ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра МСИБ

Киреева Н.В., Буранова М.А., Поздняк И.С.

Изучение алгоритма RED в среде NS-2

Методические указания к выполнению лабораторной работы

по специальностям: 210700, 090106

Самара, 2014

1. Цель работы

Изучить основные механизмы обеспечения качества обслуживания (QoS). Изучить принципы функционирования алгоритма активного управления очередями RED (и его производных). Получить навыки моделирования в среде NS-2.

2. Рекомендуемые источники

1. Галкин А.М., Кучерявый Е.А., Молчанов Д.А. NS-2
2. Кучерявый Е.А. Управление трафиком и качество обслуживания в сети Internet.
3. Кучерявый Е.А. NS-2 Как универсальное средство имитационного моделирования сетей связи.
4. Храпов С.В. Использование специализированной системы моделирования ns-2 для исследования телекоммуникационных сетей.
5. Яновский Г.Г. Качество обслуживания в сетях IP. Журнал «Вестник связи», №1, 2008
6. Kevin Fall, Kannan Varadhan The NS Manual.

1. Общие понятия

1.1 QoS

QoS (*Quality of Service* — качество обслуживания) — этим термином в области компьютерных сетей называют вероятность того, что сеть связи соответствует заданному соглашению о трафике, или же, в ряде случаев, неформальное обозначение вероятности прохождения пакета между двумя точками сети.

1.2 Параметры QoS

Для большинства случаев качество связи определяется четырьмя параметрами:

- Полоса пропускания (*Bandwidth*), описывает номинальную пропускную способность среды передачи информации, определяет ширину канала. Измеряется в bit/s (bps), kbit/s (Kbps), Mbit/s (Mbps), Gbit/s (Gbps).
- Задержка при передаче пакета (*Delay*), измеряется в миллисекундах.
- Колебания (дрожание) задержки при передаче пакетов — джиттер.
- Потеря пакетов (*Packet loss*). Определяет количество пакетов, потерянных в сети во время передачи.

Для простоты понимания канал связи можно представить в виде условной трубы, а пропускную способность описать как функцию двух параметров: диаметра трубы и её длины.

Когда передача данных сталкивается с проблемой «бутылочного горлышка» для приёма и отправки пакетов на маршрутизаторах, то обычно используется метод FIFO: первый пришел — первый ушёл (First In — First Out). При интенсивном трафике это создаёт заторы, которые разрешаются крайне простым образом: все пакеты, не вошедшие в буфер очереди FIFO (на вход или на выход), игнорируются маршрутизатором и, соответственно, теряются безвозвратно. Более разумный метод — использовать «умную» очередь, в которой приоритет у пакетов зависит от типа сервиса —ToS. Необходимое условие: пакеты должны уже нести метку типа сервиса для создания «умной» очереди. Обычные пользователи чаще всего сталкиваются с термином QoS в домашних маршрутизаторах с поддержкой QoS. Например, весьма логично дать высокий приоритет пакетам VoIP и низкий — пакетам FTP, SMTP и клиентам файлообменной сети.

1.3 Модели QoS

- Негарантированная доставка — Best Effort Service. Наличие марки ToS Best Effort Service не является механизмом тонкого регулирования и является признаком простого увеличения пропускной способности без какого-либо выделения отдельных классов трафика и регулирования.
- Интегрированный сервис — Integrated Service (IntServ). Согласно RFC 1633, модель интегрированного обслуживания обеспечивает сквозное (End-to-End) качество обслуживания, гарантируя необходимую пропускную способность. IntServ использует для своих целей протокол резервирования сетевых ресурсов RSVP, который обеспечивает выполнение требований ко всем промежуточным узлам. В отношении IntServ часто используется термин «резервирование ресурсов» (Resource reservation).
- Дифференцированное обслуживание — Differentiated Service (DiffServ). Описана в RFC 2474 и RFC 2475. Обеспечивает QoS на основе распределения ресурсов в ядре сети и определенных классификаторов и ограничений на границе сети, комбинируемых с целью предоставления требуемых услуг. В этой модели вводится разделение трафика по классам, для каждого из которых определяется свой уровень QoS. DiffServ состоит из управления формированием трафика (классификация пакетов, маркировка, управление интенсивностью) и управления политикой (распределение ресурсов, политика отбрасывания пакетов). DiffServ является наиболее подходящим примером «умного» управления приоритетом трафика.

1.4 Приложения, требующие QoS

Определённое качество обслуживания может потребоваться для ряда сетевых приложений, в частности:

- потоковые мультимедиа-приложения требуют гарантированную пропускную способность канала;
- VoIP и видеоконференция требуют небольших значений джиттера и задержки;
- ряд приложений, например, удалённая хирургия, требуют гарантированного уровня надёжности.

2. Алгоритм RED

Алгоритм управления очередями «Вероятностное заблаговременное обнаружение перегрузки» (Random Early Detection) заложил целое направление работ, посвященных управлению перегрузками в сетях данных посредством модификации и/или разработки новых алгоритмов управления очередями. RED позволяет контролировать нагрузку в рамках очереди маршрутизатора и при обнаружении перегрузки или состоянии близкому к перегрузке осуществлять вероятностный сброс пакетов. Сброс пакета осуществляется на базе вычисления вероятности, поэтому:

1) Можно соблюдать принцип «справедливого распределения ресурсов», и, соответственно, избегать возникновения проблемы «Lock-Out»;
2) Осуществлять последовательный сброс пакетов, принадлежащих различным соединениям, т.е. избегать «глобальной синхронизации».

Пример функционирования алгоритма RED представлен на рисунках 2.1 и 2.2. Алгоритм RED ориентирован на работу с протоколом TCP, поэтому сброс пакета позволит источнику нагрузки уменьшить размер окна и, таким образом, понизить нагрузку.

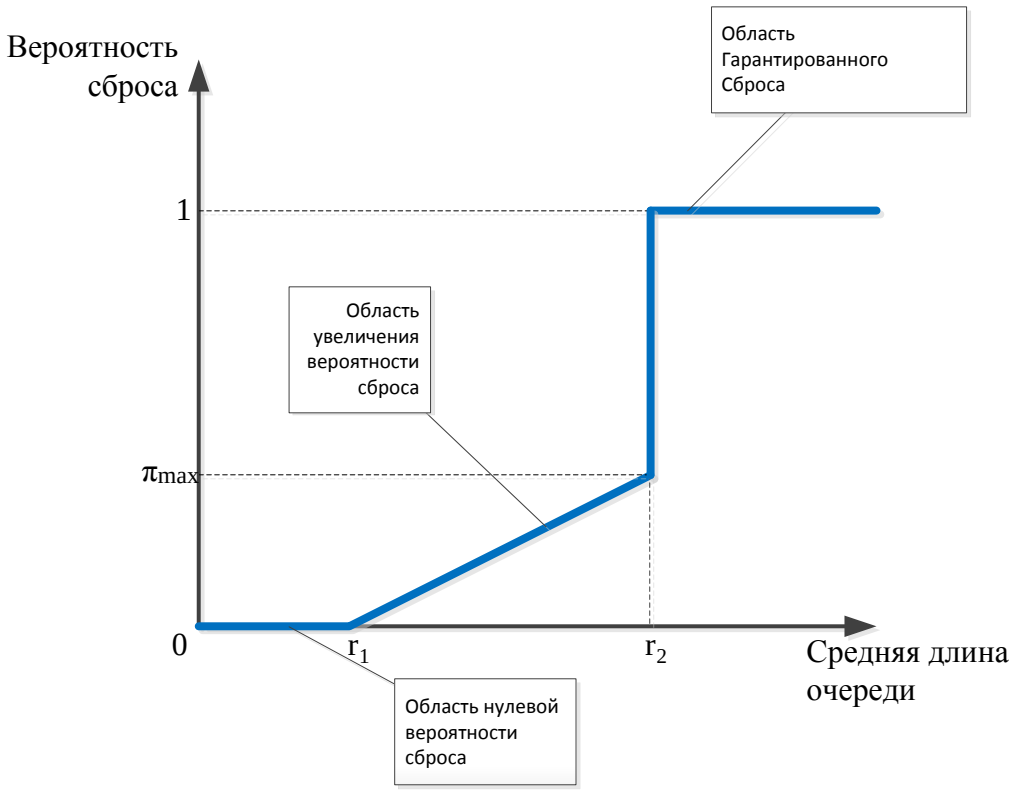

Рисунок 2.1 Принцип действия алгоритма RED

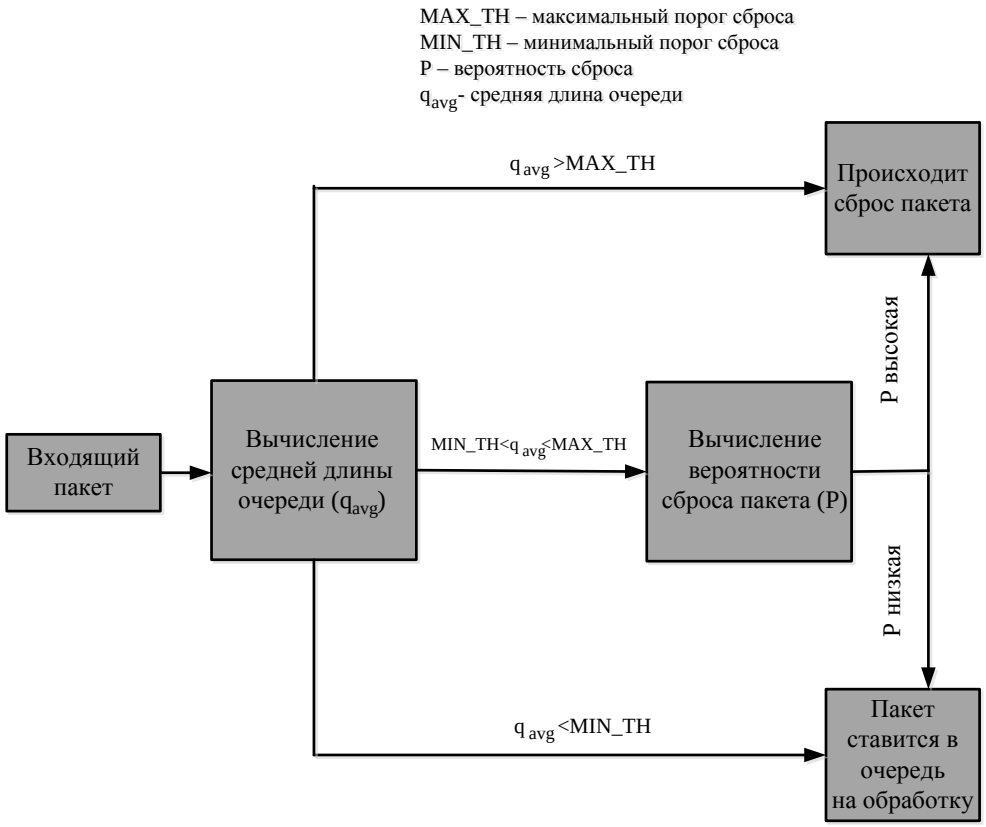

Рис. 2.2 Принцип действия алгоритма RED

Существует несколько разновидностей алгоритма RED (Рисунок 2.3):

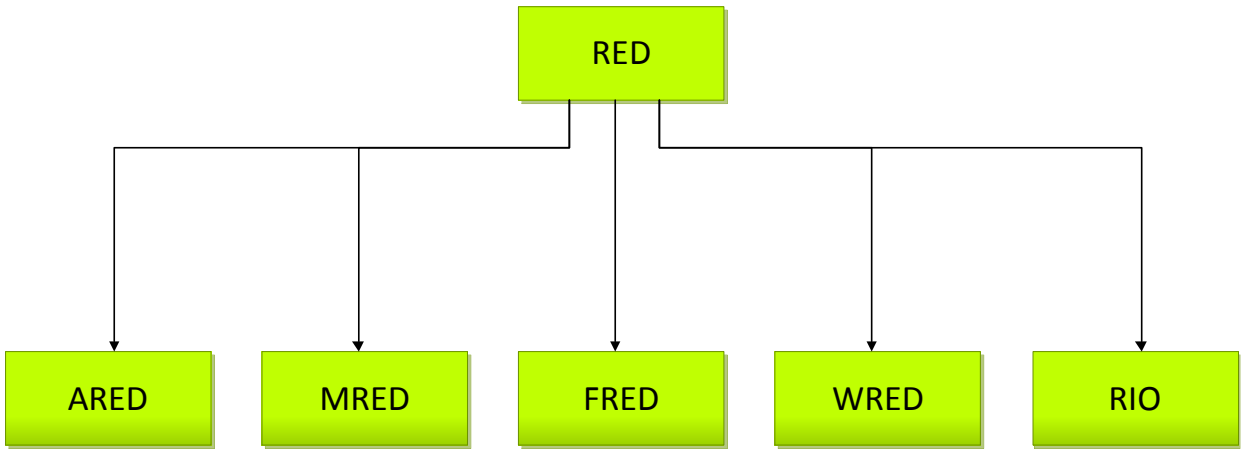

Рисунок 2.3 Разновидности алгоритма RED

Рассмотрим общий принцип функционирования базового алгоритма RED. Алгоритм состоит из двух частей: оценка среднего размера очереди и принятие решения о сбросе вновь поступившего пакета. При поступлении каждого нового пакета, используя фильтр высоких частот совместно с «экспоненциально взвешенной скользящей средней» EWMA (low-pass filter with an exponentially weighed moving average), RED определяет средний

размер очереди avg. Далее значение avg сравнивается со значениями ранее определенных границ: верхней max_th и нижней min_th, которые определяют степень нагрузки в очереди.

В случае если значение avg принадлежит интервалу [0; min th], то поступающий пакет помещается в очередь, если же значение avg принадлежит интервалу [min_th; max_th], то вычисляется вероятность сброса пакета P_a, и либо с вероятностью P_0 осуществляется сброс поступающего пакета, либо с вероятностью (1 — P_0) поступающий пакет помещается в очередь. Если же значение среднего размера очереди avg превышает значение max_th, то поступающий пакет обязательно сбрасывается. Заметим, что вероятность сброса пакета является функцией от переменной avg, т.е. с повышением нагрузки повышается вероятность сброса поступающего пакета.

Следует отметить, что RED также имеет возможность осуществлять не сброс пакетов, а маркировку. Маркировка может быть произведена посредством изменения какого-либо бита в каком-либо опциональном поле в заголовке пакета с целью извещения протокола транспортного уровня, например, TCP, о надвигающейся перегрузке. Тогда в этом случае, если в приемнике реализован протокол, понимающий маркирование пакета, то протокол приемника должен понизить размер окна и, тем самым, сообщить источнику о необходимости понижения нагрузки. В результате этого источник должен понизить размер окна и, тем самым, понизить количество пакетов, передаваемых в сеть за некоторый промежуток времени. Реализация маркировки пакета в RED более эффективна, чем реализация сброса в случае функционирования на транспортном уровне протокола, подобного TCP, а в случае если осуществляется управление потоками UDP, то RED должен реализовать функцию сброса.

Таким образом, RED состоит из двух различных алгоритмов: с помощью алгоритма вычисления среднего размера очереди фактически вычисляется степень пачечности нагрузки, которая может быть принята буфером; алгоритм подсчета вероятности маркировки или сброса пакета на основе известного значения нагрузки (т.е. значения средней длины очереди) определяет как часто маршрутизатор маркирует или сбрасывает пакеты. Целью функционирования RED в маршрутизаторе является такая реализация маркировки/сброса пакетов, при которой будет соблюдаться принцип «справедливого распределения ресурсов» и избегание «глобальной синхронизации».

2.1 Принцип функционирования расширенного алгоритма RED

Принимая во внимание, что представленный выше алгоритм RED являлся базовым, т.е. дающим общее понимание логики функционирования, рассмотрим далее расширенную версию алгоритма RED. Именно расширенная версия RED применяется для реализации на реальных маршрутизаторах. Прежде чем перейти к рассмотрению принципа

функционирования расширенного RED, необходимо определить используемые переменные.

Глобальные переменные:

avg: средний размер очереди;

q_time: момент времени, когда очередь стала пустой;

count: количество пакетов, пришедших в очередь с момента последнего сброса.

Фиксированные параметры алгоритма:

w: весовой коэффициент очереди;

min_th: нижняя граница;

max_th: верхняя граница;

тах_р: максимальное значение вероятности P_a.

Остальные переменные:

P_a: текущая вероятность маркировки/сброса пакета;

q: текущий размер очереди;

time: текущее время;

£ (time) : линейная функция времени.

Расширенный алгоритм RED функционирует следующим образом. При активации алгоритма происходит инициализация переменных count и avg. Далее с приходом каждого нового пакета в систему осуществляется ряд действий.

Вначале вычисляется значение среднего размера очереди avg:

$$Avg = (1 - w)avg + w * c \quad (2.1)$$

Если очередь пуста, то оценивается количество пакетов малой длины, которые могли бы быть переданы в период отсутствия пакетов в очереди:

$$Avg = avg * (1 - m) \quad (2.2)$$

Далее для вычисления среднего размера очереди avg предполагается, что за время отсутствия пакетов в очереди поступило m пакетов. Если не сделать такое предположение, то значение avg будет вычислено неправильно. После подсчета среднего размера очереди avg необходимо оценить его значение. Если avg принадлежит интервалу [min_th; max_th], то вычисляется вероятность маркировки/сброса пакета Pb, которая линейно изменяется в интервале от 0 до max_p:

$$Pb = \max\_p(avg - \min\_th)/(\max\_th * \min\_th) \quad (2.3)$$

Вероятность, на основе которой осуществляется маркировка/сброс поступающего пакета, вычисляется с использованием счетчика количества пакетов, пришедших в очередь с момента последнего сброса count — чем больше пакетов пришло, тем выше вероятность сброса. Такой подход гарантирует, что RED не будет ожидать слишком много времени, прежде чем осуществит маркировку/сброс пакета, т.е. маркировка/сброс осуществляется пропорционально нагрузке:

$$Pa = Pb/(1 - count * Pb) \quad (2.4)$$

Если значение avg превышает значение max_th, то поступающий пакет маркируется или сбрасывается в зависимости от реализации.

Дополнительной возможной опцией алгоритма RED является измерение среднего размера очереди не в пакетах, а в байтах. В этом случае значение среднего размера очереди точно отражает задержку, вносимую буфером в передачу пакета. При использовании этой опции алгоритм должен быть модифицирован для того, чтобы вероятность, с которой пакет маркируется/сбрасывается, была пропорциональна его длине:

$$Pb = max\_p(avg - min\_th)/(max\_th - min\_th) \quad (2.5)$$

$$Pb = Pb * packetsize/Maximumpacketsize \quad (2.6)$$

$$Pa = Pb/(1 - count * Pb) \quad (2.7)$$

Таким образом, пакеты большого размера, например, пакеты FTP, будут маркироваться или сбрасываться с большей вероятностью, нежели пакеты малой длины, например, пакеты Telnet.

2.2 Настройка параметров алгоритма RED

Для эффективного использования активного управления очередями, реализованных в маршрутизаторах, необходимо осуществлять корректную настройку параметров алгоритмов. Неправильная настройка параметров алгоритма может привести к значительному ухудшению параметров функционирования машрутизатора. Настройки требуют следующие параметры:

- Размер буфера
- Весовой коэффициент W
- Нижняя граница min_th
- Верхняя граница max_th
- Максимальное значение вероятности сброса max_p

Значение размера буфера оказывает влияние на задержку пакета в маршрутизаторе, поэтому его размер должен быть сконфигурирован в соответствии со значением максимальной задержки, которую может получить пакет при нахождении в маршрутизаторе.

Весовой коэффициент w определяется, исходя из гипотетических максимальных значений размера и продолжительности пачки пакетов, которая может быть без потерь принята очередью на обслуживание. Если значение w велико, то существует опасность, что алгоритм вычисления среднего размера очереди avg не сможет отфильтровать кратковременную перегрузку. С другой стороны, если значение w слишком мало, то его влияние на динамику среднего значения очереди avg также мало, поэтому значение avg не сможет достаточно быстро реагировать на изменение текущего размера очереди q, вследствие чего RED не будет иметь возможности определять состояние, близкое к перегрузке.

Для определения значения нижней и верхней границ max_th и min_th необходимо учитывать, что они зависят от желаемого максимального среднего размера очереди. Если предполагается, что нагрузка будет иметь достаточно высокую пачечность, то значение min_th должно быть достаточно большим для того, чтобы степень использования канала сохранялась на

приемлемо высоком уровне. Определение значения max_th напрямую зависит от значения средней задержки пакета в очереди: чем выше значение max_th, тем выше значение средней задержки. Также существует рекомендованное общее соотношение значений границ: значение max_th должно быть, как минимум, в два раза больше значения min_th.

Отметим, что настройка параметров алгоритма RED является достаточно сложной задачей. Значения параметров зависят, в первую очередь, от степени пачечности и характера нагрузки, которая будет поступать в буфер, в котором реализован RED, и от конфигурации сети. Существуют работы, в которых доказывается, что в связи с тем, что параметры сети носят динамический характер, то и параметры RED должны меняться динамически в зависимости от состояния сети.

2.3 Реализация алгоритма RED

Существуют различные области реализации алгоритма RED. В первую очередь, конечно, необходимо отметить реальное оборудование. К настоящему времени реализация RED является практически стандартной функцией маршрутизаторов TCP/IP, например Cisco. В связи с этим изучение влияния RED на параметры функционирования сети в настоящее время можно проводить и на базе реального оборудования. RED также реализован в операционных системах Linux (начиная с версии ядра 2.2.x), FreeBSD версии 4.5 и Solaris 2.5, на базе которых можно построить маршрутизатор. И не стоит забывать о прародителе всех реализаций — симуляторе ns2 [ns2], в котором RED был реализован в первую очередь. Именно при помощи имитационного моделирования исследователи смогли предсказать особенности поведения RED на реальных сетях.

2.4 Алгоритм ARED

Как было показано ранее, добиться высокой эффективности при использовании RED в маршрутизаторе можно, только корректно настроив его параметры. Настройка параметров является сложной оптимизационной задачей, решение которой зависит от множества факторов, в первую очередь, таких как степень пачечности и характер нагрузки, конфигурация сети и т.д.

Рассмотрим каким образом изменение значения весового коэффициента w влияет на эффективность функционирования RED. Очевидно, что скорость, с которой будет нарастать нагрузка в очереди, зависит от количества соединений TCP, передающих данные через рассматриваемый маршрутизатор. В случае если количество соединений TCP невелико, скорость роста нагрузки относительно низка, поэтому значение параметра w должно быть сравнительно малым.

Однако, использование того же самого низкого значения w для случая с большим количеством соединений TCP может привести к неэффективному функционированию RED по причине того, что существует опасность, что алгоритм не сможет отреагировать на кратковременную перегрузку. С другой

стороны, установка слишком высокого значения w может привести к повышенной вероятности сброса пакетов даже при прохождении через маршрутизатор нескольких соединений TCP. Интуитивно можно предположить, что динамическое изменение параметров RED может позволить адаптировать этот алгоритм для динамической нагрузки.

Вскоре был предложен адаптивный алгоритм RED (Adaptive RED, далее — ARED), базовым принципом которого является динамическое изменение параметров. Вычисление новых значений параметров осуществляется на основе данных о нагрузке за определенный промежуток времени. ARED динамически измеряет значение параметра max p, основываясь на недавних значениях средней длины очереди avg.

Рассмотрим алгоритм функционирования ARED. Если avg принимает значение меньшее, чем значение нижней границы min_th, то параметр max p принимает достаточно малое значение — предполагается, что интенсивность поступающей нагрузки мала. Очевидно также, что при инициализации ARED параметр max_p принимает достаточно малое значение в связи с тем, что очередь пуста, т.е. ее средний размер меньше нижней границы. Как только средний размер начинает превышать max th, вычисляется новое значение параметра max_p, которое существенно выше предыдущего.

Таким образом, ARED адаптируется к повысившейся нагрузке — вероятность сброса поступающего пакета остается меньше единицы, в то время как если бы в рассматриваемом маршрутизаторе был применен RED, то вероятность сброса поступающего пакета была бы равна единице. Достаточно важным моментом в работе ARED является его поведение в областях, близких к верхней и нижней границам. В случае если среднее значение размера очереди осциллирует около нижней границы min_th, то ARED постепенно снижает значение параметра max_ p, т.к. нагрузка невысока и вероятность сброса поступающего пакета можно понизить. В случае если среднее значение размера очереди осциллирует около верхней границы max_th, то ARED постепенно увеличивает значение параметра max_p, т.к., возможно, для предотвращения перегрузки необходимо повысить вероятность сброса пакета. Принцип действия алгоритма ARED изображен на рисунке 2.4.

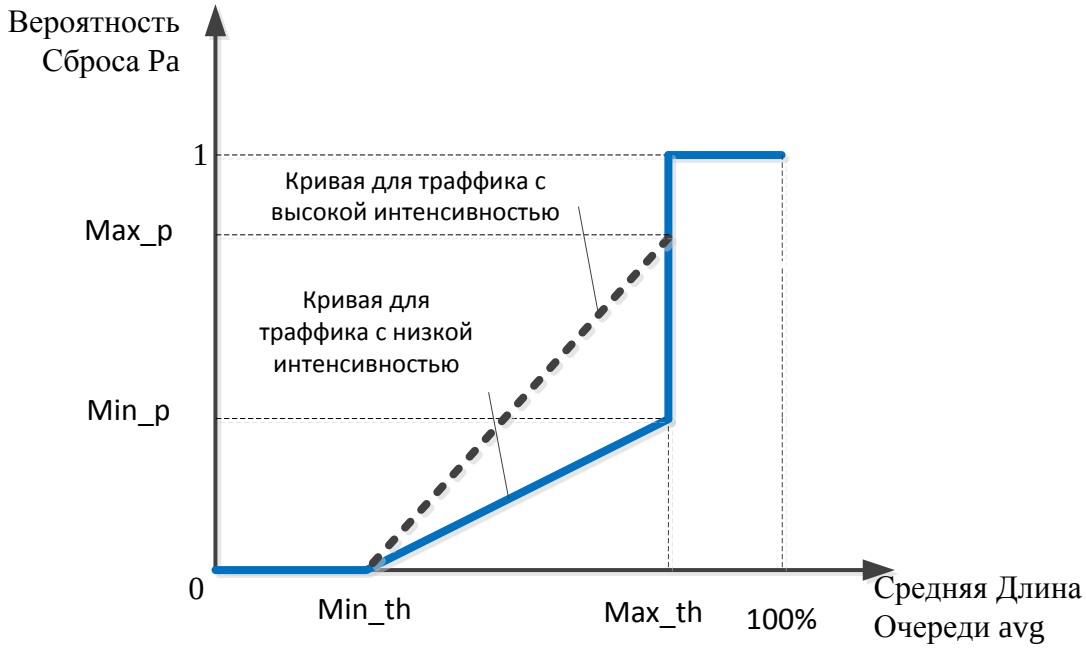

Рисунок 2.4 Принцип действия алгоритма ARED

2.5 Алгоритмы класса MRED

Для маршрутизаторов, поддерживающих архитектуру DiffServ и AF PHB, было предложено модифицировать алгоритм RED для того, чтобы иметь возможность более гибко управлять поступающей нагрузкой. Известно, что одним из основных свойств архитектуры DiffServ является возможность назначения пакетам различных приоритетов. Пакеты классифицируются на базе информации третьего уровня (IP) моделей OSI и TCP/IP по значению поля «тип услуги» (Type of Service, далее — TOS), находящегося в заголовке. В терминах архитектуры DiffServ данное поле носит имя DS. Назначение уровня приоритета осуществляется при поступлении пакета в домен DiffServ пограничным входящим узлом (Ingress node), в результате чего каждый пакет, находящийся в домене DiffServ, имеет свой приоритет и при поступлении в любой маршрутизатор в рамках этого домена должен быть обработан в соответствии со значением поля DS. Текущие спецификации AF PHB определяют четыре класса с тремя уровнями приоритета пакета для каждого. Пакеты, принадлежащие одному классу, перенаправляются независимо от пакетов, принадлежащих другому классу, при этом в маршрутизаторе, находящемся в домене DiffServ, для каждого реализованного класса PHB AF должны быть распределены ресурсы, однако не требуется, чтобы все классы PHB AF были реализованы. Таким образом очевидно, что в DiffServ-маршрутизаторах необходимо реализовывать алгоритмы активного управления очередями, которые способны различать поступающие пакеты по их приоритету и, соответственно, приоритету эти пакеты обрабатывать. Базовый алгоритм RED, описанный выше, не способен обеспечить обработку пакетов в соответствии с их приоритетами, поэтому его необходимо модифицировать путем введения дополнительных наборов

параметров для каждого приоритета. Вскоре была предложена классификация алгоритмов, базирующихся на RED и реализующих поддержку приоритетов пакетов, и было дано общее название для подобных алгоритмов — «многоуровневый RED» (Multilevel red, далее — MRED).

Алгоритмы класса MRED могут быть разбиты на четыре категории – классификация с примерами изображена на рисунке 2.5. Категория SAST (Single Average Single Threshold, «одно среднее значение, один набор значений границ») представлена базовым алгоритмом RED, категория SAMT (Single Average Multiple Threshold, «одно среднее значение, несколько наборов значений границ») — алгоритмом WRED (Weighed RED, взвешенный RED), категория MAMT (Multiple Average Multiple Threshold, «несколько средних значений, несколько наборов значений границ») — алгоритмом RIO (RED In & Out).

Далее в этой главе рассмотрим модификации MRED, позволяющие осуществлять управление очередью, принимая во внимание приоритеты поступающих пакетов.

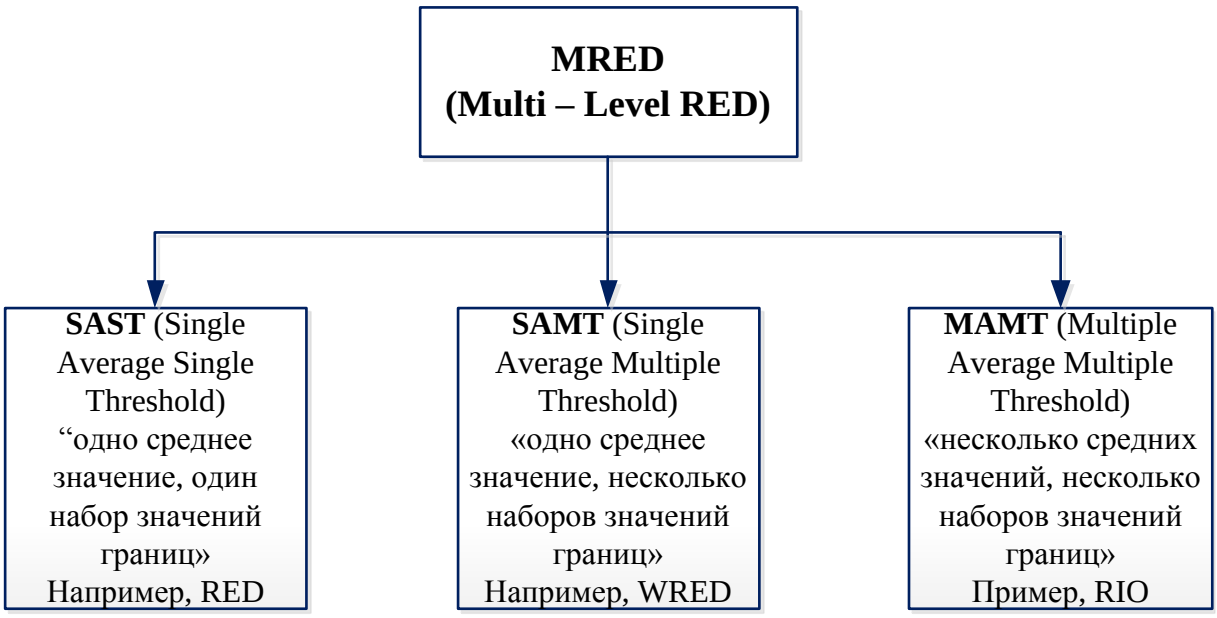

Рис 2.5 Классификация MRED

2.6 Алгоритм WRED

Отличие WRED от RED состоит только лишь в том, что для пакетов каждого приоритета определен отдельный набор параметров. Например, рассмотрим простейший случай, когда в очередь маршрутизатора поступают пакеты с двумя приоритетами — далее под пакетом с низким приоритетом будем понимать маркированный пакет, а под пакетом с высоким приоритетом — обычный немаркированный пакет. Функция зависимости вероятности сброса пакета от значения среднего размера очереди ведет себя следующим образом: при одинаковой нагрузке должен соблюдаться принцип, что вероятность сброса маркированного пакета должна превышать вероятность сброса обычного пакета.

Параметризация алгоритма WRED достаточно сложна, т.к. количество значений, которые необходимо установить, прямо пропорционально количеству приоритетов, с которыми пакеты могут поступать в рассматриваемый маршрутизатор. Например, если учесть, что при реализации AF PHB количество приоритетов может возрасти до 12, то для полной параметризации WRED необходимо установить значения $12 * \{minX\_th, maxX\_th, maxX\_p, avgX\}$ параметров, где X — номер приоритета. Частный случай функции вероятности сброса пакета для двух приоритетов алгоритма WRED представлен на рисунке 2.6:

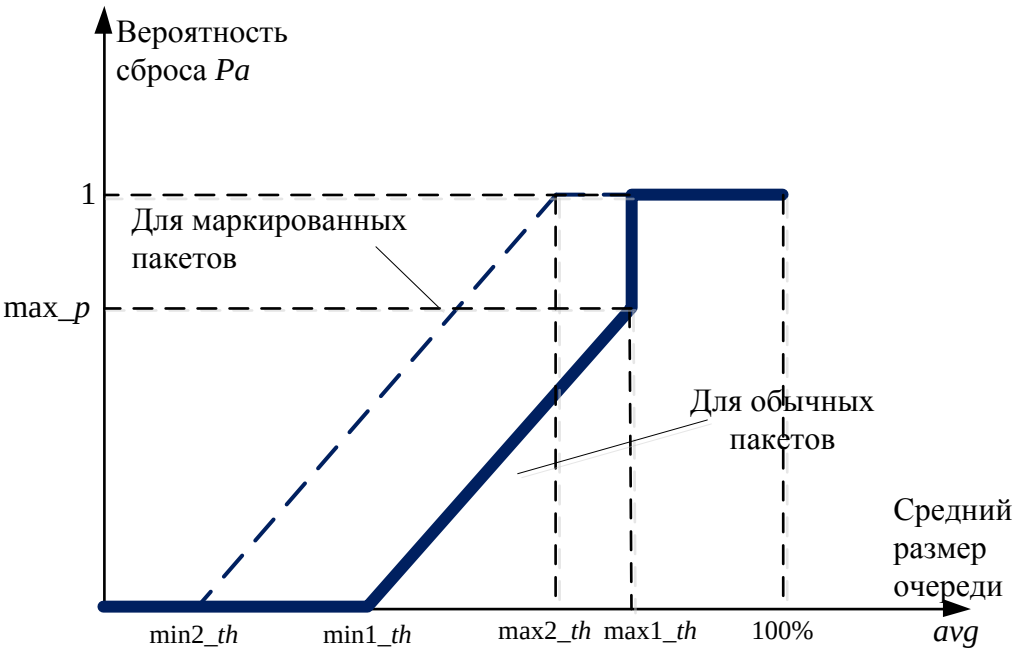

Рисунок 2.6 Алгоритм WRED

2.7 Алгоритм RIO

Алгоритм управления очередью RIO (RED In & Out) полностью аналогичен алгоритму RED за исключением того, что он параметризируется двумя наборами значений параметров — каждый набор для отдельного класса пакетов. Предполагается, что существует два класса пакетов — немаркированные (in-profile), т.е. когда пакет принадлежит потоку, значения параметров которого не превышают заранее определенных величин, и маркированные (out- profile), т.е. когда пакет принадлежит потоку, значения параметров которого превышают заранее определенные величины. RIO различает оба класса пакетов посредством анализа заголовка и поэтому имеет возможность осуществлять сброс маркированных пакетов с большей вероятностью, нежели немаркированных, таким образом обеспечивая защиту от сброса немаркированным пакетам.

В связи с тем, что алгоритм RIO является модификацией алгоритма RED, очевидно, что принципы функционирования этих алгоритмов одинаковы. Фактически RIO является усложненной версией RED, т.к. су-

ществует необходимость контролировать два набора параметров: один — для немаркированных пакетов, другой — для маркированных.

Рассмотрим принцип функционирования алгоритма RIO. Очевидно, что при поступлении пакета в очередь, необходимо определить к какому классу относится данный пакет. Если пакет немаркированный (далее для простоты будем называть такой пакет In-пакет), то осуществляется вычисление среднего размера очереди avg_in, которая состоит только из In-пакетов. Если пакет маркированный (далее для простоты будем называть такой пакет Out-пакет), то осуществляется вычисление среднего размера всей очереди avg_total. Здесь стоит отметить важную деталь, что вероятность сброса In-пакета зависит от среднего размера очереди, состоящей только из In-пакетов, в то время как вероятность сброса Out-пакета зависит от среднего размера очереди, состоящей из пакетов обоих классов, In-пакетов и Out-пакетов.

Заметим, что все пакеты независимо от их класса поступают в одну физическую очередь типа FIFO. Очередь, состоящая только из In-пакетов, физически не существует, ее размер подсчитывается специальной процедурой. На рисунке представлены функции вероятности сброса пакета в зависимости от среднего размера очереди для In-пакетов и Out- пакетов. Как видно и как можно предположить, исходя из принципов функционирования алгоритма RED, для каждого класса пакетов необходимо определить следующие параметры функционирования:

- Min_in: нижняя граница для In-пакетов;
- Max_in: верхняя граница для In-пакетов;
- Min_out: нижняя граница для Out-пакетов;
- Max_out: верхняя граница для Out-пакетов;
- Max_in_p: максимальное значение вероятности сброса для In-пакетов;
- Max_out_p: максимальное значение вероятности сброса для Out-пакетов;

Таким образом, суммируя вышесказанное, определяем наборы параметров, на базе которых вычисляется вероятность сброса поступающего пакета: для In-пакетов {min_in, max_in, max_in_p, avg_in} и для Out-пакетов {min out, max__out, max_out_p, avg_total}. Посредством определения этих параметров определяется три области функционирования системы относительно In-пакетов: «нормальное функционирование» $avg\_in \in [0; \min\_in]$, «предотвращения перегрузки» $avg\_in \in [\min\_in; \max\_in]$ и «управления перегрузкой» $avg\_in \in [\max\_in; \infty]$, а также три области функционирования системы относительно Out-пакетов: «нормальное функционирование» $avg\_total \in [0; \min\_out]$, «предотвращение перегрузки» $avg\_total \in [\min\_out; \max\_out]$ и «управления перегрузкой» $avg\_total \in [max\_out; \infty]$. Алгоритм RIO представлен на рисунке 2.7.

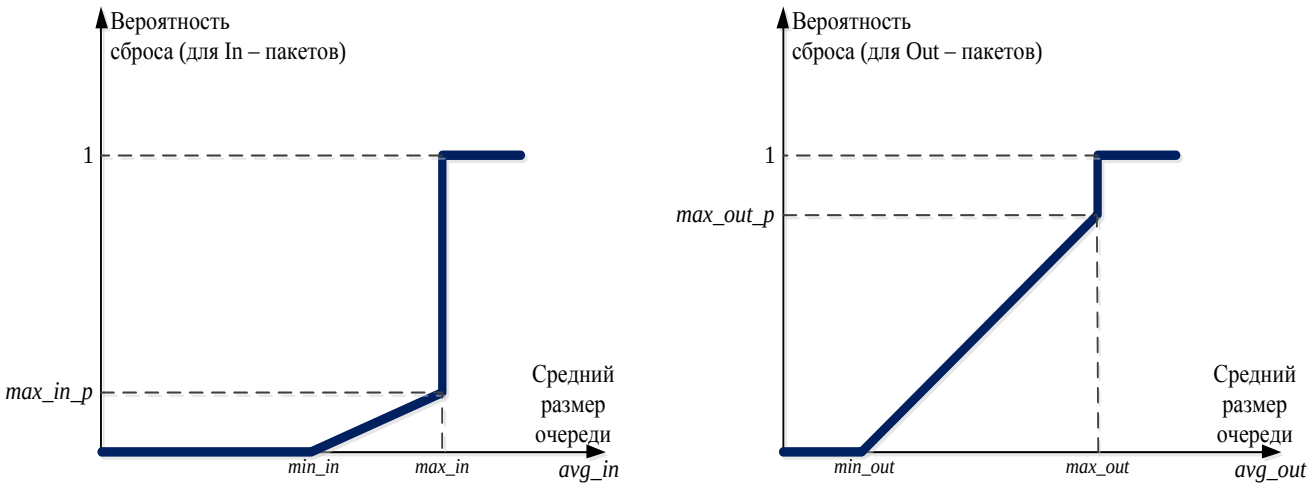

Рис 2.7 Алгоритм RIO

2.8 Алгоритм FRED

Одним из достоинств алгоритма RED считается принцип «справедливого распределения ресурсов». Однако более глубокие исследования этого вопроса показали, что при определенных условиях нельзя говорить, что RED позволяет всем соединениям честно конкурировать в борьбе за сетевые ресурсы. RED ориентирован на работу с соединениями TCP, в то время как нагрузка, создаваемая протоколом UDP, также поступает в туже самую очередь и, по причине отсутствия механизма управления нагрузкой, этот тип нагрузки бесконтрольно (со стороны источника) заполняет буферные пространства и создает перегрузку. Для исследования влияния нагрузки UDP на соблюдение принципа «справедливого распределения ресурсов» были введены определения для базовых типов нагрузки в сети интернет:

- «Неадаптивная» нагрузка (Non-adaptive): нагрузка этого типа забирает столько сетевых ресурсов, сколько для него требуется (или если количество ресурсов меньше, чем ей требуется – то все оставшиеся), при наступлении перегрузки источник не снижает скорость передачи и, соответственно, игнорирует сброс пакетов. В качестве примера можно привести некоторые аудио и видеоприложения;
- «Устойчивая» нагрузка (robust): соединения TCP с малым значеним RTT. Эти соединения достаточно быстро функционируют и скорость реакции на возникающие перегрузки высока. Количество пакетов, находящихся в буфере, достаточно велико. Примером может служить трафик HTTP;
- «Хрупкая» или «неустойчивая» нагрузка (fragile): Соединения TCP с большим значением RTT и/или малой скоростью. Время реакции этих соединений на перегрузку велико и количество пакетов, находящихся в буфере, сравнительно мало. Трафик, генерируемый интерактивными приложениями, например, Telnet, можно отнести к этому типу нагрузки.

Таким образом, в реальной сети очередь любого маршрутизатора в некоторый момент времени содержит определенное число пакетов,

относящихся ко всем трем типам нагрузок – неадаптивной, устойчивой и хрупкой. При наступлении перегрузки интенсивность неадаптивной нагрузки не уменьшится, в то время как интенсивность устойчивой и хрупкой нагрузок понизятся.

Суммарная нагрузка упадет, а неадаптивный трафик, пока нагрузка от TCP соединений мала, начнет занимать ресурсы и вероятность сброса для этого типа нагрузки временно существенно понизится. Однако, увеличение размеров окон cwnd соединений TCP, относящихся к устойчивой и хрупкой нагрузке, приведет к новой перегрузке. Причем сложится ситуация, когда неадаптивная нагрузка будет занимать достаточно большое количество ресурсов, устойчивая нагрузка – меньше, чем неадаптивная, а хрупкая – совсем мало по причине достаточно медленного обновления размеров окон cwnd из-за высокого значения RTT.

Вероятность сброса для поступающего пакета высока, так как значение среднего размера очереди также высоко, т.е. пакеты соединений TCP будут сбрасываться даже если нагрузка, создаваемая ими, низка. Отметим, что в этом случае хрупкая нагрузка подвержена большим потерям, нежели устойчивая, в связи с тем, что время реакции на перегрузку высоко. В данном примере принцип «справедливого распределения ресурсов» не выполняется.

Для решения проблемы несоблюдения справедливого распределения ресурсов при функционировании RED была разработана его модификация FRED (Flow RED). Алгоритм FRED позволяет эффективно изолировать неадаптивную нагрузку от TCP-соединений, а также обеспечить защиту пакетов низкоскоростных соединений TCP от несправедливого сброса. Выполнять столь сложные функции алгоритму FRED позволяет возможность управления состоянием для каждого потока, пакеты которого находятся в очереди маршрутизатора. Здесь и далее под понятием «состояние» будем понимать набор параметром, характеризующих конкретный отдельный поток.

Рассмотрим алгоритм функционирования FRED. Общие принципы функционирования эквиваленты принципам RED. В FRED дополнительно введены следующие параметры:

- Min_q: минимальное количество пакетов, которое поток имеет возможность поместить в буфер без потерь; задается для каждого потока;
- Max_q: максимальное количество пакетов, которое поток имеет возможность поместить в буфер без потерь; задается для каждого потока;
- Avgcq: среднее значение размера очереди, состоящей только из пакетов определенного потока; задается для каждого потока;
- Q_len: количество пакетов в буфере, относящихся к некоторому потоку; задается для каждого активного потока;
- Strike: аддитивная переменная, определяющая, как часто поток не реагировал снижением скорости на возникшую перегрузку, задается

для каждого потока. Потоки с высоким значением этой переменной пенализируются.

2.8.1 Защита хрупких соединений

FRED позволяет каждому потоку поместить в буфер Min_q пакетов без потерь. Все остальные пакеты потока, общее количество пакетов которого превышает значение Min_q, будут проходить операцию вероятностного сброса, т.е. поступающий пакет может быть как помещен в очередь, так и сброшен. Фактически граница min_q используется алгоритмом для неявного резервирования буферного пространства для хрупких соединений. Поступающий пакет всегда будет принят в очередь, если средний размер очереди для рассматриваемого потока ниже, чем определено значением min_q, несмотря на нагрузку, создаваемую в маршрутизаторе неадаптивными и устойчивыми потоками.

2.8.2 Управление устойчивыми потоками

Когда количество активных пакетов мало, т.е. N<<min_th/min_q, FRED позволяет некоторым потокам поместить в буфер количество пакетов, существенно превышающее min_q. Как только средний размер очереди превысит значение min_th, каждый поступающий пакет будет проходить операцию вероятностного сброса. Можно заметить, что FRED, так же как и RED, в таком случае может нарушить принцип «справедливого распределения ресурсов»: к каждому поступающему пакету будет применятся одинаковый алгоритм вероятностного сброса, невзирая на то, сколько ресурсов занимает поток, к которому относится этот пакет.

В алгоритме FRED предусмотрена процедура исправления подобного недостатка. Когда количество активных потоков мало, FRED динамически увеличивает все значения Min_q до значения среднего размера очереди соответствующего потока avgcq. Для простоты значения avgcq вычисляются путем деления значения среднего размера всей очереди в маршрутизаторе avg на количество активных потоков.

2.8.3 Управление неадаптивными потоками

Как было показано ранее, неконтролируемая неадаптивная нагрузка может привести к таким достаточно тяжелым последствиям для маршрутизатора и сети в целом, как захват ресурсов и блокировка соединений TCP. С целью управления нагрузкой неадаптивных потоков в FRED реализована функция подсчета того, сколько раз определенный поток не снижал скорости передачи при поступлении перегрузки или, другими словами, насколько часто пытался поместить пакет в очередь при превышении значения max_q.

Для этого в FRED реализована аддитивная переменная strike, значение которой увеличивается на единицу каждый раз, когда неадаптивный поток пытается поместить поступающий пакет в очередь или когда размер очереди

пакетов этого потока уже достиг значения max_q. Потоки с высоким значением параметра strike не могут помещать в очередь пакетов больше, чем значение параметра avgcq, т.е. неадаптивный поток не может использовать больше буферного пространства, чем поток в среднем. Такая процедура позволяет адаптивным потокам (устойчивым и хрупким) помещать в буфер пачки пакетов и в то же время не позволяет неадаптивным потокам полностью занимать буферное пространство.

2.8.4 Подсчет среднего размера очереди

Алгоритм RED инициирует процедуру подсчета среднего размера очереди при поступлении каждого нового пакета в маршрутизатор. Таким образом, динамика поведения очереди может быть представлена в дискретном виде, где учитываются только моменты поступления пакетов. В этом случае не учитываются моменты ухода пакетов из очереди на обслуживание.

Например, предположим, что некоторый пакет поступает в маршрутизатор в момент времени 0 и значения среднего и моментального размеров очереди равны 500 пакетам. Пусть далее в течение промежутка времени, равного времени обслуживания 250 пакетов, в маршрутизатор не поступило ни одного пакета, а потом поступает пакет. Подсчитываются моментальный размер очереди, который будет равен 250 пакетам, и средний размер очереди – близкий к 500. В итоге, неправильное вычисление avg повлечет за собой снижение эффективности использования сетевых ресурсов и неправильный сброс поступающих пакетов.

В алгоритме FRED подсчет значения среднего размера очереди осуществляется как при поступлении пакета, так и при уходе на обслуживание. Таким образом, динамика поведения очереди отражается гораздо лучше, по сравнению с RED, и значения вычисляемых параметров более точны.

2.8.5 Другие алгоритмы.

Как можно заметить, каждый алгоритм активного управления очередями имеет свои достоинства и недостатки. Некоторый гипотетический алгоритм может функционировать устойчиво и с высокой эффективностью, но при этом быть достаточно сложным и дорогим для реализации в реальном оборудовании, другой же алгоритм может обладать рядом недостатков и быть легко реализуем. Время идет и еще десять лет назад было невозможно представить себе реальный коммерческий маршрутизатор, в котором, кроме маршрутизации, реализованы дополнительные сложные функции, создающие высокую нагрузку на процессор.

Однако уже сегодня во многих маршрутизаторах реализованы относительно простые алгоритмы управления RED и WRED, и недалек тот день, когда алгоритмы, кажущиеся сейчас «тяжеловесными», могут быть реализованы. Поэтому нельзя сбрасывать со счетов те алгоритмы, которые

сейчас кажутся сложными для реализации – их время определенно придет, причем достаточно скоро.

Исследователи за десять лет существования принципа активного управления очередями, разработали достаточно большое количество новых подходов и алгоритмов. Представленные в этом разделе алгоритмы заложили общие принципы и создали фундамент, на котором строятся дальнейшие исследования и разработки в этой области:

- REM (Random Exponential Marking – «вероятностная заблаговременная экспоненциальная маркировка»), представляет собой одну из реализаций целого направления в области построения новых алгоритмов активного управления очередями, идея которого заключается в поиске оптимальной функции зависимости вероятности сброса/маркировки пакета от среднего размера очереди. Конкретно для REM предлагается использовать экспоненциальную функцию. Использование алгоритма REM в беспроводных сетях позволяет улучшить эффективность функционирования системы и добиться уменьшения значений задержки пакета и вероятности его потери.
- SRED (Stabilized RED – «стабилизированный RED», - этот алгоритм позволяет улучшить параметры функционирования RED, в том числе, сохранять размер очереди на определенном уровне вне зависимости от количества активных источников, причем, в отличие от FRED, оценка количества активных источников производится без сохранения состояния о каждом потоке;
- RED-PD (RED with Preferential dropping – «RED с возможностью привилегированного сброса») Этот алгоритм предложен сравнительно недавно, в 2001 году, и может быть охарактеризован как эффективный и достаточно простой для реализации. Это связано с тем, что с одной стороны RED-PD осуществляет контроль за источниками, а с другой – управление состоянием производится только для потоков, создающих высокую нагрузку.

Заключение

Суммируя материал, представленный в данной главе, определим структуру CQS-маршрутизатора (рис. 2.8). Такой маршрутизатор является существенно более сложным, по сравнению с традиционным, т.к. в нем реализовано целое множество функций поддержки качества обслуживания и управления трагиком. Отметим, что именно сетевой уровень является уровнем, на, котором находятся основные механизмы обеспечения качества обслуживания — управление, маршрутизация, классификация и т.д.

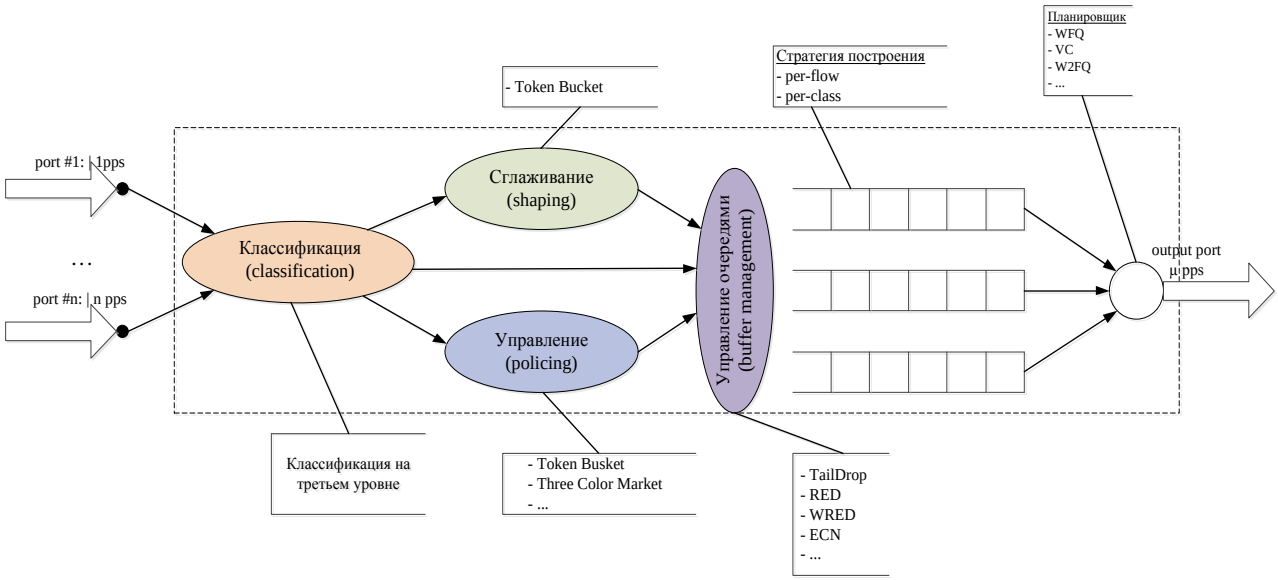

Рис 2.8 Структура CQS-маршрутизатора

3. Визуализатор NAM

Ns2 содержит средство анимации результатов моделирования – nam (Network Animator). Nam графически воспроизводит имитационную модель (топология сети, анимация прохождения пакетов по сети, постановки их в очередь и т.д.) и наглядно показывает алгоритмы работы протоколов, дисциплин обслуживания очередей. Он может служить наглядным пособием не только в научных, но и в учебных целях. Запустить nam можно либо с помощью команды nam <nam-file>, где <nam-file> – имя трейс-файла nam, созданного ns2, либо выполнить запуск прямо из скрипта моделирования OTcl.

Интерфейс пользователя содержит зону анимации, несколько меню и кнопок. В меню “views” находится четыре пункта:

- New view – создает новый вид той же анимации. Пользователь может увеличивать и прокручивать изображение в новом окне. Все виды работают синхронно.
- Show monitors – показывает окно в нижней части экрана, где осуществляется мониторинг.
- Show autolayout – показывает окно в нижней части экрана, которое содержит окна для ввода данных и кнопки для настроек автоматического расположения.
- Show annotation – показывает пункт в нижней части экрана с примечаниями по мере увеличения модельного времени.

Под панелью меню находятся кнопки перемотки назад, запуска анимации в обратную сторону, остановки анимации, запуска анимации, перемотки вперед и выхода из nam, индикатор текущего времени анимации и движок изменения скорости анимации (текущая скорость изображена над ним).

Также возможно увеличения и уменьшения изображения с помощью кнопок расположенных в левой части экрана. Изменять текущее время анимации пользователь может при помощи движка времени анимации. Под движком времени анимации находится панель автопланировки топологии сети (изначально может отсутствовать).

Существует три параметра для настройки процесса автоматической планировки:

- Ca – константа притягивающего воздействия между узлами, которая контролирует силу сжатия между узлами в зоне анимации.
- Cr – константа отталкивающего воздействия между узлам, которая контролирует силу отталкивания между узлами в зоне анимации.
- Количество итераций – определяет сколько раз запускать процедуру автопланировки.

Для маленьких топологий с десятками узлов использование исходных параметров (с 20-30 итерациями) достаточно для изображения приемлемого вида сети. Но для больших топологий необходимо изменение этих параметров.

3.1 Программы для построения графических зависимостей

Часто целью обработки трейс-файлов является построение различных графических зависимостей. Существует большое количество программ для построения графиков во всех операционных системах. Некоторые из них являются специализированными для ns2. Перечислим наиболее популярные из них:

- Программа GNUPlot
- Встроенный Модуль NS-2 XGRAPH
- Программа Tracegraph

3.1.1 Модуль XGRAPH

Программа XGraph является частью пакета ns2 «все в одном». Но также XGraph может быть установлен и отдельно от ns2. Преимущество XGraph заключается в том, что он может создавать рисунки в формате postscript, Tgif и др. XGraph может быть вызван внутри команды tcl, результатом чего будет являться вывод графика на экран сразу же после завершения моделирования. В качестве входных данных XGraph использует один или более ASCII-файлов, каждый из которых содержит набор строк, состоящих из пары точек x-y. Например, xgraph f1 f2 выведет на одной картинке графики из файлов f1 и f2.

3.1.2 Программа GNUPlot

Gnuplot – это широко доступное бесплатное программное обеспечение как для систем Unix, так и Windows.

Самый простой способ использования Gnuplot – это вывод на экран текста plot ‘fn’ , где файл fn имеет две колонки. Одна колонка – это точки по оси абсцисс, вторая – по оси ординат. Точки могут соединяться линиями различных типов:

plot ‘fn’ w lines 1

Вместо единицы пользователь может поставить любую другое число.

В качестве альтернативы можно использовать разные виды точек:

plot ‘fn’ w points 9

Рассмотрим другие некоторые возможности Gnuplot.

set size 0.6,0.6 set pointsize 3
set xrange [90.0:120.0]
plot ‘fn1’ w lines 1, ‘fn2’ w lines 8, ‘fn3’ w points 9
set key 100,8

Первая строка устанавливает размер кривой по отношению к установкам по умолчанию. Вторая строка устанавливает размер точек (делает их более жирными). Третья строка ограничивает ось x в диапазоне от 90 до 120. диапазон оси y устанавливается командой set yrange [ ]. Четвертая строка накладывает три кривых в один рисунок. Кривые взяты из трех разных файлов: fn1, fn2 и fn3. Пятая строка показывает, куда надо поместить легенду. Также, вместо точных координат можно указывать ключевые слова: ‘left’ (слева), ‘right’ (справа), ‘top’ (сверху), ‘bottom’ (снизу), ‘outside’ (на наружной части графика) и ‘below’ (под графиком). Например:
set key below

Также можно использовать команду set nokey, которая запрещает вывод легенды на экран. Информация, выводимая в легенде по умолчанию, – это имя соответствующего файла. Если необходимо озаглавить объект именем, отличающимся от имени файла, то необходимо это указать в команде plot:
plot ‘fn1’ t “математическое ожидание“ w lines 1, \
‘fn2’ t “дисперсия” w lines 2

Тогда в легенде появятся названия графиков «математическое ожидание» и «дисперсия».

Если необходимо использовать одну и ту же последовательность команд несколько раз, то можно создать файл (например, g1.com), содержащий эти команды и загружать его, когда необходимо:
load ‘g1.com’

Gnuplot может быть использован для извлечения колонки из файла:
plot ‘queue.tr’ using 1:($4/1000) t “кБайты” w lines 1, \
‘queue.tr’ using 1:5 t “пакеты” w lines 2

В данном примере первая кривая строится, используя первый столбец файла queue.tr в качестве оси x, а в качестве оси y – четвертый столбец, деленный на 1000. При построении второй кривой в качестве оси абсцисс используется первый столбец, а в качестве оси ординат используется пятый столбец. При написании команд порядок инструкций using, t и lines необходимо соблюдать.

3.1.3 Программа TraceGraph

Программа TraceGraph выгодно отличается от предыдущих программ тем, что она специально была разработана для ns2. В качестве входных данных она использует трейс-файлы ns2. То есть, в случае использования TraceGraph предварительная обработка данных (например, с помощью AWK) не нужна. Программа TraceGraph была разработана в 2001 году в

Технологическом Университете города Вроцлав (Польша). Автором программы является Ярослав Малек (Jaroslaw Malek). Последняя версия TraceGraph 2.04 выпущена в сентябре 2006 года. Программа является бесплатным ПО (только версия 2.02). Она доступна в Интернете на сайте www.geocities.com/tracegraph. TraceGraph работает как на платформах Unix/Linux, так и на Windows при условии, что установлена программа MathLAB (MathWorks Inc.), либо ее библиотеки, которые также доступны на сайте (файл mginstaller.exe). Для установки библиотек необходимо запустить файл и задать директорию (например, C:\mathlib). Далее в папку C:\mathlib\bin\win32 необходимо скопировать все файлы tracegraph 2.02. TraceGraph поддерживает все существующие форматы трейс-файлов ns2. Ниже приведены некоторые возможности TraceGraph 2.02:

- возможность построения 238 различных двумерных зависимостей;
- возможность построения 12 различных трехмерных зависимостей;
- возможность построения 8 различных гистограмм;
- зависимости и статистика по задержкам, вариациям задержек (джиттерам), временам обработки, периодам кругового обращения (RTT), количеству промежуточных узлов, производительности и др.;
- зависимости и статистика по всей сети, отдельно по узлам и звеньям;
- все результаты могут быть сохранены в текстовые файлы, а графики могут быть сохранены в формате tiff или jpeg;
- информация об осях x, y и z: минимум, среднее, максимум, среднее отклонение, средняя линия;
- может быть построен любой график, сохраненный в виде текстового файла, состоящего из двух или трех столбцов.

4. Разработка схемы для NS-2

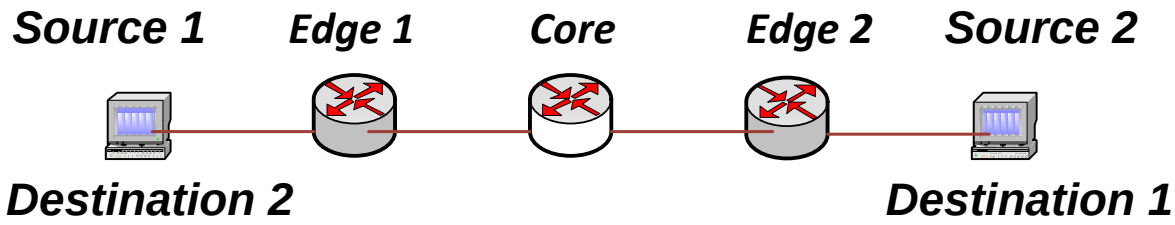

Рис 4.1 Схема в NS-2

Разберем схему подробнее. Наша схема состоит из 5 функциональных узлов: 2 компьютера, которые являются оконечными объектами схемы (Source1/Destination2 и Source2/Destination1) и 3 маршрутизатора, два из которых являются пограничными (Edge1 и Edge2) и один центральный (Core).

Задание топологии сети в ns2:

```
set s1 [$ns node] //Задание первого оконечного узла
set e1 [$ns node] //Задание первого оконечного маршрутизатора
set core [$ns node] // Задание центрального маршрутизатора
set e2 [$ns node] // Задание второго оконечного маршрутизатора
set dest [$ns node] //Задание второго оконечного узла
```

Установим связь между узлами топологии:

```
$ns duplex-link $s1 $e1 10Mb 5ms DropTail
$ns simplex-link $e1 $core 10Mb 5ms dsRED/edge
$ns simplex-link $core $e1 10Mb 5ms dsRED/core
$ns simplex-link $core $e2 5Mb 5ms dsRED/core
$ns simplex-link $e2 $core 5Mb 5ms dsRED/edge
$ns duplex-link $e2 $dest 10Mb 5ms DropTail
```

Таким образом мы задаем дуплексный канал между узлами s1 и e1 и между узлами dest и e2, с использованием алгоритма DropTail на этих участках. Между маршрутизаторами e1 и core, а также между e2 и core мы устанавливаем 2 симплексных канала с использованием алгоритма RED.

Целесообразно будет задать ориентацию для визуализатора NAM, чтобы он не изменил топологию нашей сети:

```
$ns duplex-link-op $s1 $e1 orient right
$ns duplex-link-op $e1 $core orient right
$ns duplex-link-op $core $e2 orient right
$ns duplex-link-op $e2 $dest orient right
```

Данными строками мы показываем, что наша сеть располагается в одну линию.

В нашей сети трафик будет идти от source1 до destination1 и от source2 до destination2. Исходя из этого зададим буферы виртуальных очередей:

```
set qE1C [[$ns link $e1 $core] queue]
set qE2C [[$ns link $e2 $core] queue]
set qCE1 [[$ns link $core $e1] queue]
```

```
set qCE2 [[$ns link $core $e2] queue]
```

Далее зададим параметры алгоритма RED для маршрутизаторов:

```
$qE1C meanPktSize $packetSize
$qE1C set numQueues_ 1 // Здесь задаётся количество физических очередей
$qE1C setNumPrec 2 // Здесь задается количество виртуальных очередей
$qE1C addPolicyEntry [$s1 id] [$dest id] TokenBucket 10 $cir0 $cbs0
$qE1C addPolicyEntry [$dest id] [$s1 id] TokenBucket 10 $cir1 $cbs1
$qE1C addPolicerEntry TokenBucket 10 11
$qE1C addPHBEntry 10 0 0
$qE1C addPHBEntry 11 0 1
$qE1C configQ 0 0 20 40 0.02
$qE1C configQ 0 1 10 20 0.10
```

В последних двух строчках задаются параметры виртуальных очередей: $qE1C configQ 0 0 20 40 0.02 – Здесь мы указываем что если количество пакетов в очереди будет от 20 до 40 то вероятность отбрасывания будет равна 0.02. Соответственно в строке $qE1C configQ 0 1 10 20 0.10 Говорится о том, что если число пакетов в очереди будет от 10 до 20 то вероятность отбрасывания пакета будет равна 0.1.

Необходимо настроить трафик в нашей сети. Для этого создадим 2 агента: TCP и UDP, а после этого прикрепим к ним CBR трафик.

Создание агента UDP:

```
set udp0 [new Agent/UDP] //Создание агента UDP
$ns attach-agent $s1 $udp0 //Прикрепление агента UDP к узлу s1
```

Создание источника CBR-трафика и присоединение его к агенту udp0:

```
set cbr0 [new Application/Traffic/CBR] //Создание источника трафика
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0 //Присоединение трафика к агенту
```

Создание агента TCP:

```
set tcp1 [new Agent/TCP] //Создание TCP агента
$ns attach-agent $dest $tcp1 //Прикрепление агента к узлу Dest
```

Создание источника CBR-трафика и присоединение его к агенту tcp1:

```
set cbr1 [new Application/Traffic/CBR] //Создание источника трафика
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $tcp1 //Присоединение трафика к агенту
```

Создание агента-получателя для udp0

```
set null0 [new Agent/Null]
$ns attach-agent $dest $null0
```

Создание агента-получателя для tcp1

```
set sink1 [new Agent/TCPSink]
$ns attach-agent $s1 $sink1
```

Таким образом, мы настроили трафик в нашей сети. У нас имеется два CBR потока. Один идет от s1 до dest с помощью UDP, а второй идет от dest до s1 с помощью TCP.

Задание

1. Построить сеть аналогичную, приведенной в пункте 4 с помощью NS-2. Задать источники трафика согласно номеру варианта.
2. Реализовать визуализацию полученной схемы в NAM
3. Получить графики изменения задержки и джиттера во времени с помощью любой из программ, приведенных в пункте.
4. Оформить отчет.

Варианты выполнения лабораторной работы

1	2	3	4	5	6	7	8	9	0
CBR1	Exp	Exp	CBR1	CBR2	Парето	Парето	Парето	Exp	CBR2
Exp	CBR1	Exp	CBR1	Парето	Exp	Парето	CBR2	Парето	CBR2

CBR1 (Constant Bit Rate) – Источник трафика с постоянной скоростью 800000 бит/с

CBR2 (Constant Bit Rate) – Источник трафика с постоянной скоростью 1600000 бит/с

Exp – Источник трафика с распределением по экспоненциальному закону

Парето – Источник трафика с рапределением по закону Парето

Содержание отчета

1) Цель работы
2) Рисунок схемы
3) Часть кода, в которой задаются источники трафика
4) Графики джиттера и задержки, полученные в любой из программ с пунктов 4.1.1-4.1.3.
5) Вывод о проделанной работе

Контрольные вопросы

1) Дайте определение понятию QoS. Какими параметрами он характеризуется?
2) Перечислите модели QoS. Расскажите о перечисленных моделях.
3) Расскажите об алгоритме RED. Перечислите другие алгоритмы, производные от RED.
4) Алгоритм ARED.
5) Алгоритм MRED.
6) Алгоритм WRED.
7) Алгоритм FRED.
8) Алгоритм RIO.

Приложение

```
set ns [new Simulator]
set cir0  1000000
set cbs0     3000
set rate0 2000000
set cir1  1000000
set cbs1    10000
set rate1 3000000
set testTime 85.0
set packetSize 1000
set s1 [$ns node]
set e1 [$ns node]
set core [$ns node]
set e2 [$ns node]
set dest [$ns node]
$ns duplex-link $s1 $e1 10Mb 5ms DropTail
$ns simplex-link $e1 $core 10Mb 5ms dsRED/edge
$ns simplex-link $core $e1 10Mb 5ms dsRED/core
$ns simplex-link $core $e2 5Mb 5ms dsRED/core
$ns simplex-link $e2 $core 5Mb 5ms dsRED/edge
$ns duplex-link $e2 $dest 10Mb 5ms DropTail
$ns duplex-link-op $s1 $e1 orient right
$ns duplex-link-op $e1 $core orient right
$ns duplex-link-op $core $e2 orient right
$ns duplex-link-op $e2 $dest orient right
```

```
set qE1C [[$ns link $e1 $core] queue]
set qE2C [[$ns link $e2 $core] queue]
set qCE1 [[$ns link $core $e1] queue]
set qCE2 [[$ns link $core $e2] queue]
$qE1C meanPktSize $packetSize
$qE1C set numQueues_ 1
$qE1C setNumPrec 2
$qE1C addPolicyEntry [$s1 id] [$dest id] TokenBucket 10 $cir0 $cbs0
$qE1C addPolicyEntry [$dest id] [$s1 id] TokenBucket 10 $cir1 $cbs1
$qE1C addPolicerEntry TokenBucket 10 11
$qE1C addPHBEntry 10 0 0
$qE1C addPHBEntry 11 0 1
$qE1C configQ 0 0 20 40 0.02
$qE1C configQ 0 1 10 20 0.10
$qE2C meanPktSize $packetSize
$qE2C set numQueues_ 1
$qE2C setNumPrec 2
$qE2C addPolicyEntry [$dest id] [$s1 id] TokenBucket 10 $cir0 $cbs0
$qE2C addPolicyEntry [$s1 id] [$dest id] TokenBucket 10 $cir1 $cbs1
$qE2C addPolicerEntry TokenBucket 10 11
$qE2C addPHBEntry 10 0 0
$qE2C addPHBEntry 11 0 1
$qE2C configQ 0 0 20 40 0.02
$qE2C configQ 0 1 10 20 0.10
$qCE1 meanPktSize $packetSize
```

```
$qCE1 set numQueues_ 1
$qCE1 setNumPrec 2
$qCE1 addPHBEntry 10 0 0
$qCE1 addPHBEntry 11 0 1
$qCE1 configQ 0 0 20 40 0.02
$qCE1 configQ 0 1 10 20 0.10
$qCE2 meanPktSize $packetSize
$qCE2 set numQueues_ 1
$qCE2 setNumPrec 2
$qCE2 addPHBEntry 10 0 0
$qCE2 addPHBEntry 11 0 1
$qCE2 configQ 0 0 20 40 0.02
$qCE2 configQ 0 1 10 20 0.10
set udp0 [new Agent/UDP]
$ns attach-agent $s1 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packet_size_ $packetSize
$udp0 set packetSize_ $packetSize
$cbr0 set rate_ $rate0
set null0 [new Agent/Null]
$ns attach-agent $dest $null0
$ns connect $udp0 $null0
set tcp1 [new Agent/TCP]
$ns attach-agent $dest $tcp1
```

```
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $s1 $sink1
        proc finish {} {
    global ns
    exit 0
}
$qE1C printPolicyTable
$qE1C printPolicerTable
$ns at 0.0 "$cbr0 start"
$ns at 0.0 "$cbr1 start"
$ns at 20.0 "$qCE2 printStats"
$ns at 40.0 "$qCE2 printStats"
$ns at 60.0 "$qCE2 printStats"
$ns at 80.0 "$qCE2 printStats"
$ns at $testTime "$cbr0 stop"
$ns at $testTime "$cbr1 stop"
$ns at [expr $testTime + 1.0] "finish"
$ns run
```