Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Грабовой Филипп Николаевич

Исследование кэширования TCP RE и его реализация для ОС Linux

Бакалаврская работа

Научный руководитель:
ст. преп. Зеленчук И. В.

Рецензент:
тьютор Ханов А. Р.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engineering

Grabovoy Philipp

Evaluation and implementation of TCP RE caching for Linux

Bachelor's Thesis

Scientific supervisor:
assistant Zelenchuk Ilya

Reviewer:
tutor Khanov Arthur

Saint-Petersburg
2016

Оглавление

Введение

Несмотря на огромное количество существующих Интернет-ресурсов, зачастую возникает ситуация, когда много людей обращаются к одному источнику в сети, запрашивая однинаковую информацию. Это приводит к тому, что часть сетевой инфраструктуры занимается передачей идентичных данных[1]. Кэширование трафика — один из подходов к улучшению этого аспекта, позволяющий уменьшить нагрузку на каналы связи.

На данный момент активно применяются технологии кэширования, работающие на уровнях прикладных протоколов, в особенности кэширование HTTP[14] ввиду популярности этого вида трафика[15]. Вместе с этим развиваются более общие идеи кэширования, основанные на работе с пакетами протокола транспортного уровня TCP. Ввиду этой особенности они способны кэшировать данные разных протоколов прикладного уровня. В этом состоят принципы работы пакетного кэширования TCP RE (*англ.* Redundancy Elimination — устранение избыточности). В статье “Comparison of Caching Strategies in Modern Cellular Backhaul Networks” [17] сравниваются различные подходы к кэшированию на примере трафика мобильных сетей, при этом TCP RE показывает лучший результат. Однако остается ряд открытых вопросов по этой технологии:

1. Насколько эффективно кэшируются различные виды трафика?
2. Как реализовать кэширующие компоненты, где их разумно будет применять?
3. Насколько кэширование будет загружать систему, снижая пропускную способность канала?

Цель данной работы — исследовать кэширование TCP RE и дать развернутые ответы на эти вопросы. В связи с этим программный комплекс состоит из трех компонент: анализатора кэширования TCP RE;

системы генерации и анализа файлов захвата трафика; модулей для ОС Linux, реализующих кэширование TCP RE.

Анализатор предствляет из себя приложение пользовательского уровня. Удобен тем, что позволяет захватывать и анализировать пакеты и из открытого онлайн-соединения и из файлов с записями сессий (pcap-файлы). Последняя функциональность очень важна, т. к. исследование одной и той же сессии через переиспользование pcap-файла упрощает процесс измерения эффективности кэширования.

Автомазированная система генерирации файлов захватов трафика и их последующего анализа упрощает процесс выявления закономерностей в сетевых пакетах, передающих одинаковые данные, позволяет легко корректировать сценарии сессий.

Устройство кэша в виде двух модулей применимо для снижения нагрузки на канал между двумя серверами (на которых они соответственно установлены). Реализация таких кэширующих компонент для ОС Linux и эксперимент на стенде позволят измерить загруженность системы.

Эксперименты направлены на измерение эффективности кэширования, поэтому в ходе их проведения важны два типа показателей: те, которые характеризуют выигрыш от применения алгоритма и те, которые показывают нагрузку на канал связи. Их вычисление требует различных подходов, поэтому они будут измерены отдельно друг от друга.

Выигрышем от применения схемы кэширования TCP RE разумно считать долю сэкономленного трафика. Как ее вычислить? Можно рассматривать отношение $\frac{\#\text{ повторившихся пакетов}}{\#\text{ прошедших пакетов}}$ (здесь и далее под пакетами подразумеваются данные протокола прикладного уровня сетевого пакета), и выбрать в качестве модели вычисления получение одного и того же ресурса (веб-страницы, файла) два раза подряд — самый очевидный сценарий, при котором одинаковые пакеты должны возникать. Но недостаток такой статистики состоит в том, что данные существенно разных размеров одинаково влияют на ее величину. Это может существенно повлиять на результаты, если в ходе эксперимента возникает

какой-то “шумовой” трафик (например, от других одновременно работающих приложений). Возможна фильтрация кэшируемых пакетов по IP-адресам, однако этот способ фактически будет зависим от позиции проведения эксперимента и работы DNS-серверов, разрешающих имя запрашиваемого хоста при использовании CDN (*англ.* Content Delivery Network — сеть доставки контента)[2]. Другой подход состоит в изменении статистики путем добавления весов. Если в качестве веса пакета взять его размер и вычислять показатель $\frac{S_r}{S_t}$, где S_r — суммарный размер повторившихся пакетов, S_t — суммарный размер всех прошедших пакетов, то влияние побочного трафика небольших размеров уменьшится до приемлемого уровня. И в целом такая модель больше сооветствует поведению пользователя — обычно программы, работающие в фоновом режиме, не отключаются при целевом посещении Интернет-ресурсов. Таким образом, эксперименты этого класса состоят в первичном и повторном обращении к ресурсу и измерении выбранной меры экономии трафика.

Нагрузку на канал связи можно оценить путем сравнения двух пропускных способностей: во время сессии с выключенными модулями ядра, осуществляющими TCP RE кэширование, и во время аналогичной по действиям сессии, но уже с включенным функционалом. Таким образом можно оценить затраты на процесс поиска пакета в кэше и изменения данных.

Постановка задачи

1. Разработка анализатора кэширования TCP RE
2. Создание системы автоматизированной генерации и анализа pcap-файлов
3. Разработка кэша TCP RE в виде модулей ядра
4. Измерение эффективности кэширования и нагрузки на канал связи

Обзор подходов к кэшированию

В этой части рассматриваются основные подходы, применяемые для кэширования данных в сети, идеи, на которых они основаны, их преимущества и недостатки.

HTTP Кэширование

HTTP экшировнаие — это вид кэширования, который поддерживается на уровне протокола прикладного уровня HTTP[6] и используется для кэширования веб-трафика. Основан на идентификации данных адресами URL[13].

Схема функционирования HTTP кэша с использованием одного промежуточного кэширующего шлюза представлена на рис. 1. Клиент запрашивает данные по URL адресу, если их нет в кэше, то оттуда происходит запрос на сервер, полученный ответ кэшируется и отправляется клиенту. Затем, если эти данные снова будут запрошены, произойдет проверка идентичности сохраненных версий с серверными (по временным меткам и другим параметрам) и, в случае успеха, данные будут взяты из кэша. В таком случае по каналу связи кэш — сервер передается меньшее количество информации, он разгружается.

Рис. 1: Схема HTTP кэширования с одном шлюзом

Встроенность кэша в протокол очень удобна в ходе разработки интернет-ресурсов, так как позволяет явно указывать параметры кэширования размещаемых объектов. Схема эффективно работает, так как боль-

шое количество современного трафика использует протокол HTTP, однако одинаковые ресурсы, доступные по разным Интернет-адресам, кэшируются как различные объекты. Это происходит при наличии элементов с псевдонимами (*англ.* Aliases); CDN, в этом случае одинаковые ресурсы расположены на разных серверах[3].

Префиксное HTTP кэширование

Префиксное кэширование основано на идентификации ресурса его префиксом. Схема работы также поразумевает наличие промежуточного кэширующего шлюза, как и в случае с HTTP кэшированием, однако момент разрешения пользовательского запроса отличается: кэширующий сервер в любом случае закачивает часть данных ресурса — тот самый префикс — и с помощью них определяет, есть ли этот объект в кэше. В сравнении с HTTP кэшированием у метода появляется возможность решить проблему одинаковых ресурсов с разными URL с помощью запросов и работы с самими данными. Но существенной проблемой подхода является наличие т. н. False Positives — неверных срабатываний, — неточностей, связанных с тем, что по префиксу не удается однозначно определить ресурс и потому возможно получение неверного контента, который совпадает по префиксу с совпавшим. Вероятность неудачи уменьшается в зависимости от длины префикса[17]. Также для увеличения точности подход может быть изменен, возможны варианты кэширования ресурса на основе произвольной последовательности байт (необязательно префикса), объединенных последовательностей и др.

TCP RE кэширование

Кэширование TCP RE работает на уровне транспортного протокола, устраняя избыточную передачу одинаковых данных в сетевых пакетах.

Схема работы метода представлена на рис. 2. Структурно комплекс представляет из себя два промежуточных кэширующих устройства с несколько разной функциональностью: оба одинаково моделируют структуру данных для хранения (и состояние обоих структур идентично), но

по-разному работают с сетевыми пакетами. Основной объект кэширования — это данные прикладного уровня сетевого стека[4] в пакете, без заголовков протоколов канального, сетевого и транспортного уровней. Кэшированию подвергается только поток по направлению из внешней сети к пользователю (т. к. передача популярного трафика в эту сторону наиболее вероятна), поэтому пользовательский запрос маршрутизируется без изменений. Но пакет ответа, попав на верхний кэш (кэш #2), анализируется на предмет наличия в нем (попадания в кэш). Если пакет новый, то его достачно добавить в кэш, не изменяя (и нижний кэш, #1, также добавит его к себе). Попадание означает, что пакет может быть идентифицирован положением в структуре данных, и поэтому достаточно переслать не его, а сам идентификатор. В случае, если его длина меньше длины исходных данных пакета, по каналу связи между двумя кэшами проходит меньше трафика.

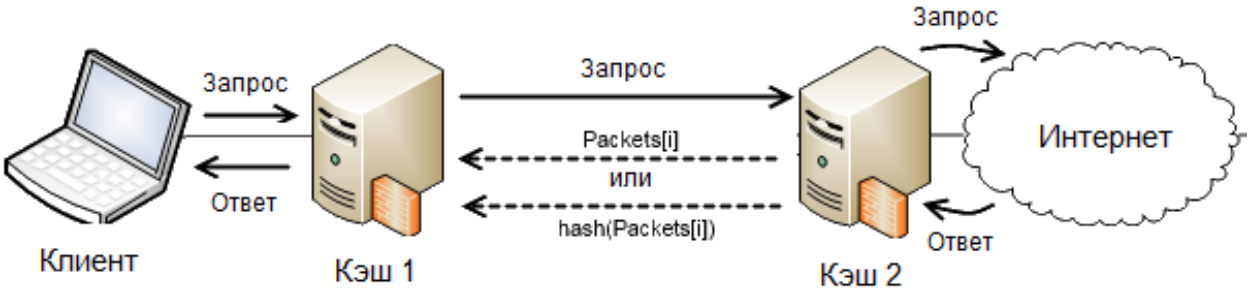

Рис. 2: Схема TCP RE кэширования

Преимущества подхода работы с пакетами состоят в том, что кэширование фактически одинаково применимо для данных совершенно различных протоколов прикладного уровня: и для HTTP, и для FTP, и для других. Отдельный момент состоит в том, что эффективность кэширования может быть различной в зависимости от этого протокола: к примеру, по заголовкам сетевого и транспортого уровней можно только косвенно понять, что передающиеся данные — шифрованные (об этом может говорить номер порта — например, по умолчанию для HTTP, использующего шифрование TLS, он 443[12]), которые очень слабо поддаются распознаванию и кэшированию. При этом кэш не имеет воз-

можности сформировать ответ на запрос пользователя, он вынужден пересылать его дальше в сеть — это означает, что экономии на внешнем по отношению к рассматриваемой системе трафике без дополнительных улучшений не добиться.

Также эффективность кэширования существенно зависит от работы самого сервера с запрашиваемым ресурсом, а именно от того, как данные будут разбиты на сегменты для пересылки с помощью сетевых пакетов, имеющих ограничение по максимальному размеру, определяемое показателями MTU (*англ.* Maximum Transmission Unit — максимальная единица передачи данных) протокола канального уровня, исходя из характеристик используемого интерфейса[11]. Эксперименты показали, что разбиение ресурса не всегда одинаково. Существуют модификации метода TCP RE, способные нивелировать этот эффект, но при этом добавляющие дополнительные задержки в работу. Возможна работа с другой единицей кэширования — объединением нескольких последовательных пакетов. Определить, что пакеты переносят смежные данные одного ресурса, можно с помощью заголовков IP и TCP, по IP-адресам и портам источника и конечного хоста, полю SYN заголовка TCP и длине самих данных[4]. Но этот способ вводит дополнительные ограничения на работу кэшей, ведь появляется необходимость объединять пакеты, приходящие, возможно, последовательно — это влечет появление задержки в работе кэшей, зависящей совместной передачи этих пакетов по сети. Нужно также разделять накопленные, но еще не закэшированные пакеты по хостам источника и пунктам назначения[5].

В силу особенностей схемы функционирования системы кэширования TCP RE она может быть успешно использована в мобильных сетях. В их устройстве предусмотрено соединение сетей радиосвязи (*англ.* RAN — Radio Access Networks), с которыми взаимодействуют пользовательские терминалы, с опорными (*англ.* CN — Core Networks), где находятся дата-центры провайдера, через транзитные сети (*англ.* BN — Backhaul Networks)[20]. Размещая модули кэширования между сетями радиосвязи и опорными (рис. 3), можно добиться уменьшения нагрузки на транзитные каналы связи[17].

Рис. 3: Схема UMTS сети с модулями TCP RE

Сравнение рассмотренных методов кэширования

В табл. 1 представлена сравнительная характеристика трех рассмотренных ранее методов кэширования, их качественные отличия. В работе “Comparison of Caching Strategies in Modern Cellular Backhaul Networks” можно найти сравнения эффективностей применения HTTP, префиксного HTTP и модификации TCP RE (с объединением нескольких пакетов) для мобильного трафика[17].

Тип кэширования	Зависимость от типа протокола верхнего уровня	Загрузка ресурса из сети	Ложные срабатывания
HTTP	есть	в случае промаха	нет
Префиксное HTTP	есть	частичная, в случае промаха — полная	да
TCP RE	нет	в любом случае	нет

Таблица 1: Сравнение видов кэширования

Таким образом, кэширование TCP RE является более гибким по сравнению с HTTP и префиксным HTTP кэшированиями, добавляя при этом необходимость в любом случае скачивать ресурс с сервера-источника.

1. Используемые технологии

Глава описывает сторонние программные продукты, активно применявшиеся в процессе разработки, и знакомит с их основными использованными функциями.

1.1. Библиотека pcap и использующие ее утилиты

Библиотека pcap, написанная на языке C, представляет из себя мощный инструмент для создания приложений, работающих с потоком сетевых данных, перехватываемых как из онлайн соединения, так и из файла[9]. При работе с онлайн-соединением библиотека получает копии пакетов, не встраиваясь в сетевой стек и не ухудшая качество соединения. Удобным является процесс последовательной обработки пакетов из источника, позволяющий работать с ним как с итератором. Реализована возможность выбора сетевого устройства для перехвата из имеющихся интерфейсов. Также полезна возможность задать фильтр, отсеивающий ненужные пакеты. У библитотеки существуют обертки для использования во многих языках программировая: java, python и др.

Многофункциональная утилита tcpdump использует библиотеку pcap. Основное ее назначение — перехват пакетов из окрытого Интернет-соединения и их последующая запись в pcap-файл. Также возможна настройка фильтров.

Программа wireshark также использует libpcap. Она облагает графическим интерфейсом, что облегчает процесс анализа трафика и отладки. Есть возможность управлять проверками корректности пакетов: соответствие SYN и ACK полей в заголовках транспортного уровня пакетов из одного потока, правильность подсчитанных чек-сумм и др. Полезными также являются утилиты tshark для сбора различной статистики и editcap, позволяющая конвертировать файлы захватов трафика из одного формата в другой.

1.2. Инструменты для обработки аргументов командной строки

В созданном программном комплексе возникает потребность использовать управление программами (в особенности анализатором) с помощью аргументов командной строки, так как это позволяет задать параметры кэширования и др.

Libtclap — удобная библиотека для шаблонного парсинга аргументов командной строки, написанная на языке C++. Есть возможность указывать тип аргумента, значение аргумента по умолчанию и добавлять исключающие друг друга аргументы.

Для обработки аргументов командной строки в скриптах, автоматизирующих запись и анализ pcap-файлов, используется модуль argparse, возможности которого тоже достаточно широки[10].

1.3. Selenium Webdriver

Selenium Webdriver активно применяется веб-разработчиками для тестирования работы сайтов, так как он, являясь с практической точки зрения набором драйверов и клиентских библитек для работы с браузерами, очень хорошо специфицирует сам интерфейс управления браузером (на его основе разрабатывается соответсвующий стандарт[16]). С помощью такого драйвера можно моделировать поведение пользователя на сайте, проверяя правильность логики работы ресурса (переход на нужную страницу при нажатии на элемент, выполнение сценария и проч.) или даже автоматизированно создавать контент (к примеру, посты на форуме).

Хотя над Selenium Webdriver существуют библиотеки-обертки, упрощающие работу со сложными элементами интерфейса (такими как checkbox и select), задачи этого исследования были решены с помощью базового функционала, имеющегося в драйвере. Selenium был применен в скриптах для автомазизированной генерации pcap-файлов. Активно использовались команды для нахождения элемента на странице по XPath

(*англ.* XML Path Language[18]), имитации нажатия клавиш над элементом, кликов.

1.4. Xvfb и PyVirtualDisplay

Xvfb (*англ.* X virtual framebuffer) — сервер, позволяющий перенаправить вывод графики на оперативную память с видеокарты . Использовался в связке с Selenium Webdriver для автоматизации выполнения сценариев на удаленной машине (через модуль pyvirtualdisplay, представляющий из себя обертку над xvfb на python).

1.5. Netfilter Framework и сетевые пакеты в ядре Linux

Netfilter — это межсетевой экран, встроенный в операционную систему Linux. Позволяет достаточно просто встроиться с помощью модуля ядра в TCP/IP стек ОС, зарегистрировав функцию-обработчик, которая будет вызываться при каждом проходящем пакете.

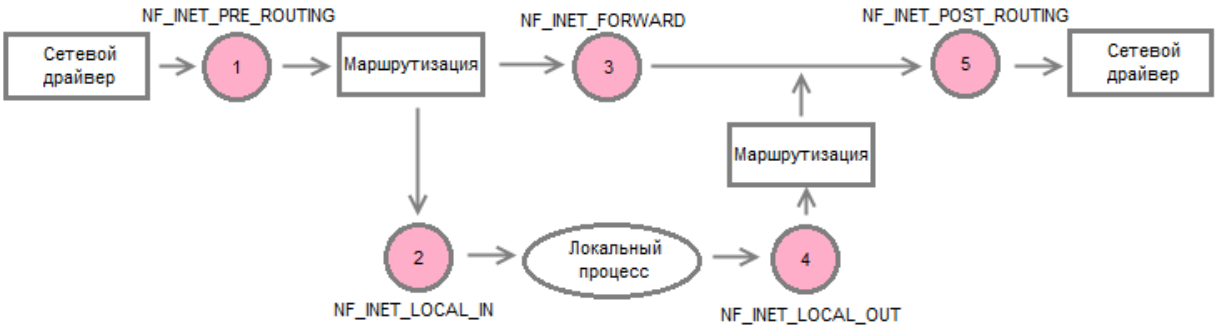

Рис. 4: Netfilter Framework: места встраивания

На рис. 4 представлена упрощенная схема движения пакетов в системе, отмечены места, в которые Netfilter позволяет встроиться. Видно, что в этих местах обрабатываются пакеты разного назначения:

- NF_INET_PRE_ROUTING — сюда попадают все пакеты, пришедшие от сетевой карты. К этому моменту успешно пройдены

т.н. sanity checks — проверки корректности полей в заголовках сетевого и транспортного уровня, таких как чек-суммы, длины данных и др.

- NF_INET_LOCAL_IN — функции этого места вызываются, если в ходе маршрутизации установлено, что пакет предназначен этому хосту.
- NF_INET_FORWARD — место для пакетов из сети, которые впоследствии будут маршрутизированы (в случае, если хост занимается маршрутизацией).
- NF_INET_LOCAL_OUT — обрабатываются пакеты, сгенерированные локальными процессами.
- NF_INET_POST_ROUTING — место обработки пакетов перед отправкой на сетевую карту.

В одно и то же место могут быть встроены несколько обрабатывающих функций, в этом случае они последовательно вызываются в порядке возрастания приоритетов. Из-за специфики работы метода TCP RE кэширования модули встраиваются в места NF_INET_PRE_ROUTING и NF_INET_POST_ROUTING, с минимальным и максимальным приоритетами соответственно.

Обработка пакета также может завершиться по-разному: он может быть отброшен (NF_DROP), принят (NF_ACCEPT, итерации по функциям продолжаются), изъят (NF_STOLEN, необходимо освобождение памяти), поставлен в другую очередь (NF_QUEUE), обработан еще раз (NF_REPEAT), принят и сразу направлен далее (NF_STOP, итерации с функциями не продолжаются). Так как модули при необходимости лишь меняют пакет, в их работе используется опция NF_ACCEPT.

Каждый пакет в ядре Linux представлен в виде структуры sk_buff, которая фактически является оберткой над буффером с пакетом. Структуры объединены в двусвязный список (который можно менять, например при удалении пакета) и содержат много дополнительных полей (в

т. ч. место для частично подсчитанной чек-суммы, локальных данных, нужных пользователю и др.). Но т. к. в процессе кэширования модифицироваться будет сам пакет, лучше использовать структуры tcphdr и iphdr для более удобного доступа к полям с данными сетевых протоколов.

Еще один компонент межсетевого экрана, использовавшийся во время настройки стенда — программа iptables. С помощью нее была настроена маршрутизация пакетов из сети с выделенным виртуальным адаптером хоста во внешнюю. Утилита позволяет изменять таблицы правил, по который пакеты обрабатываются на хосте[8].

1.6. VirtualBox и окружение виртуальной машины

В процессе подготовки модуля использовалась программа VirtualBox, позволяющая запускать виртуальные машины и работать с ними. Виртуальная машина и хост были объединены в одну выделенную сеть с помощью виртуального адаптера хоста — такой способ организации сети упрощает процесс встраивания модулей в схему. Дополнительная маршрутизация пакетов из этой сети в основную реализована с помощью настроек iptables и сервиса dnsmasq.

Для автоматизации управления виртуальной машиной (и упрощения системы скриптов) используется протокол ssh, предоставляющий возможность удаленного управления ПК[19]. С его помощью были автоматизированы процессы конфигурации виртуальной машины со стороны хоста, запуск и выключение модулей, проведение тестов и др.

2. Описание решения

В главе рассмотрены архитектура и важные детали реализации программного комплекса.

2.1. Устройство кэша

При решении задачи кэширования пакетов выбор модели устройства кэша – важный пункт, но эффективность модели определяется в первую очередь тем, как пользователи посещают Интернет-ресурсы. В работе не исследуется вопрос эффективности кэширования пользовательских сессий в зависимости от политики, изначально была выбрана модель LFU (*англ.* Least Frequently Used, название подразумевает вытеснение наименее часто использующегося элемента) как показавшая лучший результат в применении к мобильному трафику[17] по сравнению с моделью LRU (*англ.* Least Recently Used — вытеснение наименее давно использовавшегося). Есть работы, рассматривающие эту задачу более детально[7].

Рис. 5: Общая схема устройства LFU кэша

На рис. 5 представлена схема устройства кэша LFU в общем виде: данные сохранены так, что удаление наименее часто используемого происходит быстро. Однако при этом также должна быть возможность быстро обновлять значение частоты элемента, перед этим найдя его в кэше. Эти соображения подталкивают на использование схемы кэширования, изображенной на рис. 6.

В такой конфигурации скорость поиска минимального по частоте элемента оптимизирована благодаря использованию сбалансированно-

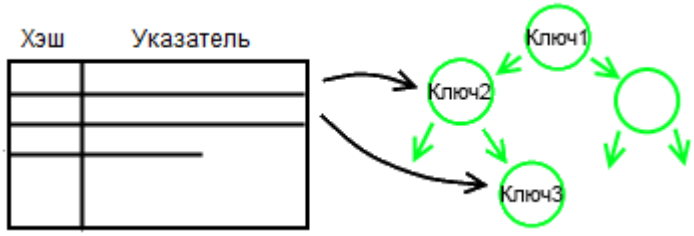

Рис. 6: Детальная схема устройства LFU кэша

го дерева поиска, ключи которого — частоты (и поэтому теоретическая сложность удаления/добавления элемента — $O(\log n)$, где n - количество элементов в дереве). Поиск элемента в кэше также осуществляется быстро за счет использования хэш-таблицы, хранящей указатель на вхождение элемента в дерево, в ней ключом выступает хэш-значение элемента (средняя сложность поиска элемента потому составляет $O(1)$). Операция обновления частоты (увеличения на единицу) является композицией поиска, удаления старого значения из дерева и добавления нового, поэтому ее теоретическая сложность также составляет $O(\log n)$.

2.2. Анализатор TCP RE

Анализатор — пользовательское приложение, моделирующее кэширование TCP RE и вычисляющее различные статистики: Byte Hit Rate, количество промахов и попаданий, количество пакетов с совпавшими хэш-значениями. Источником пакетов может быть как открытое онлайн-соединение, так и pcap-файл. Схема работы показана на рис. 7.

Анализатор написан на языке C++, использует библиотеки pcap и tlap. Его архитектура включает два класса (рис. 8):

1. Класс Cache моделирует контейнер для хранения пакетов в соответствии с рассмотренной архитектурой кэша. Используемые контейнеры из стандартной библиотеки: хэш-таблица (std::unordered_map) с ключами хэш-значениями и черно-красное дерево (std::set) с частотами в качестве ключей. Элементами кэширования выступают данные прикладного уровня сетевого пакета, хэшируются они с помощью md5 (реализация библиотеки openssl).

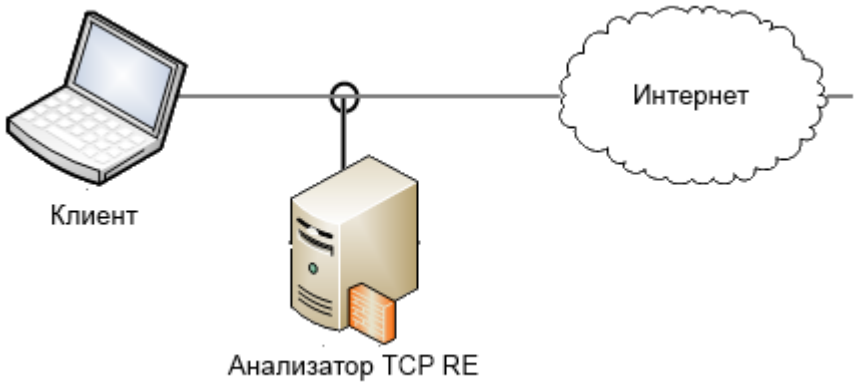

Рис. 7: Схема работы анализатора TCP RE

2. Класс NetSniffer перехватывает нужные пакеты (с помощью libpcap с использованием фильтров) и передает их классу Cache. Параметры кэширования задаются с помощью агрументов командной строки и обрабатываются с использованием библиотеки tclap (важные параметры: устройство-источник пакетов, размер кэша).

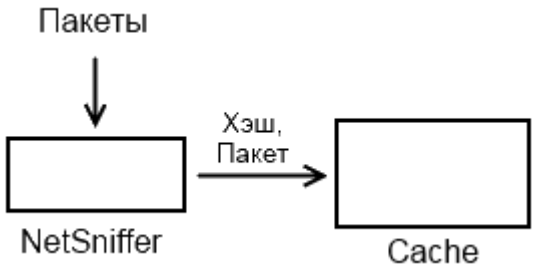

Рис. 8: Архитектура анализатора TCP RE

Корректность модели кэширования тестируется с помощью фреймворка Google Test. Проверяется правильность значений статистики Byte Hit Rate при добавлении:

1. различных пакетов
2. одного пакета много раз
3. одного набора пакетов два раза

2.3. Генерация и анализ pcap-файлов

Созданная система скриптов решает три задачи:

1. Выполняет сценарии, моделируя пользователя.

   В экспериментах были проанализиованы три типа сценариев: посещение веб-страниц, просмотр видео, загрузка больших файлов. Для автоматизации первых двух используется Selenium Webdriver (и Xvfb, при использовании на удаленной машине), загрузка файлов осуществляется вызовом консольной программы wget. Список ресурсов хранится в файле и может быть измененен и дополнен.. Для записи файлов с трафиком используется программа tcpdump. Эти скрипты написаны на языке python.

2. Анализирует файлы захватов трафика с помощью анализатора.

   Этот тип скриптов может как проанализировать уже записанные pcap-файл, так и сгенерировать новые (с помощью скриптов первого типа) и работать с ними. Написаны на языке python.

3. Настраивает необходимую конфигурацию стенда.

   Такой технический тип скриптов нужен для автоматизации процесса сборки стенда (настройки удаленного управления через ssh, конфигурации сети и проч.) перед проведением экспериментов. Написаны на языке bash.

2.4. Модули ядра TCP RE

Были созданы два модуля ядра для ОС Linux (написанные на языке C), способные проводить кэширование TCP RE в соответствии со схемой, представленной на рис. 2. Обе компоненты состоят из двух частей: функций, модифицирующих сам пакет, и кэша для их хранения.

Кэш для хранения пакетов одинаков у обоих модулей, структурно он сопадает с описанной ранее моделью LFU кэша. Важным условием функционирования модулей является идентичность данных, хранимых

в кэше. В качестве структур данных используются интрузивные контейнеры (черно-красное дерево и хэш-таблица) — структуры для создания универсальных списков, деревьев, хеш-таблиц и т.п. В этом случае специальные структуры-узлы (*англ.* nodes) связаны нужным образом (например, образуют двусвязный список), и при этом они сами являются полем пользовательской структуры. Доступ к структуре с данными через структуру-узел осуществляется с помощью макроса container_of. Хэш-таблица содержит не прямые ссылки на структуры с закэшированными пакетами, а списки по совпадающим значениям хэш-функции (md5).

Работа модуля состоит в инициализации структуры кэша и регистрировании функции-обработчика в нужном месте с помощью фреймворка Netfilter. Общие схемы устройства представлены на рис. 9 и 10.

1. Верхний модуль принимает пакет и подвергает его кэшированию: если данных не было в кэше, они туда добавляются, и пакет не меняется (произошел промах); если данные есть в кэше (произошло попадание), пакет усекается, передается хэш-сумма.

2. Нижний модуль, получив пакет, проверяет его на закэшированность (по длине и значению в зарезервированных полях заголовка tcp). Так как кэши находятся в одинаковом состоянии, попадание/промах в верхнем равносильно попаданию/промаху в нижнем. Поэтому если пакет закэширован, нижний модуль сможет его восстановить, если нет — также, как и верхний — добавить.

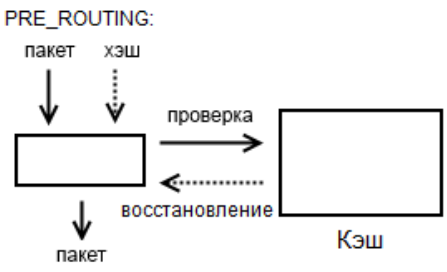

Рис. 9: Архитектура нижнего модуля

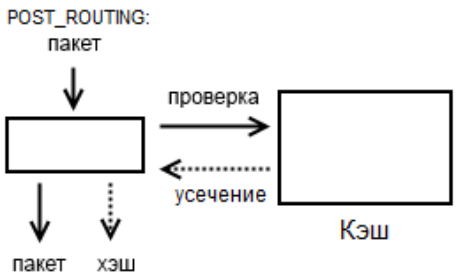

Рис. 10: Архитектура верхнего модуля

3. Эксперименты

3.1. Кэширование различных типов трафика

В ходе экспериментов были измерены эффективности кэширования трех видов трафика, каждый со своими сценариями:

1. Веб. Сценарий предполагает запрос веб-страницы и выполнение нескольких действий (ожидание, выбор и клик по элементу, переход на другую страницу того же сайта т др.).

2. Видео. Сценарий предполагает просмотр видео в браузере: запрос страницы видео-хостинга и ожидание нужно количества времени.

3. Загрузка файлов. Сценарий состоит в загрузке файла с помощью wget).

Эффективность кэширования измерялась величиной Byte Hit Rate, полученной после выполнения двух одинаковых сценариев. В этом случае $Bhr \approx \frac{\delta_r \cdot N \cdot \bar{S_r}}{2N \cdot \bar{S_{all}} + e}$ где N — количество пакетов в одной сессии (оно примерно одинаковое), $\bar{S_{all}}$ — средний размер всех пакетов, $\bar{S_r}$ — средний размер повторившихся пакетов, δ_r — доля (от N) повторившихся пакетов, e — размер шумовых пакетов, $e \ll 2N \cdot \bar{S_{all}}$. Значит, $\delta_r \approx 2Bhr \cdot \frac{\bar{S_{all}}}{\bar{S_r}} \approx 2Bhr$ при $\bar{S_{all}} \approx \bar{S_r}$ (считая, что повторящиеся пакеты такие же по распределению размеров, как и остальные).

Было выбрано несколько популярных Интернет-источников:

- Веб-ресурсы:

    1. stackoverflow.com — один из сайтов профессиональной тематики
    2. pikabu.ru — представитель развлекательных порталов
    3. echo.msk.ru — пример новостного сайта
    4. dotafire.com — сайт, посвященный популярной игре

    Для каждого сайта были запрограммированы простые сценарии.

- Видео-хостинги:

    1. youtube.com
    2. rutube.ru

       Были также опробованы twitch.tv и hitbox.com, но рекламная политика этих сайтов накладывает ограничения с точки зрения автоматического запуска и просмотра видео. Поэтому было увеличено количество видео на платформах youtube и rutube.

- Загружаемые файлы:

    1. Дистрибутив Ubuntu с ubuntu.com
    2. Драйвер для видеокарты с nvidia.com
    3. Клиент программы Cocatrice с bintray.com
    4. Pcap-файл с download.iwlab.foi.se
    5. Клиент программы Wireshark с ftp.uni-kl.de

Парные сессии записывались последовательно с помощью Selenium WebDriver. Для каждого типа трафика использовалось 20 и более пар сессий, каждая пара была проанализирована на пустом кэше заведомо большого размера для получения значения выигрыша от кэширования конкретного ресурса. Получившиеся величины Byte Hit Rate:

1. Веб: $29.9 \pm 0.8\%$
2. Видео: $28.4 \pm 0.9\%$
3. Загрузка файлов: $27.1 \pm 2.0\%$

Цифры показывают, что во всех случаях есть какой-то ненулевой процент совпавших пакетов (от 50% до 60% исходя из приведенной выше формулы). Результат не идеальный: лучше всего было бы при значениях рассматриваемой статистики около 50% — это бы означало, что совпали почти все пакеты; но и не пессимистичный — совпала немалая часть пакетов.

Это подтверждает изначальное предположение, что ресурс, будучи запрошен несколько раз, разбивается на пакеты таким образом, что часть из них одинакова (теперь известно, какая это в среднем часть). Поэтому кэширование TCP RE работоспособно и может быть использовано на практике.

3.2. Загруженность системы

3.2.1. Описание стенда

Для проведения измерений нагрузки на канал связи был собран стенд со схемой, подобной схеме функционизования кэшей TCP RE (рис. 2 и 11).

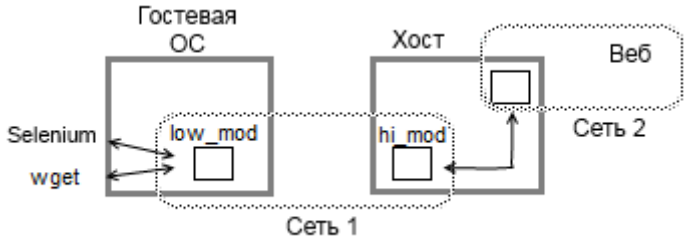

Рис. 11: Схема стенда для тестирования нагрузки

Стенд состоит из двух компьтеров — хоста и виртуальной машины, запущенной с помощью VirtualBox. Эти две машины соединены в одну сеть с помощью виртуального адаптера хоста, для работы в этой сети настроены модули кэширования. Хост также связан с внешними Интернет-ресурсами через другую сеть и настроен на маршрутизацию пакетов из гостевой ОС в Интернет с помощью iptables (это позволяет загружать файлы и запрашивать веб-страницы на стороне гостевой системы, что и делается в автоматизированном режиме с помощью ssh, Selenium, xvfb и wget).

3.2.2. Измерение нагрузки на канал связи

В ходе эксперимента гостевая ОС загружала из Интернета файл ($\approx$ 350 МБ) в двух режимах: с заново включенным кэшем TCP RE (его

размер составлял 256 МБ) и без него. Измерялась скорость передачи данных во время загрузки файла. Такой способ проведения эксперимента устойчив к шумовому трафику (так как размер передаваемых данных велик), не зависит от быстродействия драйвера браузера (если сравнивать его с запросами веб-страниц) и подразумевает, что кэш будет как добавлять, так и удалять пакеты. Однако при этом не моделируется наихудший по производительности вариант. Например, изначально добавление происходит в пустой кэш, потому вытеснять пакеты поначалу не нужно. Еще важным фактором, влияющим на заполненность кэша и трудность поиска в нем, является средняя длина списков в хэш-таблице — понятно, что при большом кэше с ростом количества переданных пакетов это значение может возрастать (это зависит от природы проходящих пакетов — насколько у популярных пакетов совпадают хэш-значения), что отрицательно сказывается на скорости поиска пакета в кэше.

Для получения более устойчивого среднего значения загрузки файлов были проведены более 45 раз, итоговый график значений представлен на рис. 12. По оси x отложено время с начала загрузки файла, по оси y — количество килобайт, переданных по каналу за последние 10 секунд. После 130-ой секунды все большинство загрузок завершило выполнение (и еще 20 секунд передает малое количество данных), поэтому эти показатели нас не интересуют. Из величин на отрезке от 10 до 120 секунд средние скорости загрузок отличаются мало, минимальное их отношение равно 0.9, среднее — 0.95.

Опыт показал, что применение кэширования TCP RE снижает пропускную способность сети, но не сильно. В случае посещения веб-страниц с точки зрения пользователя разница вообще может быть незаметна.

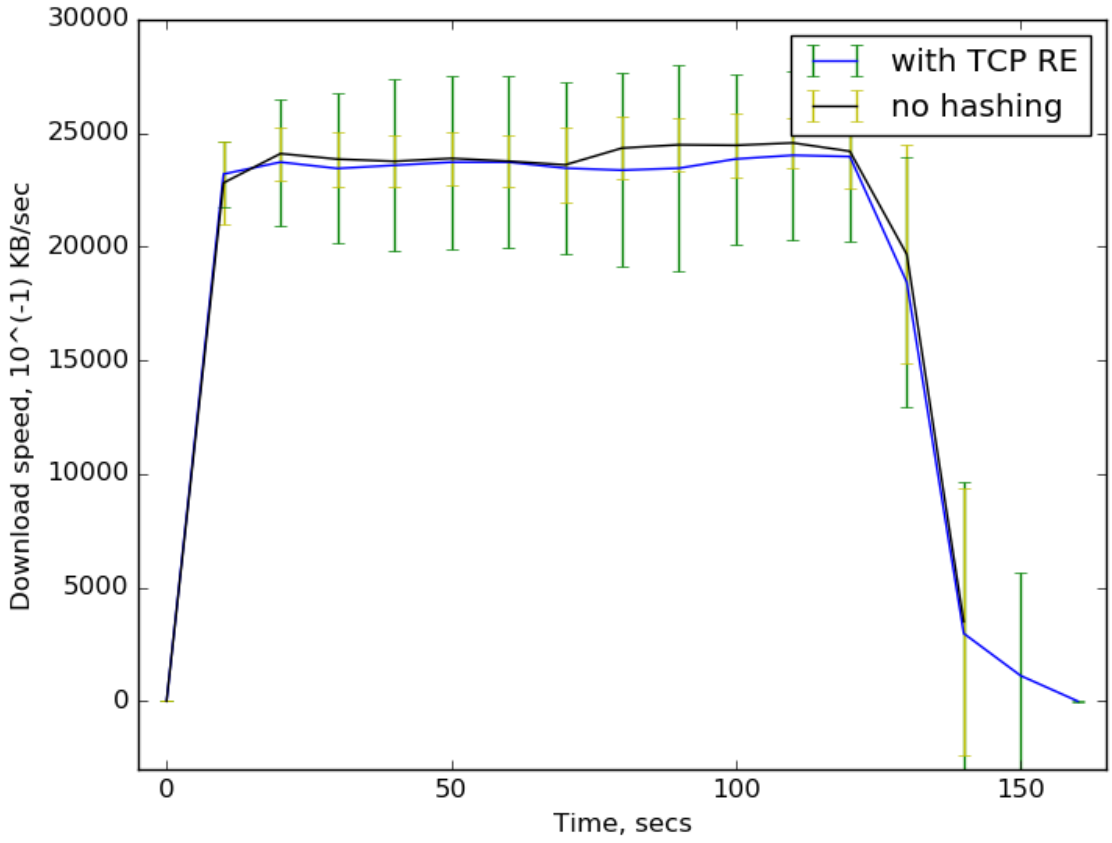

Рис. 12: Результаты измерений скорости загрузки файлов

4. Заключение

В результате исследовательской работы были решены следующие задачи:

1. Создан анализатор кэширования TCP RE
2. Спроектирована и реализована система для автоматизированной генерации pcap-файлов и их анализа
3. Разработаны модули ядра, реализующие кэширование TCP RE
4. Выполнены измерения эффективности кэширования TCP RE:
   - Эффективности кэширования различных видов трафика:
     - Веб: $29.9 \pm 0.8\%$
     - Видео: $28.4 \pm 0.9\%$
     - Загрузка файлов: $27.1 \pm 2.0\%$
   - Максимальное падение средней пропускной способности составило 10%, в среднем — 5%.

Список литературы

[1] Cisco San Jose. CA, “Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update”, 2012-2017 // Cisco Public Information. — 2013.

[2] Wein Joel M, Kloninger John Josef, Nottingham Mark C et al. Content delivery network (CDN) content server request handling mechanism with metadata framework support. — 2007. — US Patent 7,240,100.

[3] Doyle Ronald P. et al. The trickle-down effect: Web caching and server request distribution // Computer Communications. — 2002. — Vol. 25, no. 4. — P. 345–356.

[4] Fall Kevin R, Stevens W Richard. TCP/IP illustrated, volume 1: The protocols. — addison-Wesley, 2011.

[5] Ihm Sunghwan. Pushing the Limits of Web Caching: Eliminating Redundant Network Traffic // URL http://www.krnet.or.kr/board/data/dprogram/1794/B2-1-KRnet2013.pdf (дата обращения 9.05.2016). — 2013.

[6] J. Wang. A survey of web caching schemes for the internet // ACM SIGCOMM Computer Communication Review. — 1999. — Vol. 29, no. 5. — P. 36–46.

[7] LRFU (least recently/frequently used) replacement policy: A spectrum of block replacement policies / Donghee Lee, Jongmoo Choi, Jong-Hun Kim et al. // IEEE Trans. Comput. — 1996. — Vol. 50, no. 12. — P. 1352–1361.

[8] Linux 2.4 Packet Filtering. — URL: `http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.html`. (дата обращения 17.05.2016).

[9] Manpage of PCAP. — URL: `http://www.tcpdump.org/manpages/pcap.3pcap.html`. (дата обращения 28.04.2016).

[10] Manpage of argparse for python 3. — URL: https://docs.python.org/3/library/argparse.html. (дата обращения 10.05.2016).

[11] Mogul Jeffrey C, Deering Steven E. Rfc 1191: Path MTU Discovery // URL https://tools.ietf.org/html/rfc1191 (дата обращения 14.05.2016). — 1990.

[12] Rescorla Eric. Rfc 2818: Http over tls // URL https://tools.ietf.org/html/rfc2818 (дата обращения 12.05.2016). — 2000.

[13] Rfc 1738: Uniform resource locators (URL) / T. Berners-Lee, L. Masinter, M. McCahill et al. // URL https://tools.ietf.org/html/rfc1738 (дата обращения 12.04.2016). — 1994.

[14] Rfc 2616: hypertext transfer protocol–http/1.1, 1999 / Roy Fielding, Jim Gettys, Jeffrey Mogul et al. // URL http://www.rfc.net/rfc2616.html (дата обращения 10.04.2016). — 2009.

[15] Thompson K., Miller G. J., Wilder R. Wide-area Internet traffic patterns and characteristics // Network, iEEE. — 1997. — Vol. 11, no. 6. — P. 10–23.

[16] WebDriver. — URL: https://www.w3.org/TR/webdriver/. (дата обращения 17.05.2016).

[17] Woo S. et al. Comparison of Caching Strategies in Modern Cellular Backhaul Networks // Computer Networks: The International Journal of Computer and Telecommunications Networking. — 2015. — Vol. 85. — P. 51–62.

[18] XML Path Language. — URL: https://www.w3.org/TR/xpath-31/. (дата обращения 17.05.2016).

[19] Ylonen Tatu, Lonvick Chris. The secure shell (SSH) protocol architecture // URL https://tools.ietf.org/html/rfc4251 (дата обращения 5.05.2016). — 2006.

[20] Гельгор А.Л., Попов Е.А. Сотовые сети мобильной связи стандарта UMTS: учебное пособие. — СПб. : Изд-во Политехн. ун-та, 2011.