Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Лебединский Юрий Евгеньевич

Исследования методов управления процессами реконфигурации ПКС при выполнении конкурирующих запросов построения маршрутов

КУРСОВАЯ РАБОТА

Научный руководитель:
к. ф.-м. н.
В. О. Писковский

Москва, 2022

Аннотация

Работа посвящена проблеме реконфигурации программно-конфигурируемых сетей при выполнении конкурирующих запросов построения маршрутов.

В данной работе рассматривается применимость решения проблемы конфликта транзакций в реляционных базах данных для ПКС. В рамках курсовой работы разрабатывается концепция блокировок, как мера решения данной проблемы, а также формулируются необходимое и достаточное условия отсутствия потери пакетов за время реконфигурации. Для проверки этих условий реализован экспериментальный стенд и проведено исследование.

Содержание

Введение

Программно-конфигурируемые сети

Программно-конфигурируемые сети (ПКС) — это концепция построения сети, в которой контур управления сетью разделён с контуром передачи данных [1]. Логика управления находится на отдельном сервере, называемом контроллером. Приложения взаимодействуют с контроллером через северный интерфейс. Такая концепция позволяет упростить настройку и модификацию сети, а также получить независимый от производителя сетевого оборудования интерфейс взаимодействия между уровнями.

Таким образом, в архитектуре программно-конфигурируемой сети можно выделить два основных контура (рис. 1):

1. Контур передачи данных, на котором находятся сетевые устройства.

2. Контур управления. В этом контуре находятся контроллер и приложения. Контроллер предоставляет интерфейс для приложений и управляет устройствами передачи данных. Приложения определяют поведение устройств в контуре передачи данных.

OpenFlow

На сегодняшний день протокол OpenFlow [2] является основным протоколом, реализующим концепцию программно-конфигурируемых сетей. OpenFlow — это открытый стандарт, описывающий протокол удаленного управления OpenFlow коммутаторами.

OpenFlow коммутатор работает следующим образом. У каждого получаемого пакета копируется заголовок. Далее выполняется поиск правил (записей) в таблицах коммутации, которым соответствует этот заголовок. Каждое правило содержит операции, которые требуется выполнить над подходящим пакетом. Такими операциями могут быть, например, изменение заголовка пакета, пересылка пакета на определенный порт. Из найденных правил выбирается наиболее приоритетное. Если не найдено правило с большим приоритетом, выполняется последнее, которому соответствуют все пакеты. В этом случае пакет инкапсулируется и коммутатор отправляет контроллеру PacketIn сообщение. Контроллер, в свою очередь, формирует новое правило и отправляет ответное сообщение, с помощью которого оно устанавливается.

Контур управления может изменять конфигурацию ПКС путем добавления, удаления и модификации записей в таблицах коммутатора. Процесс изменения таблиц коммутации в коммутаторах называется реконфигурацией ПКС. При независимой работе приложений возможна ситуация перекрытия правил, которая может привести к некорректной работе некоторых из приложений.

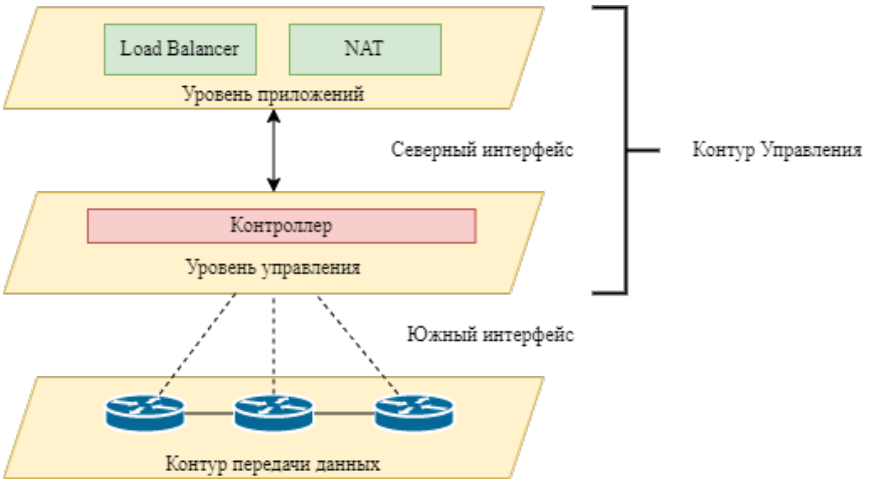


Рис. 1: Контуры управления и передачи данных архитектуры ПКС

Конфликты в ПКС

Реконфигурация требуется для множества случаев, например, для поддержания и восстановления маршрутов движения пакетов при подключении и отключении аппаратуры, для оптимизации таблиц коммутации во избежание их переполнения, а также для балансировки нагрузки каналов связи. Так как процесс реконфигурации выполняется в ответ на запрос от коммутатора, разные запросы могут приводить к возникновению противоречащих изменений, то есть к конфликтам. Для разрешения конфликтов, в данной работе предлагается использование теории сериализации (упорядочивания). Данная теория успешно применяется в работе транзакционных систем обработки информации.

Рассмотрим понятие конфликта на следующем примере [3]:

У нас есть следующая топология (рис. 2). Требуется распределить все данные, получаемые из генератора равномерно по серверам-получателям и каналам так, чтобы максимально успользовать ресурсы сети и серверов. Для этого на контроллере работают 2 приложения: End-point load-balancing (EpLB) и Path load balancing (PLB) для балансировки трафика между серверами и каналами, соответственно. Первое приложение изменяет поле получателя в заголовках пакетов, так чтобы пакеты перенаправлялись на разные сервера. Для работы второго приложения каждый пакет отправляется на контроллер, анализируется и перенаправляется на конкретный порт. Мы ожидаем следующее распределения трафика (рис. 2).

Несмотря на это, может произойти и следующая ситуация (рис. 3). Так как приложения работают одно за другим, то первый случай может произойти только если сначала работает EpLB, а потом PLB, и при этом их адресные пространства совпадают. Если приложения работают в том же порядке, но их адресные пространства различны, то пакеты, обработанные EpLB приложением будут проигнорированы PLB приложением.

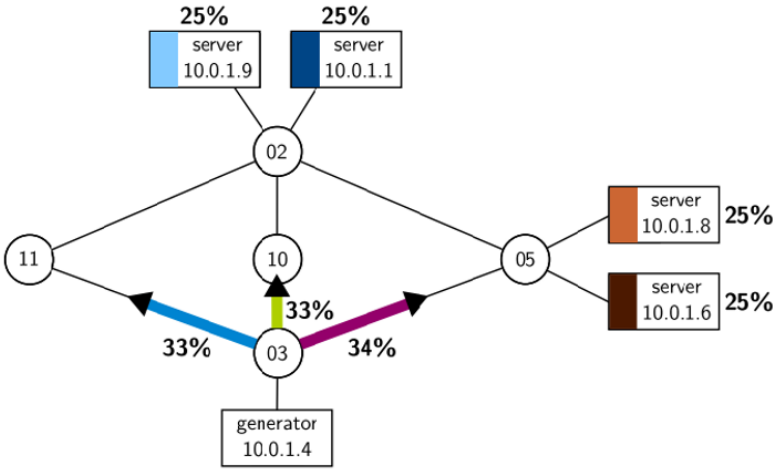


Рис. 2: Ожидаемое распределение трафика

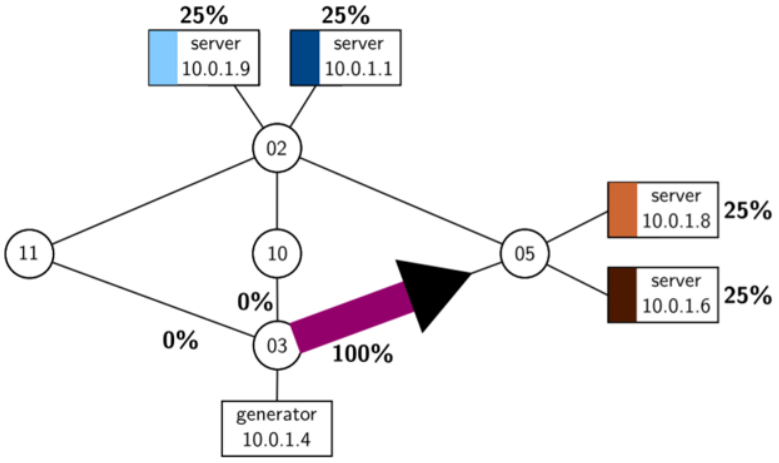


Рис. 3: Возможный результат работы приложений

Это приведет к распределению трафика как на (рис. 3).

1 Постановка задачи

Определим конфигурацию сети как совокупность таблиц коммутации [4]: $Net = tab_w : w \in W$, где tab — таблица коммутации, соответствующая одному коммутатору, W — множество коммутаторов в конфигурации. Изменения таблиц коммутации производятся в ответ на запросы от контроллера. Контроллер может отправлять следующие команды:

1. $Add(w, r)$ — добавление правила r в таблицу коммутатора w. Net;
2. $Delete(w, y, z, n)$ — удаление из таблицы коммутатора w всех правил приоритета n, соответствующих шаблону (y, z);

3. $Modify(w, y, z, \alpha, n)$ — замена инструкций во всех правилах коммутации приоритета n в таблице коммутатора w, соответствующих шаблону y, z, на новую инструкцию α.

Определим последовательность команд $\Gamma = com_1, com_2, .., com_m$. Применение последовательности команд Γ к конфигурации сети Net означает суперпозицию $com_m(com_{m-1}(..(com_1(Net))))$. Процедурой реконфигурации назовем последовательность команд P^j для j-того запроса реконфигурации сети. Расписание реконфигурации S — последовательность операций из множества процедур $P^j(j = \overline{1,k})$.

Изменения конфигурации являются распространенным источником нестабильности в сетях, приводящим к сбоям в работе и нарушениям безопасности. Даже если первоначальная и окончательная конфигурации корректны, то сам процесс перестроения часто проходит через промежуточные стадии, в процессе которых могут возникать нарушения ограничений, налагаемых на работу сети. Под проблемами реконфигурации в данном контексте понимаем наличие промежуточных состояний сети ... ,x_i, ... , x_j, ... , где x_j – состояние сети на j-том шаге реконфигурации и равно , где (j<m), которые в условиях выполнения нескольких конкурентных процедур реконфигурации сети могут вести к возможным нарушениям политик безопасности, вызванных задержкой при выполнении команд и процедур по изменению состояния сети.

Данная работа является продолжением выпускной квалификационной работы В.А. Шапошникова. Её целью является разработка и оценка методов построения расписания реконфигурации S, обеспечивающего отсутствие потерь пакетов.

2 Обзор существующих решений

Проведем сравнение существующих решений по следующим параметрам:

- Требования для непротиворечивой реконфигурации сети [4]:
  - Согласованность пакетов. Множество правил, согласно которому обрабатывается отдельный пакет в сети, должно быть или полностью старым, или полностью новым.
  - Согласованность потоков. Все пакеты в одном потоке должны быть обработаны с помощью одной конфигурации в сети (строгая согласованность) или префикс потока должен быть обработан одной конфигурацией сети, а суффикс потока должен быть обработан другой конфигурацией (слабая согласованность).
  - Отсутствие противоречий в целом. Новое множество правил, вычисляемых контроллером, должны быть непротиворечивым. Это означает, что в итоговой реконфигурации не должно быть аномалий.

- Требовательность к ресурсам. Работа сети во многом зависит от требований, предъявляемых решением к вычислительным ресурсам сетевых устройств, коммутаторов, например, объем буферной памяти, наличие и объем ассоциативной тернарной памяти, производительность и прочее. Для целей нашего обзора сформулируем следующие критерии к сетевым устройствам:
  - тип, объем и производительность памяти;
  - вычислительная производительность;
  - пропускная способность устройства, как обобщающая характеристика.
- Масштабируемость: насколько данное решение может быть применимо при увеличении
  - количества элементов сети;
  - конкурирующих запросов на ее реконфигурацию.

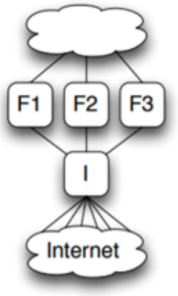
2.1 Time sensitive network

Time Sensitive Networks (TSN) [5] – набор стандартов, который позволяет обеспечивать строгую синхронизацию и детерминированную связь по Ethernet между участниками распределенной сети без каких-либо дополнительных соединений, сохраняя тем самым в рамках одной и той же сети максимальные преимущества в части коммуникации.

Ключевые принципы TSN:

1. синхронизация времени: все устройства в сети должны быть синхронизированы от общего эталона времени и образовывать единую инфраструктуру;
2. фиксированная низкая задержка: должна быть гарантирована своевременная доставка чувствительного к задержкам трафика;
3. надежность: должен быть определен набор компонентов для обеспечения оптимальной надежности и безопасности;
4. управление ресурсами: при масштабировании сети требуются дополнительные инструменты для обеспечения лучшей управляемости и видимости.

Непротиворечивая реконфигурация сети. Непротиворечивость в рамках согласованности пакетов достигается в силу того, что все устройства в сети являются синхронизированными, в силу единой инфраструктуры выполняется как согласованность пакетов, так и согласованность потоков. Требование надежности обеспечивает отсутствие противоречий в целом.



Configuration I

	Type	Action
I	U, G	Forward F_1
	S	Forward F_2
	F	Forward F_3
F_1	*SSH*	Monitor
	$*$	Allow
F_2	$*$	Allow
F_3	$*$	Allow

→

Configuration II

	Type	Action
I	U	Forward F_1
	G	Forward F_2
	S, F	Forward F_3
F_1	*SSH*	Monitor
	$*$	Allow
F_2	*SSH*	Monitor
	$*$	Allow
F_3	$*$	Allow

Рис. 4: Пример работы метода тегирования пакетов

Масштабируемость. TSN хорошо масштабируема по количеству запросов, так как данное решение направлено на интернет вещей, в котором важна безопасность и возможность обработки большого числа данных. Но в случае масштабируемости по количеству хостов, методы реконфигурации значительно усложняются, что крайне негативно сказывается на больших сетях.

Требовательность к ресурсам. Данная система является комплексной и вычислительно сложной. Поэтому данную систему не имеет смысла применять в небольших сетях. Сложность системы снижается с помощью разделения функционала на стандарты, но как было описано выше, данное решение полностью раскрывается исключительно в больших сетях.

2.2 Тегирование пакетов

Идея этого решения заключается в использовании дополнительного поля в пакете для хранения номера его версии [6]. Для лучшего понимания принципа работы рассмотрим пример:

Пусть имеется 4 возможных типа трафика, которые проходят через фильтрующие коммутаторы F_1, F_2, F_3 и коммутатор I:

- F_1 – коммутатор, который обрабатывает так называемый Неизвестный (U=Unknown) и Гостевой (G=Guest) трафик. Так как данный трафик является сомнительным, его требуется анализировать в целях защиты от возможного злоумышленника. Коммутатор F_1 проверяет (Monitor) трафик.

- F_2 – коммутатор, обрабатывающий трафик от студентов (S=Student). Данный тип трафика является доверенным, он не проверяется на наличие вредоносного кода.

- F_3 – коммутатор, который обрабатывает трафик факультета (Faculty). Данный тип трафика, как и в случае F_2, является доверенным.

В качестве эксперимента мы будем обрабатывать трафик U на F_1, трафик G на F_2, остальной трафик на F_3. В случае, если мы сразу отправим трафик G на F_2, то появится угроза безопасности сети. В таком случае, для реконфигурации мы должны выполнить следующие действия:

1. Обновить коммутатор I, чтобы перенаправлять трафик типа S на F_3.
2. Ждать, пока не пройдут все старые пакеты типа S через F_2.
3. Обновить F_2, добавив опцию проверки на коммутатор (Monitor).
4. Обновить I, чтобы весь трафик типа G переотправлялся на F_2.

Если бы пакеты не были тегированы, то мы бы не могли выполнить 2 пункт, так как не знаем, какой конфигурации принадлежит пакет.

Непротиворечивая реконфигурация сети. Согласованность пакетов и потоков обеспечивается за счет того, что старые пакеты отличаются от новых в значении тега, и пока не пройдут все старые пакеты, новые не начнут обрабатываться. Итоговое отсутствие противоречий не гарантируется.

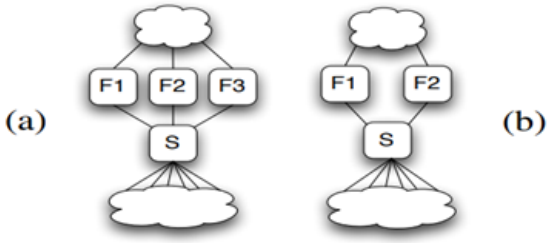
Масштабируемость и требовательность к ресурсам. Ожидание новыми потоками обработки старых влечет увеличение задержек по обработке пакетов при увеличении количества элементов сети, и, в особенности, при увеличении конкурирующих запросов на ее реконфигурацию. Так же дополнительное поле в пакете требует дополнительных затрат на память коммутатора.

2.3 Хранение нескольких конфигураций

Данное решение [7] заключается в загрузке двух наборов правил в каждый коммутатор со специальным битом в заголовке. При обработке пакета, с помощью специального бита в пакете анализируется, какой версии конфигурации он соответствует и, если это необходимо, выставляет специальный бит в поле заголовка (для определения набора правил для данного пакета).

Для лучшего понимания принципа работы рассмотрим пример:

На данной схеме P – выходной порт, T – тип пакета. Пусть в начальной конфигурации у нас имеются четыре коммутатора F_1, F_2, F_3 и S. Коммутатор S соединен с F_1, F_2, F_3 портами 1-2, 3-4, 5-6 соответственно. Трафик, выходящий через порты 1-2, является сомнительным и требует проверки на F_1 (в данном случае пакеты типа A и B). Трафик, выходящий через остальные порты, является доверенным и проверять его не надо. В качестве эксперимента перейдем к другой конфигурации, теперь мы будем обрабатывать сомнительный трафик на коммутаторах F_1,F_2, через порты 1-2 и 3-4 соответственно. А остальной трафик будет обрабатываться на коммутаторе F_3. В таком случае процесс реконфигурации должен выполняться в следующем порядке:



Configuration I

	P	T	Action
S	1, 2 3, 4 5, 6		Fwd F_1 Fwd F_2 Fwd F_3
F_1		A, B	Monitor Allow
F_2			Allow
F_3			Allow

→

Configuration II

	P	T	Action
S	1 2 3-6		Fwd F_1 Fwd F_2 Fwd F_3
F_1		A, B	Monitor Allow
F_2		A, B	Monitor Allow
F_3			Allow

Рис. 5: Пример работы метода хранения нескольких конфигураций

1. Загрузить дополнительные таблицы коммутации на F_1, F_2, F_3 и S.

2. Обновить коммутатор S. Теперь будут иметься две версии пакетов в сети, старые и новые. Новые пакеты хранят противоположный бит в заголовке, если старые хранят – 0, то новые – 1, и наоборот.

3. Дожидаться окончания действия предыдущих пакетов (когда все пакеты, идущие от S, будут новые). В данном случае у нас нет простоя в сети, так как дополнительные таблицы на коммутаторах позволяют обрабатывать несколько версий пакетов.

4. Удалить старые таблицы коммутации.

Непротиворечивая реконфигурация сети. Непротиворечивость в рамках согласованности пакетов достигается в силу того, что после загрузки новых наборов правил в каждый коммутатор, старые пакеты обрабатываются старой конфигураций, а новые - новой. В таком случае выполняется и согласованность в плане пакетов, и в плане потоков. Итоговое отсутствие противоречий никак не проверяется.

Масштабируемость. В таблице правил каждого коммутатора должны присутствовать данные двух конфигураций, т.е. доступная память коммутатора, в таком случае, делится пополам. Это может привести к большим проблемам, как и в случае увеличение количества элементов сети, так и в случае конкурирующих запросов на ее реконфигурацию. Также это может привести к ограничению функциональности, так как используемый бит распознавания версии потока фактически частично уменьшает функционал (он может занимать VLAN тэг или метку MPLS).

Требовательность к ресурсам. Данное решение подразумевает наличие избыточного количества коммутаторов для обеспечения корректной работы контура передачи данных. После загрузки таблиц нет никаких задержек по обработке пакета (в зависимости от бита применяются соответствующие правила).

2.4 Верификатор

Поведение сети описывается своими политиками. Описать поведение сети можно анализируя программы контроллера, т.н. верификация потоков управления, или анализируя установленные правила коммутации, верификация потоков данных. Для решения задачи анализа программ контроллера приходится рассматривать все потенциальные состояния сети, в которые может привести множество приложений контроллера, что в условиях отсутствия общего стандарта программирования на уровне NBI (North Bound Interface) — практически сложно реализуемая задача. Анализ потоков данных — более достижимая задача. К таким средствам анализа потоков данных относится и средство VerMont [8], разработанный в Центре прикладных компьютерных сетей.

VerMont решает задачу верификации (проверки) в оперативном режиме отправляемых с контроллера команд реконфигурации сети на предмет непротиворечия с политиками безопасности поведения сети. Средство становится в разрыв между контроллером и коммутатором, моделирует результат выполнения команды реконфигурации и, в зависимости от результата соответствия принятым в сети политикам, пропускает команду или блокирует ее выполнение. Дополнительным удобством верификатора является то, что для решения задачи верификации политик маршрутизации в программно-конфигурируемых сетях может быть использовано две модели: статическая и динамическая.

Статическая модель предназначена для анализа статических свойств ПКС, в ней не принимается во внимание контроллер и недопустимо внесение изменений в таблицы маршрутизации контроллеров. Данная модель имеет следующие преимущества:

1. Простота устройства: семантика модели определяется на основе бинарных отношений.

2. Расширяемость модели: модель позволяет без особых трудностей включать в себя более сложные средства коммутации пакетов.

3. Композиционные свойства модели: семантика составных элементов определяется при помощи простых средств композиции булевых функций.

Динамическая модель описывает поведение контроллера и коммутаторов, которые взаимодействуют с контроллером по каналам управления. Данное решение базируется

на теории автоматов. В этой модели контроллер представлен в виде автомата, который получает на вход сообщения от коммутатора и генерирует в качестве результата последовательность команд для реконфигурации ПКС. В данном случае, в качестве ограничения для множества допустимых состояний автомата выступают валидные заголовки пакетов. Данная модель имеет следующие преимущества:

1. Простота устройства: простота описания контроллера с помощью теории автоматов.

2. Расширяемость модели: учитывается ситуации контроля ПКС сети с помощью нескольких контроллеров.

3. Сочетаемость модели: данная модель сочетается с традиционными моделями описания распределенных сетей.

Непротиворечивая реконфигурация сети. Данное решение подразумевает, что перед внесением реконфигурационных изменений идет предварительная проверка на непротиворечивость данной операции реконфигурации. В данном случае итоговая реконфигурация будет применена в случае, если итоговая реконфигурация будет полностью непротиворечивой. Требование покрывает согласованность потоков и пакетов в целом.

Требовательность к ресурсам и масштабируемость. VerMont спроектирован для работы в сетях, состоящих из не более чем 30 коммутаторов и обслуживающих не более 1000 потоков с интенсивностью обработки не более 1 OF команды секунду. Показатели количества коммутаторов и потоков могут быть увеличены на порядок за счет снижения интенсивности обработки OF команд на эту же величину.

2.5 Метод сериализации

Развиваемый в работе метод базируется на решении аналогичных проблем, возникающих в системах управления баз данных [9]. В данном случае, проводятся аналогии понятий присущих ПКС и СУБД. Основная идея заключается в применении концепции блокировок СУБД для решения проблем реконфигурации и построения плана выполнения конкурирующих запросов. Во время реконфигурации ведется блокировка элементов сети, в результате которой возникновение противоречивых правил на коммутаторах невозможно, так как блокировка не позволяет выполниться операциям. Рассмотрим понятие блокировки для ПКС. Для начала введем понятие процедуры реконфигурации.

Назовем процедурой реконфигурации вектора $P^i = ((O^i_1, e^i_1), \ldots, (O^i_n, e^i_n))$, где O^i_k — операция; e^i_k — объекты реконфигурации k-й операции в i-й процедуре. Это означает, что на некотором шаге k некой процедуры реконфигурации P были последовательно выполнены операции O_1-O_n над, соответственно, объектами e_1-e_n. В данном определении

	Верификатор	TSN	Тегирование	Хранение нескольких конфигураций	Сериализация
Согласованность пакетов	Соблюдается	Соблюдается	Соблюдается	Соблюдается	Динамически настраивается
Согласованность потоков	Соблюдается	Соблюдается	Соблюдается	Соблюдается	Динамически настраивается
Отсутствие противоречий	Соблюдается	Соблюдается	Не соблюдается	Не соблюдается	Динамически настраивается
Требовательность к ресурсам по памяти	Требовательна	Требовательна	Требовательна	Не требовательна	Не требовательна
Масштабируемость по количеству элементов сети	Тяжело масштабируема	Тяжело масштабируема	Тяжело масштабируема	Тяжело масштабируема	Масштабируема
Масштабируемость по количеству запросов	Тяжело масштабируема	Легко масштабируема	Тяжело масштабируема	Тяжело масштабируема	Динамически настраивается
Возможность построения динамического плана	Невозможно	Невозможно	Невозможно	Невозможно	Возможно построение

Таблица 1: Сравнение существующих решений

под объектами понимаются сущности графа сети (хосты, коммутаторы, маршрутизаторы и т.д.). Операции же — это команды реконфигурации сети.

Процедура $P = ((O_1, e_1), \ldots, (O_n, e_n))$ заблокировала объект e на шаге i [9], если выполняются следующие условия:

- $\exists j \leq i : O_j = lock, e_j = e$;
- $\nexists k : j < k < i, O_k = unlock, e_k = e$

Непротиворечивая реконфигурация сети. Использование подхода сериализации (упорядочивания) в плане запросов ведет к обеспечению непротиворечивой работы сети при условии отсутствия аномалий, аналогичных для работы реляционных СУБД.

Требовательность к ресурсам и масштабируемость. Предлагаемый подход отвечает условиям требований к ресурсам и масштабируемости, как доказавший свою применимость на практике.

2.6 Сравнение результатов

Как мы можем видеть из таблицы 1, TSN соответствует параметрам безопасности: согласованность пакетов, согласованность потоков, отсутствие противоречий, а также данное решение хорошо себя показывает в масштабируемости по количеству запросов. Но данное решение больше подходит для интернета вещей. Также оно имеет существенные недостатки: высокая требовательность к ресурсам по памяти и по времени, плохо масштабируемо по количеству элементов сети, отсутствие возможности построения динамического плана.

По сравнению с TSN, хранение нескольких конфигураций также соблюдает согласованность потоков, согласованность пакетов и отсутствие противоречий, но оно намного менее требовательно к ресурсам по памяти и времени. Основной проблемой

данного метода является масштабируемость. Решение, основанное на СУБД, позволяет динамически настраивать параметры безопасности и масштабируемости. С помощью $\varepsilon-$согласованности можно увеличивать возможное опасное окно реконфигурации, улучшая масштабируемость системы. Также данное решение является единственным, допускающим динамическое построение непротиворечивого плана выполнения конкурирующих запросов реконфигурации сети. Целью данной работы является улучшение и доработка данного метода.

3 Исследование и решение задачи

3.1 Метод сериализации

Предлагаемый метод сериализации состоит в развитии подхода, сформулированного в [9], и адаптации алгоритмов, разработанных в рамках теории упорядочивания Serializability Theory для обеспечения эффективного функционирования транзакционных информационных систем и поддержания глобальной сериализуемости в системах с несколькими базами данных. Суть метода заключается в построении глобально непротиворечивого плана выполнения конкурентных запросов на реконфигурацию ПКС в целом.

3.2 Проведение аналогов понятий СУБД на ПКС

В теории транзакций СУБД основывается на следующих понятиях:

- таблица — набор записей (строк);
- транзакция — совокупность операций над данными (чтения, удаления, вставки, модификации), являющаяся неделимой (атомарной) [10]. Неделимость означает, что либо результаты всех операций, входящих в транзакцию, отображаются в БД, либо воздействие всех этих операций полностью отсутствует.
- блокировка — временное ограничение на выполнение некоторых операций обработки данных. Блокировка может быть наложена на отдельную запись (строку) в таблице, несколько записей, несколько таблиц или всю базу данных.
- аномалия — ситуация в таблице БД, которая приводит к противоречивости БД (хранящиеся данные являются некорректными) или затрудняют работу с БД в целом, например, появление данных-дубликатов.

Существует несколько типов аномалий: Dirty Write, Dirty Read, Lost Update, Fuzzy Read, Phantom, Read Skew, Write Skew. Рассмотрим понятие аномалии на примере Dirty Read.

Transaction A	Transaction B
write x = 5	
	x' = read x
rollback	
	write x = x' + 1

Таблица 2: Пример аномалии Dirty read

Пусть параллельно выполняются 2 транзакции. До начала их работы поле с обозначением x имеет значение 1. Тогда, если транзакции работают как на табл. 2, транзакция B прочитает незакоммиченные данные. После этого транзакция A сделает откат, то есть значение x = 1. Если после отката транзакции A транзакция B использует прочитанное значение, например в x запишет x' + 1, результат будет такой, будто отработали обе транзакции.

Проведем следующие аналогии:

- пользователь БД – приложение;
- таблица БД – таблица маршрутизации коммутатора;
- строка таблицы БД – запись в таблице маршрутизации;
- транзакция – процедура реконфигурации;
- блокировка в СУБД – блокировка объекта процедуры реконфигурации.

Аналогия понятия аномалии для СУБД и для ПКС проведена в статье [9] для всех приведенных типов аномалий.

3.3 Условия непротиворечивой реконфигурации ПКС

В СУБД транзакция должна обладать свойствами ACID: Atomicity — Атомарность, Consistency — Согласованность, Isolation — Изолированность, Durability — Устойчивость. Введем данные понятия для процедуры реконфигурации.

Каждая процедура реконфигурации соответствует работе одного приложения. Приложение создает последовательность операций – процедуру реконфигурации, которые требуется выполнить. Совокупность этих операций является атомарной, так как создается одним приложением.

Устойчивость — следующее свойство: если таблица коммутатора изменена, и эти изменения сохранены, то введенные правила остаются в таблице коммутации до их последующих изменений. Оно осуществляется средствами коммутаторов.

Изолированность гарантируется применением сериализации с учетом аномалий, рассмотренных в [9].

3.3.1 Понятие согласованности

Согласованность в базах данных — согласованность данных друг с другом, целостность данных, а также внутренняя непротиворечивость. Введём данное понятие для терминов ПКС.

Рассмотрим процедуру реконфигурации $P = ((O_1, e_1), .., (O_n, e_n))$. Введем для нее функции времени выполнения реконфигурации $time(P)$ и получения множества объектов реконфигурации $E(P)$. $time(P) = (t_{st}, t_{en})$, где t_{st} – время начала, а t_{en} – время завершения процедуры реконфигурации P. $E(P)$ – множество объектов, для которых проводится реконфигурация.

Процедуры реконфигурации P_1 и P_2 являются согласованными, если:

$$(E(P_1) \cap E(P_2)) \; and \; ((time(P_1) \cap time(P_2)) \leq \varepsilon) = \emptyset$$

И дополнительно выполнено:

$$\forall P \; \forall packet \in E(P) : TTL^{p}_{t_{en}} > 0$$

где *packet* - пакет, соответствующий шаблону одного из объектов реконфигурации e_j.

В случае если $\varepsilon = 0$, то это - полная согласованность, никаких рисков нет. При росте ε возрастает вероятность появления конфликта. Требование к TTL означает, что реконфигурация должна быть своевременной. В сети должны остаться пакеты, для которых данная реконфигурация предназначалась.

Таким образом, для того чтобы не возникало аномалий, надо потребовать, чтобы операции над одними и теми же участками сети происходили в разное время (в таком случае $\varepsilon = 0$). В данной работе рассматривается именно это требование. Повышение ε означает, что может существовать сегмент времени, в котором на одних и тех же объектах работают в одно время некоторые объекты реконфигурации. Это приведёт к снижению безопасности сети, но повысит скорость обработки.

Сформулируем условия обеспечения согласованности процедур реконфигурации в части времени их выполнения (см. раздел 3.3.1) в зависимости от характеристик потока и объема буфера сетевых устройств контура данных.

3.4 Условие реконфигурации без потерь

Рассмотрим коротко процесс обработки пакетов на коммутаторе с формированием сообщения на контроллер Packet-in и последующей загрузкой недостающих правил. Процесс реконфигурации отдельного коммутатора состоит из следующих стадий:

1. Анализ заголовка пакета.

2. В случае, если для данного пакета правило коммутации не было найдено, формируется сообщение контроллеру packet-in, которое уведомляет последний об отсутствии правила для анализируемого заголовка.

3. Контроллер формирует и отправляет по каналу управления сформированные команды реконфигурации обратно на коммутатор.

4. Команды реконфигурации исполняются контроллером.

В данных условиях потребуются следующие предположения:

Предположение (1). Не ограничивая общности рассуждений, можно считать, что сообщения packet-in на контроллер приходят только от так называемых «граничных» коммутаторов, которые получают входной поток от внешних коммутаторов. При получении такого сообщения, контроллер формирует пакет команд для реконфигурации не только коммутатора, отправившего packet-in, но коммутаторов всего сегмента сети, включая так называемые «внутренние» коммутаторы, то есть те, которые получают входной поток только от коммутаторов, принадлежащих W.

Сформулируем условия для такого способа реконфигурации внешнего по отношению к W окружения коммутаторов, при котором в коммутаторах S_k из множества W за период реконфигурации не сбрасывается ни один пакет.

Следуя описанной выше процедуре реконфигурации и [2] введем следующие обозначения:

- t_k^1 – время анализа заголовка пакета;
- t_k^2 – время прохождения packet-in от коммутатора S_k к контроллеру для принятия решения;
- t_k^d – время формирования контуром управления процедуры реконфигурации для загрузки в коммутаторы;
- t_k^3 – время прохождения команды на реконфигурацию от контроллера до коммутаторов;
- t_k^4 – время загрузки команд на реконфигурацию и модификацию коммутационных матриц;
- v_k – объём буфера коммутатора с номером k;
- f_{ij} – скорость потока данных от коммутатора i к коммутатору j;
- K – количество всех коммутаторов, рассматриваемой подсети;
- L – множество номеров граничных коммутаторов;

- M_i – количество входящих потоков в коммутатор S_i;
- N_i – количество исходящих потоков из коммутатора S_i.

Обозначим период реконфигурации t_k как промежуток времени от начала анализа заголовка пакета до применения новых правил на коммутаторах:

$$t_k = t_k^1 + t_k^2 + t_k^d + t_k^3 + t_k^4 \tag{1}$$

Необходимое условие. *Если в процессе реконфигурации не сбрасывается ни один пакет, то выполнено следующее условие:*

$$\sum_{i \in L} (\sum_{j=1}^{M_i} f_{ij} - \sum_{j=1}^{N_i} f_{ij}) * t_i \leq \sum_{i=1}^{K} v_i \tag{2}$$

Доказательство. Сброс пакетов происходит в случае переполнения буфера, т.е если общее число пакетов в буфере коммутаторов превосходит его размер. Тогда отсутсвие сброса пакетов на каждои коммутаторе можем записать следующим образом: $\forall i$ S_i : $(\sum_j^{M_i} f_{ij} - \sum_j^{N_i} f_{ij}) * t_i \leq v_i$. Левая часть неравенства описывает общее количество накапливаемых пакетов в буфере коммутатора за общее время реконфигурации. Просуммировав данное выражение по всем коммутаторам, получим требуемое неравенство. □

Достаточное условие. *Для существования такого способа реконфигурации коммутаторов W, при котором в коммутаторах S_k, при условии неограниченности пропускных способностей каналов между этими коммутаторами, за период реконфигурации не сбрасывается ни один пакет, достаточно выполнения неравенства 2.*

Доказательство. Так как потоки между коммутаторами из W могут быть любыми, можем обеспечить заполнение буферов каждого коммутатора S_k пропорционально его объему. Тогда объем пакетов, поступивших за время t_k на коммутатор S_k равен:

$$v_k \frac{\sum_{k=1}^{K} f_k t_k}{\sum_{k=1}^{K} v_k}$$

Так как по условию значение дроби ≤ 1, объем пакетов, полученных за время t_k коммутатором S_k меньше доступного объема этого буфера. □

Непосредственно из доказательства следует, что если пропускных способностей каналов достаточно, чтобы для каждого коммутатора обеспечить суммарные входящие потоки, пропорциональные буферам, то условие отсутствия потерь пакетов так же выполнено.

Рассмотрим случай ограниченных пропускных способностей каналов.

Рис. 6: Пример противоречия

Предположение (2). Рассмотрим множество N коммутаторов сегмента сети Ω, в котором выделим множество W коммутаторов количеством, не превышающим любое наперёд заданное значение K, для которого задан суммарный входящий поток Φ_K. Будем считать $K \ll N$, где знак много меньше означает, что количества коммутаторов Ω, не входящих в W, назовём их «внешними» по отношению к W, достаточно, чтобы обеспечить практически любую конфигурацию потоков через коммутаторы множества W, достаточную для прохождения потока Φ_K.

Достаточное условие. Для существования такого способа реконфигурации коммутаторов W, при котором в коммутаторах S_k за период реконфигурации не сбрасывается ни один пакет, достаточно выполнения неравенства 2.

Для данного условия найден контрпример (рис. 6): при времени реконфигурации, равном 1 с. и скоростях потоков, равным пропускным способностям каналов неравенство выполнено, но сброс пакетов, очевидно, происходит. Поэтому требуется разработка дополнительных свойств, которыми должно обладать множество коммутаторов W. Пока эти свойства не уточнены, но так как для конфигураций, представленных в экспериментальном исследовании, утверждение верно, можно говорить о том, что они обладают этими, пока невыявленными, свойствами.

4 Экспериментальное исследование

Целью экспериментального исследования является проверка корректности достаточного условия, сформулированного выше.

4.1 Экспериментальный стенд

Для проверки условия реализовано 2 экспериментальных стенда. Каждый из них представляет из себя сеть, запущенную в среде эмулирования сетей ns3 [11]. Использованы конфигурации BASNET и GRENA с ресурса topology-zoo [12], представленные на рис. 7 и рис. 8 соответственно. Коммутаторы, находящиеся в "левых" частях сетей генерируют трафик и отправляют его в сеть.

Общие условия:

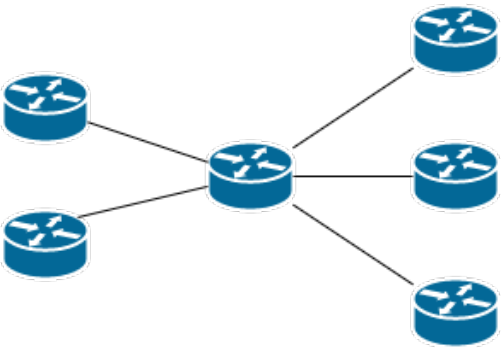


Рис. 7: Конфигурация BASNET

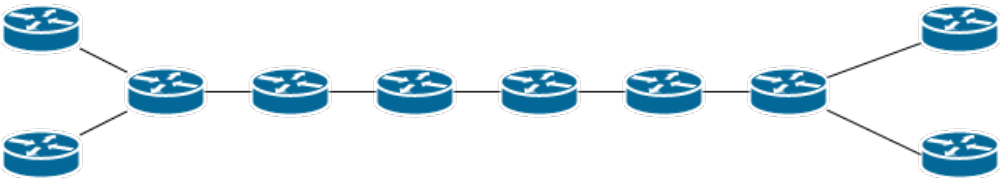


Рис. 8: Конфигурация GRENA

- Сначала запускаются "тестовые" пакеты, по одному на каждый поток. Это требуется для установления записей в таблицах коммутаторов. Для проверки условий необязательно реализовывать ПКС, так как по условию требуется разместить некоторое количество пакетов внутри подсети.
- Размер пакета = 1442 байта.
- Буферы коммутаторов являются выходными, очереди TailDrop.

4.2 Конфигурация BASNET

Рассмотрим эксперимент в конфигурации BASNET. Потоки трафика с коммутаторов $leftup$ и $leftdown$ начинаются одновременно. Коммутатор $leftup$ отправляет пакеты коммутатору $rightup$, $leftdown$ отправляет пакеты коммутаторам $rightmid$ и $rightdown$. Из коммутатора $leftdown$ выходит 2 потока, поэтому пропускная способность канала для него в 2 раза больше. Пакеты отправляются с одинаковой частотой в одинаковые моменты времени.

Опыт 1:

В данном опыте интервал отправки пакетов и скорости передачи данных подобраны так, что пакеты не будут задерживаться в очередях. Скорости передачи данных во всех каналах равны 1444 Bps (здесь и далее использованы значения, кратные размеру пакета для удобства расчетов и немного завышенные, чтобы минимизировать влияние системы), количество пакетов N, направляемых в каждый поток равно 10, 30 или 50 (соответственно, общее количнство отправляемых пакетов равно 30, 90 и 150, так как

потоков 3), интервал отправки равен 1 сек., размер очереди равен объему 1 пакета. Очевидно, что условие выполнено.

Опыт 2:

Параметры данного опыта соответствуют условию, но очередь коммутатора, находящегося в середине, будет максимально заполнена.

- N = 20;
- Left datarate = 1444Bps;
- Right datarate = 1081Bps;
- interval = 1s.

Опыт проводится для размеров очередей 2, 3, 4, 5 и 6.

Опыт 3:

Проведем опыт для разных значений N: от 16 до 21, чтобы найти, для каких достаточно размера очередей = 5, чтобы сброса пакетов не происходило. Остальные параметры такие же, как в предыдущем опыте.

Результаты опытов с конфигурацией BASNET:

Опыт 1:

Независимо от количества отправляемых пакетов, результаты опыта одинаковы. Получены все отправленные пакеты. Пакеты с $leftup$ коммутатора приходят через 1.997 секунд, с коммутатора $leftdown$ через 1.497 и 1.997 секунд после отправки. При сочетании такой скорости передачи данных в правых каналах и частоты отправки, буферы не будут заполняться. Данный опыт показывает, что стенд работает корректно.

Опыт 2:

Согласно условию: $\sum_{i \in L}(\sum_j^{M_i} w_{ij} - \sum_j^{N_i} w_{ij}) * t \leq \sum_{i=1}^{K} v_i$, рассматриваем средний коммутатор: $w_{23} = 2 * w_{13} = 2888Bps, w_{34} = w_{35} = w_{36} = 1084Bps, t = 19s$ (здесь и далее $t = (N-1) * interval$). Получаем $\sum_{i \in L}(\sum_j^{M_i} w_{ij} - \sum_j^{N_i} w_{ij}) * t = 20520B$. $\sum_{i=1}^{K} v_i = 3 * queue\_size * 1442$: для $queue\_size = 4$ получаем 17304B, для $queue\_size = 5$ – 21630B. $queue\_size = 5$ — наименьшее значение, при котором условие выполняется. На рис. 9 видно, что именно с этого значения пакеты не теряются.

Опыт 3:

Результаты опыта 3 представлены на рис. 10. Количества полученных пакетов указаны относительно количеств отправленных. Этим обуславливается разный наклон кривых. Рассмотрим условие 2 для значений N = 17, 18 при $queue\_size = 4$ и 21, 22 при $queue\_size = 5$.

- $queue\_size = 4$:

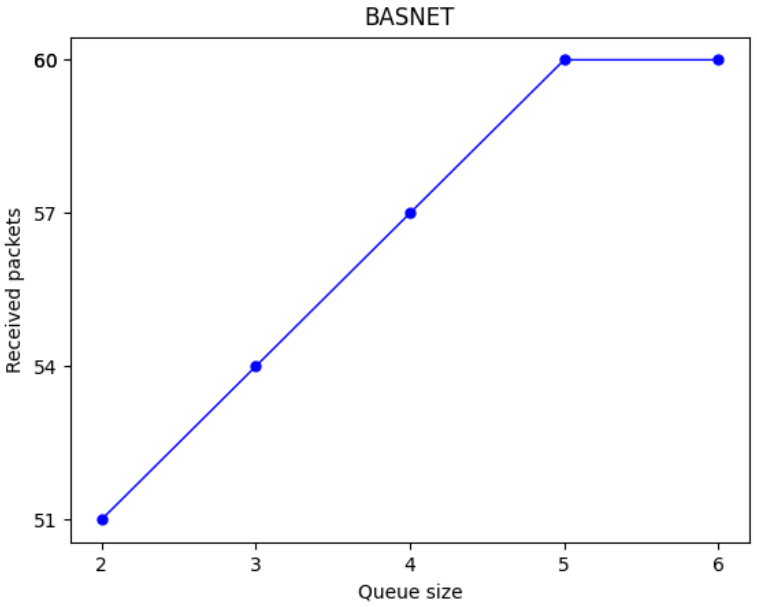


Рис. 9: Результаты опыта 2 BASNET

- N=17: 17280 < 17304;
- N=18: 18360 $\nleq$ 17304;

- $queue\_size = 5$:
  - N=21: 21600 < 21630;
  - N=22: 22680 $\nleq$ 21630.

Результаты, полученные в эксперименте, полностью соответствуют сформулированному условию. Для случаев, где оно выполнено с заданными параметрами, потерь пакетов не наблюдается.

4.3 Конфигурация GRENA

Теперь перейдем к эксперименту в конфигурации GRENA. Пакеты с коммутатора $leftdown$ отправляются на 0.1 секунду позже, чем с коммутатора $leftup$, чтобы избежать переполнения очереди от влияния обработки пакета коммутатором. Коммутатор $leftup$ отправляет пакеты коммутатору $rightup$, $leftdown$ отправляет пакеты коммутатору и $rightdown$. Пакеты отправляются с одинаковой частотой.

Опыт 1:

Для первого опыта параметры подобраны так, что очереди не заполняются. При таком соотношении скоростей передачи данных в каналах выполнено $\sum_{i=1}^{M_i} w_{ij} - \sum_{i=1}^{N_i} w_{ji} = 0$. При этом общее количество отправляемых пакетов равно 2 * N (так как 2 потока).

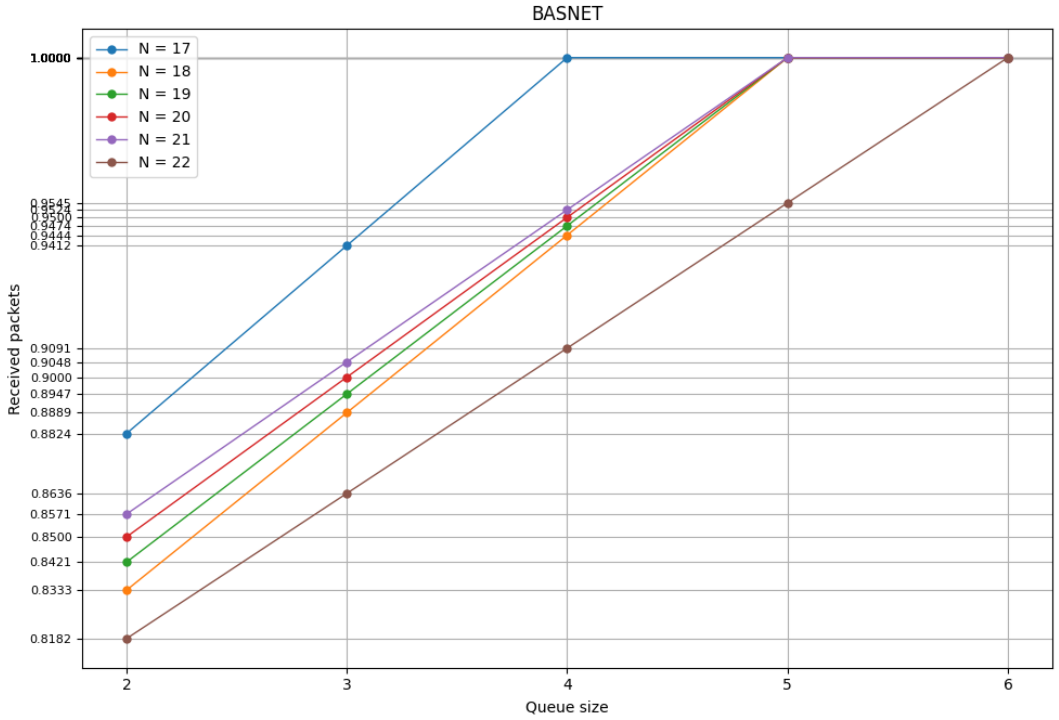


Рис. 10: Результаты опыта 3 BASNET

- N = 10/30/50;
- Left datarate = 1444Bps;
- Mid datarate = 2888Bps;
- Right datarate = 1444Bps;
- Queue size = 1p;
- interval = 1s.

Опыт 2:
Проведем опыт, для следующих параметров:

- N = 50;
- Left datarate = 1848Bps;
- Mid datarates = [3581Bps, 3465Bps, 3350Bps, 3234Bps, 3119Bps];
- Right datarate = 1444Bps;
- interval = 0.78s.

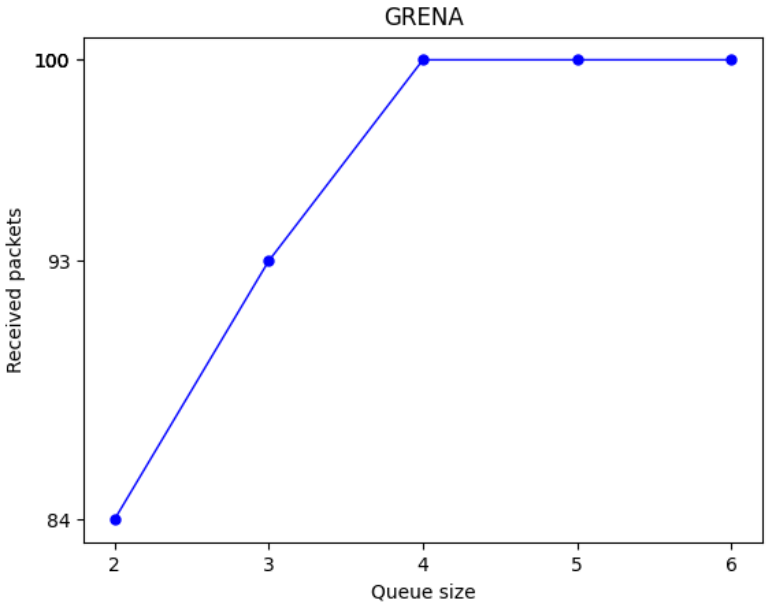


Рис. 11: Результаты опыта 2 GRENA

Опыт проводится для размеров очередей 2, 3, 4, 5 и 6.

Результаты опытов с конфигурацией GRENA:

Опыт 1:

Независимо от количества отправляемых пакетов, очередь ни одного из коммутаторов не переполняется, соответственно получены все пакеты. Пакеты с $leftup$ коммутатора доставляются на коммутатор $rightup$ через 4.494 секунды, с коммутатора $leftdown$ на $rightdown$ через 4.993 секунды после отправки. Данный опыт показывает, что стенд работает корректно.

Опыт 2:

$\sum_{i \in L}(\sum_{j}^{M_i} w_{ij} - \sum_{j}^{N_i} w_{ij}) * t = 30881B$. $\sum_{i=1}^{K} v_i = 7 * queue\_size * 1442$: для $queue\_size = 3$ получаем 30282B, для $queue\_size = 4$ — 40376B. Условие выполняется для $queue\_size \geq 4$. На рис. 11 видно, что именно с этого значения пакеты не теряются.

5 Программная реализация

5.1 Файловая структура

Код реализации можно найти по ссылке `https://git.cs.msu.ru/s02190141/experiment.git`

basnet.cc и grena.cc – экспериментальные стенды, реализующие выше описанные эксперименты.

5.2 Средства разработки

Для реализации стендов были использованы инструментальные средства:

- эмулятор компьютерных сетей – ns3 [11] версии 3.35;
- языки программирования C++ 17, Python;
- текстовый редактор Vim.

5.3 Запуск эксперимента

В каталоге scratch должны находиться файлы basnet.cc и grena.cc.

Для запуска служит следующая команда: ./waf —run "basnet ––args" или ./waf —run "grena ––args"

В качестве аргументов args могут выступать следующие параметры:

- left_datarate, mid_datarate, right_datarate — пропускная способность соединений между устройствами в левых и правых каналах, соответственно. По умолчанию 1444Bps, за исключением $leftdown$ в BASNET: для него 2888Bps.
- interval — период отправки пакетов (в секундах).
- n – количество отправляемых пакетов. Одинаково от каждого левого коммутатора, за исключением $leftdown$ в BASNET: для него количество отправляемых пакетов равно 2*n.
- q – размер буферов коммутаторов.

Пример запуска: ./waf ––run "basnet ––n=15 ––q=5"

Заключение

В работе получены следующие основные результаты:

1. На основе обзора существующих решений проблемы реконфигурации ПКС обоснован выбор метода сериализации.
2. Сформулированы и доказаны условия непротиворечивой реконфигурации ПКС.
3. Выполнена экспериментальная реализация для проведения экспериментов на базе эмулятора компьютерных сетей ns3.
4. На данном тестовом стенде были проведены эксперименты, осуществляющие проверку выполнения условий.

Список литературы

[1] *Смелянский Р.Л., Антоненко В.А.* Концепции программного управления и виртуализации сетевых сервисов в современных сетях передачи данных, учебное пособие. / Антоненко В.А. Смелянский Р.Л. — Курс, 2019.

[2] *Foundation, Open Networking.* — OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04), 2012. — June.

[3] *R., Kletzander.* A testbed for researching conflicts in SDN. / Kletzander R. // *Bachelor's thesis.* — 2017.

[4] *A. A. Grusho, I. Yu. Teryokhina.* Consistency analysis of software-defined networks reconfiguration / I. Yu. Teryokhina A. A. Grusho // *Sist. Sredstva Inf.* — 2017. — Vol. 27, no. 3. — Pp. 12–22.

[5] *Barton, R.* Management of IEEE 802.1Qci Security Policies for Time Sensitive Networks (TSN) / R. Barton, M. Seewald, J. Henry // *Technical Disclosure Commons.* — 2018.

[6] Abstractions for Network Update / M. Reitblatt, N. Foster, J. Rexford et al. // *SIGCOMM.* — 2012.

[7] Consistent updates for software-defined networks: Change you can believe in! / M. Reitblatt, N. Foster, J. Rexford, D. Walker // *HotNets.* — 2011.

[8] VERMONT – средство верификации программно-конфигурируемых сетей / В. А. Захаров, В. С. Алтухов, В. В. Подымов, Е. В. Чемерицкий // *Научно-технические ведомости СПбГПУ.* — 2015.

[9] Безопасная автоматическая реконфигурация облачных вычислительных сред / А.А. Грушо, М.И. Забежайло, А.А. Зацаринный, В.О. Писковский // *Системы и средства информ.* — 2016. — Т. 26, № 3. — С. 83–92.

[10] *Тарасов, С.В.* СУБД для программиста. Базы данных изнутри. / С.В. Тарасов. — М.: СОЛОН-Пресс, 2015. — С. 11–13, 214–232.

[11] *NS-3.* Network simulator 3. https://www.nsnam.org/. — [Дата обращения: 20.05.2022].

[12] The Internet Topology Zoo. http://www.topology-zoo.org/. — [Дата обращения: 20.05.2022].

[13] Experiment. https://git.cs.msu.ru/s02190141/experiment.git. — [Дата обращения: 24.05.2022].