Internet Engineering Task Force (IETF)
Request for Comments: 9332
Category: Experimental
ISSN: 2070-1721

K. De Schepper
Nokia Bell Labs
B. Briscoe, Ed.
Independent
G. White
CableLabs

January 2023

Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S)

Двойные связанные очереди для L4S

Аннотация

Эта спецификация определяет модель связывания алгоритмов активного управления очередями (Active Queue Management или AQM) в 2 очереди, предназначенные для потоков с разными откликами на перегрузку. Это обеспечивает для Internet способ перехода от стандартных (классических) алгоритмов TCP-Reno-friendly, которым присущи проблемы масштабирования к расширяемому (Scalable) контролю перегрузок. Это предназначено для стабильного обеспечения очень малой задержки в буферах, очень низких потерь и расширяемости пропускной способности на уровне потоков за счёт изменённого использования явных уведомлений о перегрузке (Explicit Congestion Notification или ECN). До внедрения связанных двойных очередей (Coupled Dual Queue или DualQ) элементы Scalable L4S могут быть внедрены лишь там, где можно начать работу «с чистого листа», например в частных центрах обработки данных (ЦОД).

Эта спецификация сначала объясняет, как работает Coupled DualQ, затем задаются нормативные требования. Это не зависит от применяемых конкретных AQM и в приложениях приведены примеры псевдокода.

Статус документа

Документ не относится к категории Internet Standards Track и публикуется для проверки, экспериментальной реализации и оценки.

Документ является результатом работы IETF[1] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[2]. Не все документы, одобренные IESG, претендуют на статус стандартов Internet, см. раздел 2 в RFC 7841.

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке https://www.rfc-editor.org/info/rfc9332.

Оглавление

[1]Internet Engineering Task Force - комиссия по решению инженерных задач Internet.
[2]Internet Engineering Steering Group - комиссия по инженерным разработкам Internet.

1. Введение

Этот документ задаёт модель механизмов DualQ Coupled AQM, которые могут служить сетевой частью архитектуры L4S [RFC9330]. DualQ Coupled AQM включает 2 очереди - L4S и Classic. Очередь L4S предназначена для расширяемого контроля перегрузок и может одновременно обеспечивать очень низкие задержки (доли миллисекунд а среднем) и высокую пропускную способность. Coupled DualQ действует как полупроницаемая мембрана, изолируя очередь L4S с очень малой задержкой от классической очереди при этом сохраняя связь между очередями для объединения пропускной способности так, что произвольное число приложений, которым нужна пропускная способность, получают примерно эквивалентную пропускную способность на уровне потока независимо от используемой очереди. DualQ обеспечивает это опосредованно без необходимости просматривать идентификаторы потоков на транспортном уровне и без ущерба для классического трафика применительно к одной очереди. Устройство DualQ отличается простотой и не требует настройки для общедоступной сети Internet.

1.1. Постановка задачи

Задержки становятся критически важным фактором производительности для многих (возможно, большинства) приложений Internet, например, интерактивных web-приложений и служб, передачи, видео со звуком, интерактивного видео, удалённого присутствия, мгновенного обмена сообщениями, сетевых игр, удалённых рабочих столов, облачных приложений и служб, а также удалённого управления оборудованием и производственными процессами. при достижении в сетях доступа скоростей, распространённых сегодня в развитых странах, повышение пропускной способности канала ведёт к снижению скорости отдачи, если проблема задержки не решена. [Dukkipati06]. В последнее десятилетие или около того было многое сделано для сокращения времени распространения за счёт кэширования и переноса серверов ближе к пользователям. Однако очереди остаются основным, хотя и меняющимся компонентом задержки.

Ранее малые задержки были доступны лишь для немногих избранных низкоскоростных приложений, которые ограничивают скорость отправки специально выделенной долей пропускной способности, имеющей более высокий по отношению к другому трафику приоритет, например, ускоренная пересылка Diffserv (Expedited Forwarding или EF) [RFC3246]. До сих пор не было возможности предоставить любому приложению, которому нужны малые задержки и высокая пропускная способность, стремиться полностью использовать доступные возможности, поскольку процессы поиска высокой пропускной способности сами вносили слишком большие задержки в очередях.

Для сокращения задержки в очередях, связанной с процессом поиска пропускной способности, изменялись либо сети, либо конечные системы, участвующие в процессе. В L4S отмечено, что при обоих подходах отдача сокращается.

- Новейшие механизмы AQM в сети, например, Flow Queue CoDel [RFC8290], PIE[1] [RFC8033], ARED[2] [ARED01]), позволяют сократить задержку в очередях для всего трафика, а не только для нескольких избранных приложений. Однако, независимо от возможностей AQM, (пилообразная) скорость поиска пропускной способности механизмов контроля перегрузок в стиле TCP задаёт нижний предел, по достижении которого начинаются вариации задержки или недогрузка канала. Эти механизмы AQMs настроены так, чтобы типичный поток TCP-Reno-friendly с поиском пропускной способности вносил среднюю задержку приблизительно в 2 интервала кругового обхода (round-trip time или RTT), добавляя в среднем 5-15 мсек ожидания в очереди для сочетания долгосрочных потоков и трафика web (сравн. с 500 мксек для L4S при таком же сочетании трафика [L4Seval22]). Однако для многих приложений малая задержка бесполезна, если она не является стабильно низкой. При использовании этих механизмов AQM задержка в очередях при 99-м процентиле составляет 20-30 мсек (сравн. с 2 мсек для такого же трафика в L4S).

- Недавние исследования сквозного контроля перегрузок без необходимости применения AQM в сети (например, BBR[3] [BBR-CC]) показали аналогичный минимум задержки около 20 мсек в среднем, наряду с регулярными всплесками в 25 мсек из-за зондирования пропускной способности и 60 мсек в результате замедленного старта.

L4S использует уроки Data Center TCP (DCTCP) [RFC8257], показавшего мощь взаимодополняющих изменений в сети и конечных системах. DCTCP показал необходимость двух незначительных, но радикальных изменений контроля

[1]Proportional Integral controller Enhanced - улучшенный пропорциональный интегральный контроллер.
[2]Adaptive Random Early Detection - адаптивное случайное раннее обнаружение.
[3]Bottleneck Bandwidth and Round-trip propagation time - узкие места пропускной способности и время кругового обхода.

перегрузок, которые требуются для устранения двух основных сохраняющихся причин изменчивости задержки в очередях.

1. Значительно меньшие чем в Reno вариации скорости (зубья пилы - sawteeth) в системе контроля перегрузок.
2. Перенос сглаживания (и связанной с ним задержки) из сети к отправителям.

Без п. 1 RTT классических потоков (например, Reno-friendly) составляет приблизительно от 1 до 2 базовых значение RTT между рассматриваемыми машинами. Без п. 2 реакция классического потока на изменяющиеся события задерживается в худшем случае на величину (трансконтинентального) RTT, которая может в сотни раз превышать фактическую задержку на сглаживание, требуемую для RTT типового трафика из локализованных сетей доставки содержимого (Content Delivery Network или CDN).

Эти изменения являются двумя основными свойствами так называемого расширяемого (Scalable) контроля перегрузок (DCTCP, Prague, SCReAM[1]). Оба изменения сокращают задержку лишь в сочетании с полным изменением в сети и оба доступны только при использовании ECN (а не отбрасывания) для сигнализации.

1. Сокращение размера зубьев пилы обеспечивает очень низкий порог маркировки ECN для пакетов в очереди.
2. Отсутствие сглаживания в сети означает незамедлительную сигнализацию о каждой флуктуации очереди.

Без ECN любое из этих действий привело бы к очень высокому уровню потерь. В случае ECN высокая скорость маркировки служит просто сигнализацией и не говорит об ухудшении. Отметим, что BBRv2 [BBRv2] сочетает лучшее из двух вариантов, обеспечивая расширяемый контроль перегрузок при доступности ECN и минимизируя задержки при отсутствии ECN.

Однако до настоящего времени масштабируемые элементы контроля перегрузки (например, DCTCP) не совмещались с общей очередью с поддержкой ECN и существующими классическими методами контроля перегрузок (например, Reno [RFC5681] или CUBIC [RFC8312]). Расширяемые средства контроля перегрузок настолько энергичны, что классические алгоритмы приходят в состояние самоограничения пропускной способности. Поэтому до настоящего времени элементы L4S можно было внедрять лишь там, где возможно создание среды «с нуля», например, в частных ЦОД (отсюда и название DCTCP).

Одним из способов обеспечить сосуществование классических и масштабируемых потоков является использование очередей по потокам (per-flow-queuing или FQ), как в FQ-CoDel [RFC8290]. При таком подходе пакеты классифицируются по идентификаторам потоков для изоляции разреженных (sparse) потоков от очередей в нагруженных потоках с большой задержкой. Однако для классических потоков, которым нужна сразу малая задержка и высокая пропускная способность, наличие отдельной очереди не избавляет от вреда, который поток наносит сам себе. Кроме того, для FQ требуется просмотр идентификаторов потоков, что не всегда приемлемо на практике.

Такие образом, расширяемые элементы контроля перегрузок устраняют основную причину задержек, потерь и проблем с расширяемостью, присущих классическому контролю перегрузок. Механизмы FQ и DualQ AQM могут способствовать таким сглаженным и расширяемым потокам с малой задержкой. Подход DualQ особенно полезен, поскольку идентификация потоков не всегда желательна на практике и может быть неудобной.

1.2. Контекст, область действия и применимость

L4S предполагает внесение изменений в сеть и конечные системы

Сеть
: DualQ Coupled AQM (задан в этом документе) или модификация AQM с очередями для потоков (п. b в параграфе 4.2 описания архитектуры L4S [RFC9330]).

Конечная система
: Расширяемый контроль перегрузки (раздел 4 спецификации протокола L4S ECN [RFC9331]).

Идентификатор пакета
: Компоненты L4S в сети и на конечных системах можно внедрять постепенно, поскольку пакеты L4S указываются в них выделенными для экспериментов кодами ECN в заголовке IP - ECT(1) и CE [RFC8311] [RFC9331].

DCTCP [RFC8257] является примером расширяемого контроля перегрузок для контролируемых сред и уже применяется некоторое время в ОС Linux, Windows, FreeBSD. В процессе прохождения этого документа через IETF было реализовано множество других элементов расширяемого контроля перегрузок, например, Prague для TCP и QUIC [PRAGUE-CC] [PragueLinux], BBRv2 [BBRv2] [BBR-CC], L4S-вариант SCReAM для работы в реальном масштабе времени [SCReAM-L4S] [RFC8298].

В этой спецификации основное внимание уделяется внедрению сетевой части службы L4S. После этого без каких-либо управляющих воздействий приложения могут начать использование этого нового свойства сети по мере перехода самих приложений или их ОС к расширяемым элементам контроля перегрузок, которые могут развиваться, пока их преимущества не станут доступны всем пользователям Internet.

Модель DualQ Coupled AQM позволяет включать любые механизмы AQM, разработанные для одной очереди и обеспечивающие статистическую или детерминированную вероятность маркировки/отбрасывания, определяемую динамикой очереди. В Приложениях даны примеры двух разных DualQ Coupled AQM. Во многих случаях модель упрощает базовый алгоритм контроля и требует меньше дополнительной обработки. Поэтому предполагается, что Coupled AQM подойдёт для широкого внедрения на всех типах буферов, таких как буферы в массовом недорогом домашнем оборудовании, стеках конечных систем, оборудовании операторского класса, включая серверы доступа, маршрутизаторы, межсетевые экраны и коммутаторы Ethernet, буферы в сетевых адаптерах, виртуализованных сетевых устройствах, гипервизорах и т. д.

Для общедоступной сети Internet почти все преимущества обычно будут достигаться развёртыванием Coupled AQM на любой стороне канала доступа между сайтом и Internet, который всегда является узким местом (см. параграф 6.4 в [RFC9330], где рассматривается внедрение, а также определён термин «сайт» как дом, офис, кампус или мобильное оборудование пользователя).

[1] Self-Clocked Rate Adaptation for Multimedia - самосинхронизируемая адаптация скорости для мультимедиа.

Задержки не являются единственным вопросом L4S.

- Часть названия Low Loss (малые потери) означает, что L4S обычно обеспечивает нулевые потери при перегрузке (иначе будут возникать задержки, связанные с повтором передачи) в результате применения ECN.
- Часть названия Scalable throughput (масштабируемая пропускная способность) означает, что пропускной способности на уровне потока при расширяемом контроле перегрузок следует расширяться без ограничений, избегая неизбежных проблем масштабирования в алгоритмах TCP-Friendly [RFC3649].

Первое обстоятельство явно связано с областью действия этого документа AQM. Однако второе является результатом поведения конечной системы и поэтому выходит за раки этого документа, несмотря на зависимость от AQM.

В описании архитектуры L4S [RFC9330] приведено больше деталей, включая более широкое рассмотрение совместимости расширяемого контроля перегрузок с прежними версиями в узких местах, где не развернут DualQ Coupled AQM. В статьях [L4Seval22], [DualPI2Linux], [PI2] и [PI2param] приведено полное обоснования устройства AQM как текстом, такие в более точной математической форме, а также указаны результаты оценки производительности. Основные результаты были независимо проверены с использованием контроля перегрузки Prague [Boru20] (эксперименты проводились с Prague и DCTCP, но для проверки подходит лишь первый вариант, поскольку в Prague решён ряд проблем Linux DCTCP, которые делают этот протокол непригодным для общедоступной сети Internet).

1.3. Терминология

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не следует** (SHALL NOT), **следует** (SHOULD), **не нужно** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **не рекомендуется** (NOT RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе интерпретируются в соответствии с BCP 14 [RFC2119] [RFC8174] тогда и только тогда, когда они выделены шрифтом, как показано здесь.

DualQ Coupled AQM использует 2 очереди для двух служб.

Classic Service/Queue - классическая служба/очередь
: Классический сервис предназначен для всех вариантов поведения контроля перегрузок, сосуществующих с Reno [RFC5681] (например, сам Reno, CUBIC [RFC8312], Compound [CTCP], TFRC [RFC5348]). «Классической очередью» называется очередь, обеспечивающая классический сервис.

Low Latency, Low Loss, and Scalable throughput (L4S) Service/Queue - служба/очередь с малыми задержками и потерями, в также расширяемой пропускной способностью (L4S)
: Сервис L4S предназначен для трафика с расширяемыми алгоритмами контроля перегрузок, такими как Prague [PRAGUE-CC], который был выведен из DCTCP [RFC8257]. Сервис L4S предназначен для более широкого класса трафика, нежели просто Prague, и позволяет развивать набор средств контроля перегрузок, аналогичных Prague, таких как отмечены выше (Relentless, SCReAM и т. п.). Очередью L4S называется очередь, обеспечивающая услуги L4S.

Classic Congestion Control - классический контроль перегрузок
: Поведение контроля перегрузок, способное сосуществовать со стандартным Reno [RFC5681] без существенного негативного влияния на скорость потока [RFC5033]. Поскольку скорости потоков выросли с момента разработки контроля перегрузок TCP в 1988 г., в классических алгоритмах, таких как Reno и CUBIC, для достаточно продолжительных потоков на восстановление (в результате маркировки ECN или потери пакетов) затрачиваются сотни круговых обходов (round trip), как показано в примерах параграфа 5.1 и в [RFC3649], и их число продолжает расти. Поэтому контроль за постановкой в очередь и её использованием ослабляется и малейшие нарушения (например, появление новых потоков) препятствуют достижению высокой скорости.

Scalable Congestion Control - расширяемый контроль перегрузок
: Контроль перегрузок, где среднее время от одного сигнала насыщения до следующего (время восстановления) не зависит от скорости потока при прочих равных условиях. Это обеспечивает одинаковую степень контроля над очередями и загрузкой канала независимо от скорости потока, а также гарантирует устойчивость высокой пропускной способности к нарушениям. Например, DCTCP усредняет 2 сигнала пересылки за интервал кругового обхода, независимо от скорости потока, как и другие недавно разработанные расширяемые механизмы контроля перегрузок, такие как Relentless TCP [RELENTLESS], Prague для TCP и QUIC [PRAGUE-CC] [PragueLinux], BBRv2 [BBRv2] [BBR-CC], L4S-вариант SCReAM для потоков в реальном масштабе времени [SCReAM-L4S] [RFC8298]. Для общедоступной сети Internet расширяемый (Scalable) транспорт должен соответствовать требованиям раздела 4 в [RFC9331] (Требования Prague L4S).

C
: Сокращение для Classic, например, при использовании в качестве подстрочного индекса.

L
: Сокращение для L4S, например, при использовании в качестве подстрочного индекса.
Атрибуты Classic и L4S могут применяться к очередям (queue), кодам (codepoint), идентификаторам (identifier), классификации (classification), пакетам (packet), потокам (flow). Например термин «пакет L4S» относится к пакету с идентификатором L4S, переданному из системы контроля перегрузок L4S.
Оба типа сервиса (Classic и L4S) могут справляться с некоторой долей неотзывчивого и слабо отзывающегося трафика, но в случае L4S скорость должна быть достаточно плавной или низкой, чтобы не возникала очередь (например, DNS, VoIP, синхронизация игр и т. п.). Поведение DualQ Coupled AQM задано похожим на очередь FIFO[1] в части не отвечающего и перегруженного трафика.

Reno-friendly - совместимость с Reno
: Часть классического трафика, совместимая со стандартным контролем перегрузок Reno, заданным для TCP в [RFC5681]. Спецификация TFRC [RFC5348] косвенно подразумевает, что дружественностью (совместимостью) считается «нахождение обычно в пределах двухкратного отличия скорости передачи потока TCP при одинаковых условиях». Термин Reno-friendly используется здесь вместо TCP-friendly, поскольку последнее выражение стало неточным, так как протокол TCP сейчас использует много разных вариантов контроля перегрузок, а Reno применяется не только в TCP, но и в других транспортных протоколах (например, QUIC [RFC9000]).

[1] First-In, First-Out - первым вошел - первым вышел.

DualQ или DualQ AQM

Применяется свободно в качестве сокращения Dual-Queue Coupled AQM, где из контекста очевидно Coupled AQM.

Classic ECN - классический ECN

Исходный протокол явных уведомлений о перегрузке (Explicit Congestion Notification или ECN) [RFC3168], требующий считать сигналы ECN эквивалентом отбрасывания (при генерации в сети или на отвечающем хосте).

В L4S имена, используемые для четырёх кодов в 2-битовом поле IP-ECN, сохраняются такими же, как было задано в спецификации ECN [RFC3168], т. е. Not-ECT, ECT(0), ECT(1) и CE, где ECT означает транспорт с поддержкой ECN (ECN-Capable Transport), а CE (Congestion Experienced) - возникновение перегрузки. Пакеты, помеченные кодом CE, называются ECN-marked или просто помеченными, если контекст указывает ECN.

1.4. Свойства

AQM связывает маркировку и/или отбрасывание в классической очереди с очередью L4S, чтобы поток получал примерно равную пропускную способность, независимо от применяемого механизма. Поэтому обеим очередям доступна полная пропускная способность канала и для очередей не нужно настраивать скорости. Очередь L4S позволяет расширяемому контролю перегрузок, таком как DCTCP или Prague, обеспечивать постоянно очень малую задержку без ущерба для производительности «классического» трафика Internet.

Были выполнены тысячи тестов при типовых настройках широкополосного домашнего доступа. В экспериментах использовался диапазон базового RTT до 100 мсек и скорость канала до 200 Мбит/с между ЦОД и домашней сетью с разным объёмом фонового трафика в обеих очередях. Для каждого пакета L4S механизм AQM сохранял среднюю задержку менее 1 мсек (или 2 пакетов, если задержка сериализации на медленном канале превышала 1 мсек) и не хуже 2 мсек при 99-м процентиле. Потерь в L4S AQM не возникало совсем. Детали экспериментов приведены в [L4Seval22] и [DualPI2Linux]. Субъективное тестирование с использованием очень требовательных к задержке и пропускной способности приложений через 1 общий канал доступа также описано в [L4Sdemo16] и обобщено в параграфе 6.1 описания архитектуры L4S [RFC9330].

Во всех этих экспериментах хост был подключён к домашней сети кабелем Ethernet, чтобы пользователь мог оценить возможную задержку в очередях. Следует подчеркнуть, что поддержка L4S в узком месте не может «отменить» задержки, внесённые на других каналах пути, например, в устаревшем оборудовании Wi-Fi. Однако при добавлении поддержки L4S в очереди, обслуживающей выходной канал WAN на домашнем шлюзе было бы контрпродуктивно не сократить всплески трафика на входящих соединениях Wi-Fi. Кроме того, продолжаются испытания оборудования Wi-Fi с L4S DualQ Coupled AQM на выходном интерфейсе Wi-Fi и первые результаты оценки L4S DualQ Coupled AQM в тестовой сети радиодоступа 5G с эмуляцией радиосигнала на внешнем (уличном) краю ячейки приведены в [L4S_5G].

В отличие от Diffserv EF, очередь L4S для получения малой задержки необязательно ограничивать небольшой долей пропускной способности канала. Очередь L4S может заполняться большим числом потоков, которым нужна пропускная способность (Prague, BBRv2 и т. п.), сохраняя при этом малую задержку. Очередь L4S не полагается на наличие другого трафика в классической очереди, который можно вытеснить (overtaken). Это обеспечивает малые задержки для трафика L4S независимо от наличия классического трафика. Задержка в хвосте для трафика, обслуживаемого Classic AQM, может немного меняться при наличии трафика L4S.

Ниже указаны причины, требующие наличия двух очередей.

- Большие вариации (пила) классических потоков требуют для полной загрузки канала задержки примерно в размере базового RTT.
- Расширяемым потокам не требуется очередь для сохранения загрузки канала, но они не могут сохранять достаточно малую задержку при наличии также классического трафика.

Очередь L4S имеет приоритет по задержке в интервалах, составляющих доли RTT, но в более продолжительных интервалах связывание классического AQM с L4S AQM (описан ниже) обеспечивает отсутствие предпочтений по сравнению с классическим трафиком.

2. DualQ Coupled AQM

Ниже указаны 2 основных аспекта подхода DualQ Coupled AQM.

1. Механизм Coupled AQM обеспечивающий эквивалентность классических (например, Reno или CUBIC) и L4S потоков (соответствуют требованиям Prague L4S).
2. Структура Dual-Queue, обеспечивающая отделение потоков L4S для их изоляции от обычно большой классической очереди.

2.1. Coupled AQM

В 1990-х годах была выведена «формула TCP», связывающая установившееся окно перегрузки cwnd и вероятность отбрасывания p стандартного контроля перегрузок Reno [RFC5681]. В качестве первого приближения можно считать cwnd для Reno обратно пропорциональным квадратному корню от p.

Решение ориентировано на Reno к качестве худшего варианта, поскольку при отсутствии ущерба для Reno его не будет и для CUBIC и иного трафика, совместимого с Reno. TCP CUBIC реализует режим совместимости с Reno, который актуален для типичных RTT менее 20 мсек, пока пропускная способность одного потока меньше приблизительно 350 Мбит/с. В таких случаях можно предполагать, что трафик CUBIC ведёт себя аналогично Reno. Термин «классический» применяется для совместимого с Reno трафика, включая CUBIC и, возможно, другие механизмы контроля перегрузок, предназначенные для того, чтобы не влиять сильно на скорость потока Reno.

В статье [PI2] выведено уравнение эквивалентной скорости для DCTCP, в котором значение cwnd обратно пропорционально вероятности ECN-маркировки p (а не квадратному корню). DCTCP - не единственный механизм контроля перегрузок, который ведёт себя подобным образом, поэтому термин «расширяемый» (Scalable) используется для всех похожих вариантов поведения контроля перегрузки (см. примеры в 1.2. Контекст, область действия и применимость). Термин L4S применяется для трафика, управляемого расширяемым контролем перегрузки, который соответствует также требованиям Prague L4S [RFC9331].

Для безопасного сосуществования в стационарных условиях масштабируемый поток должен работать примерно с той же скоростью, что и поток Reno TCP (при прочих равных условиях). Поэтому вероятность отбрасывания или маркировки для классического трафика (p_C) должна отличаться от вероятности маркировки для трафика L4S (p_L). Исходная спецификация ECN [RFC3168] требовала, чтобы вероятности отбрасывания и маркировки были одинаковы, но [RFC8311] обновляет [RFC3168], разрешая эксперименты с разными вероятностями.

Для сохранения стабильности классическим источникам нужно сглаживание p_C в сети, чтобы изменения были достаточно медленными. Сетевому узлу сложно знать RTT для всех потоков, поэтому AQM добавляет задержку сглаживания для худшего RTT (100-200 мсек). L4S переносит ответственность за сглаживание откликов ECN на источник, который задерживает реакции лишь по своему значению RTT, что позволяет реагировать быстрее.

Coupled AQM обеспечивает безопасное сосуществование дела вероятность классического отбрасывания p_C пропорциональной квадрату привязанной вероятности L4S p_CL. Значение p_CL является входом для вероятности мгновенной маркировки L4S p_L, но меняется так же медленно, как p_C. Это делает скорость потока Reno примерно равной скорости потока DCTCP, поскольку возведение p_CL в квадрат уравновешивается квадратным корнем из p_C в «формуле TCP» классического контроля перегрузок Reno.

Связь между вероятностью классического отбрасывания p_C и привязанной вероятностью L4S p_CL выражается как

```
p_C = ( p_CL / k )^2,                    (1)
```

где k - коэффициент пропорциональности, называемый фактором связывания (coupling factor).

2.2. Двойная очередь

Классическому трафику нужна большая очередь для предотвращения недогрузки каналов. Поэтому для трафика L4S поддерживается отдельная очередь, имеющая более высокий приоритет по сравнению с классической очередью. Приоритет задаётся по условию, чтобы не возникало ущерба для классического трафика (2.4. Общая структура DualQ Coupled AQM). Связанная маркировка гарантирует, что предоставление приоритета трафику L4S оставляет достаточно ресурсов для планирования классического трафика, чтобы каждый получал пропускную способность, эквивалентную потокам DCTCP (при совпадении прочих условий, таких как RTT).

2.3. Классификация трафика

Механизмам Coupled AQM и DualQ нужен идентификатор, чтобы отличать пакеты L4S (L) от классических (C). Тогда алгоритм связывания может обеспечить сосуществование пакетов без необходимости просмотра идентификаторов потоков, имея возможность применять маркировку и отбрасывание ко всем потокам каждого типа. Спецификация [RFC9331] требует от сети обработки кодов ECT(1) и CE в поле ECN как таких идентификаторов. Потребовалось создать отдельный документ [RFC8311], разрешивший особую обработку кода ECT(1) в экспериментах.

По соображениям политики оператор может предпочесть вывод отдельных пакетов (например, для некоторых потоков по отдельным адресам) из очереди L, даже если они идентифицированы как L4S кодом ECN. Для таких случаев протокол L4S ECN [RFC9331] говорит, что устройству «**недопустимо** менять сквозной идентификатор L4S ECN». Цель заключается в обеспечении каждому оператору возможности локальной трактовки трафика L4S, но без возможности изменить идентификацию пакетов L4S, что помешало бы другим операторам нисходящего направления принимать своё решение по обработке трафика L4S.

Кроме того, оператор мог бы применять иные идентификаторы для отнесения неких дополнительных типов пакетов к очереди L, если это по мнению оператора не нарушит работу службы L4S. Например, можно выбирать пакеты по адресам конкретных приложений или хостов, кодам Diffserv, таким как EF, Voice-Admit, поэтапное поведение NQB[1], некоторые протоколы (например, ARP и DNS), как указано в параграфе 5.4.1 [RFC9331]. Отметим, что в [RFC9331] сказано: «узлу сети **недопустимо** менять код Not-ECT или ECT(0) в поле IP-ECN на идентификатор L4S». Таким образом, очередь L не является исключительно L4S и может рассматриваться просто как очередь с малой задержкой.

2.4. Общая структура DualQ Coupled AQM

На рисунке показана общая структура, которую, вероятно, будет иметь любой механизм DualQ Coupled AQM. Эта схема приведена для облегчения понимания устройства современных DualQ Coupled AQM. Однако она не исключает других вариантов выполнения требований параграфа 2.5, минимально задающих DualQ Coupled AQM. Схема иллюстрирует лишь поведение в обычных условиях, а работа при перегрузке или с заданными оператором классификаторами описана в параграфе 2.5.1.1.

Классификатор слева разделяет входящий трафик между очередями L и C, каждая из которых имеет свой механизм AQM, определяющий вероятность маркировки или отбрасывания (p_L и p_C). В [PI2] было доказано, что предпочтительней управлять нагрузкой с помощью линейного контроллера, затем возводить результат в квадрат до использования в качестве вероятности для трафика Reno-friendly (поскольку контроль перегрузок Reno снижает нагрузку пропорционально квадратному корню увеличения вероятности отбрасывания). Поэтому AQM для классического трафика нужно реализовать в 2 этапа: i) базовое определение внутренней вероятности p' (произносится p-prime) и ii) извлечение квадратного корня, дающего p_C, т. е.

```
p_C = (p')^2.                            (2)
```

Подстановка вместо p_C в уравнение (1) даёт

```
p' = p_CL / k
```

Таким образом, медленно меняющийся ввод маркировки ECN в очередь L (связанная вероятность L4S) имеет вид

```
p_CL = k*p'                              (3)
```

Фактическая вероятность ECN-маркировки p_L для очереди L, должна отслеживать задержку непосредственно в L при перегрузке только L, а также отслеживать p_CL при связанных условиях перегрузки. Таким образом, L использует «естественный механизм Native AQM» для расчёта вероятности p'_L как функции от мгновенной задержки в очереди L. С учётом условного приоритета очереди L над C при росте очереди L механизм AQM должен применять вероятность маркировки p'_L, но значение p_L не должно опускаться ниже p_CL. Это предполагает

[1]Non-Queue-Building - построение без очереди.

```
      p_L = max(p'_L, p_CL)                (4)
```

что хорошо работает на практике.

Два преобразования p' в уравнениях (2) и (3) реализуют связывание, заданное уравнением (1).

Коэффициент пропорциональности или фактор связывания k в уравнении (1) задаёт соотношение между вероятностями индикации перегрузки (потери или маркировка) для классического трафика и L4S. Таким образом, k опосредованно задаёт соотношение скоростей классического потока и L4S, поскольку потоки (при условии их реакции) корректируют свою скорость в соответствии с вероятностью перегрузки. В Приложении C.2 даны рекомендации по выбору значения k и его влиянию на относительные скорости потоков.

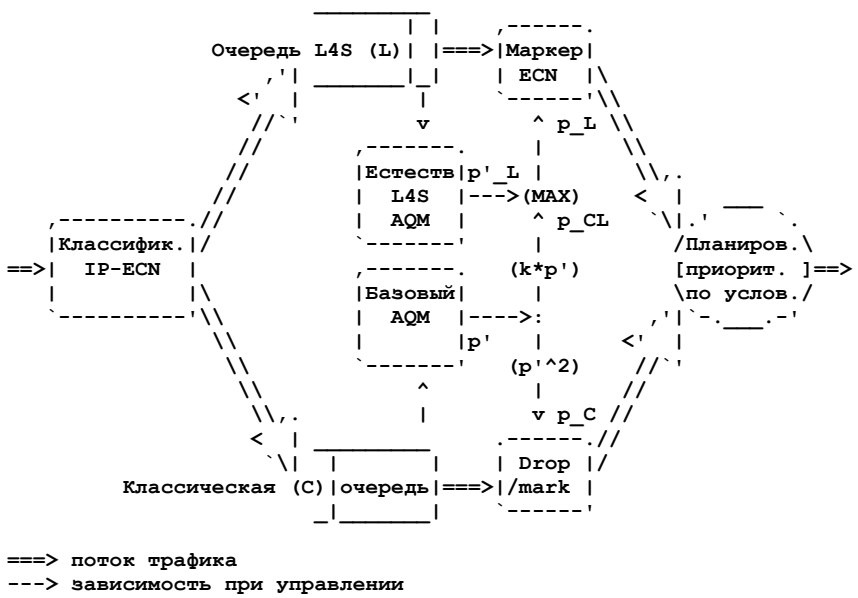

Рисунок 1. Схема DualQ Coupled AQM.

После того, как механизмы AQM применили отбрасывание или маркировку, планировщик пересылает их пакеты в канал. Даже если планировщик предоставляет приоритет очереди L, он не столь велик, как связь с очередью C. Это обусловлено тем, что по мере роста очереди C базовый механизм AQM применяет больше сигналов перегрузки для трафика L (равно как и C). Поскольку в ответ на это снижается скорость потоков L, для этого трафика используется пропускная способность меньше запланированной. Поскольку работа планировщика продолжается, пропуски заполняются трафиком C.

Предоставление приоритета очереди L обеспечивает очень малую задержку в ней, поскольку очередь L сохраняется пустой, когда трафик L контролируется связыванием. Это связывание работает лишь в одном направлении - от классического контроля перегрузки к L4S. Приоритет должен предоставляться по какому-либо условию, чтобы не допустить кратковременного подавления трафика C (4.2.2. Предотвращение краткосрочного подавления классического трафика), чтобы тот можно было тем или иным способом вытолкнуть, как описано ниже. При нормальной реакции трафика L связанная маркировка ECN предоставляет трафику C возможность прохождения даже при сильной приоритизации за счёт маркировки перегрузки трафика L, чтобы оставить некоторое пространство. Однако при наличии лишь небольшого конечного набора пакетов C (например, запрос DNS или начальное окно данных) некоторые классические AQM не обеспечивают достаточной маркировки ECN в очереди L, независимо от продолжительности ожидания пакетами C. В этом случае при сохранении занятости очереди L трафик C просто не будет учитываться планировщиком со строгим приоритетом. В идеале классическому AQM следует увеличивать вероятность связанной маркировки по мере роста продолжительности ожидания для пакетов C, но это не всегда практично, поэтому следует предоставлять приоритет очереди L при выполнении неких условий. Задание небольшого веса или предела времени ожидания для трафика C улучшает время отклика для коротких классических сообщений, таких запросы DNS, и ускоряет запуск классического потока, поскольку некоторая пропускная способность доступна сразу.

Примеры алгоритмов DualQ Coupled AQM (DualPI2 и Curvy RED) даны в приложениях A и B. Любой из них подходит для связывания маркировки и отбрасывания пакетов в DualQ.

- DualPI2 использует пропорциональный интегральный (Proportional Integral или PI) контроллер в качестве Base AQM. Этот базовый механизм AQM с квадратичным выходом и без очереди L4S может служить заменой PIE [RFC8033] и в этом случае называется PI2 [PI2]. Это принципиальное упрощение PIE, которое является более отзывчивым и стабильным при динамически меняющейся нагрузке.
- Curvy RED является производным от RED [RED], но его конфигурационные параметры основаны на задержке, что делает алгоритм нечувствительным к скорости канала и требует меньше операций на один пакет, чем RED.

DualPI2 более отзывчив и стабилен в широком диапазоне RTT, нежели Curvy RED, поэтому на момент создания документа DualPI2 чаще применялся при разработке и оценке, чем Curvy RED, который не оценён в полной мере.

Оба алгоритма AQM регулируют свои очереди в соответствии с целями, задачи временем, а не байтами. Как уже отмечено, это обеспечивает настройку, независимую от скорости выхода (из очереди). При использовании AQM в структуре DualQ это особенно важно, поскольку скорость выхода из каждой очереди может быстро меняться по мере поступления и выхода потоков для двух очередей даже при постоянной скорости общего канала.

Можно управлять очередями с другими AQM при соблюдении нормативных требований параграфа 2.5.

Две очереди могут не входить в одну иерархию очередей, например, как показано в [L4S-DIFFSERV].

2.5. Нормативные требования для DualQ Coupled AQM

Ниже приведены требования, охватывающие лишь основные аспекты DualQ Coupled AQM. Они предназначены лишь для обеспечения независимости от AQM в каждой очереди, но задают схему DualQ на основе этих AQM.

2.5.1. Функциональные требования

Реализация DualQ Coupled AQM **должна** соответствовать требованиям к поведению L4S для любого сетевого узла L4S (не только DualQ), заданным в разделе 5 [RFC9331]. В первую очередь это относится к классификации и перемаркировке, кратко описанным в параграфе 2.3. Параграф 5.5 в [RFC9331] содержит рекомендации по сокращению блочной передачи (burstiness) технологии канального уровня, служащей базой для L4S AQM.

Реализация DualQ Coupled AQM **должна** применять две очереди, по одной для каждого алгоритма AQM.

Алгоритм AQM для очереди с малой задержкой (L) **должен** быть способен применять маркировку ECN для поддерживающих ECN пакетов.

Планировщик с выводом из 2 очередей **должен** давать пакета L4S приоритет перед классическим трафиком, но этот приоритет **должен** ограничиваться, чтобы не остановить классический трафик совсем (см. 4.2.2. Предотвращение краткосрочного подавления классического трафика). Планировщик **следует** ориентировать на сохранение работы или близкое к этому поведение. Это связано с тем, что классическому трафику должна быть предоставлена возможность эффективно заполнять свободное от L4S пространство, даже если иначе оно было бы отдано трафику L4S.

[RFC9331] определяет значение маркировка ECN для трафика L4S относительно отбрасывания классического трафика. Для сосуществования классического трафика и Scalable L4S, там сказано «вероятность отбрасывания механизмом AQM классического пакета Not-ECT (p_C) **должна** быть примерно пропорциональна квадратному корню вероятности маркировки, если бы это был пакет L4S (p_L).» Термин «вероятность» (likelihood) здесь имеет смысл, позволяющий маркировке и отбрасыванию быть как вероятностными, так и детерминированными.

Для текущей спецификации из этого следует, что механизм DualQ Coupled AQM **должен** применять маркировку ECN к трафику в очереди L не реже, чем получено из вероятности отбрасывания (или маркировки ECN) в классической очереди с использованием уравнения (1). Константа k в уравнении (1) указывает относительные скорости потоков Classic и L4S, когда рассматриваемый механизм AQM является узким местом (при прочих равных условиях). Для протокола L4S ECN в [RFC9331] сказано: «Коэффициент пропорциональности (k) не стандартизован для обеспечения функциональной совместимости, но **рекомендуется** значение 2.»

Допущение, что расширяемый контроль перегрузок в Internet будет столь же энергичным, как DCTCP, гарантирует, что окно перегрузки будет примерно таким, как при стандартном контроле перегрузки TCP Reno (Reno) [RFC5681] и других дружественных к Reno механизмов контроля перегрузки, таких как TCP CUBIC в режиме Reno-friendly.

Выбор значения k определяется оператором и тот **может** применять разные значения в соответствии с рекомендациями Приложения C.2.

Если пропускная способность узкого места совместно используется разными клиентами или пользователями (например, канал доступа в Internet из сети кампуса), выбор оператором значения k будет определять распределение пропускной способности между потоками разных клиентов. Однако в общедоступной сети Internet операторы сетей доступа обычно изолируют клиентов друг от друга с помощью того или иного мультиплексирования на канальном уровне (OFDM(A) в DOCSIS 3.1, CDMA в 3G, SC-FDMA в LTE) или планирования L3 (WRR[1] для DSL), не полагаясь на контроль перегрузок в хостах [RFC0970]. В таких случаях выбор значения k влияет лишь на относительные скорости потоков в рамках доступной каждому клиенту пропускной способности, а не между клиентами. Кроме того, k не будет влиять на относительные скорости потоков, когда все потоки являются классическими (или L4S), и на пропускную способность небольших потоков.

2.5.1.1. Требования в неожиданных случаях

Гибкость, позволяющая применять классификаторы (2.3. Классификация трафика) ведёт к необходимости указания, что механизм AQM в каждой очереди должен делать с пакетами без поля ECN, ожидаемого этой очередью. Ожидается, что AQM в каждой очереди будет проверять поле ECN для определения типа передаваемого уведомления о перегрузке, затем принимать решение о применении уведомления о перегрузке для этого пакета, как описано ниже.

- Если пакет без кода ECT(1) или CE направляется в очередь L:
  - если пакет содержит код ECT(0), AQM очереди L **следует** применить маркировку CE используя вероятность, соответствующую классическому контролю перегрузки и подходящую для целевой задержки в очереди L;
  - если это пакет Not-ECT, применяемое действие зависит от применения какой-либо иной функции защиты очереди L от потоков с некорректным поведением (например, защита очередей по потокам [DOCSIS-Q-PROT] или контроль задержки):
    - если предусмотрена защита отдельной очереди, AQM очереди L **следует** игнорировать пакет и переслать его без изменений, что означает отказ от решения вопроса об отправке уведомления о перегрузке, отбрасывания и CE-маркировки пакета (например, оператор может классифицировать трафик EF, не отвечающий на отбрасывание, в очередь L вместе с отвечающим на перегрузку трафиком L4S-ECN);
    - если защита отдельной очереди не применяется, AQM очереди L **следует** применить вероятность отбрасывания пакета для классического контроля перегрузки и целевой задержки в очереди L.
- Если пакет с кодом ECT(1) направляется в очередь C:
  - AQM очереди C **следует** применять маркировку CE с использованием вероятности Coupled AQM p_CL (= k*p').

[1] Weighted Round Robin - взвешенный круговой перебор.

В приведённых выше требованиях уровень «**следует**» обусловлен тем, что операторские классификаторы предназначены лишь для гибкости. Поэтому могут быть уместны дополнительные действия в конкретных обстоятельствах оператора. Примером может служить ситуация, когда оператор знает, что неких унаследованный трафик с одним кодом использует отклик на перегрузку, связанный с другим кодом.

Если DualQ Coupled AQM обнаруживает перегрузку, этот механизм **должен** применять классическое отбрасывание для обоих типов трафика с поддержкой ECN, пока перегрузка не закончится. Применять отбрасывание при постоянно высокой маркировке ECN рекомендуется в разделе 7 спецификации ECN [RFC3168] и параграфе 4.2.1 [RFC7567].

2.5.2. Требования к управлению

2.5.2.1. Настройка конфигурации

По умолчанию механизму DualQ Coupled AQM **не следует** требовать какой-либо настройки для использования в узких местах общедоступной сети Internet [RFC7567]. Оператор может настраивать указанные ниже параметры, например, для тонкой настройки, не связанной с Internet.

- Необязательные классификаторы пакетов в дополнение к полю ECN (2.3. Классификация трафика).
- Ожидаемое типовое значение RTT, которое **может** служить для определения задержки в очереди Classic AQM в её рабочей точке, чтобы предотвратить недогрузку канала типичными одиночными потоками, например,
  - для алгоритма PI2 (Приложение A) целевая задержка в очереди зависит от типового значения RTT;
  - для алгоритма Curvy RED (Приложение B) целевая задержка в очереди для желаемой рабочей точки curvy ramp настраивается с учётом типового значения RTT;
  - при использовании иных Classic AQM вероятно потребуется рабочая точка для очереди на основе типового значения RTT и в этом случае точку **следует** задавать в единицах времени.

  Рассчитанная вручную рабочая точка может настраиваться напрямую, например, для каналов с большим числом потоков, где недостаточное использование одним потоком маловероятно.
- Ожидаемое максимальное значение RTT, которое может служить для установки параметров стабильности Classic AQM, например,
  - для алгоритма PI2 (Приложение A) параметры усиления алгоритма PI зависят от максимального RTT;
  - для алгоритма Curvy RED (Приложение B) параметры сглаживания, выбираемые для фильтрации переходных процессов в очереди, зависят от максимального RTT.

  Рассчитанные вручную параметр стабильности с учётом максимального RTT можно настраивать напрямую.
- Коэффициент (фактор) связывания k (Приложение C.2).
- Ограничение приоритета L4S по условию. Это зависит от планировщика, но значение **следует** указывать отношениям максимальной задержки пакетов C и L, например,
  - для планировщика WRR отношение весов L и C, равное w:1, означает, что максимальная задержка пакетов C в w раз больше максимальной задержки пакетов L;
  - для планировщика FIFO с временным сдвигом (TS-FIFO) (см. параграф 4.2.2) смещение tshift означает, что максимальная задержка пакетов C на tshift больше, чем для пакетов L. Значение tshift можно указывать числом типовых интервалов RTT, а не абсолютной задержкой.
- Максимальная вероятность маркировки Classic ECN p_Cmax, после которой начинается отбрасывание.

2.5.2.2. Мониторинг

Экспериментальному механизму DualQ Coupled AQM **следует** позволять оператору отслеживать по запросу указанные ниже операционные данные статистики по очередям с настраиваемыми интервалами выборки для мониторинга производительности и, возможно, учёта.

- Число пересланных битов, по которому рассчитывается уровень загрузки.
- Общее число пакетов по 3 категориям: принятые, представленные AQM и пересланные. Разница между первыми двумя будет указывать степень отбрасывания хвоста, не связанного с AQM. Разница между двумя последними указывает проактивное отбрасывание в AQM;
- Число промаркированных ECN пакетов, отброшенных пакетов без ECN и отброшенных пакетов ECN, которые могут комбинироваться с тремя общими счётчиками пакетов, указанными выше, для расчёта вероятностей маркировки и отбрасывания.
- Задержка в очереди (без учёта задержки на сериализацию головы пакета и получение доступа к среде передачи), рассмотренная в примечаниях ниже.

В отличие от других данных статистики, задержку в очереди нельзя зафиксировать в простом накопительном счётчике. Поэтому тип получаемых сведений о задержке в очереди (средняя, процентиль и т. п.) будет зависеть от ограничений реализации. Для облегчения сравнительной оценки разных реализаций и подходов реализации **следует** поддерживать среднее значение и 99-й процентиль для задержки в очереди (на очередь в интервале измерения). Относительно простым способом будет хранение грубой гистограммы задержки в очереди. Это можно сделать с использованием небольшого числа ячеек с настраиваемыми границами, которые представляют непрерывные диапазоны значений задержки в очереди. В интервале выборки каждая ячейка будет аккумулировать число пакетов своего диапазона. Максимальная задержка по очередям за интервал измерения также **может** записываться для диагностики отказов и аномальных событий.

2.5.2.3. Обнаружение аномалий

Экспериментальному механизму DualQ Coupled AQM **следует** асинхронно сообщать об аномальных условиях.

Время начала и продолжительность перегрузки

Следует использовать использовать механизм гистерезиса для предотвращения «хлопков» (flapping) возникновения и прекращения перегрузки, вызывающих «лавину» событий. Например, выход из состояния перегрузки может инициировать отчёт, а также запустить таймер. После этого если в течение отсчёта таймера AQM фиксирует начало и завершение перегрузки любое число раз, события аккумулируются без передачи отчёта, пока AQM не выйдет первый раз из состояния перегрузки по завершении отсчёта таймера.

2.5.2.4. Внедрение, сосуществование, расширяемость

В [RFC5706] предложено включать внедрение, сосуществование и расширение как требования к управлению. Смысл введения DualQ Coupled AQM заключается в обеспечении возможности внедрения и сосуществования расширяемого контроля перегрузки (как поэтапной замены сегодняшнего совместимого с Reno контроля перегрузок, не расширяемого с ростом произведения пропускной способности и задержки). Поэтому нет необходимости повторять мотивирующие проблемы, уже рассмотренные во введении и детализированные в описании архитектуры L4S [RFC9330].

Описания конкретных алгоритмов DualQ Coupled AQM в приложениях охватывают масштабирование их конфигурационных параметров, например в части RTT и частоты выборки.

3. Взаимодействие с IANA

Этот документ не требует действий IANA.

4. Вопросы безопасности

4.1. Малые задержки без обработки по потокам

Архитектура L4S [RFC9330] сравнивает подходы DualQ и FQ для L4S. При рассмотрении вопросов приватности обосновывается подход DualQ, поскольку в этом случае пользователям, желающим зашифровать идентификаторы потоков (например, с помощью IPsec или иного шифрованного туннеля VPN), не нужно жертвовать малой задержкой ([RFC8404] рекомендует избегать таких компромиссов в отношении приватности).

Обсуждение вопросов безопасности архитектуры L4S [RFC9330] включает параграфы, посвящённые управлению относительной скоростью потоков (параграф 8.1) и управлению потоками, вызывающими чрезмерную задержку в очереди (параграф 8.2). Отмечено, что интересы пользователей в части задержки не сталкиваются так же, как для пропускной способности. Если кто-то получает больше пропускной способности общего канала, кто-то другой получает меньше, тогда как задержки можно сократить для каждого без ущерба для кого бы то ни было. Отмечено также, что сегодня в Internet планирование обычно вынуждает разделять пропускную способность между «сайтами» (например, домовладениями, организациями или мобильными пользователями), но необходимость в планировании пропускной способности для отдельных приложений и управлении ею возникает нечасто.

В свете приведённых выше аргументов управление скоростью по потокам может не потребоваться, а в доверенных средах (например, в частных ЦОД) это, несомненно, маловероятно. Поскольку трудно избежать сложностей и побочных эффектов при управлении скоростью по потокам, его требуется отделять от базового AQM, например, на основании политики. С учётом этого, DualQ Coupled AQM обеспечивает малую задержку, не предопределяя управления скоростью по потокам.

Тем не менее, интересы пользователей могут конфликтовать, например, в результате аварии или злого умысла. Тогда управление скоростью по потокам может стать необходимым. Такое управление при необходимости может быть предоставлено как модульное расширение DualQ. Если нужна защита от чрезмерной задержки в очереди, в DualQ можно добавить защиту очередей по потокам (например, [DOCSIS-Q-PROT]).

4.2. Обработка неотзывчивых пакетов и перегрузки

В отсутствие управления по потокам важно, чтобы базовый механизм DualQ Coupled AQM не предоставлял неотзывчивым потокам большей преимуществ, чем AQM с одной очередью и, по меньшей мере справлялся бы с перегрузками. Перегрузка возникает, когда входная нагрузка существенно или постоянно превышает выходную пропускную способность. Определение не является точным, поскольку значительность и постоянство не являются количественными мерами.

Требуется найти компромисс между сложностью и риском нанесения одним классом трафика ущерба другим. При перегрузке службе L4S с высоким приоритетом придётся пожертвовать некоторыми аспектами производительности. В зависимости от уровня перегрузки разные решения могут смягчать воздействие на различные факторы, например, пропускную способность, задержку или отбрасывание пакетов. Выбор должен сделать разработчик или политика оператора, но не IETF. В последующих параграфах рассматривается обработка различных уровней перегрузки.

- Неотзывчивые потоки (L и/или C) без перегрузки, т. е. суммарный неотзывчивый трафик до внесения какого-либо отзывчивого трафика меньше пропускной способности на выходе. Этот случай обрабатывается обычным механизмом Coupled DualQ (параграф 2.1), но не рассматривается там. В параграфе 4.2.1 разъяснены цели проектирования и их достижение на практике.

- Неотзывчивые потоки (L и/или C), вызывающие постоянную перегрузку, т. е. суммарный неотзывчивый трафик постоянно превышает пропускную способность на выходе. Такие ситуации не обрабатываются обычным механизмом Coupled DualQ (параграф 2.1), но в последнем абзаце параграфа 2.5.1.1 задано требование обработки ситуаций, где трафик с поддержкой ECN может истощать трафик без поддержки ECN. В параграфе 4.2.3 рассматриваются основные варианты решения и приведены конкретные примеры.

- Кратковременная перегрузка (состояние между отсутствием перегрузки и постоянной перегрузкой). До момента, когда перегрузка будет сочтена постоянной в параграфе 4.2.2 рассмотрены варианты более оперативных механизмов в масштабе времени планировщика. Это предотвращает кратковременное истощение трафика из очереди C, делая приоритет очереди L условным, как требует параграф 2.5.1.

4.2.1. Неотзывчивый трафик без перегрузки

Когда 1 или множество потоков L и/или C не отвечают на сигналы перегрузки, но создаваемый ими трафик не превышает пропускную способность канала и связанная маркировка не достигает насыщения (менее 100%), целью DualQ AQM является поведение не хуже, чем у AQM с одной очередью.

Тесты показали, что это действительно так без каких-либо дополнительных механизмов сверх Coupled DualQ из параграфа 2.1 (см. результаты экспериментов с перегрузкой в [L4Seval22]). Возможно, вопреки интуиции, что независимо от классификации неотзывчивых потоков в очередь L или C, система DualQ поведёт себя, как будто она исключена (вычтена) из общей пропускной способности канала. После этого связывание распределит оставшуюся пропускную способность между конкурирующими потоками с реакцией на перегрузку (в любой очереди). Связанные с планировщиком детали рассматриваются в параграфе 4.2.2.

4.2.2. Предотвращение краткосрочного подавления классического трафика

Приоритет L4S должен предоставляться по условию (см. параграфы 2.4 и 2.5.1) , чтобы избежать кратковременного истощения классического трафика. Иначе, как разъяснено в параграфе 2.4, даже 1 отзывчивый поток L4S может временно заблокировать небольшой конечный набор пакетов C (например, начальное окно или запрос DNS). Блокировка будет короткой, но может быть более долгой в некоторых реализациях AQM, которые могут лишь увеличивать сигнализацию перегрузки, связанную с очередью C, когда пакеты C фактически выводятся из очереди. В этом случае возникает вопрос о принесении в жертву пропускной способности или задержки L4S (или иного правила).

Жертва производительностью L4S

Используя WRR как планировщик с приоритетом по условию, служба L4S может при перегрузке пожертвовать частью пропускной способности. Это можно считать гарантией минимальной пропускной способности для классического трафика или наибольшей задержки для пакета в начале классической очереди.

Предупреждение. Планировщик WRR может гарантировать пропускную способность для классического трафика лишь в случае, когда классические источники обеспечивают достаточный для этого трафик - сигналы перегрузки могут нарушить планирование, поскольку они определяют объем прибывающего отзывчивого трафика каждого класса для первоочередного планирования. Поэтому планирование применяется лишь для противодействия кратковременному истощению, пока не накопится сигнал о перегрузке и источники не отреагируют на него. Даже при длительной перегрузке (см. параграф 4.2.3), разумно отбрасывать пакеты из обеих очередей, что опять-таки сокращает до поступления в планировщик. Это связано с тем, что планировщик не может справиться с длительной перегрузкой, поскольку невозможно знать правильный вес планировщика в каждом случае.

Вес планирования для классической очереди следует устанавливать небольшим (например, 1/16). В большинстве вариантов трафика планировщик не будет вмешиваться (да это и не нужно), поскольку механизм связывания и конечные системы будут определят разделение пропускной способности между очередями, как будто это единый пул. Однако при чрезмерно энергичном или неотзывчивом трафике L4S вес для классического трафика при планировании будет достаточно большим, чтобы не возникало краткосрочного истощения этого трафика.

Хотя предполагается, что планирование WRR будет решать проблему лишь при краткосрочной перегрузке, имеются (достаточно редкие) случаи, когда WRR влияет на распределение пропускной способности в более продолжительном интервале. Однако это влияние невелико и не наносит вреда. В частности, когда отношение потоков L4S и Classic (например, 19:1) больше отношения отношения их весов в планировщике (например, 15:1), потоки L4S получат немного меньше равной доли пропускной способности. Например, при указанных соотношениях каждый поток L4S получит долю (15/16)/19 = 4.9%, тогда как в идеальном случае он получил бы 1/20 = 5%. В довольно специфическом случае неотзывчивого трафика, занимающего пропускную способность чуть меньше отведённой для L4S (например, 14/16 для описанного выше случая), применение WRR может существенно сокращать пропускную способность, оставляемую для любых отзывчивых потоков L4S.

Вес при планировании для классической очереди не следует делать слишком малым, иначе пакет C в начале очереди может быть чрезмерно задержан постоянно занятой очередью L. Например, если вес классической очереди составляет 1/16, максимальная задержка классического пакета в начале очереди из-за трафика L будет равна задержке сериализации пакетов размером 15 MTU.

Жертва задержкой L4S

Оператор может выбрать контроль перегрузки в классической очереди, позволив некоторой задержке «перетечь» в очередь L4S. Планировщик может вести себя как одна очередь FIFO с разным временем обслуживания, реализуя очень простой планировщик с приоритетом по условию, называемый FIFO со сдвигом по времени (time-shifted FIFO или TS-FIFO, см. планировщик MEDF[1] [MEDF]). Этот планировщик добавляет tshift к задержке в очереди для следующего пакета L4S до её сравнения с задержкой в очереди для следующего классического пакета, а затем выбирает пакет с большим значением (скорректированной) задержки. При обычных условиях планировщик TS-FIFO ведёт себя как строгий планировщик по приоритету, но при умеренной или сильной перегрузке он предотвращает истощение для классической очереди, поскольку сдвиг времени (tshift) определяет максимальную дополнительную задержку классических пакетов относительно L4S. Это позволяет контролировать умеренную перегрузку отзывчивого трафика путём внесения задержки для отсрочки вызова механизмов, описанных в параграфе 4.2.3, особенно при приближении к максимальному сигналу перегрузки.

Примеры из приложений A и B могут быть реализованы с любым из подходов.

4.2.3. Насыщение L4S ECN - отбрасывание или задержка?

В этом параграфе рассматривается постоянная перегрузка вызванная неотзывчивыми потоками L и/или C. Для сохранения примерно одинаковой пропускной способности потоков L4S и Classic во всем диапазоне нагрузки требуется определить другую стратегию управления при нагрузке выше точки, где L4S AQM постоянно находится (достигает) в насыщении вероятности маркировки ECN (100%), не оставляя места для дополнительной нагрузки. Маркировка L4S ECN будет насыщаться первой (при факторе связывания k>1), даже если насыщение общим неотзывчивым трафиком в одной или обеих очередях, превышающим пропускную способность канала.

Термин «неотзывчивый» относится и к случаям, где поток становится неотзывчивым на время, например, поток в реальном масштабе времени, которому нужно некоторое время на изменение скорости в ответ на перегрузку, или стандартный поток Reno, который обычно отвечает на перегрузку, но при превышении некоторого уровня уже не способен снижать окно перегрузки ниже разрешённого минимума в 2 сегмента [RFC5681], фактически становясь

[1]Modifier Earliest Deadline First.

неотзывчивым (отметим, что трафик L4S должен сохранять отзывчивость при размере окна меньше 2 сегментов, как указано в требованиях L4S [RFC9331]).

Насыщение вызывает вопрос о том, следует сокращать перегрузку отбрасыванием некоторого числа пакетов в очереди L4S или ростом задержки в обеих очередях (что может вызвать отбрасывание из-за переполнения буферов).

Отбрасывание при насыщении

Сохраняющееся насыщение можно определить порогом максимума для связанной маркировки L4S ECN (при условии k>1) до того, как наступит насыщение, при котором скорости потоков для разных типов трафика разойдутся. Выше этого порога вероятность отбрасывания классического трафика применяется ко всем пакетам любого типа. Эксперименты показали, что задержку в очереди можно сохранить на целевом уровне при любой перегрузке, включая неотзывчивый трафик, и дополнительных мер не требуется (параграф 4.2.3.1).

Задержка при насыщении

Когда маркировка L4S достигает насыщения, вместо отбрасывания L4S можно ограничить вероятность отбрасывания и маркировки в обеих очередях. После этого задержка будет расти в очереди с неотзывчивым трафиком (если применяется WRR) или в обеих очередях (если применяется TS-FIFO). В любом случае более высокая задержка должна контролировать временную высокую перегрузку. При более длительной перегрузке в конечном итоге произойдёт переполнение комбинированных DualQ и перегрузку будет контролировать отбрасывание в конце очереди (tail drop).

В примере реализации из Приложения A применяется лишь правило «отбрасывания при насыщении». Спецификация DOCSIS для DualQ Coupled AQM [DOCSIS3.1] реализует такое же правило с очень маленьким буфером L. Однако добавление защиты очередей по потокам [DOCSIS-Q-PROT] меняет этот подход на «задержку при насыщении», перенаправляя некоторые пакеты потока или потоков, наиболее сильно связанные с перегрузкой очереди L, в очередь C, которая имеет более высокую целевую задержку. Если перегрузка продолжается, механизм возвращается к отбрасыванию при насыщении, поскольку уровень отбрасывания в очереди C растёт для поддержания целевой задержки в C.

4.2.3.1. Защита от перегрузки неотзывчивым трафиком с поддержкой ECN

Без конкретного механизма для перегрузки неотзывчивый трафик получит большее преимущество, если он поддерживает ECN. Это преимущество невозможно обнаружить при обычных низких уровнях маркировки, однако оно становится более значительным при росте уровня маркировки, характерном для перегрузки, когда можно избежать высокого уровня отбрасывания. Эта проблема характерна для поддерживающего ECN трафика L4S и классического трафика. В связи с этим возникает вопрос об использовании отбрасывания трафика с поддержкой ECN (нужно ли и когда), как этого требует раздел 7 спецификации ECN [RFC3168] и параграф 4.2.1 рекомендация для AQM [RFC7567].

Например, эксперименты с DualPI2 AQM (Приложение A) показали что использование отбрасывания при перегрузке при связанной на 100% маркировке L4S решает пробдему неотзывчивого трафика ECN, а также проблему насыщения. При насыщении DualPI2 переходит в режим перегрузки, где механизм Base AQM управляется максимальной задержкой в обеих очередях и вводит вероятностное отбрасывание в равной степени для обеих очередей. Это оставляет лишь небольшой диапазон уровней перегрузки чуть ниже насыщения, где неотзывчивый трафик получает какое-либо преимущество от использования ECN (относительно неотзывчивого трафика без ECN) и это преимущество едва различимо (см. [DualQ-Test] и раздел IV-G в [L4Seval22]). Кроме того, перегрузка неотзывчивым потоком с кодом ECT(1) даёт не больше преимущества в пропускной способности по сравнению с ECT(0).

5. Литература

5.1. Нормативные документы

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <https://www.rfc-editor.org/info/rfc8311>.

[RFC9331] De Schepper, K. and B. Briscoe, Ed., "The Explicit Congestion Notification (ECN) Protocol for Low Latency, Low Loss, and Scalable Throughput (L4S)", RFC 9331, DOI 10.17487/RFC9331, January 2023, <https://www.rfc-editor.org/info/rfc9331>.

5.2. Дополнительная литература

[Alizadeh-stability] Alizadeh, M., Javanmard, A., and B. Prabhakar, "Analysis of DCTCP: Stability, Convergence, and Fairness", SIGMETRICS '11: Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, pp. 73-84, DOI 10.1145/1993744.1993753, June 2011, <https://dl.acm.org/citation.cfm?id=1993753>.

[AQMmetrics] Kwon, M. and S. Fahmy, "A Comparison of Load-based and Queue-based Active Queue Management Algorithms", Proc. Int'l Soc. for Optical Engineering (SPIE), Vol. 4866, pp. 35-46, DOI 10.1117/12.473021, 2002, <https://www.cs.purdue.edu/homes/fahmy/papers/ldc.pdf>.

[ARED01] Floyd, S., Gummadi, R., and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management", ACIRI Technical Report 301, August 2001, <https://www.icsi.berkeley.edu/icsi/node/2032>.

[BBR-CC] Cardwell, N., Cheng, Y., Hassas Yeganeh, S., Swett, I., and V. Jacobson, "BBR Congestion Control", Work in Progress, Internet-Draft, draft-cardwell-iccrg-bbr-congestion-control-02, 7 March 2022, <https://datatracker.ietf.org/doc/html/draft-cardwell-iccrg-bbr-congestion-control-02>.

[BBRv2] "TCP BBR v2 Alpha/Preview Release", commit 17700ca, June 2022, <https://github.com/google/bbr>.

[Boru20] Boru Oljira, D., Grinnemo, K-J., Brunstrom, A., and J. Taheri, "Validating the Sharing Behavior and Latency Characteristics of the L4S Architecture", ACM SIGCOMM Computer Communication Review, Vol. 50, Issue 2, pp. 37-44, DOI 10.1145/3402413.3402419, May 2020, <https://dl.acm.org/doi/abs/10.1145/3402413.3402419>.

[CCcensus19] Mishra, A., Sun, X., Jain, A., Pande, S., Joshi, R., and B. Leong, "The Great Internet TCP Congestion Control Census", Proceedings of the ACM on Measurement and Analysis of Computing Systems, Vol. 3, Issue 3, Article No. 45, pp. 1-24, DOI 10.1145/3366693, December 2019, <https://doi.org/10.1145/3366693>.

[CoDel] Nichols, K. and V. Jacobson, "Controlling Queue Delay", ACM Queue, Vol. 10, Issue 5, May 2012, <https://queue.acm.org/issuedetail.cfm?issue=2208917>.

[CRED_Insights] Briscoe, B. and K. De Schepper, "Insights from Curvy RED (Random Early Detection)", BT Technical Report, TR-TUB8-2015-003, DOI 10.48550/arXiv.1904.07339, August 2015, <https://arxiv.org/abs/1904.07339>.

[DOCSIS-Q-PROT] Briscoe, B., Ed. and G. White, "The DOCSIS® Queue Protection to Preserve Low Latency", Work in Progress, Internet-Draft, draft-briscoe-docsis-q-protection-06, 13 May 2022, <https://datatracker.ietf.org/doc/html/draft-briscoe-docsis-q-protection-06>.

[DOCSIS3.1] CableLabs, "DOCSIS 3.1 MAC and Upper Layer Protocols Interface Specification", CM-SP-MULPIv3.1, Data-Over-Cable Service Interface Specifications DOCSIS 3.1 Version I17 or later, January 2019, <https://specification-search.cablelabs.com/CM-SP-MULPIv3>.

[DualPI2Linux] Albisser, O., De Schepper, K., Briscoe, B., Tilmans, O., and H. Steen, "DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM", Proceedings of Linux Netdev 0x13 , March 2019, <https://www.netdevconf.org/0x13/session.html?talk-DUALPI2-AQM>.

[DualQ-Test] Steen, H., "Destruction Testing: Ultra-Low Delay using Dual Queue Coupled Active Queue Management", Master's Thesis, Department of Informatics, University of Oslo, May 2017.

[Dukkipati06] Dukkipati, N. and N. McKeown, "Why Flow-Completion Time is the Right Metric for Congestion Control", ACM SIGCOMM Computer Communication Review, Vol. 36, Issue 1, pp. 59-62, DOI 10.1145/1111322.1111336, January 2006, <https://dl.acm.org/doi/10.1145/1111322.1111336>.

[Heist21] "L4S Tests", commit e21cd91, August 2021, <https://github.com/heistp/l4s-tests>.

[L4S-DIFFSERV] Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", Work in Progress, Internet-Draft, draft-briscoe-tsvwg-l4s-diffserv-02, 4 November 2018, <https://datatracker.ietf.org/doc/html/draft-briscoe-tsvwg-l4s-diffserv-02>.

[L4Sdemo16] Bondarenko, O., De Schepper, K., Tsang, I., Briscoe, B., Petlund, A., and C. Griwodz, "Ultra-Low Delay for All: Live Experience, Live Analysis", Proceedings of the 7th International Conference on Multimedia Systems, Article No. 33, pp. 1-4, DOI 10.1145/2910017.2910633, May 2016, <https://dl.acm.org/citation.cfm?doid=2910017.2910633>.

[L4Seval22] De Schepper, K., Albisser, O., Tilmans, O., and B. Briscoe, "Dual Queue Coupled AQM: Deployable Very Low Queuing Delay for All", Preprint submitted to IEEE/ACM Transactions on Networking, DOI 10.48550/arXiv.2209.01078, September 2022, <https://arxiv.org/abs/2209.01078>.

[L4S_5G] Willars, P., Wittenmark, E., Ronkainen, H., Östberg, C., Johansson, I., Strand, J., Lédl, P., and D. Schnieders, "Enabling time-critical applications over 5G with rate adaptation", Ericsson - Deutsche Telekom White Paper, BNEW-21:025455, May 2021, <https://www.ericsson.com/en/reports-and-papers/white-papers/enabling-time-critical-applications-over-5g-with-rate-adaptation>.

[Labovitz10] Labovitz, C., Iekel-Johnson, S., McPherson, D., Oberheide, J., and F. Jahanian, "Internet Inter-Domain Traffic", ACM SIGCOMM Computer Communication Review, Vol. 40, Issue 4, pp. 75-86, DOI 10.1145/1851275.1851194, August 2010, <https://doi.org/10.1145/1851275.1851194>.

[LLD] White, G., Sundaresan, K., and B. Briscoe, "Low Latency DOCSIS: Technology Overview", CableLabs White Paper, February 2019, <https://cablela.bs/low-latency-docsis-technology-overview-february-2019>.

[MEDF] Menth, M., Schmid, M., Heiss, H., and T. Reim, "MEDF — A Simple Scheduling Algorithm for Two Real-Time Transport Service Classes with Application in the UTRAN", Proc. IEEE Conference on Computer Communications (INFOCOM'03), Vol. 2, pp. 1116-1122, DOI 10.1109/INFCOM.2003.1208948, March 2003, <https://doi.org/10.1109/INFCOM.2003.1208948>.

[PI2] De Schepper, K., Bondarenko, O., Briscoe, B., and I. Tsang, "PI2: A Linearized AQM for both Classic and Scalable TCP", ACM CoNEXT'16, DOI 10.1145/2999572.2999578, December 2016, <https://dl.acm.org/doi/10.1145/2999572.2999578>.

[PI2param] Briscoe, B., "PI2 Parameters", Technical Report, TR-BB-2021-001, arXiv:2107.01003 [cs.NI], DOI 10.48550/arXiv.2107.01003, July 2021, <https://arxiv.org/abs/2107.01003>.

[PRAGUE-CC] De Schepper, K., Tilmans, O., and B. Briscoe, "Prague Congestion Control", Work in Progress, Internet-Draft, draft-briscoe-iccrg-prague-congestion-control-01, 11 July 2022, <https://datatracker.ietf.org/doc/html/draft-briscoe-iccrg-prague-congestion-control-01>.

[PragueLinux] Briscoe, B., De Schepper, K., Albisser, O., Misund, J., Tilmans, O., Kuehlewind, M., and A. Ahmed, "Implementing the 'TCP Prague' Requirements for L4S", Proceedings of Linux Netdev 0x13, March 2019, <https://www.netdevconf.org/0x13/session.html?talk-tcp-prague-l4s>.

[RED] Floyd, S. and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, Volume 1, Issue 4, pp. 397-413, DOI 10.1109/90.251892, August 1993, <https://dl.acm.org/doi/10.1109/90.251892>.

[RELENTLESS] Mathis, M., "Relentless Congestion Control", Work in Progress, Internet-Draft, draft-mathis-iccrg-relentless-tcp-00, 4 March 2009, <https://datatracker.ietf.org/doc/html/draft-mathis-iccrg-relentless-tcp-00>.

[RFC0970] Nagle, J., "On Packet Switches With Infinite Storage", RFC 970, DOI 10.17487/RFC0970, December 1985, <https://www.rfc-editor.org/info/rfc970>.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <https://www.rfc-editor.org/info/rfc2914>.

[RFC3246] Davie, B., Charny, A., Bennet, J.C.R., Benson, K., Le Boudec, J.Y., Courtney, W., Davari, S., Firoiu, V., and D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, DOI 10.17487/RFC3246, March 2002, <https://www.rfc-editor.org/info/rfc3246>.

[RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <https://www.rfc-editor.org/info/rfc3649>.

[RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <https://www.rfc-editor.org/info/rfc5033>.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <https://www.rfc-editor.org/info/rfc5348>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC5706] Harrington, D., "Guidelines for Considering Operations and Management of New Protocols and Protocol Extensions", RFC 5706, DOI 10.17487/RFC5706, November 2009, <https://www.rfc-editor.org/info/rfc5706>.

[RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <https://www.rfc-editor.org/info/rfc7567>.

[RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <https://www.rfc-editor.org/info/rfc8033>.

[RFC8034] White, G. and R. Pan, "Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced (PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems", RFC 8034, DOI 10.17487/RFC8034, February 2017, <https://www.rfc-editor.org/info/rfc8034>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <https://www.rfc-editor.org/info/rfc8257>.

[RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <https://www.rfc-editor.org/info/rfc8290>.

[RFC8298] Johansson, I. and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia", RFC 8298, DOI 10.17487/RFC8298, December 2017, <https://www.rfc-editor.org/info/rfc8298>.

[RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <https://www.rfc-editor.org/info/rfc8312>.

[RFC8404] Moriarty, K., Ed. and A. Morton, Ed., "Effects of Pervasive Encryption on Operators", RFC 8404, DOI 10.17487/RFC8404, July 2018, <https://www.rfc-editor.org/info/rfc8404>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9330] Briscoe, B., Ed., De Schepper, K., Bagnulo, M., and G. White, "Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture", RFC 9330, DOI 10.17487/RFC9330, January 2023, <https://www.rfc-editor.org/info/rfc9330>.

[SCReAM-L4S] "SCReAM", commit fda6c53, June 2022, <https://github.com/EricssonResearch/scream>.

[SigQ-Dyn] Briscoe, B., "Rapid Signalling of Queue Dynamics", Technical Report, TR-BB-2017-001, DOI 10.48550/arXiv.1904.07044, September 2017, <https://arxiv.org/abs/1904.07044>.

Приложение A. Пример алгоритма DualQ Coupled PI2

В качестве первого конкретного примера ниже рассматривается алгоритм DualPI2, соответствующий схеме DualQ Coupled AQM на рисунке 1. Для Native L4S AQM применяется функция линейного изменения (ramp) (от времени в очереди) и несглаженной маркировкой ECNи может применяться также ступенчатая функция. Для Classic AQM применяется алгоритм PI2 [PI2], являющийся улучшенным вариантом PIE AQM [RFC8033].

Псевдокод вводится за два прохода. На первом объясняются основные понятия, а на втором - обработка граничных случаев, таких как перегрузка. Для удобства сравнения в обоих проходах применяется общая нумерация строк с буквенными суффиксами в случае добавления строк. Все переменные предполагаются действительными с плавающей запятой указывающими значения в базовых единицах (размер в байтах, время в секундах, скорость в байт/сек, alpha и beta в герцах и вероятности от 0 до 1). Предполагается, что константы, указанные с k (кило), M (мега), G (гига), u (микро), m (милли), % и т. д. преобразуются в соответствующие кратные или дробные значения в базовых единицах. В

фактической реализации на основе целых чисел потребуется обработка масштабных коэффициентов и соответствующая разрядность целых чисел (включая временные внутренние значения при расчётах).

Полный исходный код реализации для Linux доступен по ссылке https://github.com/L4STeam/sch_dualpi2_upstream и разъяснён в [DualPI2Linux]. Спецификация DualQ Coupled AQM для кабельных модемов DOCSIS и систем завершения (cable modem termination system или CMTS) приведена в [DOCSIS3.1] и разъяснена в [LLD].

A.1. Проход 1 - базовые концепции

Псевдокод манипулирует 3 основными структурами переменных: пакет (pkt), очередь L4S (lq), классическая очередь (cq). Ниже перечислены 6 функций псевдокода.

- Функция инициализации dualpi2_params_init(...) (Рисунок 2) устанавливает принятые по умолчанию значения параметров (API для установки других значений опущен для краткости).
- Функция постановки в очередь dualpi2_enqueue(lq, cq, pkt) (Рисунок 3).
- Функция извлечения из очереди dualpi2_dequeue(lq, cq, pkt) (Рисунок 4).
- Рекуррентная функция recur(q, likelihood) для дерандомизации маркировки ECN (показана в конце рисунка 4).
- Функция L4S AQM laqm(qdelay) (Рисунок 5) для расчёта вероятности маркировки ECN в очереди L4S.
- Функция базового AQM, реализующая алгоритм PI, dualpi2_update(lq, cq) (Рисунок 6). Служит для регулярного обновления базовой вероятности (p'), которая возводится в квадрат для Classic AQM, а также связывается с очередью L4S.

Используются также перечисленные ниже функции, которые не показаны полностью:

- scheduler() для выбора пакетов в начале очередей (выбор планировщика рассматривается ниже);
- cq.byt() или lq.byt() возвращает текущий размер (backlog) соответствующей очереди в байтах;
- cq.len() или lq.len() возвращает текущий размер соответствующей очереди в пакетах;
- cq.time() или lq.time() возвращает текущую задержку в соответствующей очереди (в единицах времени, см. примечания ниже);
- mark(pkt) и drop(pkt) для маркировки ECN и отбрасывания пакета.

В проведённых к настоящему времени экспериментах (на основе PIE) на каналах широкополосного доступа от 4 до 200 Мбит/с с базовым RTT от 5 до 100 мсек DualPI2 достигает высоких результатов с принятыми по умолчанию параметрами (Рисунок 2). Параметры классифицируются в зависимости от их связи с PI2 AQM, L4S AQM или схеме связывания. Константы и переменные, полученные из этих параметров, указаны в конце каждой категории. Для каждого параметра даны пояснения в точке его применения в псевдокоде с обоснованием заданных по умолчанию значений, чтобы можно было установить разумные значения в вариантах применения, отличающихся от общедоступной сети Internet.

```
1:  dualpi2_params_init(...) {          % Установка входных параметров по умолчанию
2:    % Параметры схемы Coupled DualQ
5:    limit = MAX_LINK_RATE * 250 ms                      % Размер двойного буфера
3:    k = 2                                                    % Фактор связывания
4:    % Не показано     % зависящий от планировщика вес или эквивалентный параметр
6:
7:    % Параметры PI2 Classic AQM
8:    target = 15 ms                                  % Целевая задержка в очереди
9:    RTT_max = 100 ms                    % Ожидаемое в худшем случае значение RTT
10:   % Константы PI2, выведенные из приведённых выше параметров PI2
11:   p_Cmax = min(1/k^2, 1)           % Макс. вероятн. отбрасыв./маркир. (Classic)
12:   Tupdate = min(target, RTT_max/3)                      % Интервал выборки PI
13:   alpha = 0.1 * Tupdate / RTT_max^2            % Интегральное усиление PI в Гц
14:   beta = 0.3 / RTT_max                      % Пропорциональное усиление PI в Гц
15:
16:   % Параметры L4S ramp AQM
17:   minTh = 800 us                            % Нижний порог маркировки L4S (время)
18:   range = 400 us                                    % Диапазон L4S (время)
19:   Th_len = 1 pkt                           % Нижний порог маркировки L4S (пакеты)
20:   % Константы L4S
21:   p_Lmax = 1                          % Максимальная вероятность маркировки L4S
22: }
```

Рисунок 2. Пример псевдокода заголовка для DualQ Coupled PI2 AQM.

Общая цель заключается в применении вероятности маркировки и отбрасывания для L4S и классического трафика (p_L и p_C). Значения выводятся из базовых вероятностей p'_L и p', соответственно, в соответствии с трафиком в очередях L и C. Вероятность маркировки в очереди L (p_L) зависит от базовой вероятности для этой очереди (p'_L) и вероятности p_CL, связанной с p' для очереди C (см. уравнения в параграфе 2.4).

```
1:  dualpi2_enqueue(lq, cq, pkt) { % Проверка размера и классификация lq или cq
2:    if ( lq.byt() + cq.byt() + MTU > limit)
3:      drop(pkt)                    % Отбрасывание пакета, если буфер заполнен
4:    timestamp(pkt)                    % Требуется лишь для метода пребывания
5:    % Packet classifier
6:    if ( ecn(pkt) modulo 2 == 1 )                  % Биты ECN = ECT(1) или CE
7:      lq.enqueue(pkt)
8:    else                                       % Биты ECN = not-ECT или ECT(0)
9:      cq.enqueue(pkt)
10: }
```

Рисунок 3. Пример псевдокода постановки в очередь для DualQ Coupled PI2 AQM.

Вероятности p_CL и p_C выводятся в строках 4 и 5 функции dualpi2_update() (Рисунок 6) и затем используются функцией dualpi2_dequeue(), где выводится p_L из p_CL в строке 6 (Рисунок 4). Приведённое ниже пошаговое описание кода разъясняет эту часть кода, начиная с прибытия пакета.

При поступлении пакетов сначала проверяется общее ограничение очереди, как показано в строке 2 псевдокода извлечения из очереди на рисунке 3. Это предполагает общий буфер для 2 очередей (в примечании b рассматриваются достоинства раздельных буферов). Во избежание предвзятости в отношении более крупных пакетов всегда разрешено свободное пространство 1 MTU и это преднамеренно проверяется перед постановкой в очередь.

Если передел не превышен, для пакета устанавливается временная метка в строке 4 (только при использовании метода пребывания в очереди для измерения задержки, см. примечание a ниже).

В строках 5-9 пакет классифицируется и помещается в очередь Classic или L4S в зависимости от младшего бита (least significant bit или LSB) поля ECN в заголовке IP (строка 6). Пакеты с кодом, имеющим 0 в LSB (Not-ECT и ECT(0)) помещаются в классическую очередь, а пакеты ECT(1) и CE в очередь L4S. Необязательная гибка классификация пакетов для краткости не показана (см. протокол L4S ECN [RFC9331]).

```
1:  dualpi2_dequeue(lq, cq, pkt) {  % Связывает очереди L4S и Classic
2:    while ( lq.byt() + cq.byt() > 0 ) {
3:      if ( scheduler() == lq ) {
4:        lq.dequeue(pkt)                    % Планировщик выбирает lq
5:        p'_L = laqm(lq.time())                        % Native LAQM
6:        p_L = max(p'_L, p_CL)                % Функция объединения
7:        if ( recur(lq, p_L) )                % Линейная маркировка
8:          mark(pkt)
9:      } else {
10:       cq.dequeue(pkt)                    % Планировщик выбирает cq
11:       if ( recur(cq, p_C) ) {             % probability p_C = p'^2
12:         if ( ecn(pkt) == 0 ) {        % Если в ECN указано not-ECT
13:           drop(pkt)                    % Квадратичное отбрасывание
14:           continue                     % Продолжение с начала цикла
15:         }
16:         mark(pkt)                        % Квадратичная маркировка
17:       }
18:     }
19:     return(pkt)                        % Возврат пакета и остановка
20:   }
21:   return(NULL)                         % Нет пакетов для извлечения
22: }
23: recur(q, likelihood) {    % Возврат TRUE с некоторой вероятностью
24:   q.count += likelihood
25:   if (q.count > 1) {
26:     q.count -= 1
27:     return TRUE
28:   }
29:   return FALSE
30: }
```

Рисунок 4. Пример псевдокода извлечения из очереди для DualQ Coupled PI2 AQM.

Псевдокод извлечения из очереди (Рисунок 4) вызывается всякий раз, когда нижний уровень готов переслать пакет. Планируется извлечение 1 пакета из очереди (если она не пуста), а затем управление возвращается вызвавшему, чтобы не возникало блокировки во время пересылки пакета. Принятие решения об извлечении из очереди сопряжено с решением AQM в части маркировки или отбрасывания. Применение AQM при постановке в очередь сместило бы часть обработки от критического момента извлечения пакета из очереди, однако увеличило бы задержку в очереди для сигналов управления, сделав контур управления менее точным (для типового RTT задержка в классической очереди удвоилась бы).

Код извлечения из очереди размещается в большом цикле while, поэтому при отбрасывании пакета процесс будет продолжаться, пока не будет выбран пакет для планирования. В строке 3 псевдокода извлечения из очереди планировщик выбирает очередь L4S (lq) или Classic (cq). Детали реализации планировщика не показаны (см. обсуждение ниже).

- Если запланирован пакет L4S строки 7 и 8 устанавливают для него маркер ECN с вероятностью p_L. Используется функция recur() в конце рисунка 4, которая предпочтительней случайной маркировки, поскольку она позволяет избежать задержки на рандомизацию при интерпретации сигналов перегрузки, сохраняя рассинхронизацию зубьев (пиков) потоков. Строка 6 рассчитывает p_L как большую из вероятностей L4S p_CL и Native L4S AQM p'_L. Это реализует функцию MAX(), показанную на рисунке 1, для связывания выхода двух AQM. В строке 6 p_L определяется двумя вероятностями:
  - p'_L рассчитывается для пакета в строке 5 с помощью функции laqm() (Рисунок 5);
  - p_CL поддерживается функцией dualpi2_update(), которая запускается в каждом интервале выборки Tupdate (строка 12 на рисунке 2).
- Если запланирован классический пакет, строки 10 - 17 маркируют или отбрасывают пакет с вероятностью p_C.

Алгоритм Native L4S AQM (Рисунок 5) - функция линейного изменения (ramp), похожая на упрощённый алгоритм RED:

- протяжённость рампы определяется в единицах задержки в очереди, а не в байтах, поэтому конфигурация не зависит от скорости извлечения из очереди;
- применяется мгновенное значение задержки в очереди, что позволяет избежать сложностей сглаживания, а также внесения худшего значения RTT в сглаживание задержки в сети (см. параграф 2.1);
- рампа линейно растёт от 0 до 1, а не до промежуточного значения p'_L, как в RED, поскольку не требуется сохранять малую вероятность маркировки ECN;

- маркировка не обязана быть случайной и используется детерминизм для сокращения задержки, требуемой для сглаживания случайных шумов сигнала.

```
1:  laqm(qdelay) {                % Возвращает вероятность Native L4S AQM
2:    if (qdelay >= maxTh)
3:      return 1
4:    else if (qdelay > minTh)
5:      return (qdelay - minTh)/range  % Деление можно выполнить сдвигом
6:    else
7:      return 0
8:  }
```

Рисунок 5. Пример псевдокода Native L4S AQM.

Для ramp-функции нужны 2 параметра конфигурации - нижний порог (minTh) и ширина рампы (диапазон) - , задаваемых в единицах времени, как показано в строках 17 и 18 функции инициализации на рисунке 2. Функцию рампы можно настроить как этап (примечание c).

Хотя статья о DCTCP [Alizadeh-stability] рекомендует порог маркировки ECN в 0,17*RTT_typ, так также сказано, что порог может быть существенно ниже при едва ли большей недогрузке канала (поскольку амплитуда зубьев в DCTCP достаточно мала). На основе множества экспериментов принятый по умолчанию нижний порог маркировки ECN для общедоступной сети Internet на рисунке 2 (target) считается хорошим компромиссом, даже будучи значительно меньше RTT_typ.

```
1:  dualpi2_update(lq, cq) {          % Обновление p' в каждом Tupdate
2:    curq = cq.time()  % Используется время в очереди первого входного пакета Classic
3:    p' = p' + alpha * (curq - target) + beta * (curq - prevq)
4:    p_CL = k * p'     % Связанная вероятн. L4S = базовая вероятн. * фактор связывания
5:    p_C = p'^2                  % Классическая вероятность = (базовая вероятность)^2
6:    prevq = curq
7:  }
```

Рисунок 6. Пример псевдокода обновления PI для DualQ Coupled PI2 AQM
(контроль диапазона [0,1] для p' опущен для простоты, см. ниже).

Вероятность связанной маркировки p_CL зависит от базовой вероятности (p'), актуальность которой поддерживается выполнением основного алгоритма PI (Рисунок 6) в каждом интервале Tupdate.

Отметим, что p' зависит лишь от времени пребывания в классической очереди. В строке 2 текущая задержка в очереди (curq) вычисляется как время нахождения первого (head) пакета в классической очереди (cq). Функция cq.time() (не показана) вычитает время, установленное при постановке в очередь, из текущего времени (см. примечание a ниже) и неявно принимает для текущей задержки в очереди значение 0, если очередь пуста.

Алгоритм основан на строке 3, которая является классическим контроллером PI, которые меняет значение p' в зависимости от a) разницы между текущей (curq) и целевой (target) задержкой в очереди и b) изменения задержки в очереди с момента последней выборки. Имя PI отражает то, что второй фактор (скорость роста очереди) пропорционален нагрузке, тогда как первый является интегралом от нагрузки (он исключает любую оставшуюся задержку в очереди сверх целевой).

Параметр target может быть установлен на основе локальных сведений, но цель состоит в том, чтобы принятое по умолчанию значение было хорошим компромиссом для любого места в предполагаемой среде развёртывания (общедоступная сеть Internet). Согласно [PI2param], целевая задержка в очереди (строка 8 на рисунке 2) связана с типовым базовым значением RTT по всему миру (RTT_typ) двумя факторами - target = RTT_typ * g * f. Ниже приведено обоснование этих факторов и введены дополнительные корректировки. Эти два фактора гарантируют, что в значительной части случаев (скажем, 90%) вариации зубьев RTT одного потока будут помещаться в буфер без недогрузки канала. Честно говоря, эти факторы являются обоснованными предположениями, но с большим акцентом на обоснованность, нежели предположения (см. [PI2param]).

- Для RTT_typ принимается значение 25 мсек на основе средней задержки CDN, измеренной в каждой стране, с весом, пропорциональным числу пользователей Internet в этой стране, для получения средневзвешенного значения в Internet [PI2param]. Страны ранжируются по числу пользователей Internet и после охвата 90% пользователь более мелкие страны были исключены, чтобы избежать мелких выборок, которые недостаточно представительны. Данные о средней задержке CDN в Китае (самое большое число пользователей) были исключены, поскольку задержка CDN там существенно отличалась, а экспериментальный метод показался неподходящим для Китая.

- Для g принимается значение 0,38. Фактор g является геометрическим коэффициентом, характеризующим форму зубьев в распространённых классических контроллерах перегрузки. Этот фактор указывает долю амплитуды зубьев пилы задержки в очереди, которые ниже цели AQM. Например, при малых скоростях геометрический фактор стандартного механизма Reno составляет 0,5, но при высоких скоростях он стремится к 1. Согласно переписи контроллеров перегрузки, выполненной Mishra с соавторами в июле-октябре 2019 г. [CCcensus19], большая часть трафика Classic TCP применяет CUBIC. Согласно анализу [PI2param], при работе через PI2 AQM большая часть этого трафика CUBIC будет в режиме совместимости с Reno, который имеет g~0,39 (для всех известных реализаций). Остальной трафик CUBIC будет в естественном режиме (true) CUBIC с g~0,36. Без моделирования профилей зубьев для менее распространённых контроллеров перегрузок средневзвешенное отношение этих двух параметров оценено как 7:3, что даёт среднее значение g = 0,38.

- Для f принимается значение 2. Это фактор безопасности, увеличивающий целевую задержку, чтобы обеспечить распределение RTT_typ вокруг среднего значения. В ином случае целевая очередь позволит избежать недогрузки лишь для пользователей со значениями меньше среднего. Параметр также обеспечивает запас прочности для путей, которые выходят за пределы дистанции между пользователем и его локальной CDN. В настоящее время нет данных о дисперсии задержки в очереди относительно среднего значения в каждом регионе, поэтому есть много способов сделать это допущение более обоснованным.

- [PI2param] рекомендует target = RTT_typ * g * f = 25 ms * 0,38 * 2 = 19 мсек. Однако оправдано дальнейшее уточнение, поскольку цель меняется с годами. Здесь использованы данные 2019 г., где упоминается глобальный индекс Speedtest, предлагающий сократить RTT_typ на 17% для стационарных пользователей и на

12% для мобильных в интервале 2020 - 2021 гг. Поэтому здесь рекомендуется использовать по умолчанию target = 15 мсек на момент написания (2021 г.).

Операторы могут использовать данные и обсуждение из [PI2param] для настройки более подходящей для их среды цели. Например, оператор может усомниться в допущениях этого документа, таких как цель исключения недогрузки для большинства однопотоковых передач (с учётом использования во многих крупных передачах нескольких потоков для исключения ограничения масштабирования классических потоков).

Два фактора усиления в строке 3 на рисунке 6 (alpha и beta) определяют, насколько сильно каждый из 2 элементов (интегральный и пропорциональный) меняет p'. Они выражаются в единицах «на секунду задержки» или Гц, поскольку преобразуют разницу задержки в очереди в изменение вероятности (вероятность имеет значения от 0 до 1). Alpha и beta определяют, насколько должно изменяться значение p' после каждого интервала обновления (Tupdate). Для меньших Tupdate значение p' следует менять на одну и ту же величину за секунду, но более мелкими и частыми долями. Значение alpha зависит от Tupdate (строка 13 функции инициализации на рисунке 2). Лучше обновлять p' как можно чаще, но значение Tupdate, вероятно, будет ограничено производительностью оборудования. Как показано в строке 12, интервал обновления следует делать достаточно малым, чтобы обновление происходило хотя бы 1 раз за время, требуемое для «осушения» целевой очереди (target) при условии, что она обновляется не меньше 3 раз в течение максимального RTT. По умолчанию Tupdate имеет значение 16 мсек в эталонной реализации Linux, поскольку его следует округлять до значения, кратного 4 мсек. Для скоростей канала от 4 до 200 Мбит/с и максимального RTT в 100 мсек в ходе широкого тестирования подтверждено, что Tupdate = 16 мсек вполне достаточно (это же значение рекомендует спецификация PIE [RFC8033]). Выбор alpha и beta также определяет диапазон стабильной работы AQM. Механизм AQM должен менять p' как можно быстрее в ответ на изменение нагрузки без перекомпенсации и соответствующий флуктуаций очереди. Поэтому значения alpha и beta зависят также от RTT в ожидаемом худшем случае потока (RTT_max).

Максимальное значение RTT контроллера PI (RTT_max в строке 9 на рисунке 2) не является абсолютным максимумом, но для долгосрочных потоков с RTT больше этого значения возникает большая нестабильность (изменчивость очереди). Задержка распространения на полпути вокруг планеты и обратно в стеклянном волокне составляет 200 мсек. Однако почти никакой трафик не проходит по такому пути в результате значительной консолидации трафика Internet в интервале 2007 - 2009 гг. [Labovitz10], а также обслуживания большей и растущей доли трафика Internet (примерно 2/3 на момент написания документа) в CDN или облачных системах, размещённых вблизи пользователей. Internet может снова измениться, но сейчас, проектирование для максимального RTT в 100 мсек является хорошим компромиссом между ускорением контроля очередей при низком RTT и некоторой нестабильностью с случаях длинных путей.

Рекомендуемые производные значения констант усиления alpha и beta можно аппроксимировать для Reno через PI2 AQM выражениями alpha = 0,1 * Tupdate / RTT_max^2; beta = 0,3 / RTT_max, как показано в строках 13 и 14 на рисунке 2. Они получены из анализа стабильности в [PI2]. Для принятых по умолчанию значения Tupdate = 16 мсек и RTT_max = 100 мсек это даёт alpha = 0,16, beta = 3,2 (расхождения обусловлены округлением). Эти принятые по умолчанию значения проверены для широкого диапазона скоростей каналов, целевой задержки и моделей трафика со смешанными и похожими значениями RTT, короткими и долгими потоками и т. п.

В крайних случаях p' может выходить из диапазона [0,1], поэтому результирующее значение p' должно ограничиваться (не показано в псевдокоде). Затем, как разъяснено выше, из нового значения p' выводится связанная и классическая вероятность в строках 4 и 5 на рисунке 6 как p_CL = k*p' и p_C = p'^{2}.

Поскольку вероятность связанной маркировки L4S (p_CL) включает коэффициент k, параметры динамического усиления alpha и beta также умножаются на k для очереди L4S. Таким образом, эффективный коэффициент усиления для очереди L4S имеет значение k*alpha (с принятым по умолчанию alpha = 0,16 Гц и k = 2, эффективное значение для L4S alpha = 0,32 Гц).

В отличие от PIE [RFC8033], значения alpha и beta не требуется корректировать в каждом Tupdate с учётом p'. В PI2 значения alpha и beta не зависят от p', поскольку возведение в квадрат для классического трафика уже настраивает их. Это разъяснено в [PI2], где также указано, почему этот подход устраняет потребность в большей части эвристики, добавленной в PIE.

Тем не менее, разработчики могут захотеть добавить отдельные детали в любой механизм AQM. Например, эталонная реализация Linux DualPI2 включает указанное ниже (не показано в псевдокоде).

- Классическая и связанная маркировка или отбрасывание (на основе p_C и p_CL от контроллера PI) не применяются к пакету, если размер агрегированной очереди в байтах < 2 MTU (до включения пакета в очередь или извлечения его в зависимости от места применения AQM).
- В планировщике WRR «кредит», указывающей, какой очереди следует передавать, меняется лишь при наличии пакетов в обеих очередях (т. е. при фактической конкуренции за ресурсы). Это означает, что поток L с подобающим темпом передачи не может быть задержан WRR. Кретид WRR сбрасывается в пользу очереди L, когда канал бездействует.

Разработчики могут добавлять и другую эвристику, например обычную [RFC8033] или улучшенную [RFC8034] защиту от всплесков (burst).

Примечания

a. Скорость «осушения» может меняться, если она запланирована для двух очередей или учитывает флуктуации беспроводной среды. Для автоматической адаптации к изменениям скорости очередь должна измеряться во времени, а не в байтах или пакетах [AQMmetrics] [CoDel]. Задержку в очереди можно измерить напрямую как время пребывания в очереди путём записи времени постановки в очередь каждого пакета и вычитания его из системного времени при извлечении пакета из очереди. Если временные метки сложно реализовать в оборудовании, задержку в очереди можно предсказать путём деления размера очереди на прогнозируемую скорость выхода, которую можно узнать точно для некоторых сетевых технологий (см., например, DOCSIS PIE [RFC8034]).

   Однако время пребывания неудобно для обнаружения пиков. Например, если всплеск (burst) пакетов попадает в пустую очередь, время пребывания полностью отражает задержку всплеска лишь при извлечении из очереди

последнего пакета, хотя размер всплеска известен очереди с момента постановки в неё последнего пакета этого всплеска, и можно было бы сообщить о перегрузке раньше. Для устранения этой проблемы каждый пакет в начале очереди можно помечать при выходе из очереди на основе ожидаемой задержки последнего пакета, как описано ниже, а не по задержке самого пакета в голове очереди из-за предшествующих пакетов. Недогрузка при пиках трафика в [Heist21] указывает конкретный случай, когда всплеск трафика значительно снижает использование очереди L. Если этот эффект будет применяться шире, использование задержки за головой очереди может повысить производительность.

Определение задержки за головой очереди можно реализовать делением отставания (backlog) при извлечении из очереди на скорость канала или умножением отставания на задержку в единицах отставания. Детали реализации будут зависеть от знания скорости канала, а если она неизвестна, можно использовать среднее значение задержки в единицах отставания. Эта задержка включает сериализацию, а также получение доступа к общей среде, поэтому детали будут сильно зависеть от канальной технологии. Этот подход менее чувствителен к ошибкам синхронизации и требует меньше операций и памяти, чем эквивалентный показатель масштабируемого времени пребывания в очереди, где время пребывания масштабируется отношением размеров очереди при извлечении пакета из очереди и помещении в неё [SigQ-Dyn].

b. Строка 2 в функции dualpi2_enqueue() (Рисунок 3) предполагает реализацию, где lq и cq используют общий буфер в памяти. Другая реализация ACK может использовать разные буферы для каждой очереди и в этом случае прибывающие пакеты нужно сначала классифицировать, чтобы определить какой буфер нужно проверять на предмет наличия места. Выбор подхода определяется компромиссом, поскольку общий буфер позволяет занять меньше памяти, а раздельные буферы изолируют очередь L4S от отбрасывания хвоста из-за больших всплесков классического трафика (например, Classic Reno TCP во время замедленного старта при большом значении RTT).

c. Имеется некоторая обеспокоенность тем, что применение ступенчатой функции из DCTCP для Native L4S AQM требует от конечных систем сглаживания сигнала для неоправданно большого числа интервалов кругового обхода, чтобы обеспечить достаточную точность. В первоначальных экспериментах с имеющимся DCTCP рампа показала себя не хуже, чем шаг. Поэтому рекомендуется настраивать рампу вместо ступени, что позволит алгоритмам контроля перегрузки исследовать более быстрые алгоритмы сглаживания.

Рампа применяется чаще, нежели ступени, поскольку оператор может превратить рампу в ступенчатую функцию, используемую в DCTCP, установив нулевой диапазон. В строке 5 на рисунке 5 не возникает деления на 0, поскольку при minTh = maxTh не может возникнуть условий для расчёта рампы.

A.2. Проход 2 - граничные случаи

Этот параграф содержит второй проход по псевдокоду с добавлением деталей для 2 граничных случаев - малой скорости канала и перегрузки. На рисунке 7 повторена функция извлечения из очереди (Рисунок 4) с деталями для обоих граничных случаев, а рисунок 8 повторяет базовый алгоритм PI (Рисунок 6) с деталями для перегрузки. Функции инициализации, извлечения из очереди, L4S AQM и recur остаются неизменными.

Скорость канала может быть настолько низкой, что сериализация одного пакета из очереди может занимать время, превышающее пороговую задержку, при которой маркировка ECN начинает применяться для очереди L. Поэтому параметр нижнего порога маркировки требуется задавать в пакетах, а не по времени (Th_len по умолчанию имеет значение в 1 пакет, строка 19 на рисунке 2), чтобы рампа гарантированно не вызывала чрезмерной маркировки на медленных каналах. Когда реализация знает скорость канала, она может установить этот минимум во время настройки. Например, величина в 1 MTU будет делиться на скорость канала для получения времени сериализации, затем, если нижний порог рампы Native L AQM меньше этого времени сериализации, реализация может повысить пороги для сдвига верхней части рампы к значению 2 MTU. Такой подход применяется в DOCSIS [DOCSIS3.1], поскольку настроенная скорость канала предназначена для DualQ.

Приведённый здесь псевдокод применяется, когда скорость соединения неизвестна, что характерно для программных реализаций, которые могут быть развёрнуты там, где канал используется совместно с другими очередями. В строках 5a - 5d на рисунке 7 вероятность естественной маркировки L4S (p'_L) обнуляется, если в очереди имеется лишь 1 пакет (в принятой по умолчанию конфигурации).

> Примечание для разработчиков Linux. В Linux проверка того, что очередь превышает Th_len перед маркировкой Native L4S AQM фактически выполняется при постановке в очередь, а не при извлечении, поскольку в ином случае это исключило бы маркировку последнего пакета в пике. Результат проверки передаётся от постановки в очередь к извлечению из неё через логическую переменную (флаг) в метаданных пакета.

Считается, что возникла постоянная перегрузка, когда вероятность классического отбрасывания/маркировки достигает p_Cmax. Выше этого порга классическая вероятность отбрасывания применяется к обеим очередям (L и C) независимо от поддержки ECN в пакетах. Пакеты ECT, которые не отброшены, могут помечаться ECN.

В строке 11 функции инициализации (Рисунок 2) максимальная вероятность классического отбрасывания p_Cmax = $\min(1/k^2, 1)$ или 1/4 для принятого по умолчанию фактора связывания k = 2. На практике было установлено, что 25% является хорошим значением для порога, чтобы сохранить беспристрастность между пакетами с поддержкой и без поддержки ECN. Это защищает очереди как от временной перегрузки отзывчивыми потоками, так и от более продолжительной перегрузки от любого неотзывчивого трафика, которые ложно объявляет реакцию на ECN.

Когда вероятность маркировки Classic ECN достигает порога p_Cmax ($1/k^2$), с ней связывается вероятность для очереди L4S (p_CL), которая будет равна 100% для любого k (в соответствии с уравнением (1) из параграфа 2.1). Поэтому для удобочитаемости константа p_Lmax указана как 1 в строке 21 функции инициализации (Рисунок 2). Это сделано для гарантии того, что очередь L4S начнёт отбрасывать пакеты, как только маркировка ECN достигнет 100% и не сможет расти дальше. Требования Prague L4S [RFC9331] указывают, что при обнаружении контролем перегрузки L4S отбрасывания пакета, он возвращается к откликам, сосуществующим с контролем перегрузки Classic. Таким образом, при отбрасывании пакетов очередью L4S правильно отбрасывать их пропорционально p'^2, как будто они являются классическими.

Каждая из двух очередей проверяет перегрузку в строках 4b и 12b функции извлечения из очереди (Рисунок 7). Строки 8c - 8g отбрасывают пакеты L4S с вероятностью p'^2, строки 8h - 8i маркируют оставшиеся пакеты с вероятностью p_CL. С учётом p_Lmax = 1, все оставшиеся пакеты будут промаркированы, поскольку для достижения блока else в строке 8b должно выполняться условие p_CL >= 1.

```
1:  dualpi2_dequeue(lq, cq, pkt) {  % Связывает очереди L4S и Classic
2:    while ( lq.byt() + cq.byt() > 0 ) {
3:      if ( scheduler() == lq ) {
4a:       lq.dequeue(pkt)                          % Запланировано L4S
4b:       if ( p_CL < p_Lmax ) {                  % Проверка насыщения
5a:         if (lq.len()>Th_len)                  % >1 пакета в очереди
5b:           p'_L = laqm(lq.time())                    % Native LAQM
5c:         else
5d:           p'_L = 0  % Подавление маркировки в очереди с 1 пакетом
6:          p_L = max(p'_L, p_CL)              % Функция комбинирования
7:          if ( recur(lq, p_L)                   %Линейная маркировка
8a:           mark(pkt)
8b:       } else {                                    % Насыщение
8c:         if ( recur(lq, p_C) ) {           % вероятность p_C = p'^2
8e:           drop(pkt) % Возврат к классич. отбрас. из-за перегрузки
8f:           continue                    % Продолжение с начала цикла
8g:         }
8h:         if ( recur(lq, p_CL) )         % Вероятность p_CL = k * p'
8i:           mark(pkt)       % Линейная маркировка оставшихся пакетов
8j:       }
9:      } else {
10:       cq.dequeue(pkt)             % Запланирован классический пакет
11:       if ( recur(cq, p_C) ) {             % Вероятность p_C = p'^2
12a:        if ( (ecn(pkt) == 0)                      % ECN = not-ECT
12b:             OR (p_C >= p_Cmax) ) {     % Перегрузка отключает ECN
13:           drop(pkt)     % Квадратичное отбрасывание, повтор цикла
14:           continue                    % Продолжение с начала цикла
15:         }
16:         mark(pkt)                        % Квадратичная маркировка
17:       }
18:     }
19:     return(pkt)                         % Возврат пакета и остановка
20:   }
21:   return(NULL)                          % Нет пакетов для извлечения
22: }
```

Рисунок 7. Пример псевдокода извлечения из очереди для DualQ Coupled PI2 AQM (включая код для граничных случаев).

Строка 2a базового алгоритма PI (Рисунок 8) обрабатывает перегрузку очереди L4S, когда классический трафик мал или отсутствует. Это необходимо, поскольку алгоритм PI поддерживает подходящую вероятность отбрасывания для регулирования перегрузки, но это зависит от размера классической очереди. Если классическая очередь мала или её нет, естественная функция обновления PI (Рисунок 6) ничего не отбрасывает даже при перегрузке очереди L4S, поэтому приходиться принимать на себя функцию отбрасывания хвоста (строки 2 и 3 на рисунке 3).

```
1:  dualpi2_update(lq, cq) {           % Обновление p' в каждом Tupdate
2a:   curq = max(cq.time(), lq.time()) % Использование большего времени
3:    p' = p' + alpha * (curq - target) + beta * (curq - prevq)
4:    p_CL = p' * k  % Coupled L4S prob = base prob * coupling factor
5:    p_C = p'^2             % Классич. вероятн. = (базовая вероятн.)^2
6:    prevq = curq
7:  }
```

Рисунок 8. Пример псевдокода обновления PI для DualQ Coupled PI2 AQM (включая код перегрузки).

Вместо этого строка 2a полной функции обновления PI (Рисунок 8) гарантирует, что Base PI AQM в строке 3 определяется тем, какая из двух задержек в очередях больше, но строка 3 всегда использует цель Classic (по умолчанию 15 мсек). Если задержка в очереди L больше лишь потому, что классического трафика мало или нет совсем, она обычно все равно гораздо ниже цели Base AQM. Это связано с тем, что трафик L4S регулируется также низким порогом своего Native AQM (строки 5a - 6 алгоритма извлечения из очереди на рисунке 7). Таким образом, Base AQM будет сведён к 0 и не внесёт вклада. Однако при перегрузке очереди L трафиком, не отвечающим на маркировку, max() в строке 2a на рисунке 8 позволяет очереди L плавно перевести Base AQM в режим перегрузки даже при отсутствии или малом объёме классического трафика. После этого Base AQM будет сохранять для очереди L классическую цель (по умолчанию 15 мсек) сбрасывая пакеты L.

Выбор технологии планировщика очень важен для защиты от перегрузки (см. параграф 4.2.2).

- Рекомендуется понятный планировщик с поддержкой весов, такой как WRR. Поскольку вес для классического трафика мал (например, 1/16), его точное значение не важно, поскольку он обычно не влияет на распределение пропускной способности. Вес важен только для предотвращения краткосрочного истощения классического трафика неотзывчивым трафиком L4S (см. параграф 4.2.2). Это обусловлено тем, что распределение пропускной способности между очередями обычно определяется связанным сигналом перегрузки, который переопределяет действия планировщика, заставляя источники L4S оставлять для классических потоков примерно равную пропускную способность.

- Как вариант, можно применять FIFO со сдвигом по времени (TS-FIFO). Этот механизм выбирает пакет в начале очереди, который ждёт дольше всего, со смещением относительно классического трафика на величину tshift. Для реализации TS-FIFO функция scheduler() в строке 3 кода извлечения из очереди реализуется просто как функция scheduler() в нижней части рисунка 9 в Приложении B. Для общедоступной сети Internet подходит значение tshift = 50 мсек. Для частных сетей с меньшим диаметром, подойдёт значение около 4*target. Планировщик TS-FIFO очень прост, но может потребовать усложнения, связанного с устранением некоторых недостатков (именно поэтому не рекомендуется применять его вместо WRR).

  - TS-FIFO не обеспечивает полной изоляции задержки в очереди L4S от неконтролируемых всплесков в классической очереди;
  - использование времени пребывания для TS-FIFO осмысленно лишь при поддержке для пакетов временных меток;
  - даже при поддержке временных меток время пребывания для пакета из головы очереди всегда «застывшее» (stale), поэтому можно применять более оперативную (мгновенную) меру задержки в очереди (см. примечания в Приложении A.1).
- Планировщик со строгим приоритетом не подходит, как было отмечено в параграфе 4.2.2.

Приложение B. Пример алгоритма DualQ Coupled Curvy RED

В качестве другого примера DualQ Coupled AQM ниже представлен псевдокод на основе алгоритма Curvy-RED. Хотя механизм AQM разработан для целочисленной арифметики, для упрощения понимания сначала используются действительные числа с плавающей запятой (Рисунок 10). Затем показана возможная оптимизация для целочисленной арифметики (Рисунок 11). Для удобства сравнения нумерация строк на рисунках общая с использованием буквенных суффиксов в добавленных строках.

B.1. Псевдокод Curvy RED

Псевдокод манипулирует 3 основными структурами переменных: пакет (pkt), очередь L4S (lq), классическая очередь (cq). Ниже перечислены 6 функций псевдокода.

- Функция инициализации dualpi2_params_init(...) (Рисунок 2) устанавливает принятые по умолчанию значения параметров (API для установки других значений опущен для краткости).
- Функция извлечения из очереди dualpi2_dequeue(lq, cq, pkt) (Рисунок 4).
- Функция планирования scheduler(), выбирающая пакет в начале одной из 2 очередей.

Используются также перечисленные ниже функции, которые не показаны полностью:

- функция извлечения из очереди, идентичная применяемой в DualPI2 функции dualpi2_enqueue(lq, cq, pkt) (Рисунок 3);
- mark(pkt) и drop(pkt) для маркировки ECN и отбрасывания пакета;
- cq.byt() или lq.byt() возвращает текущий размер (backlog) соответствующей очереди в байтах;
- cq.time() или lq.time() возвращает текущую задержку в соответствующей очереди (в единицах времени, см. примечания в Приложении A.1).

Поскольку Curvy RED был оценён до DualPI2, некоторые улучшения для DualPI2 не оценивались в Curvy RED. В приведённом псевдокоде добавлены прямые улучшения на основе допущения, что они обеспечат аналогичные преимущества, но это не было подтверждено экспериментально. Улучшения включают i) планировщик с приоритетом по условию вместо строго приоритета, ii) временной порог для Native L4S AQM и iii) поддержка ECN для Classic AQM. Недавняя оценка показала, что нижний порог маркировки ECN (minTh) значительно повышает производительность, поэтому он включён в псевдокод.

Защита от перегрузки не включена в псевдокод Curvy RED, чтобы не отвлекать внимание от основных характеристик. Её можно добавить таким же способом, как в Приложении A.2 для псевдокода DualPI2. Native L4S AQM использует ступенчатый порог, но можно использовать и рампу, подобно описанной для DualPI2. Планировщик использует простой алгоритм TS-FIFO, но его можно заменить на WRR.

Алгоритм Curvy RED не поддерживался и не оценивался в той же степени, что и DualPI2. В первых экспериментах на каналах широкополосного доступа со скоростью 4 - 200 Мбит/с и базовым RTT от 5 до 100 мсек алгоритм Curvy RED показал хорошие результаты с установленными по умолчанию параметрами, как показано на рисунке 9.

```
1:  cred_params_init(...) {       % Установка параметров по умолчанию
2:    % DualQ Coupled framework parameters
3:    limit = MAX_LINK_RATE * 250 ms         % Размер двойного буфера
4:    k' = 1                          % Фактор связывания как степень 2
5:    tshift = 50 ms              % Сдвиг времени планировщика TS-FIFO
6:    % Константы, выведенные из параметров Classic AQM
7:    k = 2^k'                    % Фактор связывания из уравнения (1)
6:
7:    % Параметры Classic AQM
8:    g_C = 5              % Параметр сглаживания EWMA как степень 1/2
9:    S_C = -1    % Фактор масштабирования Classic ramp как степень 2
10:   minTh = 500 ms       % Ниже этой задержки нет Classic drop/mark
11:   % Константы, выведенные из параметров Classic AQM
12:   gamma = 2^(-g_C)                    % Параметр сглаживания EWMA
13:   range_C = 2^S_C                         % Диапазон Classic ramp
14:
15:   % Параметры L4S AQM
16:   T = 1 ms          % Порог задержки в очереди для Native L4S AQM
17:   % Константы, выведенные из приведённых выше параметров
18:   S_L = S_C - k'  % Фактор масштабирования L4S ramp как степень 2
19:   range_L = 2^S_L                             % Диапазон L4S ramp
20: }
```

Рисунок 9. Пример псевдокода заголовка для DualQ Coupled Curvy RED AQM.

Параметры классифицированы по принадлежности к Classic AQM, L4S AQM и схеме связывания. Константы и переменные, выведенные из этих параметров включены в конце каждой категории. Это исходные (raw) входные параметры алгоритма. Модуль настройки конфигурации может воспринимать более значимые параметры (например,

RTT_max и RTT_typ) и преобразовывать их в необработанные (raw), как было сделано для DualPI2 в Приложении A. При необходимости параметры объясняются в псевдокоде.

Псевдокод извлечения из очереди (Рисунок 10) вызывается всякий раз, когда нижний уровень готов переслать пакет. Функция планирует извлечение из очереди 1 пакета (при наличии) и возвращает управления вызвавшей стороне, чтобы не возникало блокировки во время пересылки пакета. Принимая решение об извлечении из очереди, функция также принимает требуемые решения AQM в части отбрасывания или маркировки. Применение AQM при постановке в очередь сместило бы часть обработки от критического момента извлечения пакета из очереди, однако увеличило бы задержку в очереди для сигналов управления, сделав контур управления очень неточным.

```
1:  cred_dequeue(lq, cq, pkt) {       % Связывает очереди L4S и Classic
2:    while ( lq.byt() + cq.byt() > 0 ) {
3:      if ( scheduler() == lq ) {
4:        lq.dequeue(pkt)                          % Запланировано L4S
5a:       p_CL = (Q_C - minTh) / range_L
5b:       if ( ( lq.time() > T )
5c:          OR ( p_CL > maxrand(U) ) )
6:          mark(pkt)
7:      } else {
8:        cq.dequeue(pkt)                       % Запланировано Classic
9a:       Q_C = gamma * cq.time() + (1-gamma) * Q_C % Classic Q EWMA
10a:      sqrt_p_C = (Q_C - minTh) / range_C
10b:      if ( sqrt_p_C > maxrand(2*U) ) {
11:         if ( (ecn(pkt) == 0)  {                    % ECN = not-ECT
12:           drop(pkt)       % Квадратичное отбрасывание, повтор цикла
13:           continue                     % Продолжение с начала цикла
14:         }
15:         mark(pkt)
16:       }
17:     }
18:     return(pkt)                          % Возврат пакета и остановка
19:   }
20:   return(NULL)                           % Нет пакета для извлечения
21: }
22: maxrand(u) {      % Возвращает максимальное из u случайных значений
23:   maxr=0
24:   while (u-- > 0)
25:     maxr = max(maxr, rand())                  % 0 <= rand() < 1
26:   return(maxr)
27: }
28: scheduler() {
29:   if ( lq.time() + tshift >= cq.time() )
30:     return lq;
31:   else
32:     return cq;
33: }
```

Рисунок 10. Пример псевдокода извлечения из очереди для DualQ Coupled Curvy RED AQM.

Код предполагает применение AQM при извлечении из очереди[1]. Весь код извлечения из очереди размещён в большом цикле while поэтому при отбрасывании пакета процесс будет продолжаться, пока не будет выбран пакет для планирования. Если обе очереди пусты, функция возвращает NULL в строке 20. Строка 3 определяет очередь (L4S (lq) или Classic (cq)), из которой планировщик с условным приоритетом возьмёт пакет. Планировщик TS-FIFO, показанный в строках 28-33, подойдёт, если простота имеет первостепенное значение[2]. В каждой очереди решение о пересылке, отбрасывании или маркировке принимается, как описано ниже (для простоты принимается U = 1):

L4S

Если проверка в строке 3 указывает извлечение из очереди L4S, проверка в строках 5b и 5c решает вопрос о маркировке. Строка 5b сравнивает задержку в очереди L4S (lq.time()) с порогом ступени T[3], а строка 5c похожа на случайную маркировку ECN в RED, но i) решение зависит от времени в очереди, а не от байтов для возможности изменения скорости канала без перенастройки, ii) маркировка в очереди L4S зависит от логического ИЛИ двух проверок - по времени в этой и другой (Classic) очереди, iii) проверки выполняются для мгновенного значения времени в очереди L4S и сглаженного среднего в очереди Classic, iv) очередь сравнивается с максимальным из U случайных значений (при U = 1 это совпадает с использованием одного случайного значения в RED).

В строке 5a для связанной вероятности маркировки p_CL устанавливается значение, определяемое разностью усреднённой задержки в очереди Classic (Q_C) и нижнего порога задержки в очереди (minTh), разделённой на параметр масштабирования L4S (range_L). Значение range_L представляет добавленную к minTh задержку в очереди (в секундах), при которой вероятность маркировки составит 100%. В строке 5c (если U = 1) результат сравнивается с однородно распределенной случайной величиной из диапазона от 0 до 1, что гарантирует для range_L линейный рост вероятности маркировки при увеличении времени в очереди.

Classic

Если планировщик в строке 3 выбирает классический пакет и переходит в строку 7, проверка в строке 10b решает вопрос о маркировке или отбрасывания пакета. Но пред этим строка 9a обновляет значение Q_C, которое является экспоненциально взвешенным скользящим средним значением [4] времени пребывания в классической

[1]Применение AQM при постановке в очередь сместило бы часть обработки от критического момента извлечения пакета из очереди, однако увеличило бы задержку в очереди для сигналов управления, сделав контур управления менее точным (для типового RTT задержка в классической очереди удвоилась бы). На платформе с возможностью установки меток времени, например, Linux, проще всего применять AQM при извлечении из очереди, поскольку именно тогда измеряется проведённое в очереди время.

[2]WRR лучше изолирует очередь L4S от сильный всплесков в очереди Classic, но он сложнее TS-FIFO. При использовании WRR нужно по умолчанию задавать малый вес очереди Classic (например, 1/16) вместо временного сдвига в строке 5 (Рисунок 9).

[3]Ступенчатая функция показана для простоты. Рекомендуется применять рампу (см. Рисунок 5 и обсуждение в Приложении A.1), поскольку она более распространена и может обеспечить более быстрое схождения контроля перегрузки L4S.

[4]EWMA - лишь один из возможных способов фильтрации всплесков. Могут подойти более адаптивные методы сглаживания, а также сокращение EWMA быстрее роста, например, используя меньшую из сглаженной и мгновенной задержки min(Q_C, qc.time()).

очереди, где cq.time() - текущее мгновенное время пребывания пакета в голове классической очереди (0, если очередь пуста), а gamma - константа экспоненциально взвешенного скользящего среднего значения (exponentially weighted moving average или EWMA), по умолчанию равная 1/32 (строка 12 функции инициализации).

Строки 10a и 10b реализуют Classic AQM. В строке 10a средневзвешенное время Q_C делится на параметр классического масштабирования range_C, как для при масштабировании времени в очереди для маркировки L4S. Это масштабированное время в очереди будет возводиться в квадрат для определения вероятности классического отбрасывания. До возведения в квадрат оно фактически является квадратным корнем из вероятности отбрасывания, поэтому переменная названа sqrt_p_C. Возведение в квадрат выполняется путём сравнения значения с большим из 2 случайных чисел (при U = 1), эквивалентного логической операции И для двух проверок, что обеспечивает квадратичный рост вероятности отбрасывания в зависимости от времени пребывания в очереди.

Функции AQM в каждой очереди (строки 5c и 10b) являются вариантами нового обобщённого механизма RED, называемого Curvy RED. Когда производительность этого AQM сравнивалась с FQ-CoDel и PIE, цель удержать задержку в очереди на фиксированном уровне сочли ошибочной [CRED_Insights]. Если по мере роста числа потоков AQM не позволяет контроллерам перегрузки на хосте увеличивать задержку в очереди, контроллер будет вносить аномально высокие потери и они, а не очереди, становятся основной причиной задержки коротких потоков из-за тайм-аутов и отбрасывания в хвосте очереди.

Curvy RED ограничивает задержку со смягчённой целью, что позволяет некоторое увеличение задержки при росте нагрузки. Это достигается путём увеличения вероятности отбрасывания на выпуклой кривой относительно роста скорости (квадратичная кривая в классической очереди при U = 1). Как и в RED, кривая огибает нулевое значение оси, пока очередь неглубокая. По мере роста нагрузки вводится возрастающий барьер для высокой задержки. Но, в отличие от RED, используется два параметра, а не три. Недостатком Curvy RED (например, по сравнению с PI) является неприспособленность к широкому диапазону RTT. Curvy RED можно использовать в неизменном виде при ограниченном диапазоне RTT, а иных случаях требуется механизм адаптации.

По результатам ограниченных экспериментов с Curvy RED рекомендуются значения S_C = -1, g_C = 5, T = 5 * MTU при скорости канала (около 1 мсек для 60 Мбит/с) для типичного в общедоступной сети Internet диапазона RTT. В [CRED_Insights] объяснено, почему эти параметры применимы независимо от скорости канала, на котором развёрнута эта реализация AQM и как нужно скорректировать параметры для другого диапазона RTT (например, в ЦОД). Установка k определяется политикой (см. параграф 2.5 и C.2, где указаны значения по умолчанию и варианты).

Имеется также параметр кривизны (cUrviness, U), который задаётся небольшим целым числом. Скорей всего, он будет иметь жёстко заданное значение для всех реализаций по завершении экспериментов. До сих пор эксперименты проводились при U = 1, но результаты могут оказаться лучше при U = 2 или выше.

B.2. Эффективная реализация Curvy RED

Хотя оптимизация кода зависит от платформы, ниже приведены разъяснения эффективности реализации Curvy RED как мотива.

Classic AQM в строке 10b на рисунке 10 вызывается функция maxrand(2*U), что даёт вдвое большую кривизну, нежели maxrand(U), для функции маркировки в строке 5c. Этот трюк реализует правило квадрата в уравнении (1) из параграфа 2.1 на основе того, что при заданном X от 1 до 6 вероятность того, что при двух бросках кости выпадут значения меньше X равна квадрату вероятности того, что при одном броске выпадет значение меньше X. Поэтому при U = 1 функция маркировки L4S линейна, а функция классического отбрасывания квадратична. При U = 2 функция L4S будет квадратичной, а классическая - четвёртой степени и т. д.

Функция maxrand(u) в строках 22-27 просто генерирует u случайных значений и возвращает большее из них. Обычно maxrand(u) можно запускать параллельно. Например, при U = 1 для классической очереди требуется не более 2 случайных значений, поэтому вместо вызова maxrand(2*U) в основном потоке исполнения максимальное значение в каждой паре от генератора псевдослучайных чисел можно получить отдельно и держать в буфере, готовом для передачи в классическую очередь.

```
1:  cred_dequeue(lq, cq, pkt) {   %  Связывает очереди L4S и Classic
2:    while ( lq.byt() + cq.byt() > 0 ) {
3:      if ( scheduler() == lq ) {
4:        lq.dequeue(pkt)                          % Запланировано L4S
5:        if ((lq.time() > T) OR (Q_C >> (S_L-2) > maxrand(U)))
6:          mark(pkt)
7:      } else {
8:        cq.dequeue(pkt)                       % Запланировано Classic
9:        Q_C += (qc.ns() - Q_C) >> g_C             % Classic Q EWMA
10:       if ( (Q_C >> (S_C-2) ) > maxrand(2*U) ) {
11:         if ( (ecn(pkt) == 0)  {                  % ECN = not-ECT
12:           drop(pkt)       % Квадратный корень, продолжение цикла
13:           continue            % Продолжение с начала цикла while
14:         }
15:         mark(pkt)
16:       }
17:     }
18:     return(pkt)                       % Возврат пакета и остановка
19:   }
20:   return(NULL)                         % Нет пакета для извлечения
21: }
```

Рисунок 11. Оптимизированный пример псевдокода извлечения из очереди для DualQ Coupled AQM с использованием целочисленной арифметики.

Диапазоны range_L и range_C выражаются степенью 2, поэтому деление в строках 5 и 11 можно выполнить побитовым сдвигом вправо (bit-shift, >>) для целочисленного варианта псевдокода (Рисунок 11). Для этого варианта целочисленная версия функции rand(), используемая в строке 25 функции maxrand() на рисунке 10 будет возвращать целое число 0 <= maxrand() < 2^{32} (не показано), что будет умножать действительное значение вероятности из диапазона [0,1] на 2^{32}. Задержки в очереди также будут умножаться на 2^{32}, но в два этапа: i) в строке 9 время в очереди qc.ns() возвращается как целое число наносекунд, которое примерно в 2^{30} больше числа секунд, а ii) в строках 5 и 10,

исключение (-2) двух позиций при сдвиге вправо ведёт к увеличению результата в 2^2 раза для полного умножения на 2^{32}. В строке 8 функции инициализации константа EWMA gamma (g_C) представлена целочисленной степенью 2, поэтому в строке 9 целочисленного кода (Рисунок 11) требуется деление для взвешивания скользящего среднего значения, которое можно реализовать сдвигом вправо (>> g_C).

Приложение C. Выбор фактора связывания k

C.1. Зависимость от RTT

Когда классические потоки конкурируют за общую пропускную способность, относительные скорости этих потоков зависят не только от вероятности перегрузки, но и от сквозных значений RTT (базовый RTT + задержка в очереди). Скорости потоков Reno [RFC5681], конкурирующих через AQM, примерно обратно пропорциональны их RTT. В CUBIC зависимость от RTT в режиме совместимости с Reno такая же, но в других режимах механизм меньше зависит от RTT.

До первых экспериментов с DualQ Coupled AQM важность достаточно большой классической очереди для снижения зависимости от RTT при низком значении базового RTT не была должным образом оценена. В Приложении A.1.6 к спецификации протокола L4S ECN Protocol [RFC9331] приведены численные примеры, объясняющие, почему раздутые буферы скрывали зависимость классического контроля перегрузки от RTT. Там же разъяснено, почему большее сокращение задержки в очередях усиливает зависимость от RTT - проблема возможного истощения потоков с большим RTT при конкуренции с потоками, имеющими малое значение RTT.

С учётом добровольности контроля перегрузки в конечных системах, нет причин для их добровольной зависимости от RTT. Эту зависимость классического трафика от RTT невозможно «свернуть» (исключить), поэтому [RFC9331] требует от контроля перегрузки L4S значительного сокращения зависимости от RTT по сравнению со стандартным контролем перегрузки [RFC5681], по меньшей мере, при малом RTT. Зависимость от RTT должна быть не хуже, чем при использовании классических буферов подходящего размера. В соответствии с этим подходом сетевым устройствам не требуется решать проблему зависимости от RTT, хотя от этого не было бы никакого вреда, что по сути и делают очереди по потокам.

C.2. Рекомендации по контролю эквивалентности пропускной способности

Фактор связывания k определяет баланс между скоростями потоков L4S и Classic (см. параграф 2.5.2.1 и уравнение (1) в параграфе 2.1). Для общедоступной сети Internet рекомендуется значение k = 2 и обоснование этого дано ниже. Для других сред подходящее значение фактора связывания можно получить подстановкой в уравнение соответствующих значений.

Приведённые ниже математические выкладки ведут к уравнению (7), показывающему, что k = 1,64 теоретически выравнивает пропускную способность L4S и Classic, если сквозные значения RTT для них одинаковы. Однако даже при совпадении базовых RTT фактические значения RTT могут различаться потому, что классическому трафику нужна достаточно длинная очередь для предотвращения недогрузки канала, а для L4S это не нужно.

Поэтому для задания фактора связывания оператор должен решить, при каком базовом RTT он хочет выровнять пропускную способность L4S и классических потоков с учётом влияния дополнительной очереди на пропускную способность классического трафика. При таком подходе оператор может найти подходящий фактор связывания без знания точного алгоритма L4S для снижения зависимости от RTT и даже при отсутствии такого алгоритма.

В последующем рассмотрении применяются указанные ниже обозначения.

```
r - скорость пакетов [пакет/сек]
R - RTT [сек/обход]
p - вероятность маркировки ECN []
```

С классической стороны Reno представляется наиболее чувствительным и, следовательно, наихудшим классическим средством контроля перегрузки. CUBIC в режиме Reno-friendly (CReno) рассматривается как наиболее распространённый механизм контроля перегрузок, согласно ссылкам и анализу [PI2param]. В любом случае скорость классических пакетов в установившемся состоянии определяется известной формулой квадратного корня для Reno:

```
r_C = 1,22 / (R_C * p_C^0,5)          (5)
```

Со стороны L4S контроль перегрузки Prague [PRAGUE-CC] представляется эталоном для установившегося состояния зависимости от перегрузки. Prague соответствует тому же уравнению, что и DCTCP, но уравнение из описания DCTCP не используется здесь, поскольку оно подходит лишь для ступенчатой маркировки. Связанная маркировка p_CL подходит при рассмотрении эквивалентности пропускной способности с классическими потоками. В отличие от ступенчатой маркировки связанная по своей природе разнесена, поэтому используется формула для скорости пакетов DCTCP с вероятностной маркировкой, выведенную в Приложении A к [PI2]. Уравнение применяется без включения независимости от RTT (это объясняется позже).

```
r_L = 2 / (R_L * p_CL)                (6)
```

Для эквивалентности скорости потоков приравниваются скорости двух пакетов и уравнение приводится к тому же виду, что и уравнение (1) (скопировано из параграфа 2.1), чтобы их можно было приравнять и упростить для получения теоретического значения фактора связывания k*

```
    r_c = r_L
=>  p_C = (p_CL/1,64 * R_L/R_C)^2.
    p_C = ( p_CL / k )^2.             (1)
    k* = 1,64 * (R_C / R_L).          (7)
```

Фактор связывания назван теоретическим, поскольку он выражается через два RTT, что вызывает два практических вопроса - i) для нескольких потоков с разными RTT, где RTT для каждого класса трафика выводится из RTT всех потоков этого класса (фактически требуется среднее гармоническое значение), и ii) узел сети не может легко узнать RTT потоков.

Зависимость RTT от вызывается контролем перегрузки на основе окна, поэтому её нужно исключать там, а не в сети. По этой причине используется фиксированный фактор связывания в сети и зависимость от RTT сокращается у отправителей L4S. Не предполагается обновление всех классических отправителей для снижения зависимости от RTT.

Но решение проблемы лишь у отправителей L4S хотя бы не повышает зависимость от RTT (не только между отправителями L4S, но и между L4S и Classic).

Эквивалентность пропускной способности определяется для потоков при сравнимых условиях, включая одинаковые базовые значения RTT [RFC2914]. Если принять одинаковые базовые RTT (R_b) для сопоставимых потоков, можно выразить R_C и R_L через R_b. Можно аппроксимировать L4S RTT значением чуть больше базового RTT, т.,е R_L ~= R_b. R_C заменяется суммой (R_b + q_C), где значение q_C для классической очереди зависит от целевой задержки в очереди, которую оператор настроил для Classic AQM.

Приняв PI2 как пример Classic AQM, можно просто взять R_C = R_b + target (по умолчанию рекомендуется 15 мсек в Приложении A.1). Однако значение target приблизительно равно глубине очереди по вершинам зубьев пилы в контроле перегрузки, а не среднему значению [PI2param]. Таким образом, R_max = R_b + target. Положение среднего значения относительно максимума зависит от амплитуды и формы зубьев. Рассмотрим два примера: Reno [RFC5681], как наиболее чувствительный худший случай, и CUBIC [RFC8312] в режиме Reno-friendly (Creno), как наиболее распространённый алгоритм контроля перегрузок в Internet, согласно ссылкам из [PI2param]. Оба используют аддитивное увеличение и мультипликативное снижение (Additive Increase Multiplicative Decrease или AIMD), поэтому можно обобщить, используя b как фактор мультипликативного сокращения (b_r = 0,5 для Reno, b_c = 0,7 для CReno).

```
R_C  = (R_max + b*R_max) / 2
     = R_max * (1+b)/2.
R_reno = 0,75 * (R_b + target);    R_creno = 0,85 * (R_b + target)   (8)
```

Подстановка в уравнение (7) даёт при любом конкретном RTT (R_b) фиксированный фактор для каждого

```
   k_reno = 1,64*0,75*(R_b+target)/R_b
          = 1,23*(1 + target/R_b);    k_creno = 1,39 * (1 + target/R_b).
```

Оператор может выбрать базовое значение RTT, при котором он хочет получить эквивалентную пропускную способность. Например, при выборе R_b = 25 мсек как типичного базового RTT между пользователями Internet и CDN [PI2param] фактор связывания будет

```
   k_reno = 1,23 * (1 + 15/25)        k_creno  = 1,39 * (1 + 15/25)
          = 1,97                              = 2,22
          ~= 2,                               ~= 2,                  (9)
```

Такая аппроксимация применима для любого из рассмотренных выше алгоритмов Coupled DualQ, использующему фактор связывания со значением степени 2 для эффективной целочисленной реализации. Он хорошо подходит и для худшего случая (Reno).

Для проверки этого фактора связывания можно выразить отношение пропускных способностей L4S и Classic подстановкой их скоростей из уравнений (5) и (6), а также подстановкой p_C, выраженного через p_CL с использованием уравнения (1) с k = 2, как было указано для Internet

```
   r_L / r_C  = 2 (R_C * p_C^0,5) / 1,22 (R_L * p_CL)
              = (R_C * p_CL) / (1,22 * R_L * p_CL)
              = R_C / (1,22 * R_L).                              (10)
```

В качестве примера можно рассмотреть конкурирующие одиночные потоки CReno и Prague, выразив их RTT с помощью уравнения (10) через базовые RTT (R_bC и R_bL). Таким образом, R_C заменяется уравнением (8) для CReno, а R_L - функцией max(), приведённой ниже, которая представляет эффективное значение RTT для текущего контроля перегрузок [PRAGUE-CC] в принятом по умолчанию режиме независимости от RTT, поскольку это будет нижний порог эффективного RTT, применяемый для аддитивного увеличения.

```
   r_L / r_C ~= 0,85 * (R_bC + target) / (1,22 * max(R_bL, R_typ))
             ~= (R_bC + target) / (1,4 * max(R_bL, R_typ)).
```

Можно видеть, что для базовых RTT меньше target (15 мсек) числитель и знаменатель находятся на плато, что даёт желаемое ограничение зависимости от RTT.

В начале приведённых выше выкладок было обещано объяснение причины того, что приравнивание пропускной способности L4S в уравнении (6) не требуется для моделирования независимости от RTT. Это обусловлено использованием лишь одной точки с типичным значением базового RTT, выбранным оператором для расчёта фактора связывания. Эквивалентность пропускной способности будет наблюдаться, по меньшей мере, в этой точке. Тем не менее, если предположить, что отправитель Prague реализует независимость от RTT в диапазоне RTT ниже этого значения, эквивалентность пропускной способности будет распространяться и на этот диапазон.

Разработчики средств контроля перегрузок могут выбрать разные способы снижения зависимости от RTT. Каждый оператор может выбрать своё значение базового RTT и, следовательно, другое значение k, при котором он хочет получить эквивалентную пропускную способность. Тем не менее, для Internet имеет смысл выбрать значение, которое представляется типичным RTT для большинства пользователей, поскольку целевая задержка в очереди Classic AQM также выводится из типичного значения RTT для Internet.

В качестве примера, не связанного с Internet, для локализованного трафика из ЦОД конкретного ISP с использованием измеренных значений RTT было рассчитано, что значение k = 8 обеспечит эквивалентность пропускной способности и эксперименты подтвердили это с высокой точностью.

Для типовой смеси значений RTT в общедоступной сети Internet рекомендуется значение k = 2 как рабочий компромисс.

Благодарности

Спасибо Anil Agarwal, Sowmini Varadhan, Gabi Bracha, Nicolas Kuhn, Greg Skinner, Tom Henderson, David Pullen, Mirja Kühlewind, Gorry Fairhurst, Pete Heist, Ermin Sakic, Martin Duke за подробные комментарии в рецензиях (особенно к приложениям), а также за предложения по улучшению разъяснений. Спасибо Tom Henderson за сведения о выборе планировщиков и методов измерения задержки в очереди. Спасибо рецензентам направления (area) Christer Holmberg, Lars Eggert, Roman Danyliw.

Начальная работа Koen De Schepper, Bob Briscoe, Olga Bondarenko, Inton Tsang частично финансировалась European Community по программе Seventh Framework Programme в рамках проекта Reducing Internet Transport Latency (RITE)

(ICT-317700). Работа Koen De Schepper и Olivier Tilmans частично финансировалась по проектам 5Growth и DAEMON EU H2020. Работу Bob Briscoe частично финансировала Comcast Innovation Fund и Research Council of Norway в рамках проекта TimeIn. Выраженные здесь мнения принадлежат исключительно авторам.

Участники работы

Ниже перечислены те, кто внёс вклад в реализации и оценки, которые подтвердили и помогли улучшить эту спецификацию. Olga Albisser <olga@albisser.org> из Simula Research Lab, Norway (Olga Bondarenko в ранних черновиках) реализовала прототип DualPI2 AQM для Linux вместе с Koen De Schepper и провела обширные оценки, а также разработала GUI [L4Sdemo16] для визуализации производительности в реальном масштабе времени. Olivier Tilmans <olivier.tilmans@nokia-bell-labs.com> из Nokia Bell Labs, Belgium подготовил и поддерживает реализацию Linux DualPI2. Shravya K.S. написал модель для имитатора ns-3 на основе draft-ietf-tsvwg-aqm-dualq-coupled-01 (черновой вариант этого документа). На основе этой работы Tom Henderson <tomh@tomh.org> обновил модель и создал модель для варианта DualQ, указанного как часть спецификации Low Latency DOCSIS, а также выполнил различные оценки. Ing Jyh (Inton) Tsang изи Nokia, Belgium построил тестовый стенд для сквозной доставки между ЦОД и домом, где проверялись реализации DualQ Coupled AQM.

Адреса авторов

Koen De Schepper
Nokia Bell Labs
Antwerp
Belgium
Email: koen.de_schepper@nokia.com
URI: https://www.bell-labs.com/about/researcher-profiles/koende_schepper/

Bob Briscoe (editor)
Independent
United Kingdom
Email: ietf@bobbriscoe.net
URI: https://bobbriscoe.net/

Greg White
CableLabs
Louisville, CO
United States of America
Email: G.White@CableLabs.com

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru