**Степанов Е.П., Войнов Н.А.**

# Динамическое сегментирование транспортных соединений[1]

## Введение

При работе существующих алгоритмов управления перегрузкой производительность TCP соединений деградирует с ростом задержки соединения и увеличением вероятности потери пакетов, не связанной с перегрузкой. Так, с увеличением задержки TCP агенту требуется больше времени, чтобы обнаружить потерю пакетов и провести их повторную передачу. Также TCP агенту требуется больше времени, чтобы занять доступные сетевые ресурсы. В случае, если алгоритм управления перегрузкой использует потерю пакетов в качестве индикатора перегрузки, ошибки передачи нарушают сходимость скорости, ограничиваемой алгоритмом управления перегрузкой, к доступной пропускной способности. [5]

Подход с сегментированием транспортного потока (Split TCP) [1] позволяет решить данную проблему, разбивая соединение на несколько последовательных при помощи прокси сервера. Таким образом, более короткие соединения быстрее реагируют на изменения в состоянии сетевого окружения и более оптимально работают с окном перегрузки.

Однако, такой подход не всегда оказывается эффективным. Улучшение качества исходного соединения зависит от задержки в сети, от количества прокси серверов на пути соединения и от текущей загруженности прокси серверов, через которые проходит это соединение [2]. Для каждого очередного соединения такие характеристики могут быть уникальными. Таким образом, становится актуальной задача разработки и реализации адаптивного метода проксирования TCP соединений, принимающего решение о необходимости и параметрах проксирования, таких как местоположение и количество проксисерверов для каждого соединения.

В данной статье представлен адаптивный подход сегментирования транспортных соединений, выбирающий параметры сегментирования, такие как местоположение и количество прокси серверов для каждого соединения, в зависимости от текущих условий в сети. Проводится экспериментальное исследование эффективности предложенного решения и показывается преимущество данного подхода

[1]Работа выполнена при поддержке РФФИ (проект №18-07-01255-А)

по сравнению с классическим статическим сегментированием в определенных сценариях.

# 1. Split TCP

Подход с сегментированием транспортного потока (Split TCP) [1] предлагает разбить TCP соединение с большим RTT на несколько последовательных, с меньшими RTT, с помощью использования дополнительных узлов — прокси серверов, которые перехватывают новое соединение и создают несколько последовательных. За счет уменьшения RTT на каждом участке, такое решение позволяет быстрее изменить окно перегрузки, отреагировать на потерю пакета и провести его повторную передачу. Также при установке прокси серверов на стыке отличных по качеству участках гетерогенной сети появляется возможность использовать различные алгоритмы управления перегрузкой для разных физических сред, например, для беспроводной и проводной сетей.

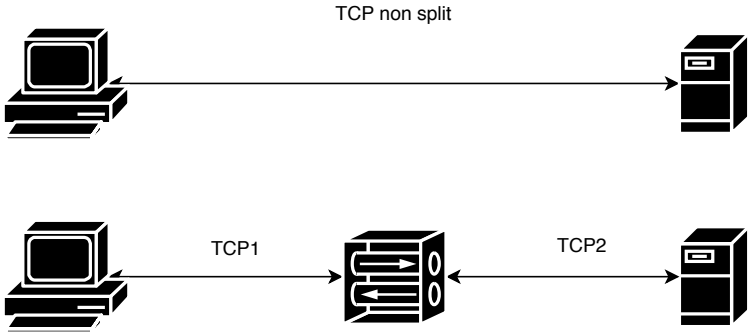


Рисунок 1. Схема работы Split TCP.

Объясним более подробно, как проксирование позволяет увеличить скорость TCP соединения. Рассмотрим соединение, изображенное на рисунке 1. Исходное end-to-end соединение без использования проксирования (обозн. non-split) делится на два последовательных: TCP1 и TCP2.

Положим:

$R_{ns}$ — скорость end-to-end non-split соединения

$R_{sp}$ — скорость split соединения

$R_1$ — скорость TCP1

$R_2$ — скорость TCP2

$T$ — круговая задержка (RTT) для non-split соединения

$q$ — вероятность потери пакета для non-split соединения

$T_2$ — круговая задержка (RTT) для TCP2
$q_1$ — вероятность потери пакета TCP1
$q_2$ — вероятность потери пакета TCP2

Согласно [5] пропускная способность TCP соединения без использования Split TCP подхода может быть оценена, как

$$R_{ns} = \frac{1}{T}\sqrt{\frac{3}{2q}}$$

При использовании Split TCP подхода пропускная способность соединения будет равна наименьшей из пропускных способностей на двух участках $R_{sp} = min(R_1, R_2)$. Предположим, что это участок TCP2.

$$R_2 = \frac{1}{T_2}\sqrt{\frac{3}{2q_2}}$$

Таким образом, преимущество, получаемое при проксировании через один узел, оценивается как

$$\frac{R_{sp}}{R_{ns}} = \frac{T}{T_2}\sqrt{\frac{q}{q_2}} > 1$$

Предполагая низкую вероятность потери пакета в проводных соединениях и гомогенность сети, на которой производится проксирование, формулу часто упрощают до

$$\frac{R_{sp}}{R_{ns}} = \frac{T}{T_2} > 1$$

Следует отметить, что полученная оценка имеет ряд ограничений.

Во-первых, оценка пропускной способности выведена для TCP Reno и может не подойти для соединений, использующих иные алгоритмы управления перегрузкой. Однако, согласно [5] качественная оценка, полученная в формулах выше, остается справедливой для таких алгоритмов, как BIC, Cubic и Compound TCP.

Также при оценке пропускной способности на самом медленном из участков предполагается, что происходит насыщенная передача данных, то есть у источника всегда есть данные для отправки, что может быть не так в случае, если отправитель — прокси-сервер, которому еще не дошли данные от источника. Но, так как данный участок является самым медленным на пути передачи, такая погрешность считается допустимой. [5]

Также стоит отметить, что полученные оценки могут быть неверны для короткоживущих соединений, которые завершаются до окончания фазы медленного старта. Также оценки не учитывают возросшее время на установку соединения, связанное с последовательной установкой соединений на каждом сегменте.

Однако, несмотря на данные ограничения, оценка наглядно показывает преимущество Split TCP подхода. Данный подход позволяет увеличить достигаемую пропускную способность по сравнению с классическим TCP протоколом для широкого класса соединений. Так, при использовании одного прокси-сервера пропускная способность может увеличиться до двух раз при условии гомогенности сети и несколько больше для гетерогенных. Нетрудно заметить, что согласно оценке увеличение количества равномерно распределенных прокси-серверов будет продолжать увеличивать достигаемую пропускную способность вплоть до достижения пропускной способности канала. Конечно, в практической реализации таких результатов сложно достичь из-за накладных расходов, возникающих при проксировании. Впрочем, согласно [2], даже при больших накладных расходах подход split TCP может давать преимущество.

Отметим, что данную оценку несложно расширить на случай нескольких прокси серверов. [5]

$$R_{sp} = \frac{1}{T_k}\sqrt{\frac{3}{2q_k}}$$

$$\frac{R_{sp}}{R_{ns}} = \frac{T}{T_k}\sqrt{\frac{q}{q_k}} > 1,$$

где

$$k = argmax\{i = 1 \dots N, T_i * \sqrt{q_i}\}$$

# 2. Динамическое сегментирование транспортных соединений

Приведенные выше формулы подтверждают практическую выгоду split TCP подхода и позволяют оценивать его возможное преимущество, выбирать расположение и количество прокси-серверов. Однако статический выбор такой конфигурации не всегда остается оптимальным: задержка соединения, служащая основной метрикой выбора таких конфигураций, меняется при различных условиях, таких как увеличение нагрузки на прокси-сервер, увеличение загрузки

очередей на пути соединения, изменение в топологии сети, маршрутах.

Существующие реализации адаптивных методов проксирования [3] обладают рядом ограничений, не позволяющих использовать их в классических сетях.

В качестве динамического метода сегментирования транспортных соединений для классических сетей предлагается следующее решение.

Предполагается, что архитектура предложенного решения строится для некоторого целевого сервера, взаимодействующего с клиентами посредством TCP протокола.

Решение подразумевает наличие нескольких прокси серверов, для возможного терминирования пользовательских соединений, направленных к целевому серверу. Для каждого очередного создания соединения к целевому серверу из данного множества прокси серверов выбирается подмножество, оптимальное с точки зрения оценки.

## 2.1 Общая схема

Схема предложенного решения состоит из нескольких компонентов:

1. Координатор (coordinator). Данный компонент является централизованным контроллером предложенного решения. Он отвечает за хранение метрик, полученных другими компонентами вычислением оптимальных путей и обслуживанием запросов от прокси-серверов. Данный компонент работает в системе в единственном экземпляре (за исключением сценариев отказоустойчивости) для всего множества прокси-серверов, обслуживающих целевой сервер.

2. Health-Checker. Данный компонент занимается вычислением необходимых метрик, таких как RTT и вероятность потерь пакетов между прокси-серверами. Также компонент производит передачу данных координатору. Данный компонент работает на каждом прокси сервере.

3. Точка присутствия (Point of Presence — PoP). Данный компонент занимается непосредственно проксированием соединений через оптимальные пути, вычисленные координатором. Данный компонент работает на каждом прокси сервере.

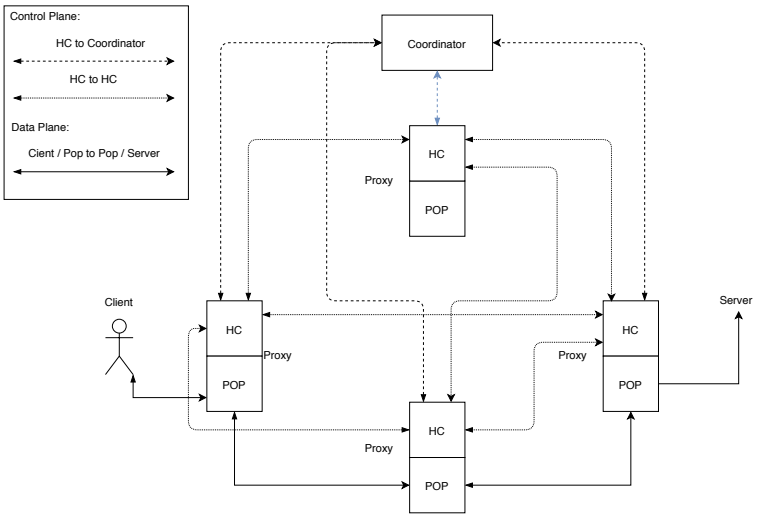


Рисунок 2. Схема предложенного адаптивного метода.

Взаимодействие компонентов схематично показано на рисунке 2.

Из данной схемы видна необходимость попадания запросов клиента на один из прокси серверов, обслуживающих целевой сервер. Предполагается, что это обеспечивает некоторый внешний компонент, не входящий в данное решение. Таким компонентом может служить DNS резолвер, направляющий клиента на ближайший в смысле задержки прокси сервер. Таким образом соединение будет всегда проходить через хотя бы один прокси сервер. Такой подход не всегда будет оптимальным, и обязательное проксирование может давать дополнительные накладные расходы без увеличения достижимой пропускной способности. Однако, вероятность данной проблемы можно существенно снизить, расположив один из прокси-серверов достаточно близко к целевому серверу, например, в том же ЦОД.

## 2.2 Координатор

Координатор содержит в себе два основных подкомпонента:

1. RPC сервер, обслуживающий запросы прокси серверов.
2. Подкомпонент, вычисляющий оптимальные пути от каждого прокси сервера до целевого сервера.

### RPC сервер

RPC сервер реализует следующие удаленные процедуры:

1. `rpc GetNodes(Node) returns (NodeList)`. Данная процедура возвращает список прокси серверов, обслуживающий целевой сервер. Данную процедуру вызывает компонент health-checker, периодически обновляя множество серверов, между которыми считаются необходимые метрики.

2. `rpc RegisterNode(Node) returns (Empty)`. Данная процедура регистрирует обслуживающий прокси сервер в системе. Вызывается компонентом "PoP" при начале работы.

3. `rpc PutMetric(Metric) returns (Empty)`. Данная процедура доставляет вычисленные метрики на координатор. Вызывается компонентом "HealthChecker" периодично.

4. `rpc GetNextHop(Node) returns (Node)`. Данная процедура возвращает адрес следующей точки внутри вычисленного оптимального пути для прокси сервера, посылающего запрос. Вызывается компонентом "PoP" периодично.

**Вычисление оптимальных путей**

Также координатор отвечает за вычисление оптимальных путей от каждого прокси сервера до целевого сервера на основе вычисленных метрик. Из всевозможных путей выбирается оптимальный, согласно оценке выше. Пути вычисляются измененным алгоритмом Беллмана-Форда. Псевдокод решения приведен ниже. Предполагается, что ассоциативный массив Metrics содержит в себе произведение RTT и вероятности потерь пакетов для каждой пары прокси серверов.

```
for v in Nodes:
d[v] = inf
next_hop[v] = DEST
d[DEST] = 0
for i in range(len(Nodes)):
    for (u,v) in Edges:
    if max(d[u], Metrics(u,v)) + delta < d[v]:
        d[v] = max(d[u], Metrics(u,v)) + delta
        next_hop[v] = u
return d, next_hop
```

Таким образом вычисляются все оптимальные пути от всех прокси-серверов до целевого сервера. Вычисленная информация сохраняется в оперативную память, откуда ее получает при обработке

запроса `GetNextHop` RPC сервер. Данный процесс запускается периодично, перевычисляя оптимальные пути.

### 2.3 Health-Checker

Данный компонент занимается оценкой метрик между прокси серверами. Он начинает свою работу с получения списка прокси серверов, периодически обновляя эту информацию. Для каждого из полученных серверов и целевого сервера запускается отдельный процесс, вычисляющий необходимые метрики (RTT и вероятность потерь пакета) и периодично отправляющий полученную информацию, усредненную за некоторый промежуток времени, на RPC сервер.

### 2.4 Точка присутствия

Данный компонент занимается непосредственно проксированием пользовательских соединений к целевому серверу. При получении очередного запроса на подключение процесс локально получает информацию о следующем узле на вычисленном оптимальном пути и подключается к нему, передавая данные, отправляемые клиентом. Таким узлом может быть как очередной прокси сервер, так и сам целевой сервер. Информация о следующем хопе периодически обновляется с помощью удаленной процедуры `GetNextHop(Node)`.

## 3. Экспериментальное исследование

В качестве демонстрации работы предложенного метода были проведены следующие эксперименты.

## 3.1 Схема тестового стенда

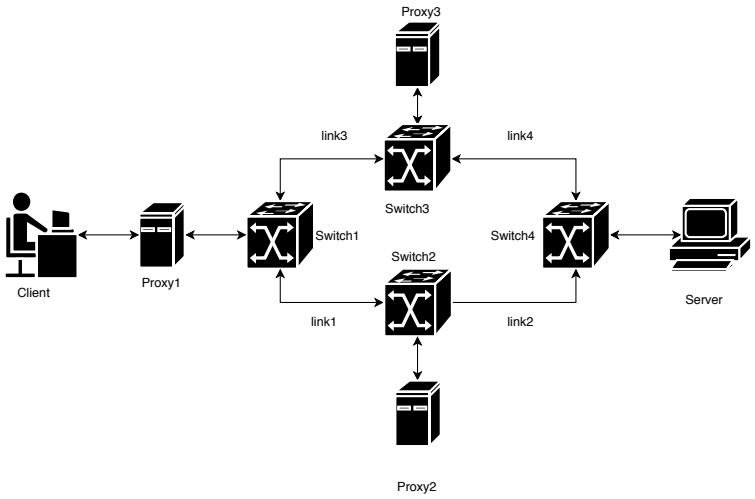


Рисунок 3. Схема тестового стенда.

Для проведения экспериментов была построена следующая топология сети. Для эмуляции представленной выше топологии использовались средства network namespaces и veth. Для задания характеристик линий связи использовались средства эмуляции Tc, Netem, TBF. Для эмуляции работы коммутаторов использовался программный коммутатор OVS. На данной топологии были заранее заданы таблицы маршрутизации: так, путь от Proxy1 до Server проходит через link1, link2. Остальные маршруты соответствуют единственным кратчайшим путям по метрике hop-count.

На прокси серверах, согласно предложенному методы были запущены компоненты Health-Checker и Point-of-Presence, компонент Coordinator был запущен на целевом сервере.

Клиентское приложение в цикле получало с HTTP-сервера файлы различного размера.

В экспериментах оценивалось время получения файлов различного размера, хранящихся на целевом HTTP-сервере. Для получения достоверной оценки, клиент выполнял GET запрос 100 различных файлов одного размера. В качестве времени получения файлов, указанного на графиках, выбиралось медианное время.

Указанные замеры проводились для следующих типов соединений:

1. Соединение без использования проксирования — в данном случае клиент отправлял GET запрос указывая IP адрес целевого

сервера.

2. Соединение с использованием динамичного проксирования — в данном случае клиент отправлял GET запрос указывая IP адрес ближайшего прокси-сервера, дальнейший путь и необходимость проксирования в других точках выбиралась предложенным адаптивным методом.

3. Соединение с использованием статического проксирования — в данном случае клиент также отправлял GET запрос, указывая IP адрес ближайшего прокси-сервера, однако во время начала экспериментов дальнейший путь и необходимость проксирования в других точках настраивалась статически, с помощью добавленных RPC процедур `rpc PutCustomNextHop(NextHop) returns (Empty)` и `rpc DropCustomNextHops(Empty) returns (Empty)`.

## 3.2 Результаты экспериментального исследования

### Эксперимент 1

В данном эксперименте статическое проксирование проводилось через цепочку Proxy1-Proxy2-Server. Фоновая нагрузка на сеть не создавалась.

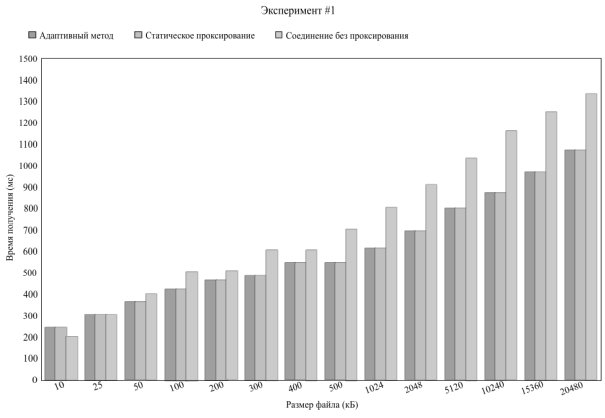


Рисунок 4. Результаты эксперимента 1.

Из графика с результатами замеров видно, что в случае стабильного состояния сети статическое и динамическое сегментирование дают одинаковые результаты, позволяя уменьшить время загрузки файлов по сравнению с соединением без использования проксирования.

### Эксперимент 2

В данном эксперименте статическое проксирование также проводилось через цепочку Proxy1-Proxy2-Server. Создавалась фоновая нагрузка с помощью генератора трафика iperf между Proxy2 и целевым сервером. Таким образом эмулировалась увеличенная клиентская нагрузка на Proxy2.

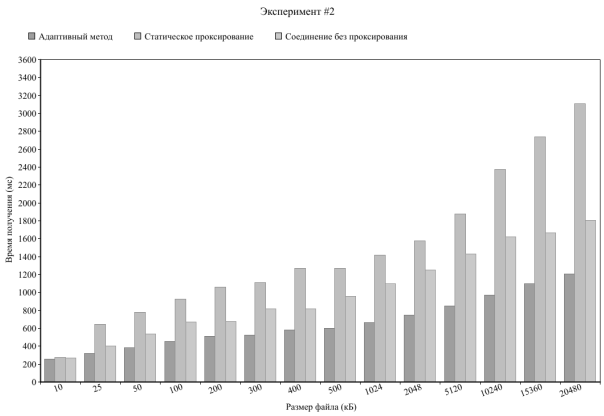


Рисунок 5. Результаты эксперимента 1.

При создании фоновой нагрузки статическое проксирование за счет повторного прохождения через узкий участок сети дает результат хуже, чем соединение без использования проксирования. Адаптивный метод, обнаруживает изменение задержки на участке между Proxy2-Server и выбирает цепочку Proxy1-Proxy3-Server в качестве оптимального пути. Таким образом адаптивный подход позволяет достичь лучших результатов при возникновении перегрузки в сети.

# 4. Заключение

В ходе проделанной работы был предложен и разработан метод адаптивной сегментации транспортных соединений для классических сетей, принимающий решение о необходимости и параметрах проксирования, таких как местоположение и количество прокси-серверов для каждого соединения, на основе аналитической модели.

Также было проведено экспериментальное исследование эффективности предложенного решения по сравнению с классическим сегментированием транспортных соединений, которое показано преимущество предложенного метода в сценариях с динамической нагрузкой в сети.

# Литература


1. J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. *Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations.* June 2001, RFC 3135.

2. Siracusano G. et al. *On the fly tcp acceleration with miniproxy* Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization. – ACM, 2016. – С. 44-49.

3. Shimamura M., Ikenaga T., Tsuru M. *Splitting tcp connections adaptively inside networks* IEICE TRANSACTIONS on Information and Systems. – 2012. – Т. 95. – №. 2. – С. 542-545.

4. Siracusano G., Bifulco R., Salsano S. *TCP proxy bypass: All the gain with no pain!* Proceedings of the SIGCOMM Posters and Demos. – ACM, 2017. – С. 88-90.

5. Varma S. *Internet congestion control.* Morgan Kaufmann, 2015.