МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. Н.И. ЛОБАЧЕВСКОГО

Радиофизический факультет

Лаборатория физических основ и технологий беспроводной связи

СИСТЕМА МОДЕЛИРОВАНИЯ СЕТЕЙ СВЯЗИ NS 2

Методическое пособие

Составители: Канаков О.И., Ковалев П., Филимонов В.

Нижний Новгород

2003

Введение

Данная работа предназначена для студентов, специализирующихся в области телекоммуникаций, в частности, сетевых протоколов и стандартов. Для выполнения работы требуется владение основными принципами построения компьютерных сетей (модель OSI, стек протоколов TCP/IP, основы маршрутизации). Желательно знакомство с основами объектной модели в программировании, а также с языками C++ и OTcl. В ходе работы студенты знакомятся с методами имитационного моделирования сетей на примере системы Ns2, распространяемой на условиях лицензии GPL в форме открытого кода. Также, работа знакомит с основными алгоритмами протокола транспортного уровня TCP и с дисциплинами обслуживания очередей, основанными на технологии RED (Random Early Detection).

1. Сведения о системе Ns2.

1.1 Имитационное моделирование

В настоящее время наблюдается активное развитие как сетей связи, так и услуг, предоставляемых этими сетями. Этот процесс требует не только разработки нового технологического оборудования, программных продуктов и стандартов, но и подготовки квалифицированных специалистов. Компьютерное моделирование, как показала практика, играет существенную роль при решении как тех, так и других задач. В процессе разработки модель, аппроксимирующая свойства и поведение исследуемой сети, позволяет решать задачи по оптимизации и управлению. Апробация тех или иных решений на модели несравнимо дешевле, чем на реальной системе, и позволяет исключить возможные ошибки. В учебном процессе моделирование позволяет наглядно проиллюстрировать основные принципы и дать обучаемым навыки работы с такими системами, которые физически им недоступны.

При моделировании сетей связи, как правило, используется один из двух подходов. Первый из них основан на теории телетрафика, которая является приложением теории систем массового обслуживания к сетям связи и базируется на принципах теории вероятности. В рамках этого подхода могут быть получены строгие математические результаты, представляемые на вероятностном языке. Недостатком подхода является сложность расчетов, делающая трудным или невозможным исследование сложных систем, особенно если они нестационарны во времени.

Второй подход заключается в непосредственном моделировании процессов, протекающих в системе (например, генерации заявок и их обслуживания), с помощью специальной программы. Такой способ моделирования называется имитационным и относится к классу методов Монте-Карло в том смысле, что все случайности, имеющие место в реальной системе (случайные задержки, ошибки передачи и т. п.), моделируются с помощью датчиков случайных чисел. Каждый запуск такой модели, таким образом, является реализацией некоторого случайного процесса. Поэтому результаты моделирования в таких системах могут различаться от запуска к запуску, и для получения достоверных сведений требуется статистическая обработка достаточно большого объема результатов.

В настоящее время существует несколько коммерческих программных продуктов, предназначенных для имитационного моделирования сетей связи. Их конкурентом является система Ns2, бесплатно распространяемая в открытых кодах на условиях лицензии GPL (GNU Public License). К преимуществам

открытого кода, помимо бесплатности, можно отнести возможность изменения и дополнения системы для нужд конкретной задачи (например, реализации новых протоколов для изучения их работы и сравнения их с существующими).

1.2 Структура Ns2

Основной частью Ns2 является симулятор Ns. Он осуществляет имитационное моделирование сетей на уровне пакетов, то есть, моделирует генерацию пакетов и прохождение их по сети. Возможно моделирование протоколов транспортного уровня UDP и различных реализаций TCP, multicast-протоколов, различных протоколов маршрутизации в проводных и беспроводных сетях, протоколов прикладного уровня FTP и Telnet, очередей с дисциплинами обслуживания DropTail и RED. Кроме того, моделируются некоторые факторы, относящиеся к физическому уровню: задержка пакетов в каналах, возникновение ошибок, видимость/невидимость узлов в беспроводных сетях (как наземных, так и спутниковых), расход энергии батарей в устройствах с автономным питанием.

Результатом работы симулятора являются выходные текстовые файлы, в которых регистрируется ход моделирования (моменты генерации/получения пакетов, состояние очередей, отброс пакетов в очередях и т. д.). Кроме того, в модель могут быть включены инструкции, вычисляющие любые величины, измерение которых требуется в конкретной задаче (задержка пакетов, пропускная способность и т. п.). Значения этих величин в ходе моделирования также могут регистрироваться в выходных файлах.

Для визуализации результатов служат аниматор NAM (Network Animator) и построитель графиков Xgraph. Кроме того, система содержит генератор топологий, упрощающий описание топологии больших сетей.

Симулятор Ns состоит из двух частей. Одна из них написана на языке C++ и должна быть перекомпилирована в случае внесения изменений и дополнений; другая написана на интерпретируемом языке OTcl (объектно-ориентированное расширение языка сценариев Tcl) и, соответственно, вообще не подлежит компиляции. При этом иерархии классов в обеих частях имеют совпадающие части и в терминологии Ns называются компилируемой и интерпретируемой иерархиями, соответственно. Взаимодействие между частями, написанными на таких принципиально разных языках программирования, осуществляется согласно спецификации, определяющей способ обращения из tcl-сценария (скрипта) к любому методу классов компилируемой иерархии и возвращение назад результатов, а также способ обращения из программы на C++ к любому методу, описанному в tcl-сценарии. Во втором случае, фактически, интерпретатор языка Tcl вызывается из C++ как функция. По замыслу создателей Ns, все методы, имеющие дело с отдельными пакетами и потому требующие высокого быстродействия, относятся к компилируемой части. Интерпретируемая же часть отвечает за менее частые события, чем передача пакетов, обеспечивающие управление ходом моделирования, и манипуляцию объектами, описанными в компилируемой части. Также, tcl-скриптом является собственно описание модели сети, подлежащей исследованию.

1.3 Принципы функционирования симулятора

С точки зрения объектного подхода в имитационном моделировании, сеть представляет собой совокупность сетевых объектов (Network Objects), каждый из которых способен определенным образом реагировать на некоторое множество событий (Events). Обо всех событиях сетевые объекты извещаются

так называемым планировщиком (Scheduler), который содержит хронологическую таблицу событий. Каждое событие имеет обязательные атрибуты: время наступления и сетевой объект, к которому относится это событие. Планировщик по очереди (в хронологическом порядке) выбирает из таблицы события и извещает соответствующие сетевые объекты о наступлении этих событий, сообщая также момент времени, в который произошло это событие. С точки зрения программиста, такое «извещение о событии» представляет собой просто вызов метода-обработчика для соответствующего объекта. Вся сопутствующая информация передается через параметры этого метода.

Помещать события в таблицу планировщика могут как сами сетевые объекты, так и пользователь при описании модели. В первом случае событие называется пакетом (так как, по смыслу предметной области, обмен пакетами это и есть способ взаимодействия между сетевыми объектами), во втором – at-событием. Последние предназначены для управления ходом моделирования.

Таким образом, любое взаимодействие между сетевыми объектами осуществляется через посредство планировщика. Отправка пакета одним сетевым объектом другому есть ни что иное, как постановка отправителем в таблицу события, адресованного получателю и помеченного тем моментом времени, когда пакет должен до этого получателя дойти (возможно, с учетом задержки при передаче). Получение пакета, соответственно, представляет собой извещение планировщиком получателя о событии и обработка этого события получателем.

При этом (если речь не идет о планировщике реального времени) модельное время не имеет никакого отношения к реальному. Планировщик извлекает события из таблицы, обращая внимание лишь на порядок их следования. Интервал между обработкой последовательных событий определяется лишь скоростью их обработки компьютером и не зависит от того, какими моментами модельного времени помечены эти события в таблице. Модельное время находит свое отражение лишь в трассировочных файлах, куда сетевые объекты записывают моменты отправки и получения пакетов (естественно, в модельном времени).

Напротив, планировщик реального времени (Real-Time Scheduler) синхронизует модельное время с реальным, ожидая между выборкой последовательных событий из таблицы по возможности в точности столько времени, сколько составляет разность между моментами, которыми помечены эти события. Такие планировщики используются, когда компьютерная модель входит в состав реальной сети. Например, создав на одном или нескольких компьютерах Ns2-модель всего российского интернета с планировщиком реального времени и подключив к такой модели два других компьютера, можно реалистично имитировать ситуацию, когда эти компьютеры связываются друг с другом, находясь в любых двух заданных точках России. Такой планировщик, однако, в данной работе не используется.

Отметим еще, что из соображений простоты все пакеты в системе имеют одинаковый формат (являются объектами одного класса), и поэтому каждый пакет, независимо от того к какому протоколу он относится, имеет в своем составе заголовки *всех* протоколов, используемых в модели. Типичная структура пакета в Ns2 изображена на рис. 1 (поле данных используется лишь в случаях, когда Ns2-модель входит в состав реальной сети; если модель чисто компьютерная, то достаточно иметь в заголовке поле, содержащее размер

пакета). Для ускорения процесса моделирования необходимо указывать явным образом, какие протоколы будут использоваться в модели, чтобы исключить заголовки неиспользуемых протоколов из заголовка пакета.

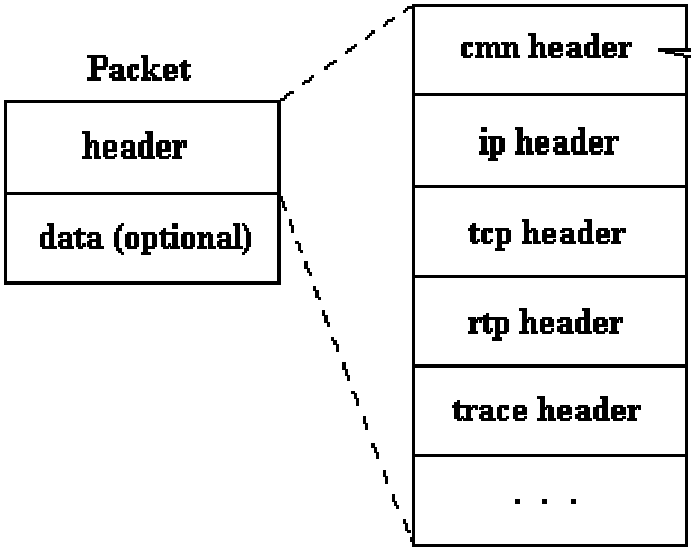


Рис. 1 Структура пакета

Сетевые объекты могут быть простыми и составными. К простым объектам относятся очереди, линии задержки, мультиплексоры, демультиплексоры, а также такие нетривиальные объекты, как агенты и приложения (см. ниже). Составные объекты формируются путем объединения простых объектов. Система Ns2 содержит большой набор готовых описаний классов составных объектов, среди которых узлы (Nodes), линии связи (Links), локальные сети (как проводные 802.3 Ethernet, так и беспроводные 802.11) и даже спутниковые сети (как на геостационарных, так и на низкоорбитальных спутниках). Однако, система может быть расширена введением в нее новых классов сетевых объектов. Особенно удобно создавать составные объекты на основе имеющихся (как простых, так и составных). При этом для описания новых классов может использоваться либо язык C++, либо OTcl. В случаях, когда важна производительность, используется C++ и перекомпилируется вся система; наоборот, когда производительность некритична и на первое место выходит легкость и быстрота разработки, используется OTcl, который вообще не требует компиляции (см. выше).

Абстракцией сетевого уровня в системе Ns2 является узел. Узел может принимать пакеты и классифицировать их по адресу и порту, являющимся полями IP-заголовка. Как было отмечено выше, узел является составным объектом. Простейшая структура узла изображена на рис. 2. Прибывающие в узел пакеты принимаются входной точкой (Entry Point) узла и классифицируются классификатором адреса (Address Classifier), представляющим собой демультиплексор. Пакеты, адресованные другим узлам, передаются на одну из линий связи, подключенных к узлу, в соответствии с маршрутной таблицей, принадлежащей классификатору адреса. Пакеты же, адресованные данному узлу, передаются классификатору порта (Port Classifier), также являющемуся демультиплексором. В зависимости от номера порта, классификатор передает пакет одному из прикрепленных к узлу агентов

(Agents), которые являются сетевыми объектами, не входящими в состав узла и отвечающими за реализацию протоколов более высокого уровня (например, транспортных протоколов UDP и различных реализаций TCP, протоколов маршрутизации и пр.). Отметим, что узел может иметь практически любое количество связей, а также портов с прикрепленными к ним агентами. Существуют также более сложные multicast-узлы, способные рассылать прибывающие пакеты нескольким адресатам, но они в данной работе не используются.

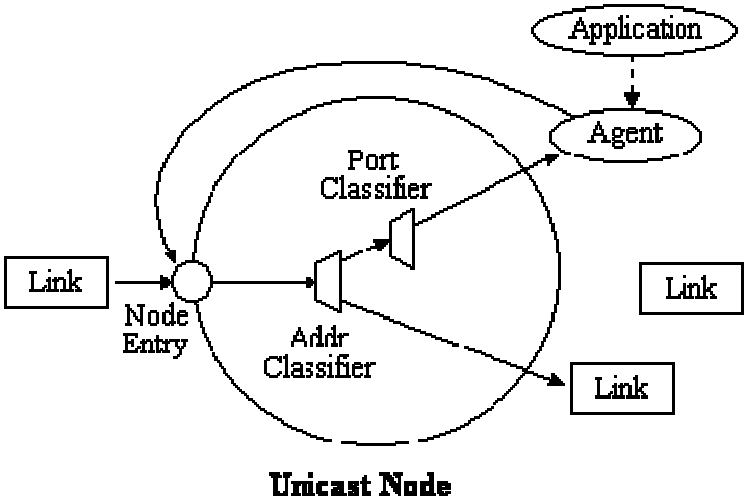


Рис. 2 Структура узла

Маршрутная таблица узла может быть как статической, так и динамической. Во втором случае она формируется агентом маршрутизации, подключенным к узлу и получающим информацию о топологии сети путем обмена специальными пакетами с аналогичными агентами на других узлах (как это и делается при динамической маршрутизации в реальных сетях).

Для создания TCP-соединения необходимо создать два агента: TCP-передатчик и TCP-приемник. TCP-приемник лишь принимает пакеты и посылает назад уведомления о получении (ACK). Поэтому такой агент в системе один: TCPSink. Передатчик же может вести себя различным образом в зависимости от порядка и моментов получения им уведомлений от адресата о получении отправленных им пакетов. Соответственно, в системе Ns2 имеется несколько вариантов TCP-передатчика, моделирующих различные реализации протокола TCP (TCP, TCP/Tahoe, TCP/Reno, TCP/Vegas), использующие разные комбинации алгоритмов TCP (Slow Start, Congestion Avoidance, Fast Recovery, Fast Retransmit).

С протоколом UDP все обстоит гораздо проще, потому что уведомления о получении там не предусмотрены. Для передачи используется агент UDP, а для приема – агент Null, просто принимающий и отбрасывающий все приходящие пакеты.

За прикладной уровень в Ns2 отвечают приложения (Applications). Приложение прикрепляется к агенту и служит для создания трафика. В Ns2 имеются приложения, моделирующие трафик, характерный для реальных протоколов прикладного уровня (FTP и Telnet), а также абстрактные генераторы трафика различного типа (например, CBR – простейший генератор трафика с

постоянным темпом выдачи пакетов). Приложения запускаются и останавливаются пользовательскими at-событиями.

Дуплексные (двусторонние) линии связи в Ns2 являются составными объектами, состоящими из двух противоположно направленных симплексных (односторонних) линий связи.

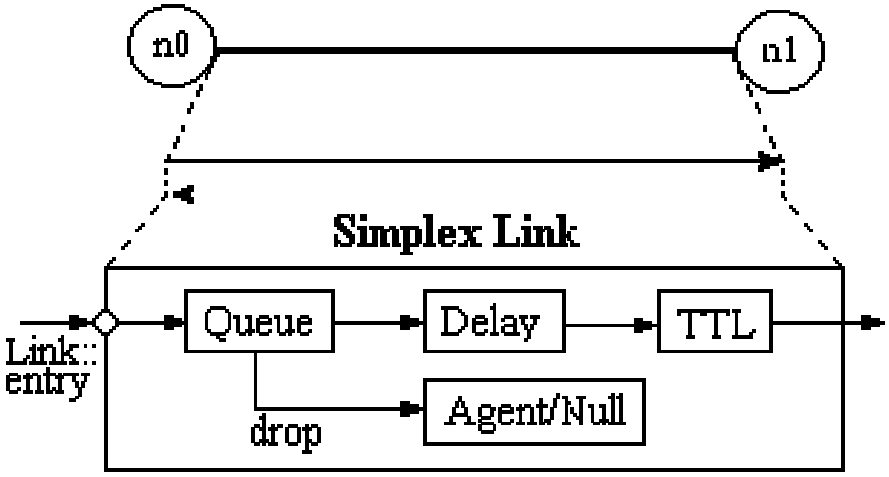


Рис. 3 Структура линии связи

Симплексная линия связи также является составным объектом (см рис. 3). Она состоит из очереди, Null-агента, в который отправляются отброшенные из очереди пакеты, линии задержки и объекта, проверяющего и уменьшающего на единицу поле времени жизни (Time to Live, TTL) пакета. Кроме того, могут быть еще объекты-мониторы, следящие за состоянием очереди (постановка в очередь, извлечение, отброс пакетов). Заметим, что все эти объекты, за исключением линии задержки, физически относятся скорее к узлу, чем к линии связи. Объяснить, почему создатели Ns2 ввели их все же в состав линии связи, можно двумя способами.

Во-первых, очередь должна «знать», с какой скоростью ей выдавать пакеты в линию связи, а эта скорость в модели определяет пропускную способность линии связи (хотя в реальности, наоборот, скорость передачи пакетов определяет*ся* пропускной способностью линии). Кроме очереди, нет ни одного другого объекта, через параметры которого можно было бы ввести в модель пропускную способность линии. Поскольку пропускная способность является атрибутом именно линии связи, то логично, чтобы очередь (а значит, и все вспомогательные объекты, которые к ней подсоединены), входила в состав линии связи.

Во-вторых, такой подход позволил более равномерно распределить сложность по сетевым объектам. Иначе от линии связи осталась бы лишь линия задержки, а узел бы существенно усложнился.

2. Сведения об алгоритмах TCP

Как известно, информация в протоколах TCP/IP передается пакетами. В этом разделе будет рассказано о том, каким образом в TCP организуется надежная доставка пакетов и их перенаправление.

2.1 Алгоритм Slow Start – медленное начало.

В старой реализации протокола TCP/IP передающая машина начинала соединение посылкой множества пакетов вплоть до размера окна, заданного

приемником (обозначим RWND) на принимающую машину. Это хорошо только в том случае, если обе машины находятся в одном сегменте сети. Если же между компьютерами есть «медленные» сегменты, может произойти переполнение стека одного из роутеров (см. рис. 4; этот и последующие рисунки построены по результатам моделирования в Ns2), что вызовет долгую задержку передачи.

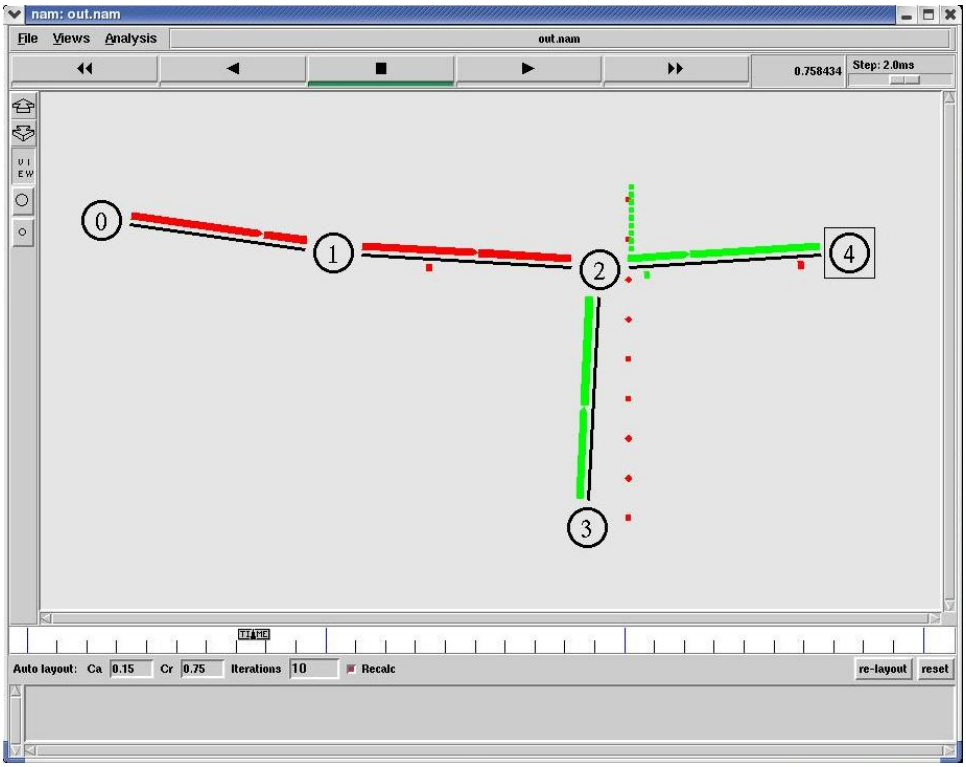


Рис. 4 Переполнение очереди

Алгоритм, созданный для того, чтобы уменьшить задержки при переполнении, называют Slow Start.

По алгоритму Slow Start передающая машина отправляет пакеты в сеть по мере того, как получает подтверждения приема пакетов. Slow Start добавляет отправителю дополнительную информацию о соединении – окно переполнения, называемое CWND.

После установления соединения с удаленным хостом происходит начальная инициализация CWND. Это размер одного сегмента, объявляемый или другой машиной, или устанавливаемый по умолчанию в 536 или 512 байт, обозначим его segsize.

Отправитель может передать **min (CWND, DWND)** данных.

CWND – это контроль потока передачи на стороне отправителя.

DWND – это контроль потока передачи на стороне получателя.

Первый базируется на результативности передачи данных. Второй – на размере буфера для данного соединения на принимающей машине.

Шаги алгоритма Slow Start можно проиллюстрировать следующим образом:

1. CWND = 1*segsize.
2. Отправитель посылает один сегмент и ждет ACK.
3. После его получения CWND увеличивается в два раза.

4. Передаются два сегмента. После получения обоих ACK размер CWND снова удваивается.

5. И т. д.

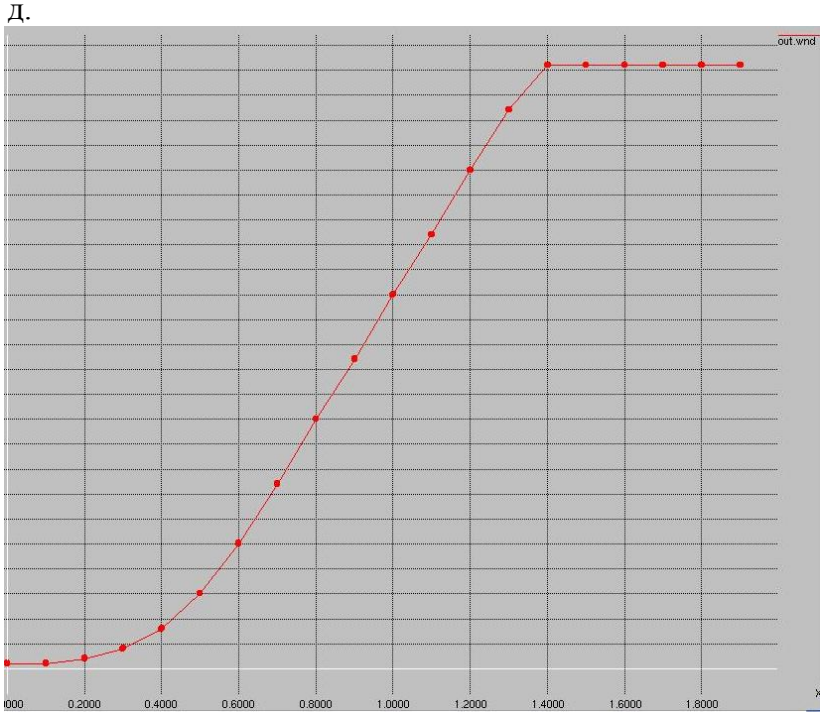


Рис. 5 Slow Start: экспоненциальный рост числа переданных пакетов

Получаем экспоненциальный рост (см. рис. 5), хотя не совсем так – ведь получатель может задерживать ответ ACK. Обычно это делается посылкой одного ACK на каждые два пакета.

Если приемник не будет этого делать, то наступит момент, когда он начнет отказываться от пакетов. Это говорит передатчику о том, что окно CWND слишком разрослось.

Тут-то и происходит переход к следующему алгоритму.

2.2 Congestion Avoidance – предотвращение переполнения.

Переполнение может произойти в том случае, если данные из «быстрого» сегмента сети передаются в более «медленный», или же когда суммарный объем множества входящих потоков данных превосходит пропускную способность принимающего канала.

Алгоритм Congestion Avoidance направлен на определение переполнения в сети и его предотвращение.

Congestion Avoidance и Slow Start – два независимых алгоритма. Но когда происходит переполнение, TCP должен замедлить передачу данных в сеть алгоритмом Congestion Avoidance, а затем призвать Slow Start стабилизировать посылки пакетов на новой, более низкой скорости.

На практике алгоритмы программируются в паре.

Для совместной работы алгоритмы требуют по две переменных на каждое соединение: CWND и пороговый размер ssthresh.

При инициализации CWND устанавливается размером в один сегмент, а ssthresh - к 65535 байтам.

Как и при Slow Start, TCP никогда не посылает больше, чем **min (CWND, RWND).**

Вероятность потери пакета где-то в канале данных («в проводах») между отправителем и получателем гораздо меньше 1%, и поэтому Congestion Avoidance воспринимает потерю пакета как сигнал о переполнении. Есть два признака потери пакета: timeout и получение дубликата АСК.

Шаги алгоритма Congestion Avoidance:

1. Когда происходит переполнение, **ssthresh=min (CWND,RWND)/2.** (При этом ssthresh не меньше 2 сегментов.)
2. Если же признак переполнения - это timeout, то дополнительно CWND=1*segsize, то есть работает «медленное начало».

При получении АСК возможны два варианта (в зависимости от того, какой алгоритм работает на данный момент):

Если CWND <= ssthresh, то TCP работает по алгоритму Slow Start, иначе – по Congestion avoidance. Соответственно CWND растет либо по экспоненциальному, либо по линейному закону.

Congestion avoidance увеличивает окно так:

CWND = CWND + segsize*segsize/CWND (в байтах) при каждом АСК.

Это – линейный рост CWND. Максимальное увеличение CWND – один сегмент за каждый RTT (round-trip time). Congestion Avoidance может срабатывать лишь единожды в каждом RTT, независимо от количества АСК. Slow Start же увеличивает CWND по количеству принятых АСК.

Разработчики уделяют большое внимание совершенствованию алгоритмов передачи данных.

В основном, их усилия направлены на математическое изучение поведения сети и внедрение дополнительных коэффициентов, получаемых из параметров сети, в формулы расчета размера окна.

2.3 Fast Retransmit – быстрая повторная передача.

Модификации к алгоритму предотвращения переполнения были предложены тоже в 1990 году.

TCP может произвести немедленный дубликат АСК, если принимающая машина получила не те сегменты. Эта идея лежит в основе экспериментального алгоритма Fast Retransmit.

Сразу после ошибки приема посылается дубликат АСК, который говорит передатчику о том, какой номер последовательности ожидается.

Теперь передатчик не знает, чем вызван повторный АСК – потерей данных, или запросом на повторную передачу.

Предполагается, что если поступило 1 или 2 повторных АСК, то это – запрос на повтор передачи. Если 3 и более – то это потеря пакетов.

В том случае, если мы получим запрос о повторении передачи – TCP сразу выполняет переподачу, не ожидая срабатывания таймера повторной передачи; затем работает Slow Start.

Иначе включается алгоритм Congestion Avoidance.

2.4 Fast Recovery – быстрое восстановление.

Суть этого алгоритма заключается в том, что после повторной передачи пакета, который, якобы, был неправильно принят по мнению Fast Retransmit, включается не Slow Start, а Congestion Avoidance.

Это улучшение особенно эффективно при большом размере окон, т. е. на «широких» каналах, так как там придется довольно долго наращивать CWND.

Также причина в том, что данные, полученные приемником во время передачи дубликата ACK, не теряются, а находятся в его буфере, и потому не имеет смысла резко снижать скорость передачи.

Шаги алгоритма:

1. Когда получен третий дубликат ACK, устанавливается **ssthresh=min (CWND,RWND)/2.**

(При этом ssthresh не меньше 2 сегментов.)

2. Повторно передается потерянный сегмент.

3. CWND = CWND + 3*segsize.

(Это увеличивает окно переполнения количеством сегментов, которые другая машина держит в памяти.)

4. При получении каждого нового дубликата ACK CWND увеличивается на segsize. Это увеличивает окно переполнения для дополнительного сегмента, который покинул сеть. Передаем пакет, если нам это позволяет новое CWND.

5. Когда прибывает следующий ACK, который означает новые данные, CWND устанавливают равным ssthresh (а он был установлен на первом шаге).

Этот ACK должен быть подтверждением переpередачи от шага 1, одного RTT. К тому же, этот ACK должен обусловить все промежуточные пакеты, посланные между потерянным пакетом и получением первого дубликата ACK. Получилось, что 5 шаг - это предотвращение переполнения. Теперь скорость снижена вдвое по сравнению с той, на которой пакет был потерян.

Алгоритм Fast Retransmit впервые был применен в 4.3BSD Tahoe и сопровождался алгоритмом Slow Start. Алгоритм Fast Recovery применили в 4.3BSD Reno.

3. Сведения о технологии RED

До введения технологии RED существовало два способа борьбы с переполнением очереди – PPD (Past Packet Drop) и EPD (Early Packet Drop). В первом случае при переполнении пакеты отбрасывались с конца, а во втором случае – с начала очереди.

Технология RED (Random Early Detection) работает по следующему алгоритму. Задаются верхняя и нижняя границы размера очереди (**max_{th}** и **min_{th}**. Обычно **max_{th}=3*min_{th}**). Если средний размер очереди меньше заданного нижнего порога, то сброса пакета не происходит. Если размер больше верхнего порога, то пакет сбрасывается. Если же средний размер очереди лежит в промежутке между порогами, то пакет сбрасывается с вероятностью, линейно зависящий от среднего размера очереди.

Примем следующие обозначения:

max_{th} – Верхний порог
min_{th} – Нижний порог
avg – Средний размер очереди
Count – Число пришедших пакетов со времени последнего сброса или со времени достижения нижней границы
Pa – Вероятность сброса пакета.
Pb – Промежуточная вероятность
Maxp – Максимальная вероятность сброса пакета
Q – Текущая длина очереди

Wq	– Весовой коэффициент для вычисления средней длины очереди
f()	– Некая линейная функция
time	– Текущее время
q_time	– Время, когда последний раз очередь опустела

Алгоритм обслуживания очереди RED тогда можно описать следующим образом:

```
Avg=0
Count=-1
Для каждого приходящего пакета {
    Count++
    Вычисление Avg
    If (minth<Avg<maxth)  { Вычисление вероятности Pa}
    If (Avg>maxth) { Pa=1 }
    If (Avg<minth) { Pa=0; Count=-1}
    If (Rnd<Pa) то { сбросить пакет;      Count=-1}
}
```

Вычисление средней длины очереди происходит по алгоритму фильтра НЧ с экспоненциальным весом:

```
If (очередь не пуста) {
  avg=(1 - Wq) * avg + Wq * q
}
Else {
    m=f(time - q_time)
    avg=(1-Wq)m * avg
}
```

Пример работы этого алгоритма показан на рис. 6. Синим цветом построен график мгновенного размера очереди, а красным – усредненного по описанному выше алгоритму.

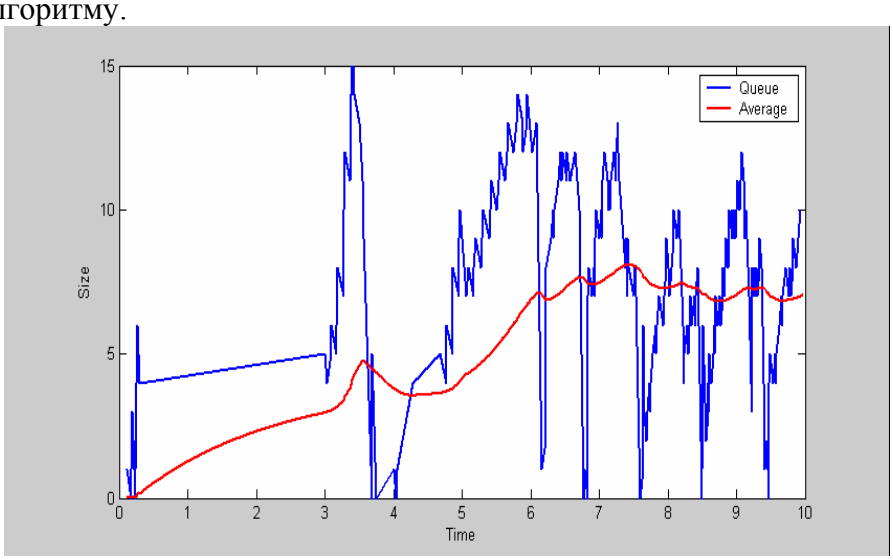


Рис. 6 Вычисление среднего размера очереди

Вычисление промежуточной вероятности сброса пакета выполняется по линейному закону. Далее вероятность несколько преобразуется для лучшего соответствия реальному положению вещей:

$$P_b = \max_p \frac{Avg - \min_{th}}{\max_{th} - \min_{th}}$$

$$P_a = \left| \frac{P_b}{1 - Count * P_b} \right|$$

График предварительной вероятности (Pb) сброса пакета представлен на рис. 7.

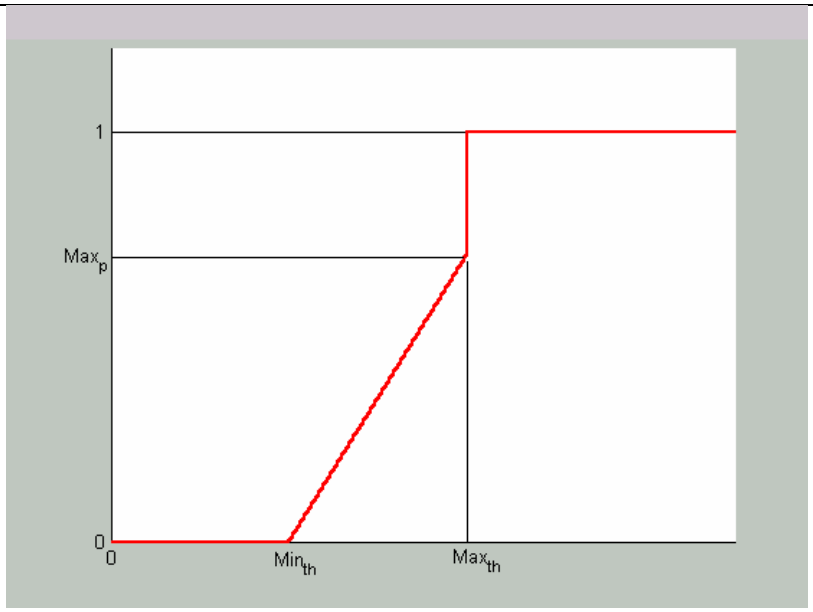


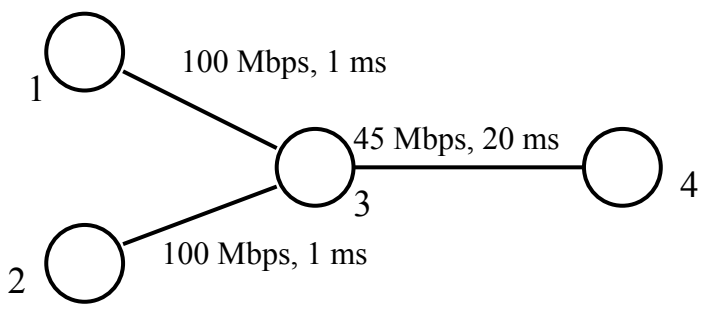
Рис. 7. Предварительная вероятность отброса пакета
в зависимости от средней длины очереди

Вычисление весового коэффициента – чрезвычайно важная задача. Если взять коэффициент слишком большим (радиотехнический эквивалент – постоянная времени ФНЧ слишком мала), то средний размер очереди будет практически совпадать с текущим размером. В результате мы получим чёткую и быструю реакцию на изменение длины очереди, но при малейшей опасности перегрузки произойдёт необоснованный сброс большого числа пакетов.

Если же выбрать коэффициент маленьким (постоянная времени ФНЧ велика), то средняя длина очереди будет слабо меняться, и мы будем сглаживать пиковые нагрузки без сброса пакетов, но, соответственно, реакция на изменение длины очереди будет медленной.

4. Пример построения модели в системе Ns 2.

Для моделирования используется следующая схема сети.



На 1-м и 2-м узлах расположены FTP-источники, посылающие пакеты на 4-й узел. Все каналы - дуплексные

4.1 Константы сети.

Для начала, зададим константы, определяющие нашу сеть.

```
set n13_throughput 100Mb
set n13_bandwidth 1ms
set n23_throughput 100Mb
set n23_bandwidth 1ms
set n34_throughput 45Mb
set n34_bandwidth 20ms
set Window 240
set PacketSize 1000
set SimTime 5.0
set StartTime 1.0
set Limit 150
```

4.2 Процедура инициализации.

Теперь, необходимо задать процедуру, инициирующую эмулятор. В ней необходимо установить генератор случайных чисел в начальное состояние, задать размер пакета, время моделирования, определить значение максимального размера очереди, а также выделить память под структуру симулятора и инициировать NAM. Как аргумент эта функция будет принимать значение максимального размера очереди и начальной установки генератора случайных чисел.

```
proc init {seed BuffSize} {
    global ns SimTime PacketSize Window

    puts "$seed"

    ns-random $seed
    set ns [new Simulator]
    set nf [open out_trace.nam w]
    $ns namtrace-all $nf

    $ns color 0 blue
    $ns color 1 red
    Agent/TCP set window_ $Window
    Agent/TCP set packetsize_ $PacketSize
    Queue set limit_ $BuffSize
}
```

4.3 Задание топологии и трафика.

Теперь необходимо определить топологию сети.

```
proc topologic {queuetype} {
    global ns n1 n2 n3 n4
    global n13_throughput n13_bandwidth
    global n23_throughput n23_bandwidth
    global n34_throughput n34_bandwidth
```

```
        set n1 [$ns node]
        $n1 color "blue"
        set n2 [$ns node]
        $n2 color "red"
        set n3 [$ns node]
        $n3 shape box
        set n4 [$ns node]

        $ns duplex-link $n1 $n3 $n13_throughput
          $n13_bandwidth DropTail
        $ns duplex-link $n2 $n3 $n23_throughput
          $n23_bandwidth DropTail
        $ns duplex-link $n3 $n4 $n34_throughput
          $n34_bandwidth $queuetype
        $ns duplex-link-op $n3 $n4 queuePos 0.5
    }
```

Последняя строка используется визуализатором NAM.

Задание трафика должно включать в себя создание агентов протокола TCP, поверх которого будут генерироваться FTP-пакеты, и подключение его к соответствующим узлам сети. Время включения 2-го узла можно задавать как детерминировано, так и случайно. Мы будем использовать первый способ. Кроме того, нужно задать так называемых SINK-агентов, посылающих ACK-пакеты отправителю и освобождающих пришедшие пакеты (Если бы мы использовали UDP-агента, то SINK был бы излишен и достаточно было бы вместо него поставить NULL-агента, который бы просто избавлялся от пришедших пакетов). После создания SINK (NULL)-агента нужно установить логическую связь между ними.

```
    proc make_traffic {} {
        global ns n1 n2 n3 n4 tcp1 tcp2 StartTime fmon_

        set tcp1 [new Agent/TCP]
        $tcp1 set overhead_ 0.00017
        $ns attach-agent $n1 $tcp1
        set snk1 [new Agent TCP/Sink]
        $ns attach-agent $n4 $snk1
        $ns connect $tcp1 $snk1
        $tcp1 set fid_ 1

        set tcp2 [new Agent/TCP]
        $tcp2 set overhead_ 0.00017
        $ns attach-agent $n2 $tcp2
        set snk2 [new Agent TCP/Sink]
        $ns attach-agent $n4 $snk2
        $ns connect $tcp2 $snk2
        $tcp2 set fid_ 2

        set ftp1 [new Source/FTP]
        $ftp1 set agent_ $tcp1
        $ns at 0.0 "$ftp1 start"

        set ftp2 [new Source/FTP]
```

```
        $ftp2 set agent_ $tcp2
        $ns at $StartTime "$ftp2 start"

        set fmon_ [$ns makeflowmon Fid]
        $ns attach-fmon [$ns link $n3 $n4] $fmon_
    }
```

Для задания случайного времени старта нужно вместо константы **$StartTime** использовать следующую конструкцию:

```
        Set StartTime [expr [ns-random] / 2147483647.0 /
          100]
```

Последние две строки кода задания трафика инициализируют мониторинг потока пакетов в канале между 3-м и 4-м узлами.

4.4 Исследование очереди.

Для исследования очереди обратимся к созданному нами объекту **fmon_**. Устанавливая для него "классификатор" (объект, позволяющий в более/менее доступной форме представлять результаты мониторинга очереди) мы можем, используя встроенные функции, узнать среднюю задержку в канале.

Код взят с сайта http://www.aciri.org/floyd/papers/

```
    proc get_queue {} {
        global ns fmon_

        set fids { 1 2 }
        set total 0.0

        foreach i $fids {
            set flow [$fcl lookup auto 0 0 $i]
            if { $flow != "" } {
                set dsamp [$flow get-delay-samples]
                set mean [$dsamp mean]
                set total [expr $total + $mean]
                puts "flow $i mean [format "%8.6f" $mean]"
            }
        }
        puts "mean delay (in seconds) [format "%8.6f"
          [expr $total / 2]]"
    }
```

В случае использования данного кода очередь в 100 пакетов соответствует задержке в 0.017 секунд.

4.5 Процедура завершения.

В процедуре завершения мы выведем значения пропускной способности канала 3-4 для каждого FTP-источника в отдельности, а также суммарную пропускную способность. Кроме того мы запустим в фоновом режиме визуализатор NAM.

```
    proc finish{} {
        global SimTime PacketSize tcp1 tcp2

        set pack1 [$tcp1 set t_seqno_]
        set tp1 [expr pack1 * $PacketSize * 8.0 / $SimTime
          / 1000000]
```

```
    puts "flow [$tcp1 set fid_] throughput [format
      "%6.2f" $tp1]"

    set pack2 [$tcp2 set t_seqno_]
    set tp2 [expr pack2 * $PacketSize * 8.0 / $SimTime
      / 1000000]
    puts "flow [$tcp2 set fid_] throughput [format
      "%6.2f" $tp2]"

    puts "running NAM..."
    exec nam out_trace.nam &
    exit 0
}
```

4.6 Постороение графика средней длины очереди.

Эта процедура не используется в данной работе, но с помощью её можно визуально наблюдать за изменением размера очереди при эмуляции технологии RED. После вызова **init** нужно вызвать процедуру:

```
proc trace_red {} {
    global ns n3 n4

    set redq [[$ns link $n3 $n4] queue]
    set tchan_ [open all.q w]
    $redq trace curq_
    $redq trace ave_
    $redq attach $tchan_
}
```

И в процедуре **finish**:

```
close $tchan_
```

После чего можно построить график по данным, содержащимся в файле **all.q**.

4.7 Моделирование DropTail.

Для моделирования удобно задавать параметры из командной строки. Поэтому в качестве аргументов нашей программы мы будем задавать размер буфера и начальное значение генератора случайных чисел.

Задание технологии обслуживания очередей DropTail предельно просто:

```
set BuffSize [lindex $argv 0]
set seed [lindex $argv 1]
init $seed $BuffSize
topologic DropTail
$ns at $SimTime "get_queue; finish"
puts "Running Simulation $seed Buff=$BuffSize"
$ns run
```

4.8 Моделирование RED.

Примем, что в качестве второго аргумента у нас передаётся нижний порог. Верхний же порог установим в 3 раза больше нижнего. Кроме того, нужно задать параметр linterm_=1/Max_p.

```
set MinTh [lindex $argv 0]
set MaxTh [expr $MinTh * 3]
```

```
puts "MinTh=$MinTh  MaxTh=$MaxTh"
set seed [lindex $argv 1]

Queue/RED set linterm_ 50
Queue/RED set thresh $MinTh
Queue/RED set maxthresh_ $MaxTh
Queue/RED set limit_ $Limit

init $seed $Limit
topologic RED
$ns at $SimTime "get_queue; finish"
puts "Running Simulation $seed Buff=$BuffSize"
$ns run
```

5. Задания.

5.1 Анализ алгоритма Slow Start.

Это простейшее задание.

Между двумя машинами передаются данные (канал в 1 Mb, например) по алгоритму Slow Start, при этом в качестве параметра скрипта используется окно, объявляемое приемником.

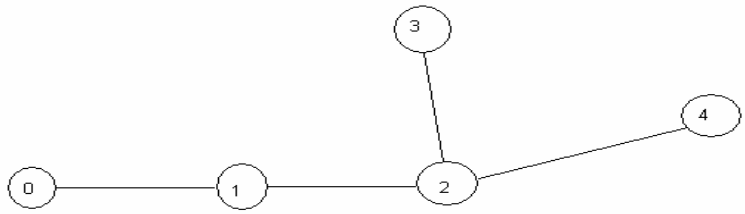
Агент должен быть настроен на алгоритм Slow Start: $tcp1 set add793slowstart_ true.

Увеличение числа посылаемых пакетов – нелинейное: $ns al 0.05 "ttcp0 set sstresh_1" (Вы увидите это по графику).

Полученный скрипт иллюстрирует механизмы Slow Start.

5.2 Анализ алгоритма Conquestion Avoidance.

Необходимо описать такую структуру сети, где все связи скоростью 1 Mb:



Параметры скрипта: размер стека на 2 машине и время проведения симуляции.

Данные постоянно передаются с 0 на 4 машину. А пакеты с третьей машины идут в интервале от 0.5 до 1.5 секунд. Увеличение числа посылаемых пакетов во всех случаях – экспоненциальное: $tcp1 set add793exponinc_ true.

Агенты должны быть настроены на алгоритм Conquestion Avoidance: $tcp1 set add793jacobsonrtt_ true.

Скрипт иллюстрирует работу Conquestion Avoidance в случае переполнения.

5.3 Анализ принципов работы алгоритма Slow Start, работающего с Conquestion Avoidance в паре.

В симуляции будет пять последовательно соединенных машин.

Скорости соединений соответственно: 10Mb, 256Kb, 1Mb, 256Mb, 10Mb.

В качестве параметров скрипта передаются размер стека у машины 1 и время проведения визуализации.

В самой программе с машины 1 на машину 5 происходит посылка данных агентом FTP. Наблюдая за процессом передачи данных, полученными

графиками, а также изменяя параметры, вы должны понять основные принципы взаимодействия реализуемых алгоритмов.

5.4 Исследование сетей с дисциплиной обслуживания RED

1. Написать скрипты, моделирующие работу технологий DropTail и RED.
2. Модифицировать полученные файлы так, чтобы по результатам их работы можно было построить графики *Mean Delay (BuffSize)* и *Total Troughput (BuffSize)*. Графики строить не менее чем по 100 точкам.
3. Замечание: при сопоставлении результатов для DropTail и RED обратить внимание на то, что значению BuffSize у DropTail соответствует значение MaxThresh у RED.
4. Построить график *Mean Delay (Total Troughput)*, где пропускная способность канала выражена в процентах от максимальной.
5. (Необязательно для выполнения) Модернизировать процедуру **`trace_red`** для построения графика величины очереди при моделировании DropTail.
6. Изменяя параметр **`q weight`** у **`Queue/RED`** построить графики зависимости пропускной способности и средней задержки от весового коэффициента.
7. (Необязательно для выполнения) Выяснить зависимость вышеуказаных величин от значения Max_p.

6. Литература

К части 1:
1. The *ns* Manual http://www.isi.edu/nsnam/ns/ns-documentation.html
2. Ns by Example http://nile.wpi.edu/NS/
3. Русскоязычный сайт по Ns2 http://yk.atm.tut.fi/ns2/

К части 2:
1. A Tour of TCP http://www.ce.chalmers.se/~fcela/tcp-tour.html
2. RFC 2001 http://rfc-editor.org