Об одном из методов балансировки программно-конфигурируемых сетей

Никитинский М.А.*
Программист-аналитик
ООО «Энергия-Инфо»
Ярославль, Россия
man@a-real.ru

Чалый Д.Ю.**
Доцент кафедры Теоретической информатики
Ярославский государственный университет им. П.Г. Демидова
Ярославль, Россия
chaly@uniyar.ac.ru

Балансировка сетевого трафика является одним из важных элементов построения и организации отказоустойчивости в коммуникационных сетях. На данный момент существует множество алгоритмов балансировки, как для стандартных подходов организации сетей, так и для программно-конфигурируемых сетей (ПКС). Нами разрабатывается асимметричный транспортный протокол, одним из его свойств является возможность использовать метод anycast для параллельной связи с несколькими серверами. В данной статье будут рассмотрены возможные варианты применения семейства асимметричных транспортных протоколов для улучшения балансировки в ПКС.

Ключевые слова — асимметричный транспортный протокол, программно-конфигурируемая сеть, алгоритм балансировки, интернет контроль сервер, интернет шлюз

I. Введение

С каждым годом все больше компаний и разработчиков вливаются на рынок сетевых технологий в области ПКС. Этот факт обусловлен тем, что применяя подход ПКС к построению сетевой инфраструктуры предприятия, достигается значительная экономия средств организации. Финансовая составляющая выражена в снижении стоимости сетевого оборудования, уменьшении затрат на потребляемую электроэнергию, а также в сокращении количества персонала обслуживающего сетевую инфраструктуру предприятия. К ведущим организациям в области разработок ПКС можно отнести следующие CISCO SYSTEMS, International Business Machines, Hewlett-Packard, Nippon Electronics Corporation и другие, в России лидером разработок является центр прикладных исследований компьютерных сетей (ЦПИ КС).

В целом большинство проведенных за все время исследований можно охарактеризовать как разработку универсального контроллера для ПКС. Однако не менее важной частью являются коммутаторы и действующие коммуникационные протоколы. Последние играют наиболее важную роль на конечных узлах сети. В связи с увеличением количества мобильных устройств, высоконагруженных и облачных систем, а также сенсорных сетей возникает задача эффективной работы протоколов непосредственно на конечных узлах сети и создания новых алгоритмов балансировки сетевого трафика. Одним из подходов к решению данной задачи является применение разрабатываемого нами транспортного протокола, в котором управляющие параметры соединения хранятся только на одном конечном узле сети, что позволит использовать новые пути балансировки сетевого трафика.

II. Некоторые алгоритмы балансировки

«Эластичное дерево» [1]. Было предложено разработчиками лаборатории Hewlett-Packard совместно с лабораторией университета Стенфорд для снижения энергозатрат в центре обработки данных (ЦОД). Идея данного подхода состоит в том, чтобы выключать часть маршрутизаторов или снижать их энергопотребление за счет перераспределения нагрузки на физических портах. Данный вид балансировки направлен на максимальную экономию энергозатрат.

Aster*x [2]. Разработан в Стенфордском университете специально под контроллер NOX. В основе данного алгоритма лежит идея балансировки по назначению трафика (например, все http запросы, приходящие на определенный порт, посылать по заранее определенному пути). Также Aster*x имеет возможность отключения балансировки относительно различных типов запросов и не применяет методы балансировки к краткосрочному трафику.

В работе [3] авторы предложили и оценили новый механизм балансировки, основанный на контроле доступа потоков. Рассматривается применение подхода ПКС для организации бесшовного роуминга при передаче данных в мобильных сетях, в конечном итоге разгружаются основные сети, увеличивается потенциал каждого потока и расширяются возможности пользователей посредствам сокращения времени ожидания и количества отброшенных пакетов. Наиболее поразительными оказались результаты, вероятностный подход снизил процент неудовлетворенных пользователей почти в пять раз. Созданная авторами модель показала прирост в 237%

* Работа поддержана грантом Министерства образования и науки РФ ID RFMEFI57614X0105.
**Работа поддержана грантом РФФИ №14-01-31539 мол_а.

в плане распределения ресурсов между потоками. Кроме того, было отмечено резкое сокращение числа снижения скорости на 300% по сравнению с аналитической моделью, а также почти на 520% меньше по сравнению с отсутствием какой либо балансировки в сети. В целом результаты данной статьи, основанные на аналитической модели, показали преимущество балансировки в контексте ПКС.

Выше представлены только некоторые алгоритмы балансировки в ПКС. Если рассматривать все алгоритмы, то можно разделить их по назначению применения: энергосберегающие, учитывающие политики QoS (Quality of Service), для мобильных устройств, для высоконагруженных систем, для балансировки при различных атаках. А также алгоритмы балансировки можно разделить на проактивные (когда правила балансировки заданы заранее) и реактивные (устанавливаются для каждого вновь пришедшего потока). В данной статье мы рассмотрим вариант проактивного алгоритма балансировки на четвертом уровне (Layer 4, L4) сетевой модели OSI/ISO (open systems interconnection / international organization for standardization) для высоконагруженных серверов.

III. Асимметричный транспортный протокол

Главной задачей протокола TCP является надежная и эффективная передача данных между конечными системами через ненадежную среду передачи - коммуникационную сеть, которая может терять, переупорядочивать и искажать передаваемые данные. Для простоты будем считать, что передача ведется в одном направлении, и будем называть *сервером* сторону, передающую данные, а *клиентом* - принимающую. При этом каждый передаваемый байт данных уникально пронумерован возрастающей последовательностью чисел. Надежность передачи данных обеспечивается при помощи механизма кумулятивных подтверждений - успешное получение каждой порции данных (которая также называется сегментом), переданных сервером, должно быть подтверждено клиентом. В том случае, если за определенное время подтверждение не пришло, сервер повторно передает данные [4, 5]. Эффективность передачи заключается в том, чтобы использовать все доступные ресурсы коммуникационной сети, не допуская ее излишних простоев и, по возможности, ее перегрузки. С другой стороны, для обеспечения надежной и эффективной передачи данных сервер и клиент обязаны сохранять информацию о соединении до ее завершения. Для этого используется блок TCB (Transmission Control Block) содержащий номера сокетов отправителя и получателя, флаги безопасности и приоритета для данного соединения, указатели на текущий сегмент и очереди повторной посылки. Такая распределенная организация взаимодействия между сервером и клиентом приводит к следующим проблемам:

- Для хранения и обработки состояния соединения требуются ресурсы, как на стороне сервера, так и на стороне клиента.
- Ограниченно количество одновременно подключаемых клиентов к серверу, из-за ограниченности ресурсов.
- Существует возможность DDoS-атак (Distributed Denial of Service) на сервер, которые могут вызвать нарушение работы сервера.

Таким образом, возникает задача модификации транспортного протокола с целью его более эффективной работы непосредственно на конечных узлах сети, по возможности, без существенного снижения производительности передачи данных. Одним из подходов к решению этой задачи является протокол Trickles [6].

Протокол Trickles был предложен в 2005 году в Корнельском университете США. Основным отличием протокола Trickles от семейства протоколов TCP является перенос всех управляющих параметров на сторону клиента, при этом серверная часть не хранит информации о транспортном соединении. Транспортный протокол, работающий по такому принципу взаимодействия сервера с клиентом, мы будем называть *асимметричным* протоколом или протоколом с *нераспределенным состоянием* соединения.

Сервер в протоколе Trickles полностью имитирует поведение сервера TCP New Reno [7]. Для контроля соединения сервер Trickles должен обладать управляющей информацией, но так как он не хранит ее, информацию о состоянии соединения передает ему клиент в каждом новом сегменте (рис. 1). Обработав запрос, сервер закрывает соединение.

Trickles соединение состоит из запросов к серверу и ответов клиенту, причем одновременно может выполняться несколько таких запросов. Мы будем называть продолжением потока (continuation) сегмент, следующий от сервера к клиенту и обратно. Последовательность продолжений потока образует элементарный поток (trickle). Таким образом, можно говорить, что в процессе передачи данных происходит параллельная работа нескольких элементарных потоков (рис. 2). Во время транзита в сети каждый элементарный поток независим от других элементарных потоков, их синхронизация происходит только на стороне клиента.

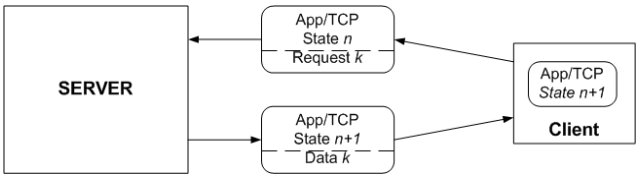

Рис. 1. Передача сегментов в протоколе Trickles.

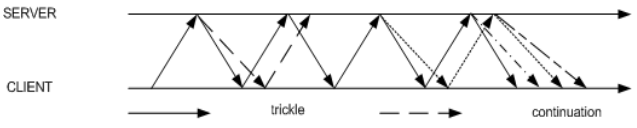

Рис. 2. Continuation и trickle.

Опишем алгоритм управления потоком протокола Trickles, так как именно эта часть протокола в большей степени влияет на его производительность. Этот алгоритм работает отдельно для каждого элементарного потока в том смысле, что когда сегмент, принадлежащий некоторому потоку, находится в транзите, в нем хранятся все необходимые параметры соединения. При этом именно серверная часть протокола преобразует, эти параметры и предпринимает дальнейшие действия по управлению элементарными потоками. Когда сегмент приходит на сервер, существует три возможных варианта развития событий: сервер отправляет один сегмент с данными в ответ на запрос, продолжая поток; сервер увеличивает количество элементарных потоков; сервер уничтожает поток, ничего не отправляя в ответ на запрос. Так как один элементарный поток в сети представлен одним сегментом, то количество работающих элементарных потоков представляет собой текущую оценку протоколом пропускной способности сети.

Механизм управления элементарными потоками имеет три режима, данный механизм стремится соответствовать аналогичному алгоритму в TCP New Reno: slow start/congestion avoidance, fast recovery и retransmit timeout.

В режиме slow start/congestion avoidance сервер Trickles [6] ассоциирует каждый пакет с номером запроса k, и принимает решения о продолжении или разделении потока, используя функцию TCPCwnd(k).

Переход в режим fast recovery осуществляет клиент, так как сервер не хранит информацию о состоянии соединения. Клиент, получив пакеты с номерами k+1 и k+2, считает, что пакет с номером k был потерян, и передает серверу запросы, в которых SACK-блоки [8] не содержат номер k. SACK-блок (selective acknowledgments) это структура данных содержащая все номера полученных пакетов. Сервер, получив такой запрос, уменьшает в два раза количество trickle в сети. Если запрос имеет номер k+1 и SACK-блок не содержит номер k, то сервер в ответ отправляет потерянный пакет с номером k. Получив подтверждение о том, что потеря была восстановлена успешно, сервер пересчитывает управляющие параметры соединения и отправляет их клиенту. Клиент, получив новые параметры соединения, считает, что режим fast recovery закончен, устанавливает новые параметры соединения и переходит в режим slow start/congestion avoidance. Таким образом, время работы режима fast recovery равно 2RTT (Round Trip Time).

Если произошло две и более потери, то клиент Trickles переходит в режим retransmit timeout. При этом сервер, получая пакет в котором SACK-блок имеет две и более потери, прекращает входящий trickle. После того как срабатывает таймер повторной передачи, клиент отправляет запрос на восстановление всех потерянных пакетов. Восстановив все потери, клиент изменяет, управляющие параметры соединения и переходит в режим slow start/congestion.

Такое изменение модели работы транспортного протокола имеет ряд преимуществ:

- Серверная часть может быть реплицирована между несколькими физическими устройствами, так как в передаваемых сегментах содержится вся необходимая информация о соединении (т.е. существует возможность одновременно работать с несколькими копиями одного сервера).
- Увеличение числа обслуживаемых клиентов сервером за счет того, что больше нет необходимости хранить служебную информацию о состоянии соединения.
- В мобильных сетях возможно более быстрое восстановление транспортного соединения при переходе с одной сети на другую, что является непростой задачей при использовании классического протокола TCP.
- Еще одним преимуществом является не возможность создания полуоткрытых соединений.

Очевидно, что класс асимметричных протоколов может облегчить функционирование высоконагруженных серверов, что было показано в [6]. Однако в этой работе не был проведен всесторонний анализ производительности протокола Trickles, что является важной задачей при разработке любого транспортного протокола.

Основными методами исследования производительности транспортных протоколов являются имитационное моделирование, либо натурные эксперименты. Нами был выбран метод имитационного моделирования, так как существует пакет ns-2 [9], в котором есть большое количество моделей различных версий протокола TCP и который является стандартом де-факто при проведении таких исследований.

Система ns-2 является объектно-ориентированным программным обеспечением, ядро которой реализовано на языке C++, а исследуемые модели коммуникационных сетей описываются на языке OTcl. Использование двух языков программирования объясняется тем, что, с одной стороны, модели должны быстро выполняться, что достигается при помощи использования языка C++, а с другой стороны, наличие интерпретируемого языка OTcl позволяет быстро разрабатывать эти модели.

Центральной концепцией системы ns-2 является агент, который является сущностью, выполняющейся на узле сети. В наших экспериментах в роли агента выступала разработанная модель протокола Trickles. Типичным сценарием построения модели протокола в системе ns-2 является следующая процедура:

- Добавить модель пакетов, которые используются протоколом для обмена данными.
- Написать класс агента, который моделирует работу протокола. Для простоты значительное количество транспортных протоколов в системе ns-2 моделируется при помощи двух агентов: один исполняет роль сервера, а второй — клиента.

- Предыдущие два пункта реализуются на языке C++. Чтобы разработанный агент стал доступен для построения исследуемых коммуникационных сетей, необходимо произвести его привязку к языку OTcl.

Создавая структуру пакета, мы опирались на вариант, предложенный в [6].

В системе ns-2 стандартный класс агента Agent включает в себя виртуальные методы, которые моделируют обработку пришедших пакетов и тайм-ауты. Для серверной части агент реализуется переопределением метода Agent::recv(), который обрабатывает пришедший запрос от клиента и сразу же отправляет ответ. Для реализации функционала клиента переопределяется этот же метод, а также метод Agent::timeout() для моделирования тайм-аута повторной передачи.

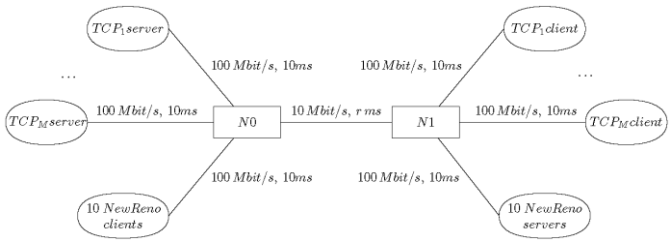

Рис. 3.Конфигурация сети, используемая в экспериментах.

IV. Результаты экспериментов

На основании анализа работы [6] нами была создана оригинальная модель протокола Trickles [10]. Эта модель использовалась для построения модели коммуникационной инфраструктуры, показанной на рис. 3. Данная инфраструктура содержит два промежуточных маршрутизатора N0 и N1, максимальная длина очереди на них равна 50 сегментам. Размер сегмента предполагается равным 1540 байтов. Информационный трафик передается от узлов, помеченных как server, к узлам, помеченным как client. В обратном направлении передаются подтверждения полученных данных.

В каждом эксперименте исследуется производительность нескольких экземпляров одного протокола, обозначенных TCP_1 server, …, TCP_M server. При этом в качестве конкретного экземпляра протокола TCP используются встроенная в пакет ns-2 модель одного из протоколов: TCP Reno, TCP SACK, TCP Vegas, TCP NewReno, а также разработанный нами протокол Trickles. В рамках каждого эксперимента моделируется передача данных в течение 600 секунд. При этом TCP_1 server, …, TCP_M server экземпляры протокола не только конкурируют друг с другом за сетевые ресурсы, но и в сети присутствует также сторонний трафик, затрудняющий передачу подтверждений, который генерируется десятью экземплярами протокола NewReno. Они включаются одновременно и работают в периоды с 150-й до 300-й секунды, а также с 450-й по 600-ю секунду. В данной серии экспериментов изменяются два параметра. Первый параметр — количество одновременно работающих экземпляров исследуемого протокола M, изменяется от 10 до 50 с шагом 1. Вторым параметром является значение задержки передачи r для канала между устройствами N0 и N1, который изменяется от 10 до 120 мс с шагом 10 мс.

Для каждого эксперимента было рассчитано две характеристики производительности. Характеристика goodput показывает, какой объем данных был передан от сервера к клиенту, и для одного соединения рассчитывается по формуле (1):

$$goodput = \frac{\text{send\_data} - \text{retransmitted\_data}}{\text{t}}, \quad (1)$$

где send_data — это общее количество сегментов, которые были переданы протоколом, retransmitted_data — это количество повторных передач, а t — общее время эксперимента. Второй характеристикой является справедливость использования общих ресурсов по Джейну [11], которая определяется по формуле (2):

$$J_{FI} = \frac{\left(\sum_{i=1}^{M} b_i\right)^2}{M \times \sum_{i=1}^{M} b_i^2}, \quad (2)$$

где b — это goodput i-го соединения, а M — количество транспортных соединений, которое разделяет общий ресурс.

Результаты экспериментов показаны на рис. 4. На оси Y в первом столбце представлено количественное значение goodput, во втором столбце fairness, на оси X представлено значение задержки между узлами N1 и N0, а точками обозначено значение goodput или fairness для каждого эксперимента в отдельности, где параметром было количество одновременно работающих экземпляров одного протокола. Как мы можем видеть, протокол Trickles передает сопоставимое количество данных по сравнению с другими протоколами. Оценка по индексу Джейна, показала достаточно честное распределение ресурсов между копиями протокола Trickles. Таким образом, результаты экспериментов на данной топологии показали, что протокол Trickles является менее эффективным протоколом по сравнению с различными версиями протокола TCP, а также имеет ряд недостатков, главным из которых является его нестабильность.

V. NewTrickles и ПКС

Анализ полученных данных в ходе экспериментов, показал, что режим fast recovery работает за время равное 2*RTT в протоколе Trickles, тогда как в большинстве протоколов семейства TCP, данный режим работает за время равное RTT [12]. А значит при больших значениях параметра RTT или при большом количестве trickle, вероятность потери пакетов и переход в режим retransmit timeout становится критичной, что в свою очередь влияет на производительность транспортного протокола в целом.

Нами был предложен новый алгоритм работы режима fast recovery для асимметричных протоколов. Клиент, получив два пакета пришедших не вовремя, считает, что в сети произошла потеря и переходит в режим fast recovery. Он повторно формирует запрос на потерянные данные и отправляет его. Тем самым обеспечивается совместимость нашего алгоритма с протоколом Trickles, в отличие от

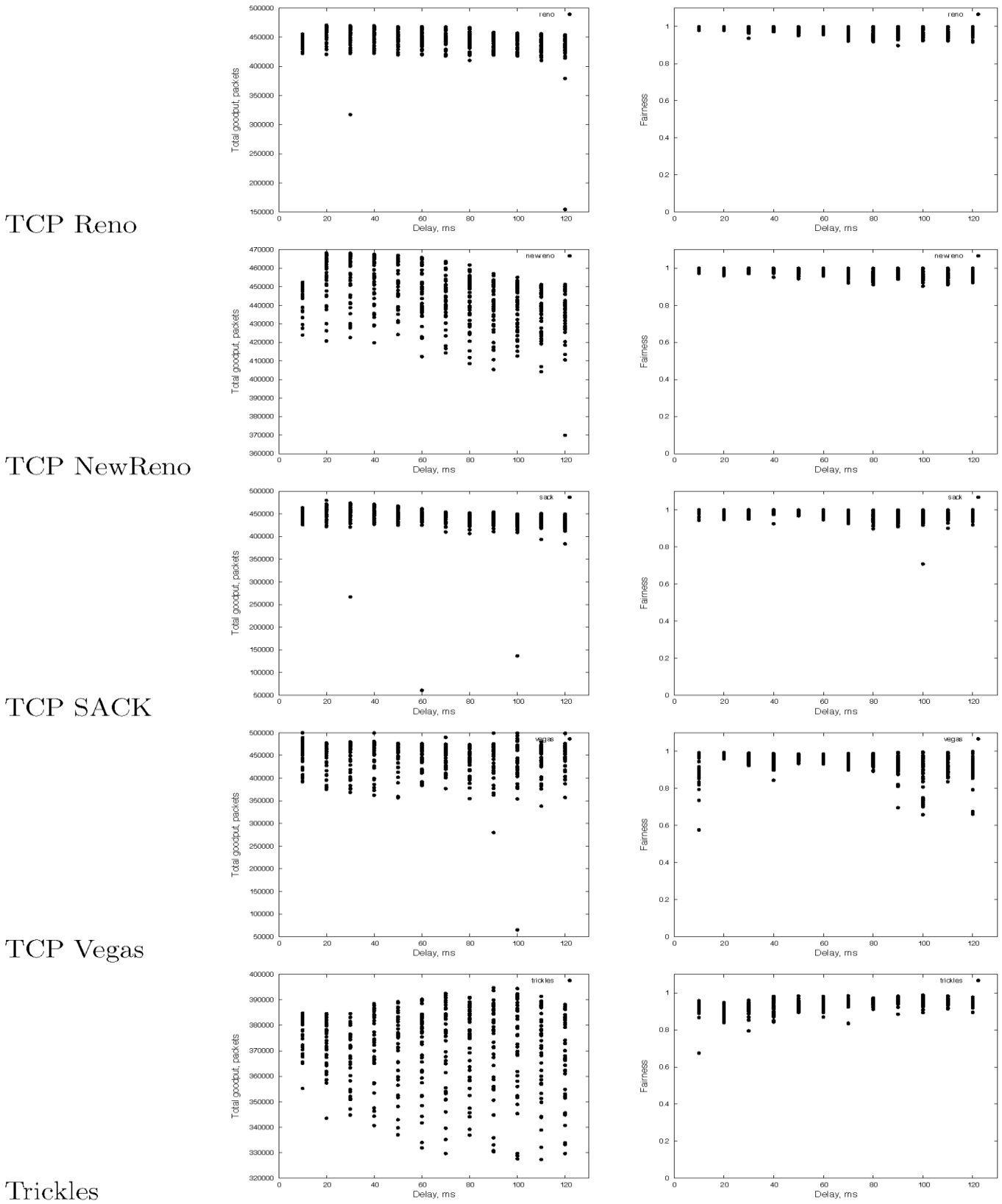

Рис. 4. Результаты экспериментов. Значение goodput представлены слева. Значение справедливого использования ресурсов по Джейну справа.

которого клиент принимает решение об уменьшении количества trickle's в сети, а не сервер. Сервер, получив запрос на определенный кусок данных, отправляет его в ответ, при этом он не знает и не получает никакой информации о том что в сети произошла потеря. Когда приходит восстановленный пакет, клиент рассчитывает новые параметры соединения, а не отправляет запрос на расчет данных параметров серверу. Рассчитав новые значения управляющих параметров соединения, клиент завершает режим fast recovery и переходит в режим slow start/congestion avoidance. Таким образом, мы получаем время работы режима fast recovery равное одному RTT, что соответствует времени восстановления большинства протоколов семейства TCP. Также изменения коснулись других режимов работы асимметричного протокола Trickles, а именно, в режиме slow start/congestion avoidance, клиент принимает решение об увеличении количества trickle в сети, в режиме retransmit timeout, клиент принимает решение об остановке всех trickle в сети.

Нами была разработана модель асимметричного транспортного протокола, использующая выше предложенные режимы работы. Данную модель мы назвали – NewTrickles. При такой организации взаимодействия между клиентом и сервером, сервер становиться простой функцией, получающей запрос, содержащий дескриптор и указание на необходимые байты файла. Относительно протокола Trickles, протокол NewTrickles:

- Снижает нагрузку на коммуникационную сеть за счет принятия решений об увеличении или уменьшении количества trickle в сети. Нагрузка снижается на входящий канал сервера.
- Снижает количество операций выполняемых сервером за счет переноса вычислений на сторону клиента, а значит, уменьшается время обработки каждого входящего пакета, что ведет к увеличению числа обслуживаемых клиентов.
- Снижается в два раза время восстановления в режиме fast recovery.

При этом все преимуществами протокола Trickles присуще NewTrickles, потому что он также является асимметричным протоколом. В настоящее время проводится апробация разработанной модели в сетевом симуляторе ns-3 (Network simulator 3).

Взрывной рост количества сетевых приложений и устройств за последние годы, привел нас к тому, что необходимо не только хранить информацию, но и иметь в сети ее копии. Так как асимметричный протокол не требует жесткого соединения с сервером, то используя метод anycast (доставка до любого ближайшего) возможна одновременная работа с дубликатами одного сервера. При таком взаимодействии клиента с серверами в ПКС мы можем получить следующие преимущества:

- Более сбалансированную нагрузку на сеть, что сможет уменьшить количество отброшенных пакетов в сети.
- Уменьшить загруженность серверов и увеличить количество обслуживаемых клиентов, за счет равномерного перераспределения запросов.
- Невозможность использования DDoS-атак.

На данный момент задача применение асимметричных транспортных протоколов в ПКС не имеет решения, так как существует всего лишь два асимметричных протокола - это протокол Trickles, разработка которого прекратилась в 2008, и протокол NewTrickles, который разрабатывается нами. Используемый асимметричными протоколами метод anycast не имеет однозначной реализации в ПКС. В связи с этим мы видим несколько вариантов применения асимметричных протоколов в динамически развивающейся области информационных технологий, в области ПКС.

Первый, предполагает использование классического коммутатора в ПКС. Данный вариант является весьма трудоемким с точки зрения контроллера ПКС. Так как контроллеру необходимо постоянно следить за состоянием коммутатора и конфигурировать сетевые сообщения для него, заставляя его думать, что он находится в классической сети.

Вторым вариантом является разбор каждого пакета контроллером сети и выбор оптимального пути доставки до серверов. Данный вариант является также трудоемким, он использует больше ресурсов контроллера, чем контроллер тратит на обработку обычного потока.

Третьим вариантом является внесение новых правил на коммутаторы OpenFlow и контроллер ПКС. Предполагается, что если контроллер, обрабатывая первый пакет потока, увидит указание на применение метода anycast или пакет будет относиться к классу асимметричных протоколов, то контроллер ПКС установит набор правил обработки потока для каждого коммутатора. Так как спецификация протокола OpenFlow не имеет четкого указания по реализации метода anycast в ПКС, то реализация данного метода представляется трудоемким процессом объединения множества правил на коммутаторах со сложной динамической поддержкой.

Четвертым вариантом решения предполагается наличие на границе сети устройства способного оценить пропускную способность сети, задержки в сети, динамически подстраиваться под изменяющуюся ситуацию, а также способного разделить входящий поток на два. В качестве такого устройства возможно использование пограничного шлюза в ПКС. При этом шлюз должен использовать контроллер ПКС как инструмент для организации каналов связи между клиентами и серверами. Тогда применение асимметричного транспортного протокола обеспечит дополнительную проактивную балансировку трафика и повысит отказоустойчивость в ПКС.

Данные варианты были рассмотрены при личной беседе с одним из ведущих разработчиков ЦПИ КС на первой международной научно-практической конференции по технологиям SDN и NFV в России, проходившей 27-29 октября 2014 года в Москве. Наиболее перспективными оказались варианты три и четыре.

VI. Заключение

В данной работе мы описали принципы работы асимметричного транспортного протокола Trickles. Рассмотрели его преимущество над протоколом TCP. Изучили базовые принципы создания модели протокола в ns-2. Показали результаты одного из проведенных опытов, а также выявили основной недостаток работы протокола Trickles. На основе выявленного недостатка разработали новый механизм работы быстрого восстановления. А также нами создан модуль асимметричного транспортного протокола для сетевого симулятора.

На данный момент мы плотно сотрудничаем с разработчиками универсального интернет шлюза компании А-Реал консалтинг. Данный шлюз предоставляет следующие возможности: firewall, proxy server, почтовый сервер, встроенный антивирус, обеспечивает поддержку IP-телефонии, многоуровневую фильтрацию трафика, DNS, DHCP и различные счетчики трафика. Мы предполагаем, создать и интегрировать в универсальный интернет шлюз модуль управления контроллером ПКС для малых и средних предприятий. Для взаимосвязи серверов и клиентов предполагается использование асимметричного транспортного протокола. Такое решение обеспечит не только необходимый функционал предприятию, но гибкость сетевого оборудования, а использование асимметричного протокола, сделает продукт более защищенным и привлекательным на рынке коммуникационных технологий.

Список литературы

[1] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 17-17, Berkeley, CA, USA, 2010. USENIX Association.

[2] N. Handigol, S. Seetharaman, M. Flajslik, R. Johari, and N. McKeown. Aster*x: Load-balancing as a network primitive. 9th GENI Engineering Conference (Plenary), November 2010.

[3] Suneth Namal, Ijaz Ahmad, Andrei Gurtov, Mika Ylianttila. SDN Based Inter-Technology Load Balancing Leveraged by Flow Admission Control. Future Network and Services (SDN4FNS), 2013 IEEE , pages 1-5.

[4] Transmission Control Protocol. DARPA Internet Program. Protocol Specification // RFC793,September, 1981. Web site: www.rfc-editor.org

[5] Paxson V., Allman M. Computing TCP's Retransmission Timer // RFC2988,November, 2000. Web site: www.rfc-editor.org

[6] Shieh, A., Myers, A. C., Sirer, E. G. A stateless approach to connection-oriented protocols // ACM Trans. Comput. Syst. 26, 3, Article 8 (September 2008), 50 pages.

[7] Floyd S., Henderson T., Gurtov A. The NewReno Modification to TCP's Fast Recovery Algorithm // RFC3782, April, 2004. Web site: www.rfc-editor.org

[8] Floyd S., Mahdavi J., Mathis M., Podolsky M. An Extension to the Selective Acknowledgement (SACK) Option for TCP // RFC2883, July, 2000. Web site: www.rfc-editor.org

[9] S. McCanne and S. Floyd. ns Network Simulator // http://www.isi.edu/nsnam/ns/

[10] M.A. Nikitinskiy, D.Ju. Chalyy. Performance analysis of trickles and TCP transport protocols under high-load network conditions // Automatic Control and Computer Sciences. December 2013, volume 47, issue 7, pp 359-365.

[11] Jain R. The art of computer systems performance analysis. // Jon Wiley and Sons, 1991.

[12] Никитинский М.А., Новый аналоговый алгоритм восстановления для асимметричных транспортных протоколов // Тенденции развития прикладной информатики: Сборник статей по итогам международной научно-практической конференции. Ярославль, 2013. С. 234-238.