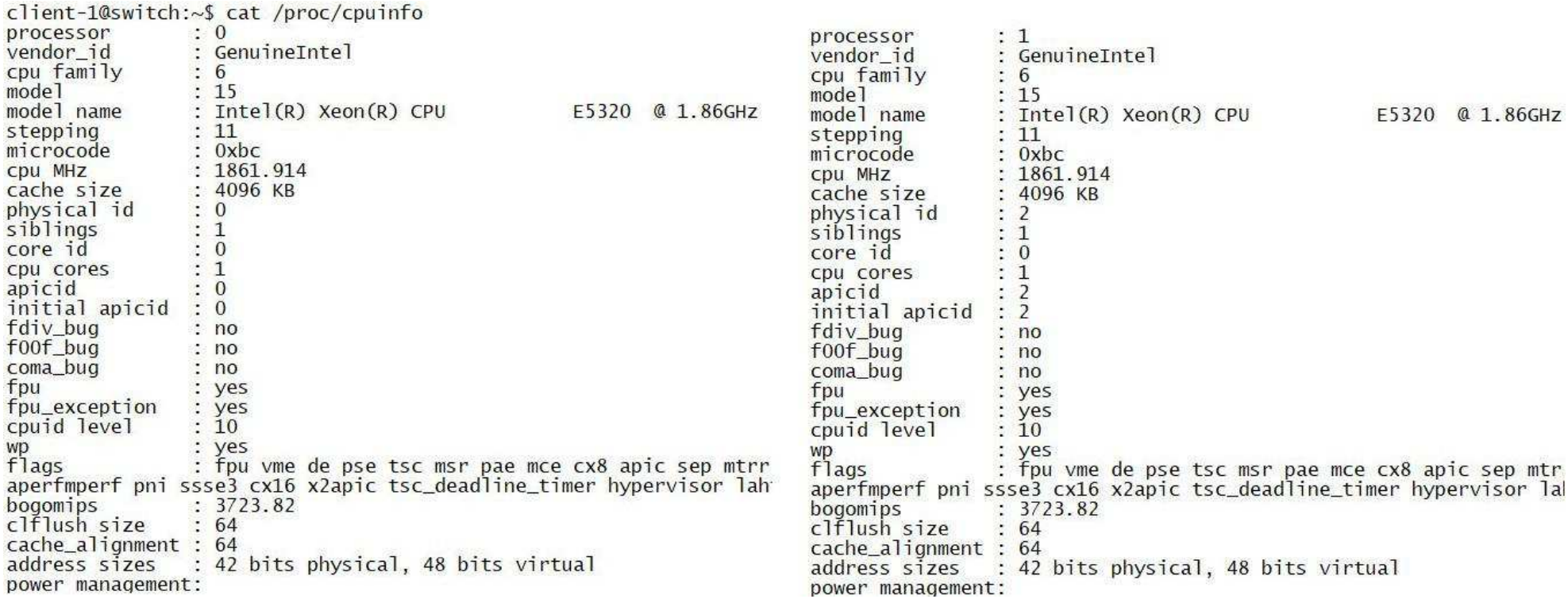

Рисунок 10. Информация о процессоре (вывод команды cat /proc/cpuinfo)

Структурная схема стенда показана на рисунке 11.

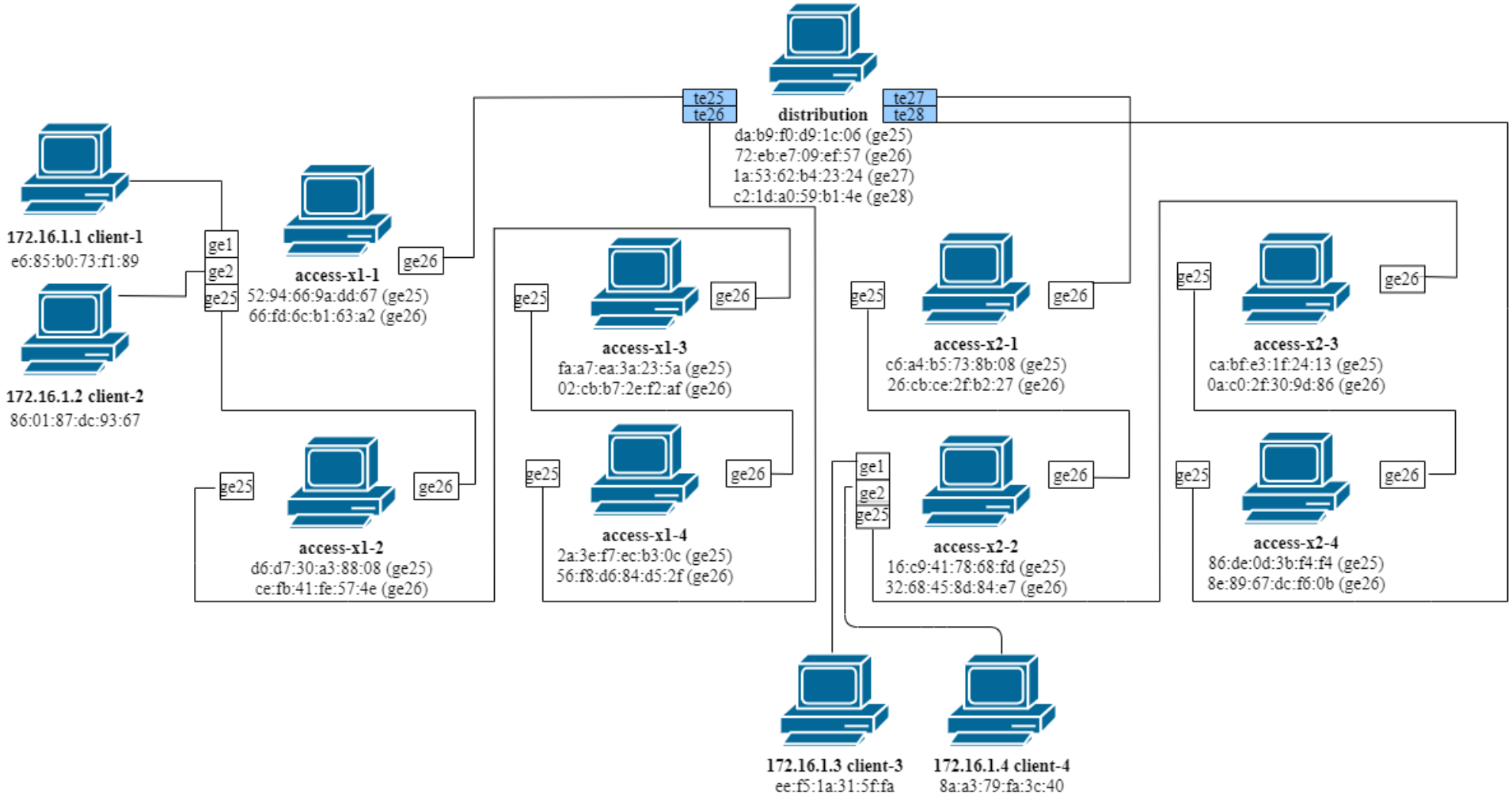

Рисунок 11. Структурная схема виртуального стенда для тестирования

Собранный виртуальный стенд значительно упрощает запуск и отладку тестируемого протокола по сравнению со стендом из физических коммутаторов и ip-телефонов.

Все ВМ со своими интерфейсами созданы и настроены с помощью скрипта setup.sh. Покажем его работу на примере коммутатора access-x1-1. Для этого сделаем вывод скрипта setup.sh для коммутатора access-x1-1:

```
user@switch:~$ cat setup.sh | grep access-x1-1
/sbin/ip netns add access-x1-1
/sbin/ip netns exec distribution /sbin/ip link set dev "ge26"
netns access-x1-1
/sbin/ip netns exec access-x1-1 /sbin/ip addr flush dev "ge26"
/sbin/ip netns exec access-x1-1 /sbin/ip link set dev "ge26"
up
/sbin/ip netns exec access-x1-2 /sbin/ip link set dev "ge25"
netns access-x1-1
/sbin/ip netns exec access-x1-1 /sbin/ip addr flush dev "ge25"
/sbin/ip netns exec access-x1-1 /sbin/ip link set dev "ge25"
up
/sbin/ip netns exec access-x1-1 /sbin/ip link add name "ge1"
type veth peer name "eth0"
/sbin/ip netns exec access-x1-1 /sbin/ip addr flush dev "ge1"
/sbin/ip netns exec access-x1-1 /sbin/ip link set dev "ge1"
up
/sbin/ip netns exec access-x1-1 /sbin/ip link set dev "eth0"
netns client-1
/sbin/ip netns exec access-x1-1 /sbin/ip link add name "ge2"
type veth peer name "eth0"
/sbin/ip netns exec access-x1-1 /sbin/ip addr flush dev "ge2"
/sbin/ip netns exec access-x1-1 /sbin/ip link set dev "ge2"
up
/sbin/ip netns exec access-x1-1 /sbin/ip link set dev "eth0"
netns client-2
```

Из полученного вывода видно, что скрипт setup.sh создает network namespaces с именем access-x1-1. Network namespaces (netns) — это логическая копия сетевого стека со своими собственными маршрутами, правилами брандмауэра и сетевыми устройствами [12]. Затем для access-x1-1 создаются и запускаются интерфейсы с именами "ge26", "ge25", "ge1", "ge2". Для интерфейса "ge1" указывается соседний интерфейс "eth0", который принадлежит netns с именем client-1. А для интерфейса "ge2" — интерфейс "eth0", который принадлежит netns с именем client-2.

Таким образом, созданы и настроены ВМ, которые выполняют функции коммутатора доступа access-x1-1 и подключенных к нему ip-телефонов client-1 и client-2.

Чтобы запустить разработанный протокол на конкретной ВМ необходимо авторизоваться на стенде под ее именем. Для примера запустим протокол на коммутаторе доступа access-x1-1.

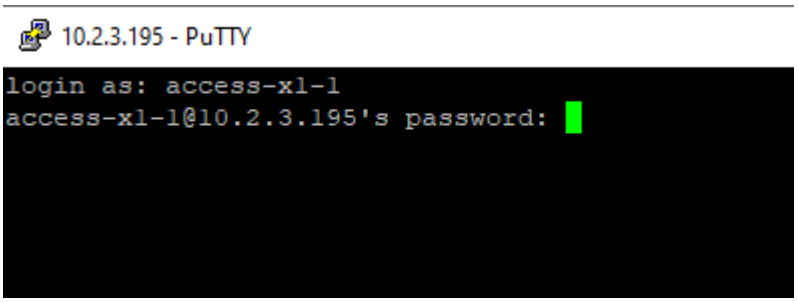

Рисунок 12. Авторизация на коммутаторе access-x1-1

После авторизации видим приветственное сообщение с системной информацией ВМ (см. рисунок 13).

```
Welcome to Ubuntu 12.04.5 LTS (GNU/Linux 3.13.0-32-generic i686)
 * Documentation:  https://help.ubuntu.com/
  System information as of Mon May 17 20:50:08 GMT-7 2021
  System load:  0.0               Processes:           112
  Usage of /:   17.2% of 6.76GB   Users logged in:     1
  Memory usage: 7%                IP address for eth0: 10.2.3.195
  Swap usage:   0%
```

Рисунок 13. Приветственное сообщение ВМ access-x1-1

После этого автоматически запускается скрипт access-x1-1.sh, который для netns с именем access-x1-1 запускает скрипт switch.py с параметром access.

```
user@switch:~$ cat access-x1-1.sh
#!/bin/sh
/usr/bin/sudo /sbin/ip netns exec access-x1-1 /usr/bin/sudo /usr/bin/python /home/user/switch.py access
```

Рисунок 14. Вывод скрипта access-x1-1.sh

Параметр access выдается только тем ВМ, к которым непосредственно подключены клиенты. В скрипте switch.py находится исходный код разработанного протокола, написанный на высокоуровневом языке программирования Python версии 2.7. После запуска данного скрипта начинает работу протокол.

Аналогично происходит запуск для остальных ВМ из стенда для тестирования.

3.3 Результаты тестирования разработанного протокола

При тестировании разработанного протокола на соответствие заявленных функций использовались утилиты ping и tcpdump.

Утилита ping генерирует ICMP пакеты, которые позволяют узнать о доступности того или иного хоста в сети.

Утилита tcpdump — это анализатор, который перехватывает сетевой трафик, проходящий через устройство, на котором запущена утилита.

На виртуальном стенде было проведено тестирование разработанного протокола, которое показало следующие результаты. На рисунке 15 изображена топология с работающими на время тестирования устройствами.

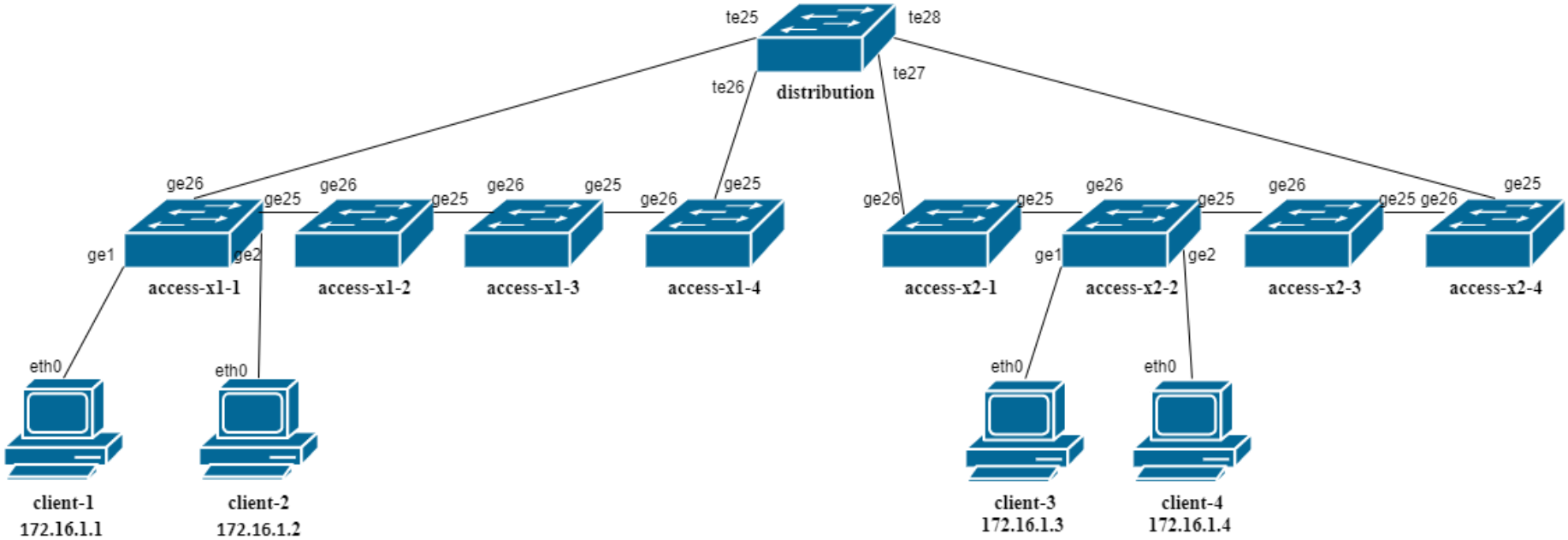

Рисунок 15. Топология сети на тестовом стенде

При запуске утилиты ping с указанием количества отправленных пакетов в 1000 шт. с client-1 на client-4, с client-2 на client-3, с client-3 на client-1, с client-4 на client-2 одновременно были получены следующие результаты (см. рисунок 16):

- ответ на ping был получен от всех конечных хостов;
- задвоения и потери пакетов не наблюдались;
- хосты-коммутаторы отработали алгоритм протокола без ошибок и корректно заполнили таблицы маршрутизации.

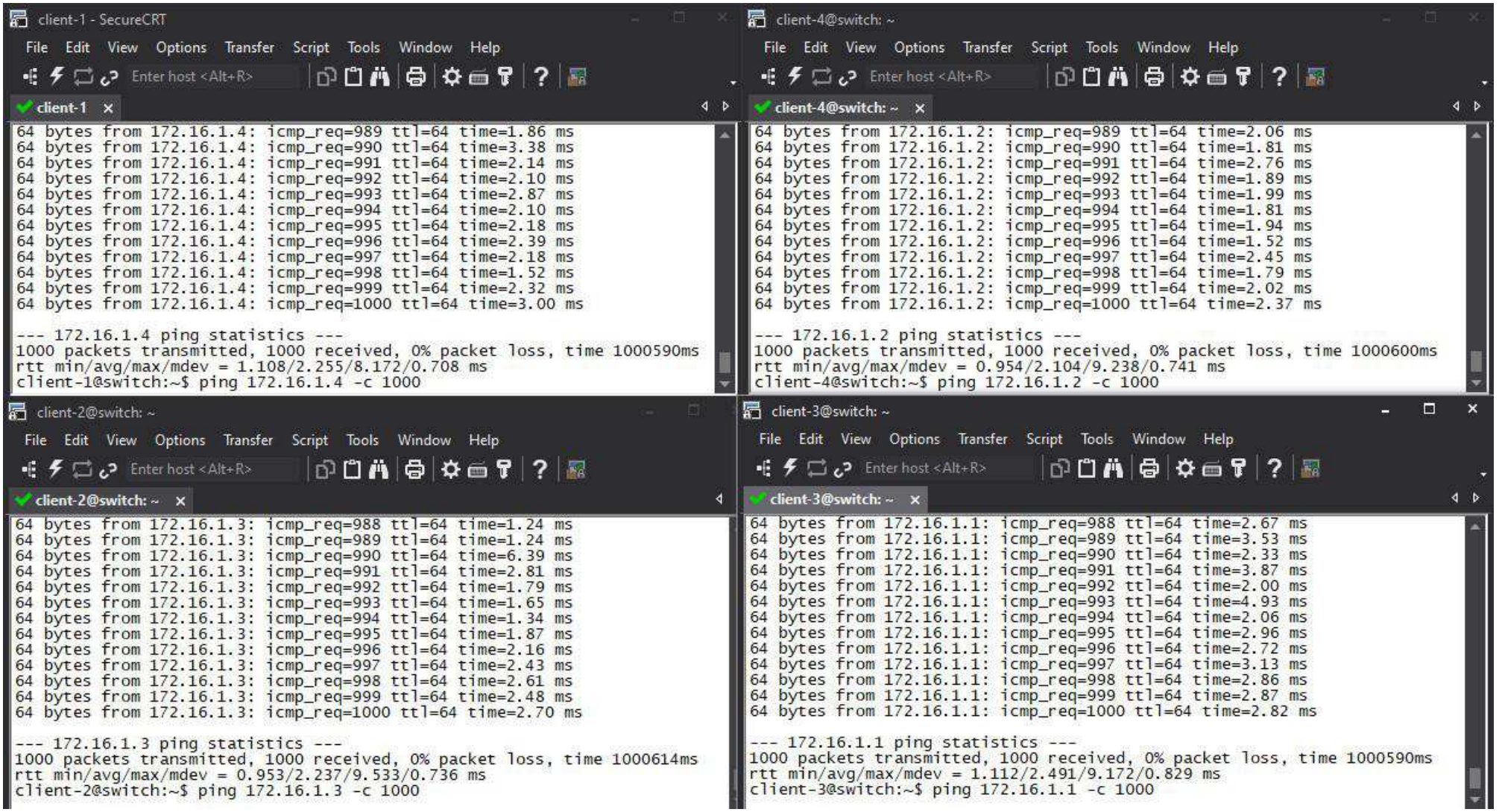

Рисунок 16. Результаты выполнения команды ping между клиентами

На рисунке 17 представлена информация с client-1 и всех коммутаторов при выполнении команды ping с client-1 на client-4.

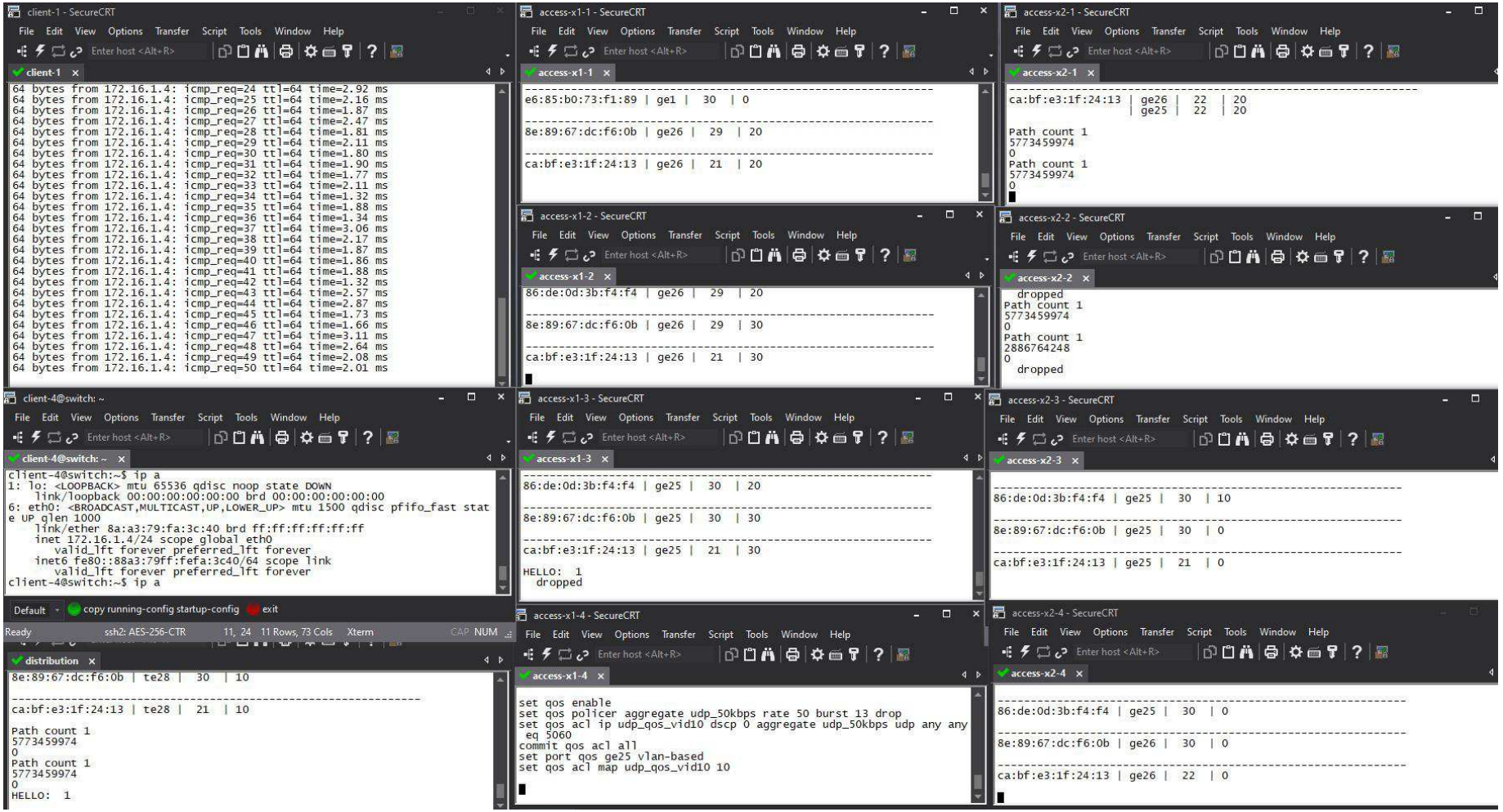

Рисунок 17. Информация с коммутаторов при выполнении команды ping между client-1 и client-4

С помощью информации с рисунка 17 можно проследить полученный при работе протокола маршрут для клиентов client-1 и client-4: client-1 ↔ access-x1-1 ↔ distribution ↔ access-x2-1 ↔ access-x2-2 ↔ client-4.

Полученный результат позволяет сделать вывод, что алгоритм протокола работает правильно и выстраивает наиболее оптимальный маршрут для трафика.

На рисунке 18 изображены сообщения Update с коммутаторов access-x1-1 и access-x2-2, к которым непосредственно подключены клиенты client-1 и client-4 соответственно. Видно, что разработанный протокол осуществляет передачу настроек качества обслуживания, введённых вручную администратором сети, по маршруту следования трафика между коммутаторами access-x1-1 и access-x2-2. Следовательно, разработанный протокол выполняет заявленную функцию.

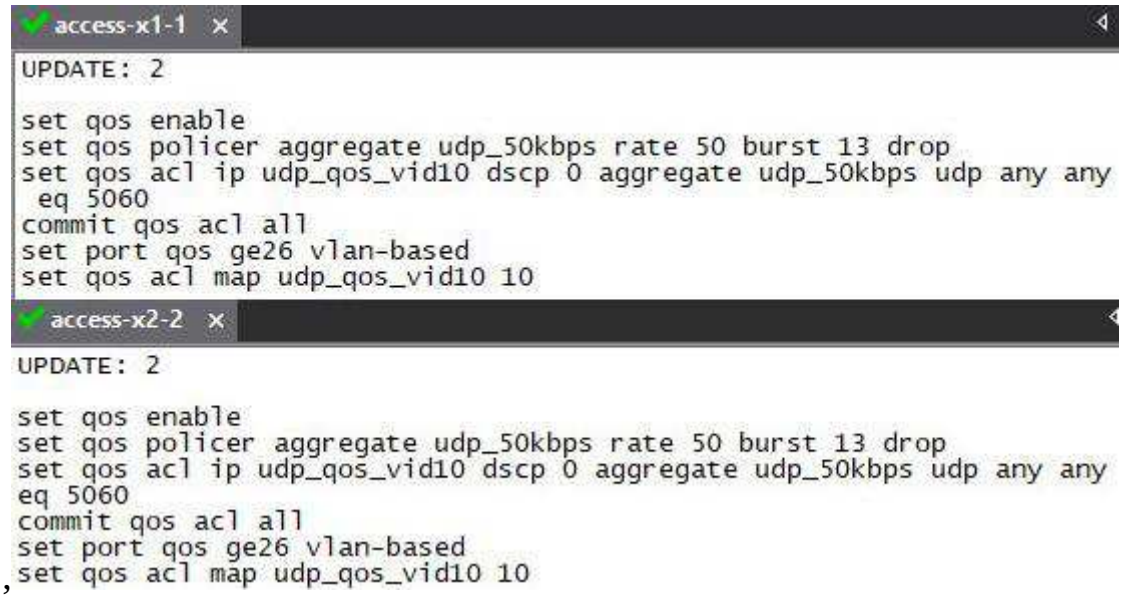

Рисунок 18. Сообщения Update с настройками качества обслуживания

Для дополнительной проверки был снят дамп с помощью утилиты tcpdump на виртуальном канале между access-x1-1 и access-x1-2 при включенном client-1 (см. рисунок 19).

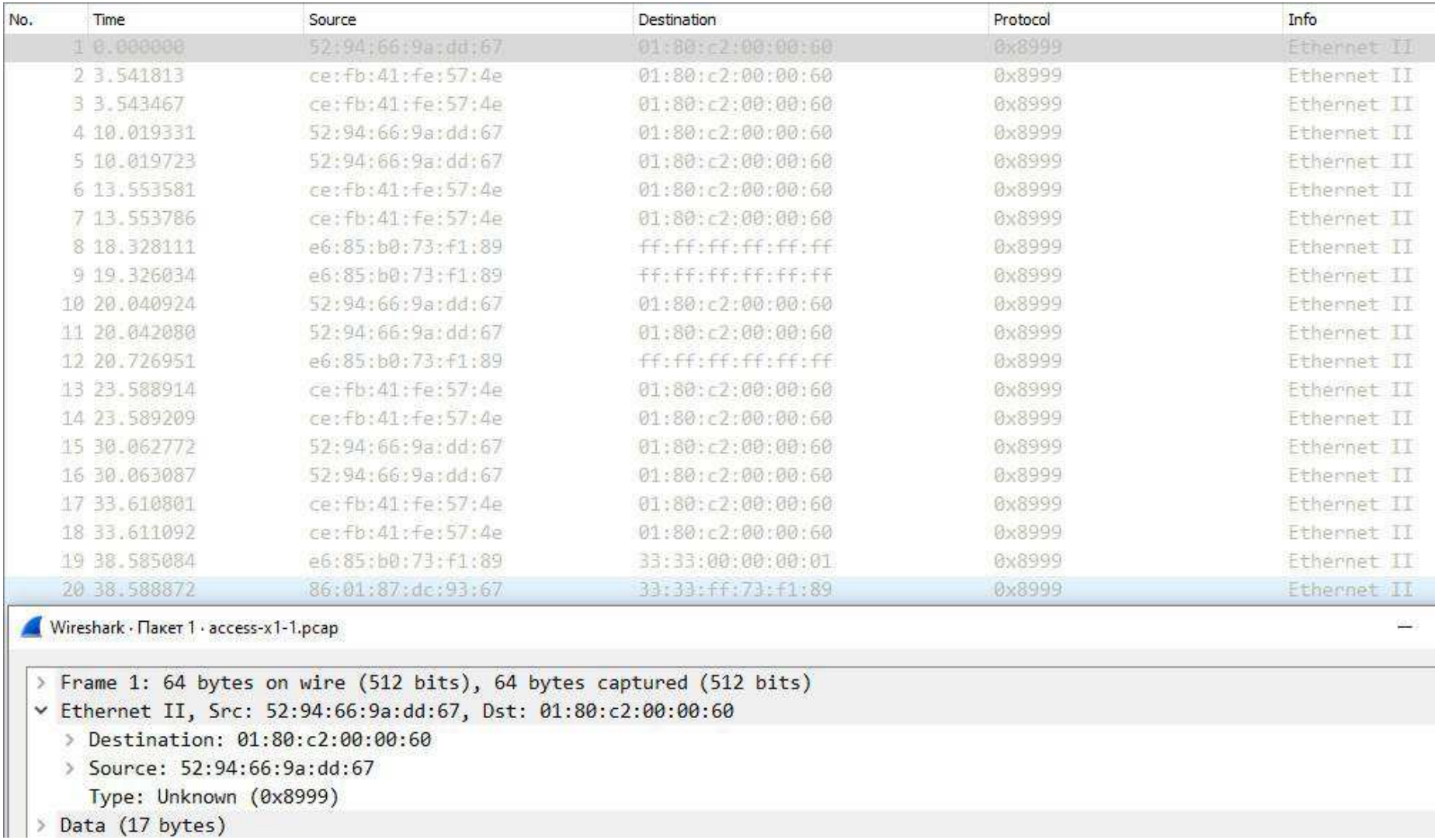

Рисунок 19. Дамп на канале между access-x1-1 и access-x1-2

С помощью информации, изображенной на рисунке 19, можно проследить за работой протокола. Сначала происходит поиск соседей, рассылка сообщений Hello — кадры № 1 и 2. Затем после установления соседских отношений коммутаторы обмениваются сообщениями Update — кадры № 5 и 7. На кадрах № 8 и 9 client-1 отправляет широковещательный запрос для построения ARP-таблицы. На кадрах № 19 и 20 происходит передача полезных данных от client-1.

В результате проведенного тестирования были получены результаты, благодаря которым, можно сделать вывод, что разработанный протокол выполняет заявленные функции, а именно позволяет на уровне коммутации использовать все доступные каналы передачи данных, без риска выхода из строя сети, а также производит рассылку служебных сообщений. С помощью разработанного протокола удалось автоматизировать передачу настроек качества обслуживания по всему маршруту движения трафика от отправителя к получателю.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы было проведено исследование существующих методов и моделей управления качеством обслуживания. Информация о них, включая достоинства и недостатки, содержится в обзоре предметной области (глава 1). На основе данной информации было принято решение о разработке нового протокола, который позволяет автоматизировать настройку качества обслуживания без внедрения SDN и автоматизированных систем, представленных на рынке.

Для реализации заявленных функций нового протокола, был разработан алгоритм и написан код на высокоуровневом языке программирования Python. Созданный скрипт был запущен на стенде из виртуальных машин, работающих на операционной системе GNU/Linux Ubuntu. Девять виртуальных машин выполняли функции коммутаторов, а четыре — конечных устройств. По результатам проведенных тестов было установлено, что разработанный протокол выполняет заявленные функции и позволяет автоматизировать настройку качества обслуживания по всему маршруту движения трафика.

Все поставленные задачи выполнены. Цель работы достигнута.

СПИСОК СОКРАЩЕНИЙ

ACL — Access Control List

API — Application Programming Interface

DiffServ — Differentiated services

DIP — Destination Internet Protocol

FSM — Finite-State Machine

GNU — GNU’s Not UNIX

ICMP — Internet Control Message Protocol

ID — Identifier

IntServ — Integrated services

IOS — Internetwork Operating System

IP — Internet Protocol

MAC — Media Access Control

Netns — Network namespaces

QoS — Quality of Service

RFC — Request for Comments

RTT — Round Trip Time

SDN — Software-Defined Networking

SIP — Source Internet Protocol

TCP — Transmission Control Protocol

ToS — Type of Service

UDP — User Datagram Protocol

VLAN — Virtual Local Area Network

VoIP — Voice over IP

ВКР — Выпускная квалификационная работа

ВМ — Виртуальная машина

ОС — Операционная система

ПО — Программное обеспечение

ЭВМ — Электронно-вычислительная машина

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кузьмин, В. В. Модели и процедуры управления трафиком в мультисервисной сети оператора связи : дис. … канд. техн. наук : 05.13.01 / Кузьмин Василий Васильевич. — Нижний Новгород, 2015. — С. 24-25.
2. Сети для самых матёрых. Часть пятнадцатая. QoS // PCNEWS.RU [Электронный ресурс] — Режим доступа: https://pcnews.ru/blogs/seti_dla_samyh_materyh_cast_patnadcataa_qos-845933.html
3. Колосов, Е.А. Разработка аппаратных и программных решений предоставления сервиса IP-телефонии / Е.А. Колосов [Электронный ресурс] — Режим доступа: http://masters.donntu.org/2008/kita/kolosov/diss/index.htm
4. Когай, Г. Д. Показатели и модели реализации качества обслуживания / Г. Д. Когай, М. К. Кисина // Молодой ученый. — 2016. — № 20 (124). — С. 157-160.
5. Барсков, А. SDN: кому и зачем это надо? / А. Барсков // Журнал сетевых решений LAN. — 2012. — № 12. — С. 22.
6. Гордеев, И. М. Разработка программного комплекса для обучения построению программно-конфигурируемых сетей SDN / И. М. Гордеев, М. В. Модель, А. С. А. Мутханна // Сборник научных статей VIII Международной научно-технической и научно-методической конференции: «Актуальные проблемы инфотелекоммуникаций в науке и образовании (Апино 2019)» — Санкт-Петербург: Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича, 2019. — 356-360 с.
7. Гилазов, М. Н. Влияние качества обслуживания на коммутацию IP-пакетов / М. Н. Гилазов // Молодой ученый. — 2015. — № 11 (91). — С. 282-285.

8. Евангели, А. Автоматизированная настройка QoS для локальных и глобальных сетей. / А. Евангели // itWeek [Электронный ресурс] — Режим доступа: https://www.itweek.ru/infrastructure/article/detail.php?ID=63591
9. Применение политик QoS на коммутаторах Catalyst серии 6500/6000. / Cisco Systems, Inc. // Cisco Systems, Inc. [Электронный ресурс] — Режим доступа: https://www.cisco.com/c/ru_ru/support/docs/switches/catalyst-6000-series-switches/12493-102.html
10. Русин, М. С. Построение полносвязанных сетей на базе коммутаторов Ethernet : дис. студента магистра : 09.04.01 / М. С. Русин. — Красноярск, 2020.
11. Самойленко, Н. OSPF / Н. Самойленко // xgu.ru [Электронный ресурс]. − Режим доступа: http://xgu.ru/wiki/OSPF
12. Rosen, R. Resource management: Linux kernel Namespaces and cgroups / R. Rosen [Электронный ресурс]. – Режим доступа: http://docplayer.net/14284170-Resource-management-linux-kernel-namespaces-and-cgroups.html
13. Дониев, Э. Т. Особенности MPLS для управления трафиком в IP-сетях / Э. Т. Дониев, З. З. Нигматов — // Молодой ученый. — 2017. — № 35 (169). — С. 1-3.
14. Суть информатизации общества [Электронный ресурс]. – Режим доступа: http://samzan.ru/118971
15. Мухамадиева, К. Б. Эффективное распределение телекоммуникационных каналов / К. Б. Мухамадиева // Молодой ученый. — 2018. — № 1 (187). — С. 19-21.
16. Ошкина, Е. В. Сетевая технология SDN (обзор, современные тенденции) / Е. В. Ошкина — // Технические науки: проблемы и перспективы : материалы V Междунар. науч. конф. (г. Санкт-Петербург, июль 2017 г.). — Санкт-Петербург : Свое издательство, 2017. — С. 3-6.

17. Li, D. et al. A survey of network update in SDN // Frontiers of computer science. — 2017. — № 1. — P. 4–12.
18. Балжинням Н., Лю Ю. SDN / программно-конфигурируемые сети / сравнительные исследования сети IP // Научная дискуссия: вопросы технических наук. — 2017. — № 2 (42). — С. 78–85.
19. Дворников А. А., Восков Л. С., Саксонов Е. А., Ефремов С. Г. Метод построения оптимального наложенного канала для беспроводной сенсорной сети // Информационные технологии. — 2016. — Т. № 11. — С. 812–818.
20. Артюхов, В. В. Организация автоматизированной системы управления IT-инфраструктурой корпоративной сети / В. В. Артюхов, Л. В. Трунова. — // Молодой ученый. — 2017. — № 43 (177). — С. 23-29.
21. Иванов, К. К. Управление в системах реального времени / К. К. Иванов. — // Молодой ученый. — 2017. — № 20 (154). — С. 139-141.
22. Проблемы обеспечения качества услуг (QoS) [Электронный ресурс]. — Режим доступа: https://siblec.ru/telekommunikatsii/multiservisnye-seti-svyazi/4-problemy-obespecheniya-kachestva-uslug-qos
23. Transition [Электронный ресурс]: Quality of Service (QoS) in High-Priority Applications. — Режим доступа: https://www.transition.com/wp-content/uploads/2016/05/qos_wp.pdf
24. Transition [Электронный ресурс]: Обзор качества обслуживания (QoS). — Режим доступа: https://technet.microsoft.com/library/hh831679.aspx
25. Национальная библиотека им. Н. Э. Баумана [Электронный ресурс]: QoS (Quality of service). — Режим доступа: https://ru.bmstu.wiki/index.php?title=QoS_(Quality_of_service)&mobileaction=toggle_view_mobile
26. QOS – приоритизация трафика [Электронный ресурс]. – Режим доступа: https://moxa.ru/tehnologii/ethernet_network/qos/

27. Cisco Collaboration System Solution Reference Network Designs. Available at: http://www.cisco.com/c/en/us/td/docs/voice_ip_comnycucm/smd/collabl 1 col lab 11/ cac.html (accessed 14.05.2018).
28. Моисеев. В. И. Методика расчета параметров линии доступа мультисервисной сети / В. И. Моисеев. Б.Я. Лихтциндер // Вестник ЮУрГУ. Серия «Компьютерные технологии, управление, радиоэлектроника». — 2018. — Т. 18, № 3. — С. 51-58.
29. Переход к технологии программно-конфигурируемых сетей [Электронный ресурс]. – Режим доступа: http://infocom.uz/2019/07/24/perexod-k-texnologii-programmno-konfiguriruemyx-setej/
30. RFC-1633 "Integrated Services in the Internet Architecture: An Overview" Braden R., Clark D., Shenker S., June 1994.
31. RFC-2474 "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", K. Nichols, December 1998.
32. RFC-2475 "An Architecture for Differetiated Services", Blake S., Black D., Carlson M. Davies E., Wang Z., Weiss W., December 1998.
33. RFC-2386 "A Framework for QoS-based Routing in the Internet", E. Crawley, August 1998
34. RFC-2597 "Assured Forwarding PHB Group", J. Heinanen, June 1999
35. RFC-2598 "An Expedited Forwarding PHB", V. Jacobson, June 1999
36. RFC-3387 "Considerations from the Service Management Research Group (SMRG) on Quality of Service (QoS) in the IP Network", M. Eder, H. Chaskar, S. Nag. September 2002
37. СТО 4.2-07-2014 Система менеджмента качества. Общие требоваши к построению, изложению и оформлению документов учебной деятельности. - Ввел. 09.01.2014. - Красноярск : СФУ, 2014. - 60 с.

ПРИЛОЖЕНИЕ А

Листинг кода разработанного протокола

```
    # Import
    from socket import *
    import struct
    import thread
    import binascii
    import os
    import time
    import sys
    import re
    from pprint import pprint
    # Enumerate network interfaces
    interfaces = os.popen("ip a | egrep -o '(te|ge)[0-9-
]+'").read().split("\n")
    interfaces.pop()
    # Is it an access switch?
    access = len(sys.argv) > 1 and sys.argv[1] == "access"
    # Create ports
    ports = [None] * len(interfaces)
    for i in range(0, len(interfaces)):
        ports[i] = socket(AF_PACKET, SOCK_RAW, htons(3))
        ports[i].bind((interfaces[i], 0))
    print "Switching on " + " ".join(interfaces)
    # tag format
    tag = "\x89\x99\0\x0a"
    # tag='\x89\x99\0\0'
```

Продолжение приложения А

```
    # message types
    message_type = {
            "hello": 1,
            "update": 2,
        }
    # mac addresses
    broadcast='\xff\xff\xff\xff\xff\xff'
    multicast = "\x01\x80\xc2\x00\x00\x60"
    # config for quality of service policy
    config_qos_policy = """
    set qos enable
    set qos policer aggregate udp_{rate}kbps rate {rate}
burst {burst} drop
    set qos acl ip udp_qos_vid{vlan_id} dscp 0 aggregate
udp_{rate}kbps udp any any eq 5060
    commit qos acl all
    set port qos {interface} vlan-based
    set qos acl map udp_qos_vid{vlan_id} {vlan_id}
    """
    # mac address table
    mac_address_table = {}
    def total_bytes(frame):
        return int(frame.encode("hex"), 16)
    def list_mac(frame):
        return (
            binascii.hexlify(frame[0])
            + ":"
```

Продолжение приложения А

```
            + binascii.hexlify(frame[1])
            + ":"
            + binascii.hexlify(frame[2])
            + ":"
            + binascii.hexlify(frame[3])
            + ":"
            + binascii.hexlify(frame[4])
            + ":"
            + binascii.hexlify(frame[5])
        )
    def list_mac_address_table():
        for mac, data in mac_address_table.items():
            print "----------------------------------------
---------------------"
            node = list_mac(mac)
            for inf in data:
                node = (
                    node
                    + " | "
                    + str(inf[0].getsockname()[0])
                    + " |  "
                    + str(inf[1])
                    + "  | "
                    + str(inf[2])
                    + "\n"
                    + "                 "
                )
            print (node)
```

Продолжение приложения А

```
    ########################################
    # Determine where frame to be switched #
    ########################################
    def switch_frame(frame, socket_in):
        ports_out = list()
        if access and len(socket_in.getsockname()[0]) == 3:
# edge port ?
            mac_address_table[frame[6:12]]  =  [[socket_in,
30, 0]]
        else:
            ether_type = frame[12:14] # get ether type from
frame
            msg_type = int(frame[14:15].encode("hex"), 16)  #
get message type from frame
            metric = int(frame[15:16].encode("hex"), 16)  #
get metric from tag
            if msg_type == message_type["hello"]:
                print("HELLO:  " + str(msg_type))
                send_update_frame(socket_in, frame)
            elif msg_type == message_type["update"]:
                print("UPDATE: " + str(msg_type))
                interface,   vlan_id,   rate,   burst   =
socket_in.getsockname()[0], 10, 50, 13
                config = generate_config(config_qos_policy,
interface, vlan_id, rate, burst)
                print(config)
```

```
            if mac_address_table.get(frame[6:12], "null") ==
"null":  # Check MAC unknown or not
                #   print   'MAC   '+list_mac(frame[6:12])+'
unknown and learned' # Unknown
                mac_address_table[frame[6:12]]             =
[[socket_in, 30, metric]]
            else:  #
                data = mac_address_table.get(frame[6:12])
                #      print      data[0].getsockname()[0]+'
'+socket_in.getsockname()[0]
                index   =   [i   for   i,   (inf,   _,   _)   in
enumerate(data) if inf == socket_in]
                if index:  # learn shorter route check socket
equals
                    # print (index[0])
                    # print (data[index[0]][2])
                    if metric == data[index[0]][2]:
                        data[index[0]]  =  [socket_in,  30,
metric]
                        mac_address_table[frame[6:12]]     =
data
                        # list_mac(frame[6:12])
                        # list_mac(data[0])
                        if int(frame[0].encode("hex"), 16) %
2 and socket_in != data[0][0]:
                            return list()
```

```
                        # print 'MAC learned. Metric in frame
'+str(metric)+'= metric in table '+str(data[2])+'. And ports
equals.'
                    elif metric > data[index[0]][2]:
                        # print 'MAC not learned. Metric in
frame '+str(metric)+'> metric in table '+str(data[2])+'. And
ports equals.'
                        return list()
                    else:
                        # print 'MAC learned. Metric in frame
'+str(metric)+'< metric in table '+str(data[2])+'. And ports
equals.'
                        # data=[[socket_in,30,metric]]
                        mac_address_table[frame[6:12]]     =
[[socket_in, 30, metric]]
                else:  # ports not equals
                    if   [i   for   i,   (_,   _,   cost)   in
enumerate(data) if cost == metric]:
                        # print 'MAC not learned. Metric in
frame '+str(metric)+'= metric in table '+str(data[2])+'. And
ports not equals.'
                        data.append([socket_in, 30, metric])
                        mac_address_table[frame[6:12]]     =
data
                    # return list()
                    elif   [i   for   i,   (_,   _,   cost)   in
enumerate(data) if cost < metric]:
```

```
                        return list()
                    else:
                        # print 'MAC learned. Metric in frame
'+str(metric)+'>= metric in table '+str(data[2])+'. And ports
not equals.
                        mac_address_table[frame[6:12]]     =
[[socket_in, 30, metric]]
        if   int(frame[0].encode("hex"),   16)   %   2   or
mac_address_table.get(frame[0:6], "null") == "null":
            for socket in ports:
                if socket != socket_in:
                    ports_out.append(socket)
        else:
            path_count                                     =
len(mac_address_table.get(frame[0:6]))
            print "Path count " + str(path_count)
            hash = (
                total_bytes(frame[30:34])                  +
total_bytes(frame[34:38]) + total_bytes(frame[27])
            )                                              #
+total_bytes(frame[42:46])+total_bytes(frame[46:50])
            # hash = total_bytes(frame[6:12])
            print (hash)
            path = hash % path_count
            print (path)
    ports_out.append(mac_address_table.get(frame[0:6])[path]
[0])
```

Продолжение приложения А

```
        return ports_out
    ########################################
    # Main switch function
    def switch(socket_in, interface):
        while 1:
            # receive frame
            frame = socket_in.recv(9014)
            #       print       'Received       frame       on
'+socket_in.getsockname()[0]+'                                  from
'+binascii.hexlify(frame[6])+':'+binascii.hexlify(frame[7])+
':'+binascii.hexlify(frame[8])+':'+binascii.hexlify(frame[9]
)+':'+binascii.hexlify(frame[10])+':'+binascii.hexlify(frame
[11])+'                                                     to
'+binascii.hexlify(frame[0])+':'+binascii.hexlify(frame[1])+
':'+binascii.hexlify(frame[2])+':'+binascii.hexlify(frame[3]
)+':'+binascii.hexlify(frame[4])+':'+binascii.hexlify(frame[
5])+' ethertype 0x'+binascii.hexlify(frame[12:14])
            #       print       'Received       frame       on
'+socket_in.getsockname()[0]+' from '+list_mac(frame[6:12])+'
to              '+list_mac(frame[0:6])+'              ethtype
'+binascii.hexlify(frame[12:14])
            # determine where to send
            ports_out = switch_frame(frame, socket_in)
            # send frame copies
            if len(ports_out) == 0:
                print "  dropped"
            else:
```

```
                for socket_out in ports_out:
                    ether_type = frame[12:14]
                    # if out socket NOT EDGE tag and to switch
from switch add tag
                    if        (not        access)        or
len(socket_out.getsockname()[0]) != 3:
                        # if frame without tag
                        if ether_type != tag[0:2]:
                            # if out_port backupX0Y
                            if
len(socket_out.getsockname()[0]) == 9:
                                new_metric_bytes           =
struct.pack("h", 50)  # Add metric 50
                                frame2 = (
                                    frame[0:12]
                                    + tag[0:2]
                                    + new_metric_bytes[1]
                                    + new_metric_bytes[0]
                                    + frame[12 : len(frame)]
                                )
                            # if out_port ethXY
                            else:
                                new_metric_bytes           =
struct.pack('h',10) # Add metric 10

frame2=frame[0:12]+tag[0:2]+new_metric_bytes[1]+new_metric_b
ytes[0]+frame[12:len(frame)]
```

Продолжение приложения А

```
                        # if frame with tag
                        else:
                            # if out_port backupX0Y
                            if
len(socket_out.getsockname()[0]) == 9:
                                new_metric                  =
int(frame[15:16].encode("hex"), 16) + 50  # Add metric 50
                                new_metric_bytes            =
struct.pack("h", new_metric)
                                frame2 = (
                                    frame[0:14]
                                    + new_metric_bytes[1]
                                    + new_metric_bytes[0]
                                    + frame[16 : len(frame)]
                                )
                            # if out_port ethXY
                            else:
                                new_metric                  =
int(frame[15:16].encode("hex"), 16) + 10  # Add metric 10
                                new_metric_bytes            =
struct.pack("h", new_metric)
                                frame2 = (
                                    frame[0:14]
                                    + new_metric_bytes[1]
                                    + new_metric_bytes[0]
                                    + frame[16 : len(frame)]
                                )
```

```
                    # if frame with tag for PC/Other
                    else:
                        if (
                            access
                            and
len(socket_out.getsockname()[0]) == 3
                            and ether_type == tag[0:2]
                        ):
                            # remove tag
                            frame2 = frame[0:12] + frame[16
: len(frame)]
                        else:
                            # do not alter tags
                            frame2 = frame
                    #       print       '       sent       to
'+socket_out.getsockname()[0]
                    socket_out.send(frame2)
    # send frame with message update
    def send_update_frame(socket, frame):
        update_frame   =   frame[0:14]   +   struct.pack("B",
message_type["update"]) + "\x00\x00"
        socket.send(update_frame)
    # generate config
    def  generate_config(config,  interface,  vlan_id,  rate,
burst):
        config = config.replace("{interface}", interface)
        config = config.replace("{vlan_id}", str(vlan_id))
```

Продолжение приложения А

```
        config = config.replace("{rate}", str(rate))
        config = config.replace("{burst}", str(burst))
        return config
    # get mac
    def get_mac(interface):
        interface_info = os.popen("ip a s dev %s" %
interface).read()
        match = re.findall(
            r"ether ((?:[0-9a-fA-F]:?){12}) brd",
interface_info, re.IGNORECASE | re.MULTILINE
        )
        mac = binascii.unhexlify(match[0].replace(":", ""))
        return mac
    # Neighbors discovery
    def neighbors(socket, interface):
        while 1:
            print ("searching for neighbors on " + interface)
            # constructing hello frame
            # hello_frame=broadcast+...+tag+'\x00\x00'
            mac_source = get_mac(interface)
            hello_frame = multicast + mac_source + tag[0:2]
+ struct.pack("B", message_type["hello"]) + "\x00\x00"
            # sending hello
            socket.send(hello_frame)
            # receiving hellos
            time.sleep(10)
```

Окончание приложения А

```
    # Start main switch function in separate threads
    for i in range(0, len(interfaces)):
        thread.start_new_thread(switch, (ports[i], ""))
    # Start control plane threads
    for i in range(0, len(interfaces)):
        if interfaces[i] in ["ge25", "ge26"] or
interfaces[i].startswith("te"):  # if core or edge port
            thread.start_new_thread(neighbors, (ports[i],
interfaces[i]))
    # Main thread will do nothing
    while 1:
        time.sleep(1)
        # list_mac_address_table()
        # mac timeout
        print "|       MAC       |" + "  port  |" + "  age  |
metric  |"
        list_mac_address_table()
        for mac, data in mac_address_table.items():
            for inf in data:
                inf[1] = inf[1] - 1
                if inf[1] == 0:
                    data.remove(inf)
            if len(data) == 0:
                del mac_address_table[mac]
```

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой

_____ О.В. Непомнящий
подпись инициалы, фамилия
«16» 06 2021 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Методы управления качеством обслуживания корпоративных сетей
тема

09.04.01 Информатика и вычислительная техника
код и наименование направления

09.04.01.05 Сети ЭВМ и телекоммуникации
код и наименование магистерской программы

Научный руководитель _____ 15.06.21 доцент, канд. физ.-мат. наук К.В. Коршун
подпись, дата должность, ученая степень инициалы, фамилия

Выпускник _____ 15.06.21 А.О. Табаков
подпись, дата инициалы, фамилия

Рецензент _____ 15.06.21 генеральный директор ООО «Интертакс» М.В. Алексеев
подпись, дата должность, ученая степень инициалы, фамилия

Нормоконтролер _____ 15.06.21 доцент, канд. физ.-мат. наук К.В. Коршун
подпись, дата должность, ученая степень инициалы, фамилия

Красноярск 2021