Network Working Group
Request for Comments: 1144

V. Jacobson[1]
LBL

February 1990

Сжатие заголовков TCP/IP для низкоскоростных последовательных каналов

Compressing TCP/IP Headers for Low-Speed Serial Links

Статус документа

В этом RFC предложен необязательный протокол для сообщества Internet и содержится приглашение к дискуссии в целях развития протокола. Документ описывает метод сжатия заголовков дейтаграмм TCP/IP для повышения производительности при работе по низкоскоростным последовательным каналам. Описана мотивация, реализация и производительность метода. Для справки приведен пример реализации на языке C. Документ может распространяться свободно[2].

Оглавление

1 Введение

По мере проникновения мощных компьютеров в домашнюю сферу растет потребность в подключении этих компьютеров к сети Internet. К сожалению такое расширение доступа в сеть связано с некоторыми сложными проблемами кадрирования на канальном уровне, выделения адресов, маршрутизации, аутентификации и производительности. На момент подготовки документа велись работы по всем этим направлениям. В данном документе описан метод, позволяющий повысить производительность работы TCP/IP по низкоскоростным (300 - 19 200 бит/с) последовательным каналам.

Предлагаемая здесь компрессия похожа на протокол Thinwire-II, описанный в работе [5]. Однако этот протокол обеспечивает более эффективное сжатие (средний сжатый заголовок имеет размер 3 байта по сравнению с 13 в Thinwire-II), а также более прост в реализации (в Unix-системах достаточно 250 строк на языке C, а среднее время компрессии или декомпрессии одного пакета составляет 90 мксек (170 инструкций) на процессоре MC68020 с тактовой частотой 20 МГц).

Эта компрессия предназначена для дейтаграмм TCP/IP[3]. Автор исследовал компрессию дейтаграмм UDP/IP и обнаружил, что их доля слишком мала и поток дейтаграмм не обеспечивает когерентности, требуемой для эффективного сжатия (например, запросы к серверу имен) или заголовки протоколов вышележащих уровней

[1]Эта работа частично финансировалась Министерством энергетики США в рамках контракта DE-AC03-76SF00098.
[2]Документ доступен в форматах ASCII и PostScript. В текстовой версии отсутствуют рисунки и не используется шрифторое выделение (курсив, полужирный и т. п.). Поскольку эта информация по мнению автора является существенной частью документа, текстовый вариант является в лучшем случае неполным, а в худшем может ввести в заблуждение. Тем, кто планирует работать с этим протоколом настоятельно рекомендуется использовать документ в формате Postscript.
[3]Связь с TCP здесь глубже, чем может казаться. В дополнении к сжатию известного формата заголовков TCP и IP некоторые особенности протокола TCP используются для упрощения протокола компрессии. В частности, гарантированная доставка TCP и потоковая модель обмена информацией позволяют избавиться от необходимости какой-либо диалоговой коррекции ошибок в протоколе сжатия (см. раздел 4).

значительно превышают по размеру заголовки UDP/IP (например, Sun RPC/NFS). Независимая компрессия заголовков IP и TCP также была исследована, но от нее пришлось отказаться, поскольку она увеличивает средний размер сжатого заголовка на 50% и удваивает размер программного кода для компрессии и декомпрессии.

2 Описание проблемы

Службы Internet, доступ к которым может осуществляться по последовательным каналам IP из дома, включают как интерактивные соединения терминального типа (например, telnet, rlogin, xterm), так и перенос больших объемов данных (например, ftp, smtp, nntp). Необходимость компрессии заголовков обусловлена желанием получать быстрый отклик в интерактивных сеансах. Т. е., эффективность линии пропорциональна отношению размера данных к суммарному размеру данных и заголовка. Если единственной задачей является перенос больших объемов данных, за счет увеличения размеров дейтаграмм эффективность можно повысить почти до 100%.

Исследования показали [15], что интерактивный отклик воспринимается плохо, если вывод «эхо-символов» происходит с задержкой более 100 - 200 мсек. Протокольные заголовки влияют на этот порог тремя путями:

(1) При очень медленной линии заголовок и данные могут не поместиться в окно продолжительностью 200 мсек — нажатие одной клавиши приводит к передаче пакета TCP/IP размером 41 байт и приему пакета с эхо-символом такого же размера. Для передачи 82 байтов в течение 200 мсек скорость в линии должна быть не меньше 4000 бит/с.

(2) Даже на достаточно быстрых линиях (не менее 4800 бит/с), когда скорости достаточно для интерактивных сессий, могут возникать нежелательные конфликты между трафиком передачи данных и интерактивными сессиями. Для эффективного использования линии размер данных в пакетах должен в 10 — 20 раз превышать размер заголовка. Для 40-байтовых заголовков TCP/IP размер максимального передаваемого в линию блока данных (MTU[2]) следует устанавливать в интервале от 500 до 1000 байтов. Даже при установке более высокого приоритета для интерактивного трафика пакеты telnet будут вынуждены ждать завершения передачи больших пакетов с данными. В предположении односторонней передачи данных среднее время ожидания составит половину времени передачи блока данных размером MTU, что при скорости 9600 бит/с и MTU=1024 будет приводить к средней задержке в 500 мсек.

(3) Для любой коммуникационной среды существует предельная скорость сигнала - предел Шеннона. Исследования AT&T [2] показали, что для типичной телефонной линии предел Шеннона составляет приблизительно 22 000 бит/с. При работе в полнодуплексном режиме модем 9 600 бит/с занимает 80% доступной полосы и разработчики модемов начали предлагать асимметричные решения для повышения эффективности использования полосы. Поскольку одинаковое количество данных в обоих направлениях передается весьма редко, можно дать для одного из направлений скорость больше 11 000 бит/с за счет мультиплексирования с разделением по времени в полудуплексном режиме (например, Telebit Trailblazer) или организации низкоскоростного «обратного канала» (например, USR Courier HST)[3]. В обоих случаях модем пытается динамически предсказать, какой из сторон требуется более высокая скорость передачи, предполагая, что на одной стороне находится человек (потребность в передаче ограничена типичной скоростью набора символов - 300 бит/с). Учет 40-байтовых заголовков делает такое распределение полосы эвристичным.

Из сказанного выше очевидно, что одной из целей компрессии является ограничение полосы, требуемой для передачи отдельных символов в терминальных сессиях, до значения не более 300 бит/с. Типовое значение максимальной скорости набора составляет примерно пять символов в секунду[4], что оставляет еще 25 символов (30 - 5) на заголовки (5 байтов заголовка на один передаваемый символ). 5-байтовые заголовки решают проблемы (1) и (3) напрямую, а проблему (2) - опосредованно. При размере пакетов 100-200 байтов заголовки легко амортизируются с обеспечением эффективности использования полосы линии 95-98%. При таком размере пакетов достигается разумный компромисс между передачей больших объемов данных и интерактивными сессиями (см. параграф 5.2).

Другой аспект разработки связан с тем, что протокол компрессии может основываться только на информации, гарантированно известной на обеих сторонах последовательного канала. Рассмотрим топологию, показанную на рисунке 1, где взаимодействующие хосты A и B находятся в разных локальных сетях (сплошные линии), соединенных между собой двумя последовательными каналами[5] (тонкие двойные линии между шлюзами C и D, E и F). Одним из вариантов компрессии является преобразования сессии TCP/IP в семантически эквивалентную сессию протокола с меньшим объемом заголовков (например, X.25). Однако в результате транзитивности маршрутизации при наличии множества путей возможно, что часть трафика от A к B пойдет по пути A-C-D-B, а другая часть - по пути A-E-F-B. Возможно, что трафик от A к B пойдет по пути A-C-D-B, а трафик от B к A - по пути B-F-E-A. Ни один из шлюзов не может рассчитывать на то, что он увидит все пакеты сеанса TCP и алгоритм компрессии для работы в такой топологии не может быть привязан к синтаксису соединения TCP.

Физический канал рассматриваемый, как два независимых односторонних соединения (по одному для каждого направления), вносит минимальные требования к топологии, маршрутизации и конвейерам. Стороны каждого одностороннего соединения имеют договоренность лишь о последнем переданном в канал пакете (пакетах). Таким образом, хотя любая схема компрессии включает общее состояние, это состояние локализовано в пространстве и времени в соответствии с принципом общей судьбы Дэйва Кларка (Dave Clark) [4]: «Две стороны не могут согласовать

---

[2]Maximum transmission unit.

[3]Хорошее обсуждение возможностей 2-проводных телефонных линий приведено в главе 11 работы [1]. В частности, здесь рассеяно широко распространенное заблуждение о возможностях модемов с подавлением «эхо-символов» (CCITT V.32). Подавление эхо может обеспечить каждой стороне 2-проводной линии полную пропускную способность, но обеспечивает полной скорости линии, поскольку сигнал удаленной стороны добавляется к локальному «шуму». Предел Шеннона в 22 кбит/с обусловлен физическими параметрами 2-проводных телефонных линий.

[4]См. работу [13]. Ускорение набора или многосимвольные последовательности (например, при нажатии на клавиши управления курсором) могут увеличивать скорость в 2-4 раза по сравнению со средней. Однако потребность в полосе остается примерно постоянной, поскольку алгоритм TCP Nagle [8] агрегирует трафик с интервалом поступления <200 мсек.

[5]Отметим, что, несмотря на то, что конечными точками TCP являются хосты A и B, в этом примере компрессия и декомпрессия должны выполняться на шлюзах, соединенных последовательными каналами (т. е., между C и D, E и F). Поскольку A и B используют протокол IP, они не могут знать об использовании низкоскоростного канала на пути передачи данных. Очевидно, что компрессия не должна нарушать модель IP и функции сжатия должны выполняться на промежуточных системах, а не в конечных точках.

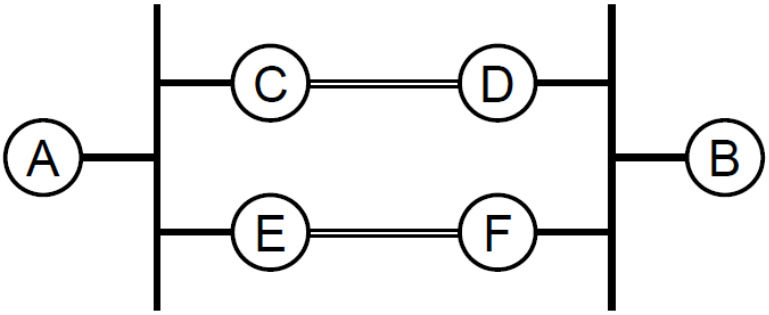

Рисунок 1. Топология, не дающая полной информации на шлюзах.

состояние лишь в случае неработоспособности канала, но в этом случае отсутствие согласования не имеет никакого значения».

3 Алгоритм компрессии

3.1 Основная идея

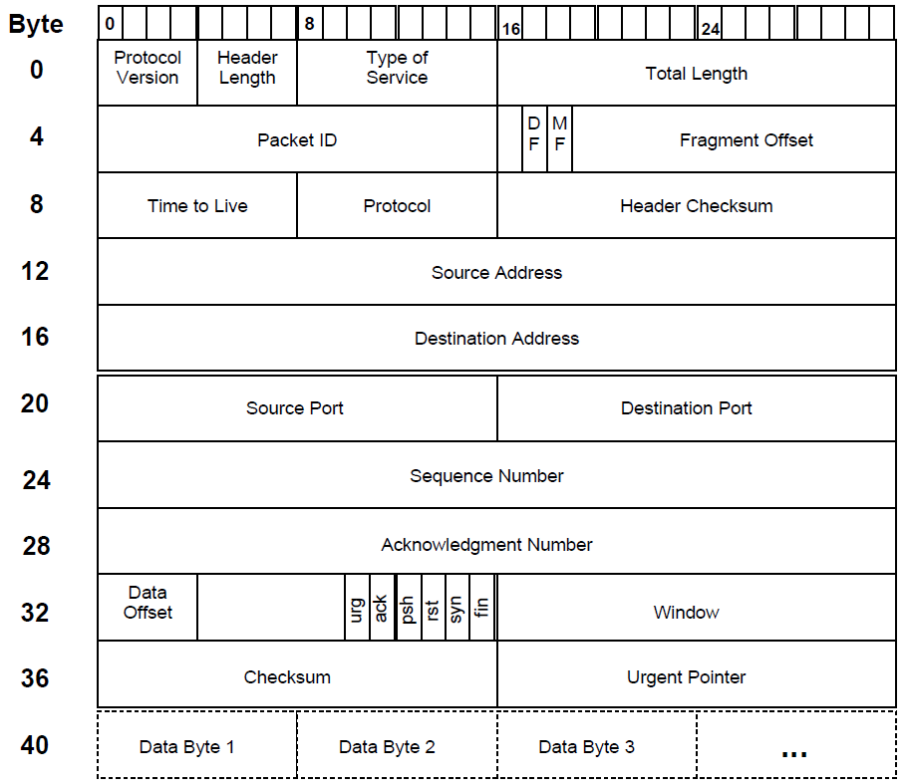

Рисунок 2. Заголовок дейтаграммы TCP/IP.

На рисунке 2 показан заголовок (минимального размера) типичной дейтаграммы TCP/IP[1]. Размер заголовка составляет 40 байтов - 20 байтов в заголовке IP и 20 - TCP. К сожалению, поскольку протоколы TCP и IP не разрабатывались совместно, все эти поля служат для решения некоторых задач и нет возможности просто опустить некоторые поля для повышения эффективности.

Однако через каждое соединение TCP обычно передаются десятки или сотни пакетов. Как много информации из каждого пакета не меняется на протяжении всего срока действия соединения? Половина (см. затененные поля на рисунке 3). Поэтому, если и отправитель и получатель отслеживают активные соединения[2] а получатель хранит копию заголовка последнего пакета из каждого соединения, отправитель может сжать заголовки вдвое, передавая лишь компактный (<= 8 битов) идентификатор соединения и 20 изменяющихся байтов, а отправитель может восстановить 20 неизменных байтов из сохраненного заголовка.

Можно сэкономить еще несколько байтов, поскольку любой разумный протокол кадрирования на канальном уровне будет говорить получателю размер принятого пакета и поле общего размера в заголовке (байты 2 и 3) является избыточным. Но в таком случае для передачи, по существу, остается лишь поле контрольной суммы заголовка IP[3] (байты 10 и 11), которое предотвращает обработку поврежденных заголовков IP на промежуточных узлах. Однако защищать то, что не передается, нет никакого смысла. Получатель может проверить контрольную сумму заголовка для реально переданного пакета (несжатой дейтаграммы), но для сжатых дейтаграмм значение поля контрольной суммы восстанавливается вместе с регенерацией остальных полей заголовка IP[4].

[1]Протоколы TCP и IP, а также их заголовки описаны в документах [10] и [11].
[2]96-битовый квартет <src address, dst address, src port, dst port> обеспечивает уникальную идентификацию соединения TCP.
[3]Остается еще поле идентификатора пакета. *Прим. перев.*
[4]Контрольная сумма заголовка IP не является «сквозной» контрольной суммой [14]. Обновление поля Time to Live заставляет пересчитывать значение контрольной суммы IP на каждом этапе пересылки. Автор получил негативный опыт, связанный с нарушением сквозной передачи аргумента в [14], и этот протокол осторожно относится к сквозной передаче контрольной суммы TCP,

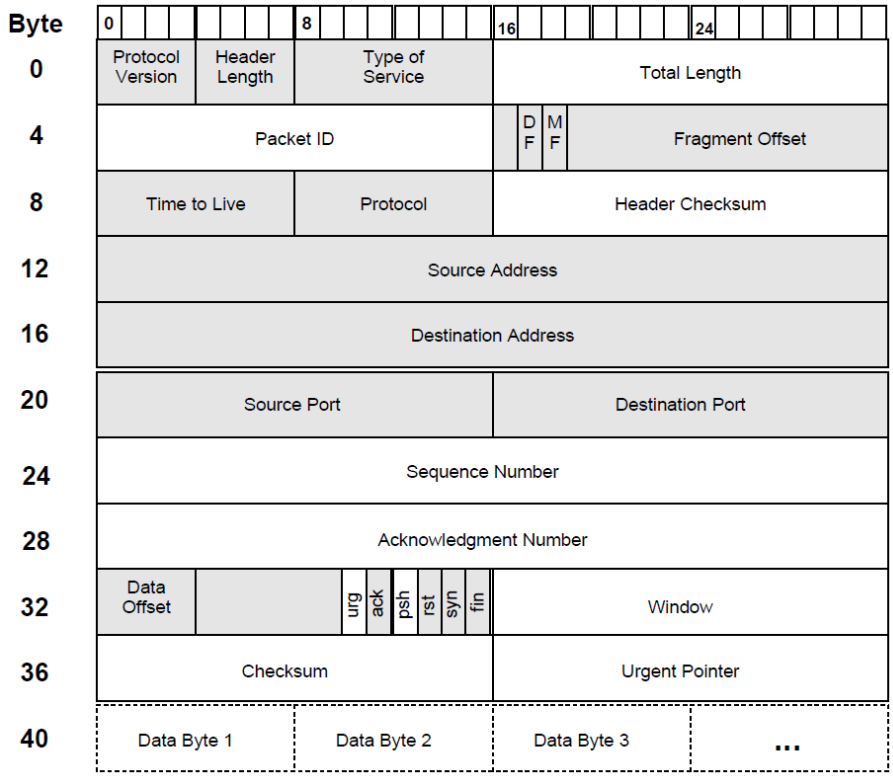

Рисунок 3. Поля с неизменными в масштабе соединения значениями.

А результате для передачи остается 16 байтов из двух заголовков. Все эти байты меняются в течение срока жизни соединения, но не все они изменяются одновременно. Скажем, при передаче файла по протоколу FTP в направлении от отправителя к получателю изменяются только поля Packet ID, Sequence Number и Checksum, а в обратном - только Packet ID, Ack, Checksum и, возможно, Window. Имея копию последнего пакета, переданного в каждое соединение, отправитель может обнаружить изменения в текущем пакете, после чего передать битовую маску, показывающую эти изменения , и сами измененные поля[1].

Если отправитель передает только меняющиеся поля, эта схема позволяет сжать заголовки в среднем до 10 байтов. Однако имеет смысл разобраться с изменениями полей. Значение Packet ID обычно берется из счетчика, который инкрементируется при каждой передаче пакета. Т. е., различие этого поля в предыдущем и текущем пакете будет выражаться небольшим положительным целым число (обычно, <256 - один байт), а зачастую будет равно 1. При передаче данных со стороны отправителя значение порядкового номера в текущем пакете будет превышать номер в предыдущем пакете на количество данных в предыдущем пакете (предполагается, что пакеты приходят по порядку). Поскольку размер пакетов IP не будет превышать 64K, изменение порядкового номера должно быть меньше 2^{16} (два байта). Таким образом, за счет передачи изменений вместо значений поле можно сэкономить еще 3-4 байта на каждый пакет.

Это ведет нас к заголовку размером 5 байтов. С учетом специальной обработки для двух наиболее типовых ситуация (интерактивные сессии и передача больших объемов данных) можно уменьшить размер заголовков до 4 байтов, но базовой схемой сжатия является дифференциальное кодирование, описанное выше. С учетом того, что приведенные выше умозаключения позволяют сократить размер заголовков до 5 байтов, представляется разумным рассмотреть опущенные детали и попытаться реализовать это на практике.

3.2 Детальное рассмотрение

3.2.1 Обзор

На рисунке 4 показана блок-схема программы сжатия. Сетевая система вызывает выходной драйвер SLIP, передавая ему пакет IP для передачи через последовательный канал. Пакет проходит через компрессор, который проверяет его принадлежность к протоколу TCP. Пакеты, не относящиеся к TCP, и «несжимаемые» пакеты TCP (см. ниже) просто маркируются, как TYPE_IP и передаются модулю кадрирования. Сжимаемые пакеты TCP отыскиваются в массиве заголовков пакетов. Если найдено соответствующее соединение, входящий пакет сжимается (насжатый) заголовок пакета копируется в массив, а пакет типа COMPRESSED_TCP передается в модуль кадрирования. Если соответствий не обнаружено, самая старая запись удаляется из массива, вместо нее копируется заголовок данного пакета, а пакет с типом UNCOMPRESSED_TCP[2] передается в модуль кадрирования. Таким способом отправитель (ре)синхронизируется с получателем по первому несжатому пакету сжимаемой последовательности пакетов.

которая не меняется при передаче (см. раздел 4).

[1]Это похоже на Thinwire-I из работы [5]. Незначительным отличием является дельта-кодирование, когда отправитель «вычитает» предыдущий пакет из текущего (трактуя пакеты, как массивы 16-битовых целых чисел) и передает 20-битовую маску, указывающую различия, за которой передаются эти различия. Если разные сессии отделены, это обеспечивает эффективную компрессию (размер типичного заголовка составляет 12-16 байтов), в которой компрессору не требуется информации о структуре пакета. Варианты такой схемы успешно применяются в течение многих лет (например, в протоколе Proteon [3] для последовательных каналов на маршрутизаторах).

[2]Пакет UNCOMPRESSED_TCP идентичен исходному пакету IP, но значение поля протокола IP заменяется номером соединения (индекс массива сохраненных заголовков).

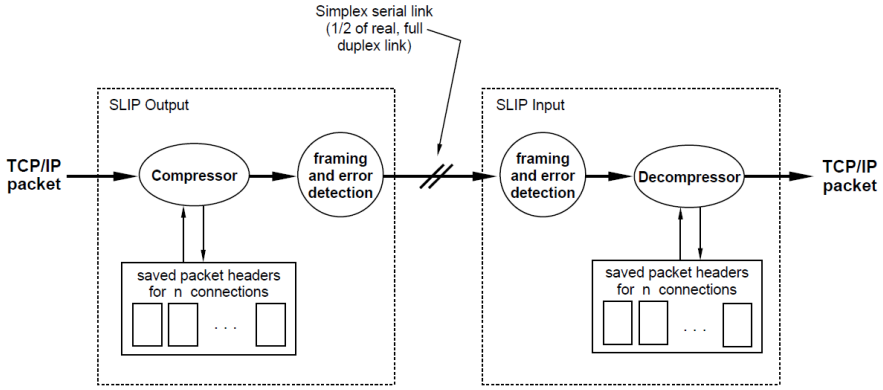

Рисунок 4. Модель компрессии-декомпрессии

Модуль кадрирования отвечает за обмен пакетными данными, тип и границы (чтобы декомпрессор мог знать, сколько байтов отправил компрессор). Поскольку в компрессии используется дифференциальное кодирование, для модуля кадрирования недопустимо изменение порядка пакетов (при работе через один последовательный канал для нарушения порядка нет причин). Он должен также обеспечить качественное детектирование ошибок и, если сжимаются номера соединений, индикацию ошибок для декомпрессора (см. раздел 4)[1].

Декомпрессор не меняет тип входящих пакетов. Пакета типа TYPE_IP он пропускает без обработки. Для пакетов UNCOMPRESSED_TCP из поля протокола IP извлекается порядковый номер соединения и восстанавливается значение IPPROTO_TCP, после чего номер соединения используется в качестве индекса для массива сохраненных получателем заголовков TCP/IP и заголовок входящего пакета копируется в массив. В пакетах типа COMPRESSED_TCP номер соединения служит в качестве индекса массива для получения заголовка TCP/IP предыдущего пакета из этого соединения. Информация из сжатого пакета используется для обновления найденного элемента массива и создается новый пакет, содержащий заголовок, выбранный из массива и «объединенный» с заголовком сжатого пакета.

Отметим, что коммуникации происходят в симплексном режиме - данные от декомпрессора к компрессору не передаются. Это означает, что декомпрессор при корректировке состояния в случае ошибок в линии опирается на повторную передачу TCP (см. раздел 4).

3.2.2 Формат сжатого пакета

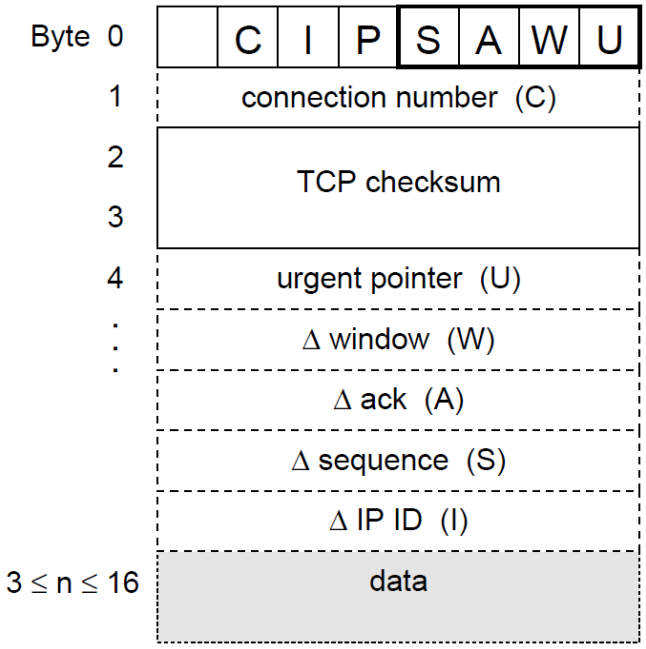

Рисунок 5. Заголовок сжатой дейтаграммы TCP/IP.

На рисунке 5 показан формат сжатого пакета TCP/IP. Здесь присутствует маска изменений[2], показывающая, какие поля, предполагающиеся переменными, изменены в действительности, номер соединения, чтобы получатель мог найти сохраненную копию последнего заголовка для данного соединения TCP, исходное значение контрольной суммы TCP для сквозной защиты целостности и для каждого установленного в маске изменений бита - величина соответствующего изменения. Необязательные поля, контролируемые маской, на рисунке обведены пунктиром. При установленном бите маски соответствующее поле присутствует в сжатом заголовке, при сброшенном - отсутствует[3].

Поскольку изменения (Δ) порядкового номера и др. полей обычно невелики (в частности, при выполнении приведенных в разделе 5 рекомендаций), все их значения представляются в формате переменного размера, хотя на практике обычно достаточно 8 битов. Изменение на величину до 255 кодируется одним байтом. Нулевое значение в поле

[1]Кадрирование на канальном уровне выходит за пределы данного документа. Для протокола компрессии подойдет любое кадрирование, имеющее перечисленные здесь свойства. Однако автор рекомендует потенциальным разработчикам обратить внимание на кадрирование SLIP, предложенное в качестве стандарта [9].

[2]Байт 0. *Прим. перев.*

[3]Бит P в показанной на рисунке маске отличается от остальных - он содержит копию флага PUSH из заголовка TCP. Флаг PUSH является курьезным анахронизмом, за сохранение которого выступают некоторые члены сообщества Internet. Поскольку флаг PUSH может меняться (и меняется) в любой дейтаграмме, при компрессии он передается в явном виде.

изменений никогда не передается, поэтому 0 служит в качестве сигнала о том, что изменение в данном случае велико и передается в двух следующих байтах[1] (сначала старший, затем младший). Изменение на 15 представляется, как 0f, на 255 - ff, на 65534 - 00 ff fe, а 0 - как 00 00 00. Такая схема обеспечивает достаточно эффективное кодирование и декодирование. Обычно обе процедуры выполняются за три машинных команды процессора MC680x0.

Значения, передаваемые в качестве порядковых номеров и номеров подтверждений TCP, различаются[2] в текущем и предыдущем пакете (если разница отрицательна или превышает 64K, пакет передается без компрессии). Значения окна также будут различаться для текущего и предшествующего пакетов. Однако разница между размерами окон может быть как положительной, так и отрицательной, поскольку поле размера окна имеет размер 16 битов. Указатель срочных данных передается, если установлен флаг URG (если указатель срочных данных изменился, а флаг URG не установлен, пакет передается без сжатия). Для идентификаторов пакетов значения передаются в форме разницы между текущим и предыдущим значением. Однако, в отличие от остальных сжимаемых полей, при сброшенном флаге I в маске компрессии предполагается, что идентификатор изменился на 1, а не остался неизменным.

Имеется два важных особых случая:

(1) порядковые номера и номера подтверждений изменяются на значение размера данных а последнем пакете; размер окна и URG не меняются;

(2) порядковый номер изменяется на значение размера данных а последнем пакете, номер подтверждения, размер окна и URG не изменяются.

Случай (1) относится к терминальному трафику с включенным эхо-выводом, (2) - к терминальному трафику без эхо и односторонней передаче данных. Некоторые комбинации битов S, A, W и U в маске изменений служат для индикации этих специальных случаев. Флаг U (срочные данные) устанавливается редко, поэтому в качестве таких комбинаций выбраны S W U (случай 1) и S A W U (случай 2). Для предотвращения путаницы в случае реального изменения типа S * W U, пакет передается без сжатия.

Поскольку состояние активности соединений меняется достаточно редко (например, пользователь обычно будет работать не менее нескольких минут в окне telnet прежде, чем перейдет в другой окно), бит C позволяет опускать номера соединений. Если этот бито сброшен, предполагается, что пакет относится к тому же соединению, что и последний (сжатый или несжатый) пакет. Если бит C установлен, номер соединения передается в байте, следующем сразу за маской изменений[3].

Из сказанного выше представляется очевидным, что сжатый терминальный трафик обычно выглядит примерно так (в шестнадцатеричном представлении: 0B c c d, где 0B указывает на случай (1), c c - два байта контрольной суммы TCP, а d - введенный символ. Команды редактора vi или emacs, а также команды FTP в направлении переноса данных put или get имеют вид 0F c c d ... , а запросы просмотра в FTP - 04 c c a, где a задает размер запрошенных данных[4].

3.2.3 Обработка в компрессоре

Модуль компрессии вызывается с пакетом IP для обработки и структурой состояния компрессии для выходной последовательной линии. На выходе модуля возвращается пакет, готовый к окончательному кадрированию и «тип» этого пакета для канального уровня.

Как отмечено в предыдущем разделе, компрессор преобразует каждый входящий пакет в TYPE_IP, UNCOMPRESSED_TCP или COMPRESSED_TCP. Пакет TYPE_IP представляет собой не измененную копию[5] исходного пакета и его обработка никак не меняет состояния компрессора.

Пакет UNCOMPRESSED_TCP отличается от исходного пакета лишь тем, что в поле протокола IP (байт 9) вместо числа 6 (протокол TCP) указывается номер соединения. В дополнение к этому массив состояний, связанный с этим соединением, обновляется копией заголовков IP и TCP из входного пакета, а номер соединения записывается в качестве последнего, чей пакет был передан в линию (для описанной в Приложении C сильной компрессии).

Пакет COMPRESSED_TCP содержит данные (если они есть) из исходного пакета, но заголовки IP и TCP полностью заменяются сжатым заголовком. Массив состояний для соединения и данные о номере соединения обновляются, как для типа UNCOMPRESSED_TCP.

Процедура выбора типа в компрессоре включает следующие операции:

- если пакет не относится к протоколу TCP, он передается, как TYPE_IP;
- если пакет является фрагментом IP (т. е., поле смещения фрагмента отлично от 0 или установлен флаг наличия других фрагментов), он передается, как TYPE_IP[6];

---

[1]Если изменение невозможно указать 16 битовым значением, это приводит к тому, что пакет передается без компрессии.

[2]Разница вычисляется с использованием арифметики дополнения до 2.

[3]Поле номера соединения имеет размер 1 байт, т. е., число активных соединений TCP не может превышать 256. Почти за два года работы автору не доводилось наблюдать ситуаций, когда одновременно реально использовалось более 16 соединений (даже в тех случаях, когда канал SLIP был использован в качестве шлюза за сильно загруженным 64-портовым терминальным мультиплексором). Таким образом, отмеченное ограничение не представляется существенным и позволяет использовать поле протокола в пакетах UNCOMPRESSED_TCP использовать для передачи номеров соединений, что упрощает обработку таких пакетов.

[4]Очевидно также, что маска изменений меняется достаточно редко и зачастую ее можно опустить. Фактически, можно добиться несколько большего эффекта, сохраняя последний сжатый пакет (он не будет превышать 16 байтов, поэтому издержки на хранение не будут велики) и проверяя наличие каких-либо отличий (кроме контрольной суммы TCP). При отсутствии отличий, передается типа пакета, означающий «сжатый пакет TCP, не отличающийся от последнего» и сам пакет, содержащий только контрольную сумму и данные. Однако такое решение увеличивает сложность при сравнительно небольшом (не более 25%) повышении эффективности (см. Приложение C).

[5]Не обязательно реально дублировать исходный пакет любого из этих трех типов. Отметим, что компрессор не может увеличить размер дейтаграммы. Как видно из кода, приведенного а Приложении A, протокол можно реализовать так, что все изменения заголовка будут осуществляться «на месте».

[6]Заголовок TCP содержится только в первом фрагменте, поэтому требуется проверка поля смещения. Первый фрагмент может содержать полный заголовок TCP и, следовательно, может сжиматься. Однако проверка полноты заголовка TCP увеличивает сложность и размер кода, поэтому с учетом аргументов, приведенных в работе [6], представляется разумным передавать все фрагменты IP без сжатия.

- если установлен любой из битов управления TCP (SYN, FIN или RST) или сброшен бит ACK, пакет считается несжимаемым и передается с типом TYPE_IP[1].

Если пакет не относится к одному из трех перечисленных выше вариантов, он будет передан с типом UNCOMPRESSED_TCP или COMPRESSED_TCP.

- Если не обнаружено состояния соединения, соответствующего IP-адресам и портам TCP отправителя и получателя, то или иное состояние (вероятно, использованное наиболее давно) заменяется новым, а пакет передается с типом UNCOMPRESSED_TCP.

- Если найдено соответствующее пакету состояние, содержащийся в нем заголовок сравнивается с заголовком текущего пакета для обнаружения неожиданных изменений (например, проверяется, что все измененные поля соответствуют отмеченным на рисунке 3). Поля протокола IP, смещения фрагмента, наличия других фрагментов, SYN, FIN и RST уже были проверены, а адреса и номера портов проверялись при поиске состояния. Поэтому проверяются оставшиеся поля версии протокола, размера заголовка, типа обслуживания, запрета фрагментирования, времени жизни, смещения данных, опций IP и TCP (при их наличии). Если любое из этих полей текущего заголовка отличается от соответствующего поля сохраненного заголовка, для пакета устанавливается тип UNCOMPRESSED_TCP.

Если все «неизменные» поля соответствуют, предпринимается попытка сжатия пакета.

- Если установлен флаг URG, кодируется поле срочных данных (отметим, что оно может иметь нулевое значение) и устанавливается бит U в маске изменений. К сожалению, при сброшенном флаге URG поле срочных данных все равно требуется сравнивать с предыдущим заголовком[2]; если поле изменилось, пакет передается, как UNCOMPRESSED_TCP.

- Определяется различие между текущим и сохраненным размером окна и при отличном от нуля значении разница кодируется с установкой флага W в маске изменений.

- Определяется различие между полями Ack. Если разница отрицательна или превышает 2^{16} - 1, пакет передается с типом UNCOMPRESSED_TCP[3]. В противном случае отличная от нуля разница кодируется с установкой флага A в маске изменений.

- Определяется различие между полями порядковых номеров. Если разница отрицательна или превышает 2^{16} - 1, пакет передается с типом UNCOMPRESSED_TCP[4]. В противном случае отличная от нуля разница кодируется с установкой флага S в маске изменений.

После определения изменений U, W, A и S может выполняться проверки необходимости специального кодирования.

- Если флаги U, S и W установлены, изменения соответствуют одному из специальных вариантов кодирования. Пакет передается с типом UNCOMPRESSED_TCP.

- Если установлен только флаг S, проверяется соответствие разницы размеру пользовательских данных в последнем пакете (сравнивается значение поля размера последнего пакета за вычетом размера заголовков TCP и IP с результатом изменения S). если значения совпадают, в маске изменений устанавливается значение SAWU (специальный случай «односторонней передачи данных») и отбрасывается кодирование изменения порядкового номера (декомпрессор может восстановить его, поскольку от знает значения полей общего размера и размеров заголовков последнего пакета).

- Если установлены только биты S и A, проверяется величина изменения для обоих и соответствие изменения размеру переданных пользовательских данных. При совпадении в маску изменений помещается значение SWU (интерактивный трафик с эхо-символами), а кодированные изменения отбрасываются.

- Если не изменилось ничего, проверяется отсутствие в пакете пользовательских данных (это может говорить о дубликате подтверждения или пробе размера окна) или их присутствие в предыдущем пакете (это говорит о повторе передачи в соединении без конвейеров). При положительном результате любой из проверок пакет передается, как UNCOMPRESSED_TCP.

В заключение заголовок TCP/IP исходящего пакета заменяется сжатым заголовком:

- рассчитывается изменение идентификатора пакета и, если оно отлично от 1[5], разница кодируется (отметим, что она может быть нулевой или отрицательной) и в маске изменений устанавливается бит I;

- если в исходной дейтаграмме установлен флаг PUSH, в маске изменений устанавливается бит P;

- заголовки TCP и IP из пакета копируются в массив состояний;

---

[1]Проверка ACK является избыточной, поскольку соответствующие стандарту реализации должны устанавливать флаг ACK во всех пакетах, за исключением начального пакета SYN. Однако эта проверка не вносит значительных издержек и предотвращает восприятие фиктивных пакетов, как корректных.
SYN-пакеты не сжимаются, поскольку лишь половина таких пакетов имеет корректное поле ACK и они обычно содержат опцию TCP (максимальный размер сегмента), которой нет в последующих пакетах. Таким образом, следующий пакет не будет сжиматься, поскольку размер заголовка TCP меняется и передача пакетов SYN с типом UNCOMPRESSED_TCP вместо TYPE_IP ничего не изменит. Сжатие пакетов FIN является спорным вопросом. В Приложении B.1 отмечено наличие в заголовке свободного бита, который можно использовать для передачи флага FIN. Однако, поскольку в соединении передается множество пакетов, не представляется разумным выделять отельный бит для флага, который появляется в единственном пакете соединения.
[2]Поле срочных данных не должно изменяться при сброшенном флаге URG, но спецификация TCP [11] не требует этого.
[3]Обе проверки можно выполнить в один прием, проверяя значение 16 старших битов разницы.
[4]Отрицательные изменения порядкового номера могут говорить о повторной передаче. Поскольку это может быть результатом отбрасывания пакета декомпрессором, передается несжатый пакет для ресинхронизации декомпрессора (см. раздел 4).
[5]Отметим, что здесь разница сравнивается с 1, а не с 0. Идентификатор пакета обычно инкрементируется на 1 в каждом передаваемом пакете, поэтому отсутствие изменения крайне маловероятно. Изменение на 1 является обычным - оно возникает всякий раз, когда в системе активно единственное соединение.

- заголовки TCP и IP удаляются и взамен устанавливается новый заголовок, включающий (в обратном порядке от начала):
  - аккумулированные и закодированные изменения;
  - контрольную сумму TCP (если новый заголовок создается «на месте», контрольная сумма может оказаться переписанной и ее значение следует брать из копии заголовка, сохраненной в массиве состояний, или иного места, в которое она была помещена до отбрасывания исходного заголовка);
  - номер соединения (если он отличается от номера в последнем, переданном через эту последовательную линию пакете); это означает также, что должен быть изменен номер последнего соединения, передавшего пакет в линию, и установлен бит C в маске изменений;
  - маска изменений.

После этого сжатый пакет TCP передается модулю кадрирования для отправки.

3.2.4 Обработка в декомпрессоре

В результате использования симплексной модели коммуникаций обработка в декомпрессоре значительно проще обработки в компрессоре. Все нужные решения уже приняты и декомпрессор просто делает то, что сказал ему компрессор.

Модуль декомпрессии вызывается с передачей ему входящего пакета[2], размера и типа этого пакета, а также структуры с состоянием компрессии для входящей последовательной линии. Модуль возвращает (возможно реконструированный) пакет IP.

Декомпрессор может получать четыре типа пакетов, три из которых генерируются компрессором, а к четвертому типу относятся псевдопакеты TYPE_ERROR, генерируемые приемным модулем кадрирования при обнаружении ошибок[3]. Первым этапом является выбор режима (переключение) по типу пакета:

- если тип пакета TYPE_ERROR или не распознан, устанавливается флаг состояния toss (отбрасывание) для того, чтобы пакеты COMPRESSED_TCP отбрасывались, пока не будет установлен бит C или получен пакет UNCOMPRESSED_TCP; в результате вызова не возвращается ничего (null-пакет);
- если пакет имеет тип TYPE_IP, возвращается копия этого пакета без изменений и состояние не изменяется;
- для пакетов типа UNCOMPRESSED_TCP проверяется индекс состояния из поля IP[4]; если индекс некорректен, устанавливается флаг toss и ничего не возвращается; в остальных случаях индекс копируется в поле последнего соединения, для которого был принят пакет, создается копия пакета[5], восстанавливается номер протокола TCP в поле IP, заголовок пакета копируется в заданный индексом элемент массива состояний и после этого возвращается копия пакета.

Если пакет не был обработан одним из перечисленных выше путей, он относится к типу COMPRESSED_TCP и для него создается новый заголовок TCP/IP по данным, содержащимся в пакете, и данным из массива состояний для заголовка последнего принятого пакета. Сначала используется заданный явно или неявно номер соединения для поиска данных в массиве состояний:

- если в маске изменений установлен бит C проверяется индекс состояния; при некорректном значении индекса устанавливается флаг toss и ничего не возвращается; в остальных случаях значение индекса устанавливается для полученного последним пакета, а флаг toss сбрасывается;
- если биты C и toss не установлены, пакет игнорируется и ничего не возвращается.

Номер соединения для которого был получен последний пакет является индексом массива состояний а первый байт (байты) принятого пакета (маска изменений и, возможно, индекс соединения) можно использовать для начала восстановления заголовка. Поскольку заголовок TCP/IP из массива состояний должен, в конечном итоге, соответствовать вновь принятому пакету, проще всего применить маску изменений из полученного пакета к этому заголовку и восстановить исходный пакет, объединяя полученный в результате заголовок с данными из принятого пакета. В приведенном далее описании слова «сохраненный заголовок» относятся к заголовку TCP/IP, хранящемуся в массиве состояний.

- Следующие два байта входящего пакета содержат контрольную сумму TCP и копируются в сохраненный заголовок.
- Если в маске изменений установлен бит P, в сохраненном заголовке устанавливается флаг TCP PUSH; в противном случае этот флаг сбрасывается.
- Если установлены четыре младших бита (S, A, W и U) маски изменений (специальный случай «однонаправленного потока данных»), определяется объем пользовательских данных в последнем пакете путем вычитания размера заголовков TCP и IP из общего размера пакета IP в сохраненном заголовке. Полученное значение добавляется к порядковому номеру в сохраненном заголовке.

---

[2]Предполагается, что кадрирование канального уровня уже удалено и пакет, а также поле его размера не включают байтов кадрирования.

[3]С пакетами TYPE_ERROR не связано никаких данных. Они нужны для того, чтобы приемный модуль кадрирования мог сказать декомпрессору о возможном пропуске в потоке данных. Декомпрессор использует такеи пакеты, как сигнал о необходимости отбрасывания пакетов до момента получения пакета с явным номером соединения (установлен бит C). Обоснование необходимости этого приведено в заключительной части параграфа 4.1.

[4]Индексы состояния используют соглашения языка C и принимают значения от 0 до N - 1, где 0 < N <= 256 - число доступных состояний.

[5]Как и для компрессора, код может быть структурирован таким образом, чтобы не нужно было делать копий, а все изменения выполнялись «на месте». Однако выходной пакет может быть больше входного, поэтому в начале буфера входящих пакетов следует оставлять 128 свободных байтов для размещения перед пакетом заголовка TCP/IP.

- Если биты S, W и U установлены, а бит A сброшен (специальный случай «терминального трафика»), определяется объем пользовательских данных в последнем пакете и полученное значение прибавляется к порядковому номеру TCP и значению поля номера подтверждения в сохраненном заголовке.

- В остальных случаях биты маски изменений интерпретируются индивидуально в порядке их установки компрессором:

  - Если установлен бит U, а также бит TCP URG в сохраненном заголовке, пакет декодируется и указывается в TCP Urgent Pointer. Если бит U не установлен, флаг TCP URG сбрасывается.

  - Если установлен бит W, следующий байт (байты) входящего пакета декодируется и полученное значение добавляется к полю размера окна TCP в сохраненном заголовке.

  - Если установлен бит A, следующий байт (байты) входящего пакета декодируется и полученное значение добавляется к значению поля TCP в сохраненном заголовке.

  - Если установлен бит S, следующий байт (байты) входящего пакета декодируется и полученное значение добавляется к значению поля порядкового номера TCP в сохраненном заголовке.

- Если в маске изменений установлен бит I, следующий байт (байты) входящего пакета декодируется и полученное значение добавляется к значению поля IP ID в сохраненном заголовке. В противном случае к значению IP ID добавляется 1.

На этом заканчивается работа с данными из заголовка входящего пакета и остаются только данные из него. Размер остающихся данных добавляется к размеру сохраненных заголовков IP и TCP, а результат помещается в поле размера IP сохраненного заголовка. Сохраненный заголовок IP обновлен, как подобает, поэтому для него рассчитывается контрольная сумма, которая записывается в поле IP checksum. После этого дейтаграмма собирается путем конкатенации сохраненного заголовка и оставшихся во входящем пакете данных. Восстановленная таким образом дейтаграмма возвращается модулем декомпрессии.

4 Обработка ошибок

4.1 Детектирование ошибок

Опыт авторов показывает, что соединения по коммутируемым телефонным линиям особенно подвержены ошибкам передачи данных. Это связано со сжатием в двух аспектах.

Во-первых, локальное влияние ошибок в сжатых пакетах. Все методы детектирования ошибок основаны на избыточности, а компрессия удаляет из заголовков TCP и IP почти всю избыточность. Иными словами, декомпрессор может принять случайные шумы в линии за корректный пакет TCP/IP[1]. Можно пытаться детектировать поврежденные пакеты по контрольным суммам TCP, но, к сожалению, далеко не все ошибки удастся обнаружить таким способом. Например, контрольная сумма TCP зачастую не позволит обнаружить две однобитовые ошибки, разделенные 16-битовым интервалом. Для модемной сигнализации V.32 при скорости 2400 бод с 4 битами на бод в любой линии с временем передачи больше 400 мксек будут повреждаться 16 битов. В работе [2] отмечено, что время передачи через абонентские телефонные линии может достигать 2 мсек.

Корректным способом решения этой проблемы является детектирование ошибок на уровне кадрирования. Поскольку кадрирование можно (по крайней мере, теоретически) адаптировать к конкретной линии, детектирование ошибок может быть облегченным или ресурсоемким в зависимости от того, что позволяет канал[2]. Если детектирование ошибок в пакетах выполняется на канальном уровне, декомпрессор просто полагает, что он будет получать информацию в случае наличия ошибки в принятом пакете (декомпрессор всегда игнорирует (отбрасывает) пакеты с ошибками, а индикация нужна для предотвращения распространения ошибок, как описано ниже).

оригинал	передано	принято	восстановлено
1: A	1: A	1: A	1: A
2: BC	1, BC	1, BC	2: BC
4: DE	2, DE	---	---
6: F	2, F	2, F	4: F
7: GH	1, GH	1, GH	5: GH

Правило «отбрасывать пакеты с ошибками» является другим аспектом взаимосвязи между компрессией и ошибками. Рассмотрим показанный на рисунке справа пример взаимодействия. Каждая запись в таблице имеет форму «начальный порядковый номер: отправленные данные» или «изменение порядкового номера, отправленные данные».

Сначала передается несжатый пакет, за которым следует четыре пакета с компрессией. В третьем пакете возникает ошибка и он отбрасывается. Для восстановления четвертого пакета получатель применяет изменение порядкового номера из входящего сжатого пакета к порядковому номеру полученного последним корректного пакета (пакет 2) и создает в результате некорректный порядковый номер для пакета 4. После ошибки порядковые номера во всех восстановленных пакетах будут иметь некорректные порядковые номера (со сдвигом на количество данных в потерянном пакете[3]).

Без той или иной проверки предшествующая ошибка будет приводить на приемной стороне к незаметной потере двух байтов из середины передачи (поскольку декомпрессор генерирует порядковые номера, пакеты, содержащие F и GH, прибудут в модуль TCP получателя с теми же порядковыми номерами, которые были бы у них, если бы пакета DE

[1]Совпадение контрольной суммы TCP с точностью до модуля.

[2]Хотя подходящее детектирование ошибок зависит от линии, CCITT CRC [9] обеспечивают хороший баланс между простотой расчета контрольных сумм и надежностью детектирования ошибок для широкого спектра каналов, включая случай относительно малых пакетов, требуемых для интерактивных сессий. Таким образом, для обеспечения совместимости следует использовать кадрирование [9], если нет веских причин для иного.

[3]Это является примером проблемы, возникающей при дифференциальном или дельта-кодировании, известной, как losing DC (потеря DC).

просто не существовало). Несмотря на то, что некоторые соединения TCP устойчивы к потере части данных[4], это не применимо на практике. К счастью, контрольная сумма TCP (поскольку она является простой суммой для содержимого пакетов с учетом порядковых номеров), позволяет обнаруживать все такие ошибки. Например, если на приемной стороне будет рассчитана контрольная сумма для двух последних пакетов описанных выше, она будет всегда отличаться от контрольной суммы пакета на 2.

К сожалению в двух случаях описанная выше защита контрольных сумм TCP может отказывать, если входящий сжатый пакет будет отнесен не к тому диалогу. Рассмотрим два активных диалога C1, C2 и случай, когда за пакетом из C1 следует два пакета из диалога C2. Поскольку номер соединения не меняется, он будет опущен для второго пакета C2. Но, если первый пакет C2 будет принят с ошибкой CRC, второй пакет C2 может быть ошибочно сочтен следующим пакетом C1. Поскольку контрольная сумма C2 является случайным значением по отношению к порядковым номерам C1, с вероятностью не ниже 2^{-16} такой пакет может быть воспринят TCP-получателем C1[2]. Для предотвращения этого после индикации ошибки CRC от модуля кадрирования получателю следует отбрасывать пакеты, пока не будет получен пакет COMPRESSED_TCP с установленным битом C или пакет UNCOMPRESSED_TCP (т. е., пакеты следует отбрасывать до явного получения номера соединения).

В заключение этого параграфа отметим, что существует два типа ошибок — повреждение пакетов и потеря синхронизации в рамках диалога. Первый тип детектируется декомпрессором, как ошибка CRC на канальном уровне, а второй детектируется получателем TCP по (гарантированно) некорректной контрольной сумме TCP. Комбинация этих двух независимых механизмов обеспечивает отбрасывание связанных с ошибками пакетов.

4.2 Восстановление после ошибок

В предыдущем параграфе было отмечено, что после ошибки CRC декомпрессор будет вносить ошибку контрольной суммы TCP с каждый декомпрессируемый пакет. Ошибка контрольной суммы не вызывает нарушения потока данных, однако диалог TCP становится бесполезным, пока декомпрессор не восстановит генерацию корректных пакетов. Как можно ускорить этот процесс?

Декомпрессор создает некорректные пакеты в результате того, что его состояние (сохраненный заголовок последнего пакета) не соответствует состоянию компрессора. Состояние декомпрессора заново синхронизируется пакетом UNCOMPRESSED_TCP. Таким образом, восстановление после ошибки может быть выполнено путем передачи несжатого пакета, когда декомпрессор предполагает отсутствие синхронизации состояний.

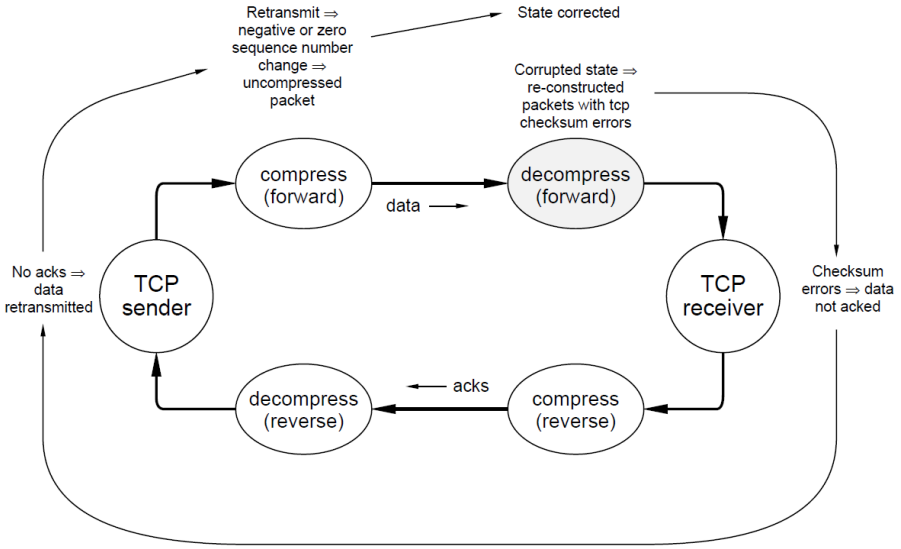

Рисунок 6. Коррекция ошибок на «прямом» пути.

На первый взгляд представляется разумным использование преимуществ полнодуплексного канала — декомпрессор может послать компрессору какой-либо сигнал, запрашивающий отправку несжатого пакета. Это совершенно не желательно, поскольку это ограничивает топологию сильнее, чем предложено в разделе 2, и требует внесения дополнительных функций в протокол для компрессора и декомпрессора. При более внимательном рассмотрении оказывается, что такой вариант просто не будет работать — сжатые пакеты достаточно малы и очевидно, что в результате занятия линии на передачу такого сигнала декомпрессор просто не получит ничего. Таким образом, пакеты будут восстанавливаться некорректно (по причине отсутствия сжатого пакета), но знать об этом будут только конечные точки TCP, а не декомпрессор.

Однако конечные точки TCP знают об ошибке, а протокол TCP обеспечивает гарантированную доставку через ненадежные среды. Это означает, что конечные точки должны будут со временем начать действия по восстановлению и этом будет служить компрессору сигналом для восстановления синхронизации с декомпрессором — передача несжатого пакета в процессе корректировки ошибок TCP.

Но как компрессор узнает о процедуре восстановления ошибок TCP? Рассмотрим схему обмена данными TCP на рисунке 6. «Обманутый» декомпрессор располагается в «прямом» (передача данных) направлении диалога TCP. Принимающий модуль TCP будет отбрасывать пакет вместо его подтверждения (по причине ошибок в контрольной сумме), а передающий модуль TCP через какое-то время повторит передачу пакета и компрессор на прямом пути обнаружит, что разность между порядковыми номерами в повторном и переданном последним пакете отрицательной (множества пакетов в пути) или нулевой (в пути один пакет). Первый случай обнаруживается при расчете порядковых номеров в процессе сжатия, второй — на этапе проверки «специального» кодирования и требует дополнительных действий. В диалогах достаточно часто происходит передача подтверждения без данных, за которым следует пакет

[4]Многие системные администраторы утверждают, что пропуски в потоке данных NNTP важнее, чем сами данные.
[2]При максимальной загрузке эта вероятность проявляется в форме одной пропущенной ошибки за 3 часа передачи по каналу 9600 бод с частотой ошибок 30%.

данных. Пакеты подтверждения и данных в этом случае будут содержать одинаковые порядковые номера, если пакет данных не является повтором. Для предотвращения необоснованной передачи несжатых пакетов следует проверять размер предыдущего пакета - наличие в пакете данных при отсутствии изменения порядкового номера говорит о

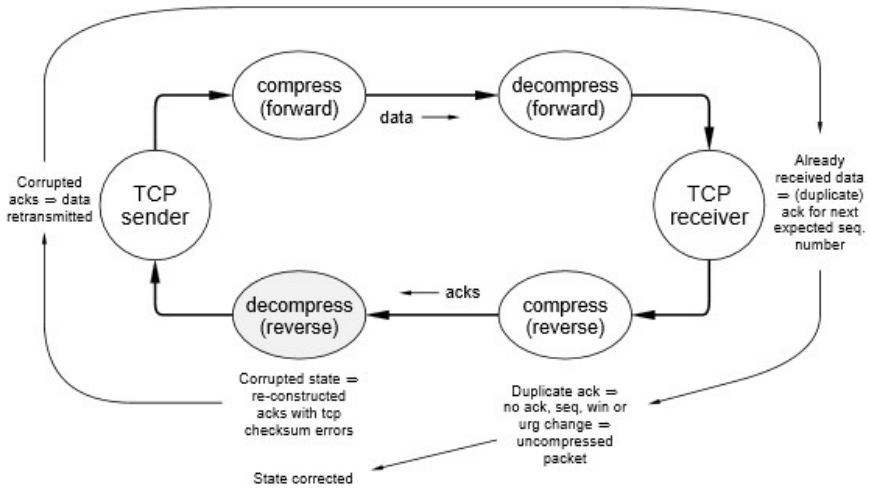

Рисунок 7. Коррекция ошибок на «обратном» пути

повторе передачи.

«Обманутый» декомпрессор обратного (ack) направления также может обнаружить ошибки (рисунок 7). Передающий модуль TCP отбрасывает подтверждения (ошибка в контрольной сумме) и по истечении некоторого времени повторит передачу. Принимающий модуль TCP в результате получит дубликат пакета и должен создать подтверждение для следующего ожидаемого порядкового номера [11, стр. 69]. Это подтверждение будет дубликатом последнего подтверждения, сгенерированного получателем, поэтому компрессор обратного пути увидит, что номер подтверждения, порядковый номер, размер окна и указатель срочности (urg) не изменились. Если это происходит для пакета, не содержащего данных, компрессор трактует его, как дубликат подтверждения в ответ на повтор передачи и генерирует пакет UNCOMPRESSED_TCP[1].

5 Параметры конфигурации и тонкая настройка

5.1 Настройка компрессии

С компрессией заголовков связаны два конфигурационных параметра. Один из параметров определяет, будут ли сжатые пакеты передаваться по конкретной линии, и (в случае положительного ответа) какое количество состояний (сохраненных заголовков пакетов) нужно резервировать. Есть также один параметр уровня коммуникационного канала — максимальный размер пакета или MTU и один конфигурационный параметр «прикладного уровня» - способ сжатия данных, который будет применяться для компрессии заголовков. С этом параграфе обсуждается конфигурация компрессии, как таковой. Вопросы MTU и способов сжатия данных рассматриваются в двух следующих параграфах.

Некоторые хосты (например, недорогие ПК) могут не иметь достаточных ресурсов процессора или памяти для реализации описываемой здесь компрессии. Имеются также каналы и приложения, для которых компрессия заголовков не требуется или не желательна. Существует также множество каналов SLIP, на которых в настоящее время не используется этот тип сжатия заголовков. В целях обеспечения совместимости драйверы последовательных линий IP, которые поддерживают компрессию заголовков, должны обеспечивать тот или иной способ отключения сжатия на уровне конфигурационных параметров (см. Приложение B.2)[2].

Если сжатие включено, компрессор должен предотвращать передачу идентификаторов соединений (connection id — индекс состояния), которые будут отбрасываться декомпрессором. Например, будет возникать «черная дыра», если декомпрессор имеет 16 гнезд для состояний, а компрессор станет использовать двадцать[3]. Если компрессор имеет слишком мало гнезд, распределитель LRU будет приводить к тому, что большая часть пакетов будет передаваться, как UNCOMPRESSED_TCP. Это приведет к существенному расходу гнезд состояний и памяти.

Экспериментируя с разными размерами в течение прошлого года, автор обнаружил, что 8 гнезд будет недостаточно (например, заметно снизится производительность), когда множество окон на многооконной рабочей станции используются одновременно или станция служит в качестве шлюза для трех и более машин. С 16 гнездами подобных проблем не наблюдалось никогда (это может быть связано с тем, что загрузке линии 9600 бит/с одновременно 16 «потоками» она становится настолько перегруженной, что дополнительное увеличение времени кругового обхода уже становится пренебрежимо малым).

---

[1]Пакет может быть пробой нулевого окна (zero-window probe), а не повтором подтверждения, однако такие пробные пакеты передаются достаточно редко и передача их без компрессии не нанесет вреда.

[2]Протокол PPP [9] позволяет конечным точкам согласовать компрессию, поэтому с ним не возникает проблем совместимости. Однако администратору системы на каждой стороне должна обеспечиваться возможность управления сжатием заголовков. Обычно по умолчанию сжатие отключено и включать его нужно после согласования.

[3]Строго говоря, не существует причин для трактовки идентификатора соединений, как индекса массива. Если состояния декомпрессора будут храниться в хэш-таблице или иной ассоциативной структуре, идентификатором соединения будет ключ, а не индекс и производительность при малом числе гнезд состояний декомпрессора будет лишь снижать производительность, не приводя к отказам. Однако ассоциативные структуры требуют более сложного кодирования и ведут к большему расходу памяти (128 байтов на гнездо), поэтому представляется разумным организовать на декомпрессоре массив гнезд и обеспечить тот или иной (возможно, неявный) способ уведомления о размере массива.

Каждое гнездо должно иметь размер, дастаточный для хранения максимального заголовка TCP/IP в 128 байтов[3], что требует выделения 2 Кбайт для 16 гнезд. При современных микросхемах памяти размером 4 Мбит расход 2 Кбайт памяти не представляется значительным и авторы рекомендуют следующие правила конфигурации:

(1) Если протокол кадрирования не поддерживает согласования, компрессору и декомпрессору следует поддерживать по 16 гнезд с номерами от 0 до 15.

(2) Если протокол кадрирования поддерживает согласование, можно выбирать любое приемлемое для обеих сторон число гнезд в диапазоне от 1 до 256[4]. Если число гнезд не согласуется или еще не согласовано, обе стороны должны предполагать 16 гнезд.

(3) Если имеется полный контроль за всеми машинами по обе стороны каждого канала и ни одна из этих машин не будет использоваться для взаимодействия с машинами, выходящими за пределы вашего контроля, вы будете свободны в выборе конфигурации и можете игнорировать приведенные выше соображения. Однако при выходе той или иной машины из под вашего контроля (весьма вероятном), будьте готовы услышать о себе много нового со стороны сообщества Internet, которое увидит некорректную конфигурацию ваших машин и не сможет с ними взаимодействовать.

5.2 Выбор MTU

Из обсуждения в разделе 2 представляется желательным ограничение максимального размера пакетов (MTU) на любой линии, где может быть интерактивный трафик и множество активных соединений (для обеспечения быстрого отклика интерактивных приложений в разных одновременных соединениях). Возникает вопрос, насколько это снижает пропускную способность? Не снижает совсем!

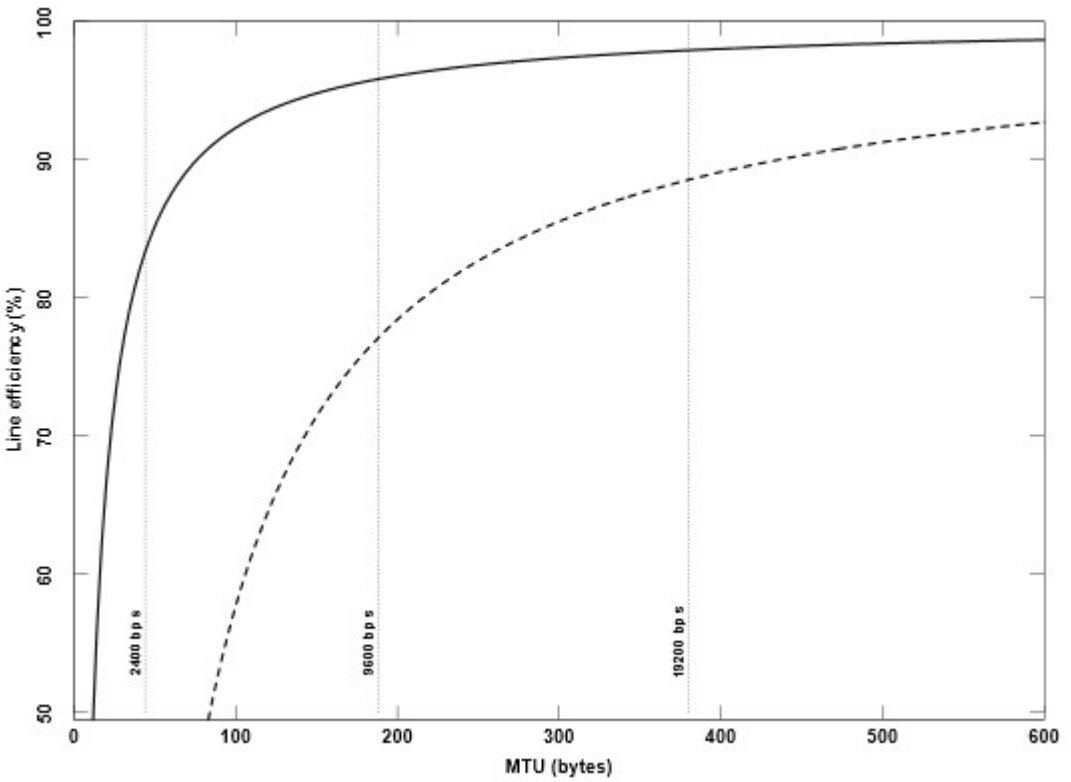

Рисунок 8. Зависимость эффективной пропускной способности от MTU

На рисунке 8 показано изменение пропускной способности[5] в зависимости от значений MTU при использовании компрессии заголовков (сплошная линия) и без нее (пунктир). Вертикальные линии из точек показывают значения MTU, соответствующие интервалу передачи пакета 200 мсек для скоростей 2400, 9600 и 19200 бит/с. Отметим, что при сжатии заголовков даже линия 2400 бит/с обеспечивает достаточно высокую эффективность (83%)[6].

[3]Максимальный размер заголовка, зафиксированный в стандартах протоколов, составляет 64 байта для IP и 64 байта для TCP.

[4]Разрешение использовать одно гнездо усложняет кодирование компрессора. Реализациям следует избегать такого случая и в реализациях компрессора может отключаться сжатие при наличии единственного гнезда.

[5]По вертикальной оси отложены проценты от скорости линии. Например, 95 означает, что 95% полосы доступны для передачи пользовательских данных (иными словами, пользователь будет видеть скорость 9120 бит/с при скорости в линии 9600 бит/с). 4 байта заголовка канального уровня (кадрирование) учтены вместе с размером сжатого или несжатого заголовка TCP/IP. Время передачи 200 мсек было выбрано в предположении асинхронной линии с использованием 10 битов на символ (8 битов данных, старт-бит, стоп-бит, без контроля четности).

[6]Тем не менее, 40-байтовый TCP MSS, требуемый для линии 2400 бит/с, может служить нагрузочным тестом вашей реализации TCP.

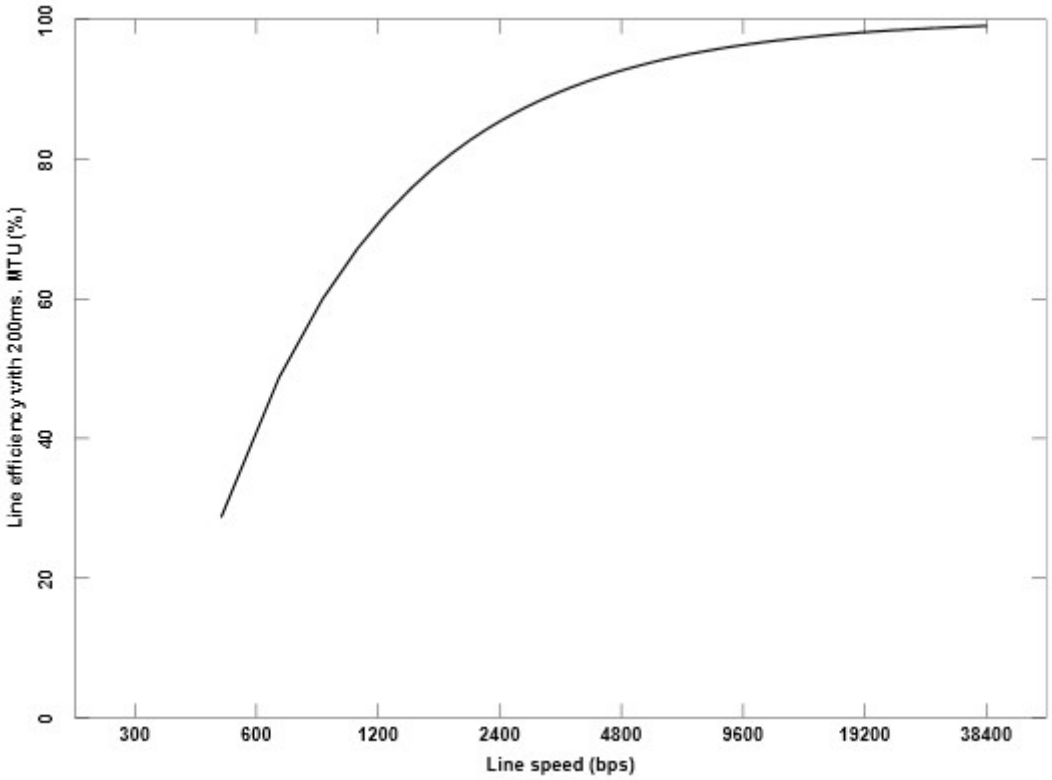

Рисунок 9. Зависимость эффективности от скорости линии.

На рисунке 9 показана зависимость эффективности линии от лоста скорости в линии в предположении, выбора MTU по времени передачи 200 мсек[1]. Наибольший изгиб кривой производительности наблюдается при скоростях около 2400 бит/с. При более низких скоростях эффективность (равно как и связанное с ней линейно значение MTU) заметно меняется со скоростью линии и высокая эффективность может достигаться только за счет увеличения времени отклика. При скоростях выше 2400 бит/с кривая становится менее наклонной и меньше зависит от скорости или MTU. Иными словами, возможно одновременное увеличение эффективности линии и снижение времени отклика.

В качестве иллюстрации отметим, что для линий 9600 бит/с с компрессией заголовков преимущества от увеличения MTU сверх 200 байтов не наблюдается. Если MTU увеличить до 576, средняя задержка возрастет на 188%, тогда как пропускная способность увеличится лишь на 3% (с 96 до 99%).

5.3 Взаимодействие с компрессией данных

С начала 1980-х стали доступны быстрые и эффективные алгоритмы сжатия данных (такие, как Lempel-Ziv[7]) и программы для их реализации в составе операционных систем (например, Berkeley Unix). При использовании низкоскоростных или протяженных линий сжатие данных до передачи стало обычной практикой. Для соединений по коммутируемым линиям такая компрессия зачастую выполняется в модемах независимо от обменивающихся данными хостов. В связи с этим возникает ряд интересных вопросов: (1) нужно ли отдельно сжимать заголовки при использовании компрессии данных? (2) как сжатие заголовков взаимодействует с компрессией данных? (3) данные следует сжимать до или после компрессии заголовков[2]?

Для решения вопроса (1) использовалась компрессия Lempel-Ziv для 446 пакетов TCP/IP типичного диалога telnet. Поскольку пакеты порождаются в результате ручного ввода символов, почти все они будут содержать лишь один байт данных и 40 байтов заголовка. Т. е., тест был по сути на компрессию заголовков TCP/IP. Коэффициент сжатия (отношение размера исходных и сжатых данных) составил 2,6. Иными словами, средний размер заголовка уменьшился с 40 до 16 байтов. Такое сжатие представляется достаточно эффективным, однако ему далеко до сжатия заголовка в 5 байтов, требуемого для быстрого интерактивного отклика, а тем более до 3 байтов (коэффициент сжатия 13,3), до которых удается сжать заголовок методами компрессии заголовков.

Рисунок 10 Варианты компрессии данных

Ответы на второй и третий вопросы несколько сложнее. Для их решения исследовалась передача файлов по протоколу FTP[3] с компрессией данных L-Z и без нее. Сжатие L-Z пытались применять в двух точках исходящего потока данных (Рисунок 10): (1) перед тем, как данные передавались модулю TCP для инкапсуляции (подобно компрессии на уровне приложения) и (2) после инкапсуляции данных (подобно компрессии в модеме). В таблице 1 приведены

[1]Для типичной асинхронной линии значение MTU для 200 мсек составляет 0,02 от скорости линии в бит/с.
[2]Для тех, кто не хочет читать остальную часть параграфа - «да», «нет», «не имеет значения», соответственно.
[3]Объем данных от пользователя telnet слишком мал для обретения преимуществ компрессии данных и, кроме того, компрессия данных неизбежно вносит ту или иную задержку. Статистика и объем данных клиентской стороны соединения FTP (ASCII) похожи на случай telnet, поэтому приведенные результаты применимы и для него.

результаты для передачи текстового (ASCII) файла размером 78 776 байтов (руководство (man) для Unix csh.1)[1] с использованием рекомендаций предыдущего параграфа (MTU 256 байтов или MSS 216 байтов; всего 368 пакетов). Показаны коэффициенты сжатия для следующих вариантов:

- файл данных (без компрессии и инкапсуляции)
- данные -> компрессор L-Z
- данные -> инкапсуляция TCP/IP
- данные -> L-Z -> TCP/IP
- данные -> TCP/IP -> L-Z
- данные -> L-Z -> TCP/IP -> L-Z
- данные -> TCP/IP -> компрессия заголовков
- данные -> L-Z -> TCP/IP -> компрессия заголовков
- данные -> TCP/IP -> компрессия заголовков -> L-Z
- данные -> L-Z -> TCP/IP -> компрессия заголовков -> L-Z

Таблица 1. Коэффициент сжатия текстовых файлов ASCII.

	Без сжатия данных	L-Z для данных	L-Z в линии	L-Z для обоих
Raw Data	1.00	2.44	-	-
+ TCP Encap.	0.83	2.03	1.97	1.58
w/Hdr Comp.	0.98	2.39	2.26	1.66

Первая колонка таблицы 1 показывает, что при отсутствии компрессии данные «расширяются на 19% («сжатие» с коэффициентом 0,83) при инкапсуляции TCP/IP и на 2% при инкапсуляции с компрессией заголовков TCP/IP[2]. Первая строка показывает достаточную эффективность компрессии L-Z для данных, обеспечивающую снижение размера более, чем вдвое. Колонка 4 показывает общеизвестный факт ошибочности представления о том, что L-Z сжимает уже сжатые данные. Интересна информация из колонок и строк с номерами 2 и 3. Эти данные показывают, что преимущества от компрессии данных превышают издержки на инкапсуляцию даже для стандартного TCP/IP. Они также говорят, что компрессия данных до инкапсуляции дает несколько лучший результат по сравнению с компрессией в модеме или модуле кадрирования. Различие невелико (3% и 66 для стандартного TCP/IP и сжатия заголовков[3]).

В таблице 2 показаны результаты таких же экспериментов для случая передачи двоичного файла размером 122880 байтов (программа ps в Sun-3). Хотя двоичные данные почти не сжимаются, результат качественно совпадает с результатом для файлов ASCII. Единственное значимое отличие показано в строке 2 — компрессия данных в модеме почти на 3% лучше компрессии у отправителя перед инкапсуляцией TCP/IP (двоичные файлы Sun и заголовки TCP/IP статистически похожи). Однако результаты для компрессии заголовков (строка 3) подобны результатам для файлов ASCII — компрессия в модеме примерно на 3% хуже, чем компрессия исходных данных[4].

Таблица 2. Коэффициент сжатия двоичных файлов.

	Без сжатия данных	L-Z для данных	L-Z в линии	L-Z для обоих
Raw Data	1.00	2.72	-	-
+ TCP Encap.	0.83	1.43	1.48	1.21
w/Hdr Comp.	0.98	1.69	1.64	1.28

6 Измерение производительности

Целью разработки программного кода для компрессии было получение достаточно простого средства для использования при скорости ISDN (64 Кбит/с) на типовой для 1989 года рабочей станции. При скорости 64 Кбит/с байт передается каждые 122 мксек, поэтому для времени сжатия/декомпрессии было (произвольно) выбрано значение 120 мксек[5].

Таблица 3. Время сжатия и декомпрессии.

Машина	Среднее время обработки пакета (мксек)	
	сжатие	декомпрессия
Sparcstation-1	24	18
Sun 4/260	46	20

[1]Выполнялось 10 экспериментов, в каждом из которых передавалось 10 файлов ASCII (4 больших сообщения электронной почты, три текста программ на языке Unix C и три документа Unix man). Результаты оказались похожими для разных типов файлов и приведенные ниже заключения применимы ко всем типам.

[2]Это получено из относительных размеров заголовков - 256/216 для инкапсуляции TCP/IP и 219/216 для компрессии.

[3]Различия обусловлены разными байтовыми картинами дейтаграмм TCP/IP и текста ASCII. Любая схема компрессии, основанная на модели Маркова (например, Lempel-Ziv), обеспечивает худшие результаты при чередовании сильно различающихся источников. Если относительные пропорции двух источников меняются (например, при увеличении MTU), разница в производительности для компрессии до и после инкапсуляции будет уменьшаться. Однако величина этого уменьшения очень мала — при увеличении increasing MTU в 4 раза (с 256 до 1024), различия между компрессией L-Z до и после инкапсуляции изменятся с 2,5% до 1,3%.

[4]Нсть и другие причины использования компрессии перед отправкой — меньшее количество символов требуется инкапсулировать и передавать модему. Автор предполагает, что варианта с компрессией данных в модеме следует избегать за исключением тех случаев, когда операционная система осложняет сжатие перед инкапсуляцией.

[5]Выбор времени не был совсем произвольным. Декомпрессия часто выполняется в течение периода наличия флага межкадрового интервала, поэтому в системах, где декомпрессия выполняется с таким же уровнем приоритета, как входные прерывания последовательного порта, превышение времени декомпрессии над временем передачи 1 символа может приводить к переполнению приемного буфера. При среднем размере кадра 5 байтов (в линии с учетом кадрирования и компрессии заголовка) время сжатия/декомпрессии, совпадающее с продолжительностью передачи 1 символа, не будет занимать более 20% доступного времени. Это представляется разумным.

Sun 3/60	90	90
Sun 3/50	130	150
HP9000/370	42	33
HP9000/360	68	70
DEC 3100	27	25
Vax 780	430	300
Vax 750	800	500
CCI Tahoe	110	140

В процессе разработки кода компрессии была создана также тестовая программа. Поначалу она применялась для сравнения разных протоколов компрессии, а потом для тестирования кода в разных компьютерных архитектурах и регрессионных тестов после «повышения» эффективности. Небольшое изменение этой тестовой программы позволило создать полезный измерительный инструмент[3]. В таблице 3 показаны временные характеристики кода компрессии на различных машинах, доступных автору (время измерялось с использованием фиксированных потоков данных telnet/ftp). За исключением архитектуры Vax, в которой используется (a) иной порядок байтов и (b) паршивый компилятор (Unix pcc), все компьютеры вписались в заданный интервал 120 мксек.

7 Благодарности

Автор признателен членам IETF[4] под руководством Phill Gross, обеспечившим поддержку и рецензирование этой работы. Несколько терпеливых тестеров, особенно Sam Leffler и Craig Leres, помогли в поиске ошибок в предварительных вариантах программ. Cynthia Livingston и Craig Partridge внимательно прочли и существенно улучшили черновые варианты этого документа. И последняя, но не менее важная благодарность корпорации Telebit modem и, в частности, Mike Ballard, поддерживавшим работу с самого начала и являющимся чемпионами в сфере последовательных линий и dial-up IP.

Литература

[1] Bingham, J. A. C. Theory and Practice of Modem Design. John Wiley & Sons, 1988.

[2] Carey, M. B., Chan, H.-T., Descloux, A., Ingle, J. F., and Park, K. I. 1982/83 end office connection study: Analog voice and voiceband data transmission performance characterization of the public switched network. Bell System Technical Journal 63, 9 (Nov. 1984).

[3] Chiappa, N., 1988. Частное сообщение.

[4] Clark, D. D. The design philosophy of the DARPA Internet protocols. In Proceedings of SIGCOMM '88 (Stanford, CA, Aug. 1988), ACM.

[5] Farber, D. J., Delp, G. S., and Conte, T. M. A Thinwire Protocol for connecting personal computers to the Internet. Arpanet Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, Sept. 1984. RFC-914.

[6] Kent, C. A., and Mogul, J. Fragmentation considered harmful. In Proceedings of SIGCOMM '87 (Aug. 1987), ACM.

[7] Lempel, A., and Ziv, J. Compression of individual sequences via variable-rate encoding. IEEE Transactions on Information Theory IT-24, 5 (June 1978).

[8] Nagle, J. Congestion Control in IP/TCP Internetworks. Arpanet Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, Jan. 1984. RFC-896.

[9] Perkins, D. Point-to-Point Protocol: A proposal for multi-protocol transmission of datagrams over point-to-point links. Arpanet Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, Nov. 1989. RFC-1134.

[10] Postel, J., Ed. Internet Protocol Specification. SRI International, Menlo Park, CA, Sept. 1981. RFC-791.

[11] Postel, J., Ed. Transmission Control Protocol Specification. SRI International, Menlo Park, CA, Sept. 1981. RFC-793.

[12] Romkey, J. A Nonstandard for Transmission of IP Datagrams Over Serial Lines: Slip. Arpanet Working Group Requests for Comment, DDN Network Information Center, SRI International, Menlo Park, CA, June 1988. RFC-1055.

[13] Salthouse, T. A. The skill of typing. Scientific American 250, 2 (Feb. 1984), 128--135.

[14] Saltzer, J. H., Reed, D. P., and Clark, D. D. End-to-end arguments in system design. ACM Transactions on Computer Systems 2, 4 (Nov. 1984).

[15] Shneiderman, B. Designing the User Interface. Addison-Wesley, 1987.

A. Пример реализации

Ниже приведен пример кода для реализации описанного в данном документе протокола.

Поскольку многие люди, которые захотят работать с этим кодом, будут знакомы с ядром Berkeley Unix и принятым в нем стилем кодирования (kernel normal form), код приведен именно в таком стиле. В коде используются подпрограммы Berkeley (макросы и расширения inline assembler) для преобразования в сетевой порядок байтов и обратно, а также для копирования и сравнения байтовых строк. Эти подпрограммы кратко описаны в Приложении A.5 для тех, кто с ними не знаком.

Этот код работает на всех машинах, перечисленных в таблице 3. Автор рад отметить, что программы работают, независимо от порядка байтов и методов выравнивания (хотя здесь использовалось допущение о том, что выравнивание Berkeley Unix может оказаться некорректным в других реализациях IP – см. комментарии к sl_compress_tcp and sl_decompress_tcp).

[3]Как тестовая программа, так и программа для измерения времени включены в доступный по протоколу ftp пакет, описанный в Приложении A (файлы tester.c и timer.c).
[4]Internet Engineering Task Force.

Было предпринято несколько попыток повысить эффективность кода. К сожалению, они оказались недостаточными. Автор приносит свои извинения за это (в частности, за «корявый» код C) и оставляет вопрос повышения эффективности, как факультативное упражнение.

Образец кода и полная реализация в Berkeley Unix доступны по протоколу ftp (анонимный доступ) на хосте ftp.ee.lbl.gov (128.3.254.68) в файле cslip.tar.Z. Это сжатый файл Unix, который следует загружать в двоичном режиме.

Ко всему коду, приведённому в приложениях, относится указанное ниже заявление об авторских правах:

```
/*
 * Copyright (c) 1989 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are
 * permitted provided that the above copyright notice and this
 * paragraph are duplicated in all such forms and that any
 * documentation, advertising materials, and other materials
 * related to such distribution and use acknowledge that the
 * software was developed by the University of California,
 * Berkeley.  The name of the University may not be used to
 * endorse or promote products derived from this software
 * without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE
 * IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE.
 */
```

A.1. Определения и данные о состоянии

```
#define MAX_STATES 16   /* должно быть >2 и <255 */
#define MAX_HDR 128     /* максимальный размер заголовков TCP+IP (по протоколам) */

/* типы пакетов */
#define TYPE_IP 0x40
#define TYPE_UNCOMPRESSED_TCP 0x70
#define TYPE_COMPRESSED_TCP 0x80
#define TYPE_ERROR 0x00 /* это не тип, который передается в линию
                         * Приемный модуль кадрирования использует этот формат,
                         * чтобы сказать декомпрессору об ошибке при передаче. */
/*
 * Биты первого октета в сжатом пакете
 */

/* биты флагов, которые изменяются в пакете */

#define NEW_C       0x40
#define NEW_I       0x20
#define TCP_PUSH_BIT       0x10

#define NEW_S       0x08
#define NEW_A       0x04
#define NEW_W       0x02
#define NEW_U       0x01

/* резерв, специальный случай вышеприведенных значений */
#define SPECIAL_I   (NEW_S|NEW_W|NEW_U)         /* интерактивный трафик с эхо */
#define SPECIAL_D   (NEW_S|NEW_A|NEW_W|NEW_U)   /* односторонний трафик */
#define SPECIALS_MASK       (NEW_S|NEW_A|NEW_W|NEW_U)


/*
 * данные о «сотстоянии» каждого активного диалога tcp в линии. Это обычно копия
 * заголовка IP/TCP из последнего пакета вместе с небольшим идентификатором
 * приемной и передающей стороны, используемым для поиска сохраненных заголовков
 */
struct cstate {
     struct cstate *cs_next;  /* предшествующее значение cstate (только передача) */
     u_short cs_hlen;         /* размер hdr (только прием) */
     u_char cs_id;            /* номер соединения для данного состояния */
     u_char cs_filler;
     union {
          char hdr[MAX_HDR];
          struct ip csu_ip;   /* заголовки ip/tcp из предыдущего пакета */
     } slcs_u;
};
#define cs_ip slcs_u.csu_ip

#define cs_hdr slcs_u.csu_hdr

/*
 * все данные о состоянии для одной последовательной линии (по одному на линию).
 */
struct slcompress {
     struct cstate *last_cs;            /* предыдущее состояние tstate */
     u_char last_recv;                  /* идентификатор предыдущего приема */
     u_char last_xmit;                  /* идентификатор предыдущей передачи */
```

```
        u_short flags;
        struct cstate tstate[MAX_STATES];  /* состояния передачи для соединения */
        struct cstate rstate[MAX_STATES];  /* состояния приема для соединения */
};

/* значения флагов */
#define SLF_TOSS 1       /* отбрасывание полученных кадров из-за ошибки на входе */

/*
 * Приведенные ниже макросы служат для кодирования и декодирования целых чисел. Они
 * предполагают, что cp указывает на буфер, где будет сохраняться (отыскиваться) следующий
 * кодируемый (декодируемый) байт. Поскольку программы декодирования выполняют
 * арифметические операции они выполняют преобразование порядка байтов.
 */

/*
 * ENCODE кодирует целое число, отличное от 0. ENCODEZ проверяет нулевое значение
 * (0 кодируется в форме 3 байтов).
 */
#define ENCODE(n) { \
     if ((u_short)(n) >= 256) { \
          *cp++ = 0; \
          cp[1] = (n); \
          cp[0] = (n) >> 8; \
          cp += 2; \
     } else { \
          *cp++ = (n); \
     } \
}
#define ENCODEZ(n) { \
     if ((u_short)(n) >= 256 || (u_short)(n) == 0) { \
          *cp++ = 0; \
          cp[1] = (n); \
          cp[0] = (n) >> 8; \
          cp += 2; \
     } else { \
          *cp++ = (n); \
     } \
}

/*
 * DECODEL принимает изменение (сжатие) байта cp и добавляет его к текущему значению поля
 * f в пакете (4-байтовое целое число с сетевым порядком байтов). DECODES выполняет те же
 * операции для 2-байтовых полей. DECODEU принимает изменение в cp помещает его в поле
 * (short) f. Cp обновляется для указания на следующее поле сжатого заголовка.
 */
#define DECODEL(f) { \
     if (*cp == 0) {\
          (f) = htonl(ntohl(f) + ((cp[1] << 8) | cp[2])); \
          cp += 3; \
     } else { \
          (f) = htonl(ntohl(f) + (u_long)*cp++); \
     } \
}
#define DECODES(f) { \
     if (*cp == 0) {\
          (f) = htons(ntohs(f) + ((cp[1] << 8) | cp[2])); \
          cp += 3; \
     } else { \
          (f) = htons(ntohs(f) + (u_long)*cp++); \
     } \
}
#define DECODEU(f) { \
     if (*cp == 0) {\
          (f) = htons((cp[1] << 8) | cp[2]); \
          cp += 3; \
     } else { \
          (f) = htons((u_long)*cp++); \
     } \
}
```

A.2. Компрессия

Эта процедура выглядит более сложной, чем на самом деле. Код делится на 4 примерно равных части. Первая часть управляет кольцевым связанным списком недавно «активных» соединений TCP[1]. Вторая часть контролирует изменения sequence/ack/window/urg и строит основную часть сжатого пакета. Третья часть служит для особых случаев кодирования, а последняя кодирует идентификаторы пакета и соединения, а также меняет исходный заголовок на сжатый.

Аргументами этой процедуры являются указатель на сжимаемый пакет, указатель на состояние компрессии для последовательной линии и флаг, управляющий сжатием идентификатора соединения (бит C).

[1]Двумя основными операциями с этим списком являются «поиск», который завершается на новом пакете для последнего использованного соединения, и перенос последней записи списка в его начало (первый пакет из нового соединения). Кольцевой список позволяет эффективно реализовать эти операции.

Компрессия выполняется «на месте», поэтому при создании сжатого пакета стартовый адрес и размер входящего пакета (поля off и len в m) будут обновляться при удалении исходного заголовка и его замене сжатым. Если создается сжатый или несжатый пакет состояние компрессии обновляется. Программа возвращает тип пакета для модуля кадрирования передачи (TYPE_IP, TYPE_UNCOMPRESSED_TCP или TYPE_COMPRESSED_TCP).

Поскольку в поля заголовка применяется 16- и 32-битовая арифметика, входящий пакет IP требуется выровнять должным образом (например, в SPARC заголовок IP выравнивается по 32-битовой границе). Если этого не делать, приведенный ниже код потребует существенных изменений (вероятно проще будет скопировать байт из входящего заголовка в переменную с корректным выравниванием и после этого выполнять операции).

Отметим, что исходящий пакет не будет иметь такого выравнивания (например, может начинаться с нечетного байта).

```
u_char
sl_compress_tcp(m, comp, compress_cid)
     struct mbuf *m;
     struct slcompress *comp;
     int compress_cid;
{
     register struct cstate *cs = comp->last_cs->cs_next;
     register struct ip *ip = mtod(m, struct ip *);
     register u_int hlen = ip->ip_hl;
     register struct tcphdr *oth;       /* последний заголовок TCP */
     register struct tcphdr *th;        /* текущий заголовок TCP */
     register u_int deltaS, deltaA;     /* временные данные общего назначения */
     register u_int changes = 0;        /* смена маски */
     u_char new_seq[16];                /* замена последнего на текущий */
     register u_char *cp = new_seq;

     /*
      * Возврат, если пакет TCP (флаг ACK или установлены другие флаги) не сжимаем
      * или является фрагментом IP. Предполагается, что вызывающая сторона уже проверила,
      * что пакет IP относится к протоколу TCP.
      */
     if ((ip->ip_off & htons(0x3fff)) || m->m_len < 40)
          return (TYPE_IP);

     th = (struct tcphdr *) & ((int *) ip)[hlen];
     if ((th->th_flags & (TH_SYN | TH_FIN | TH_RST | TH_ACK)) != TH_ACK)
          return (TYPE_IP);

     /*
      * Пакет может быть сжат и будет передаваться, как  COMPRESSED_TCP или
      * UNCOMPRESSED_TCP. В любом случае нужно найти (или создать) состояние компрессии
      * Особым случаем является использование последнего соединения, поскольку его новое
      * применение весьма вероятно и при его использовании не потребуется изменения
      * порядка.
      */
     if (ip->ip_src.s_addr != cs->cs_ip.ip_src.s_addr ||
         ip->ip_dst.s_addr != cs->cs_ip.ip_dst.s_addr ||
         *(int *) th != ((int *) &cs->cs_ip)[cs->cs_ip.ip_hl]) {

          /*
           * Пакет не первый — ищем состояние!
           *
           * Состояния хранятся в форме кольцевого связного списка и last_cs указывает на
           * конец списка. Список хранится в порядке lru путем сдвига состояния к голове
           * списка при каждом обращении к данному состоянию. Поскольку список короткий
           * и на практике искомое соединение почти всегда оказывается в начальной части,
           * поиск организован линейно. Если состояние для дейтаграммы не найдено,
           * используется самое старое (последнее) состояние из списка.
           */
          register struct cstate *lcs;
          register struct cstate *lastcs = comp->last_cs;

          do {
               lcs = cs;
               cs = cs->cs_next;
               if (ip->ip_src.s_addr == cs->cs_ip.ip_src.s_addr
                   && ip->ip_dst.s_addr == cs->cs_ip.ip_dst.s_addr
                   && *(int *) th == ((int *) &cs->cs_ip)[cs->cs_ip.ip_hl])
                    goto found;
          } while (cs != lastcs);

          /*
           * Ничего не нашлось — используем самое старое состояние. Передается несжатый
           * пакет, который сообщает другой стороне номер, использованный для данного
           * диалога. Отметим, что в силу кольцевого характера списка состояний самое
           * старое состояние указывает на самое новое и нужно лишь установить last_cs для
           * обновления связности lru.
           */
          comp->last_cs = lcs;
          hlen += th->th_off;
          hlen <<= 2;
          goto uncompressed;

found:
```

```
        /* Нашли — переносим в начало списка соединений. */
        if (lastcs == cs)
             comp->last_cs = lcs;
        else {
             lcs->cs_next = cs->cs_next;
             cs->cs_next = lastcs->cs_next;
             lastcs->cs_next = cs;
        }
    }
    /*
     * Удостоверимся, что изменилось только то, что мы хотели изменить. Сначала строка if
     * используется для проверки версии IP, размера заголовка и ToS. Второе условие
     * проверяет флаг Don't fragment, третье — TTL и протокол (протокол можно не
     * проверять, но это ничего не стоит). Четвертое условие служит для проверки размера
     * заголовка TCP, в пятом проверяются опции IP при их наличии, а в шестом — опции TCP
     * при их наличии. Если любая из этих проверок дает негативный результат,
     * дейтаграмма передается без сжатия.
     */
    oth = (struct tcphdr *) & ((int *) &cs->cs_ip)[hlen];
    deltaS = hlen;
    hlen += th->th_off;
    hlen <<= 2;

    if (((u_short *) ip)[0] != ((u_short *) &cs->cs_ip)[0] ||
        ((u_short *) ip)[3] != ((u_short *) &cs->cs_ip)[3] ||
        ((u_short *) ip)[4] != ((u_short *) &cs->cs_ip)[4] ||
        th->th_off != oth->th_off ||
        (deltaS > 5 && BCMP(ip + 1, &cs->cs_ip + 1, (deltaS - 5) << 2)) ||
        (th->th_off > 5 && BCMP(th + 1, oth + 1, (th->th_off - 5) << 2)))
         goto uncompressed;

    /*
     * Проверка изменившихся полей. Получатель ждет изменений в urgent, window, ack, seq.
     */
    if (th->th_flags & TH_URG) {
         deltaS = ntohs(th->th_urp);
         ENCODEZ(deltaS);
         changes |= NEW_U;
    } else if (th->th_urp != oth->th_urp)
         /*
          * Ах! Флаг URG не установлен, а значение urp изменилось — реализациям не
          * следует допускать этого, но RFC793 не запрещает таких изменений.
          */
         goto uncompressed;

    if (deltaS = (u_short) (ntohs(th->th_win) - ntohs(oth->th_win))) {
         ENCODE(deltaS);
         changes |= NEW_W;
    }
    if (deltaA = ntohl(th->th_ack) - ntohl(oth->th_ack)) {
         if (deltaA > 0xffff)
              goto uncompressed;
         ENCODE(deltaA);
         changes |= NEW_A;
    }
    if (deltaS = ntohl(th->th_seq) - ntohl(oth->th_seq)) {
         if (deltaS > 0xffff)
              goto uncompressed;
         ENCODE(deltaS);
         changes |= NEW_S;
    }
    /*
     * Просмотр на случай особого кодирования.
     */
    switch (changes) {

    case 0:
         /*
          * Ничего не изменилось. Если этот пакет содержит данные, а в предыдущем их не
          * было, это может говорить, что этот пакет является подтверждением ack
          * (нормально для интерактивного соединения) и пакет передается сжатым.
          * В противном случае это может быть повтором, повтором ack или пробой окна.
          * Пакет передается без компрессии, если на другой стороне отсутствует
          * сжатая версия.
          */
         if (ip->ip_len != cs->cs_ip.ip_len &&
             ntohs(cs->cs_ip.ip_len) == hlen)
              break;

         /* (пропускаем) */

    case SPECIAL_I:
    case SPECIAL_D:
         /*
          * Реальные изменения соответствуют одному из особых случаев кодирования -
          * пакет передается без компрессии.
```

```
             */
            goto uncompressed;

        case NEW_S | NEW_A:
            if (deltaS == deltaA &&
                deltaS == ntohs(cs->cs_ip.ip_len) - hlen) {
                /* Особый случай терминального трафика с эхо */
                changes = SPECIAL_I;
                cp = new_seq;
            }
            break;

        case NEW_S:
            if (deltaS == ntohs(cs->cs_ip.ip_len) - hlen) {
                /* Особый случай data xfer */
                changes = SPECIAL_D;
                cp = new_seq;
            }
            break;
        }
        deltaS = ntohs(ip->ip_id) - ntohs(cs->cs_ip.ip_id);
        if (deltaS != 1) {
            ENCODEZ(deltaS);
            changes |= NEW_I;
        }
        if (th->th_flags & TH_PUSH)
            changes |= TCP_PUSH_BIT;
        /*
         * Сохраним контрольную сумму до ее переписывания. После этого обновим состояние
         * с использованием заголовка данного пакета.
         */
        deltaA = ntohs(th->th_sum);
        BCOPY(ip, &cs->cs_ip, hlen);

        /*
         * Мы хотим использовать исходный пакет в качестве сжатого. (cp - new_seq) указывает
         * число байтов, требуемых для сжатых порядковых номеров. Кроме того потребуется 1
         * байт для маски изменений, 1 для идентификатора соединения и 2 для контрольной
         * суммы tcp. Итого, требуется (cp - new_seq) + 4 байта заголовка. hlen показывает
         * число отбрасываемых байтов исходного пакета, поэтому вычтем 2 для получения
         * нового размера пакета.
         */
        deltaS = cp - new_seq;
        cp = (u_char *) ip;
        if (compress_cid == 0 || comp->last_xmit != cs->cs_id) {
            comp->last_xmit = cs->cs_id;
            hlen -= deltaS + 4;
            cp += hlen;
            *cp++ = changes | NEW_C;
            *cp++ = cs->cs_id;
        } else {
            hlen -= deltaS + 3;
            cp += hlen;
            *cp++ = changes;
        }
        m->m_len -= hlen;
        m->m_off += hlen;
        *cp++ = deltaA >> 8;
        *cp++ = deltaA;
        BCOPY(new_seq, cp, deltaS);
        return (TYPE_COMPRESSED_TCP);

    uncompressed:
        /*
         * Обновим состояние соединения cs и передадим несжатый пакет
         * (uncompressed означает обычный пакет ip/tcp, но с идентификатором диалога
         * который может использоваться при сжатии последующих пакетов в поле протокола.
         */
        BCOPY(ip, &cs->cs_ip, hlen);
        ip->ip_p = cs->cs_id;
        comp->last_xmit = cs->cs_id;
        return (TYPE_UNCOMPRESSED_TCP);
    }
```

A.3. Декомпрессия

Эта функция служит для декомпрессии принятых пакетов. В качестве параметров передаются указатель на пакет, размер и тип пакета, а также указатель на структуру состояний компрессии приемной последовательной линии. Функция возвращает указатель на декомпрессированный пакет или 0, если в пакете были обнаружены ошибки. Если пакет относится к типу COMPRESSED_TCP или UNCOMPRESSED_TCP, функция обновляет состояние компрессии.

Новый пакет будет создаваться на месте полученного. Это означает, что требуется не менее 128 байтов свободного пространства в начале bufp для размещения восстановленных заголовков IP и TCP. Восстановленный пакет будет выравниваться по 32-битовой границе.

```
    u_char *
```

```
sl_uncompress_tcp(bufp, len, type, comp)
     u_char *bufp;
     int len;
     u_int type;
     struct slcompress *comp;
{
     register u_char *cp;
     register u_int hlen, changes;
     register struct tcphdr *th;
     register struct cstate *cs;
     register struct ip *ip;

     switch (type) {

     case TYPE_ERROR:
     default:
          goto bad;

     case TYPE_IP:
          return (bufp);

     case TYPE_UNCOMPRESSED_TCP:
          /*
           * Находим сохраненное состояние для данного соединения. При корректном индексе
           * состояния сбрасывается флаг discard.
           */
          ip = (struct ip *) bufp;
          if (ip->ip_p >= MAX_STATES)
               goto bad;

          cs = &comp->rstate[comp->last_recv = ip->ip_p];
          comp->flags &= ~SLF_TOSS;
          /*
           * Восстанавливается поле протокола IP после чего сохраняется копия заголовка
           * пакета. Поле контрольной суммы в копии обнуляется, поэтому не требуется
           * устанавливать нулевое значение при каждой обработке сжатого пакета.
           */
          ip->ip_p = IPPROTO_TCP;
          hlen = ip->ip_hl;
          hlen += ((struct tcphdr *) & ((int *) ip)[hlen])->th_off;
          hlen <<= 2;
          BCOPY(ip, &cs->cs_ip, hlen);
          cs->cs_ip.ip_sum = 0;
          cs->cs_hlen = hlen;
          return (bufp);

     case TYPE_COMPRESSED_TCP:
          break;
     }
     /* Получен сжатый пакет. */
     cp = bufp;
     changes = *cp++;
     if (changes & NEW_C) {
          /*
           * Проверим принадлежность индекса состояния к нужному диапазону и считаем
           * состояние. Если индекс корректен, сбросим флаг discard.
           */
          if (*cp >= MAX_STATES)
               goto bad;

          comp->flags &= ~SLF_TOSS;
          comp->last_recv = *cp++;
     } else {
          /*
           * Пакет имеет неявный индекс состояния. Если с момента получения последнего
           * явного индекса состояния в линии возникали ошибки, пакет отбрасывается.
           */
          if (comp->flags & SLF_TOSS)
               return ((u_char *) 0);
     }
     /*
      * Находим состояние, заполняем поле контрольной суммы TCP и устанавливаем флаг PUSH.
      */
     cs = &comp->rstate[comp->last_recv];
     hlen = cs->cs_ip.ip_hl << 2;
     th = (struct tcphdr *) & ((u_char *) &cs->cs_ip)[hlen];
     th->th_sum = htons((*cp << 8) | cp[1]);
     cp += 2;
     if (changes & TCP_PUSH_BIT)
          th->th_flags |= TH_PUSH;
     else
          th->th_flags &= ~TH_PUSH;

     /*

      * Фиксируем для состояния поля ack, seq, urg и win на основе маски изменений.
```

```
         */
        switch (changes & SPECIALS_MASK) {
        case SPECIAL_I:
             {
             register u_int i = ntohs(cs->cs_ip.ip_len) - cs->cs_hlen;
             th->th_ack = htonl(ntohl(th->th_ack) + i);
             th->th_seq = htonl(ntohl(th->th_seq) + i);
             }
             break;

        case SPECIAL_D:
             th->th_seq = htonl(ntohl(th->th_seq) + ntohs(cs->cs_ip.ip_len)
                        - cs->cs_hlen);
             break;

        default:
             if (changes & NEW_U) {
                  th->th_flags |= TH_URG;
                  DECODEU(th->th_urp)
             } else
                  th->th_flags &= ~TH_URG;
             if (changes & NEW_W)
                  DECODES(th->th_win)
             if (changes & NEW_A)
                  DECODEL(th->th_ack)
             if (changes & NEW_S)
                  DECODEL(th->th_seq)
             break;
        }
        /* Обновляем IP ID */
        if (changes & NEW_I)
             DECODES(cs->cs_ip.ip_id)
        else
             cs->cs_ip.ip_id = htons(ntohs(cs->cs_ip.ip_id) + 1);

        /*
         * В данный момент cp указывает на первый байт данных в пакете. Если мы не выравниваем
         * по 4-байтовой границе, данные нужно скопировать «вниз» так, чтобы заголовки IP и
         * TCP были выровнены. Значение cp изменяем с учетом размера заголовков TCP/IP,
         * чтобы освободить место для восстановления их. Предполагается, что при обработке
         * пакета имеется возможность добавить перед ним 128 байтов. Меняем значение размера
         * с учетом восстановленных заголовков и заполняем поле IP length.
         */
        len -= (cp - bufp);
        if (len < 0)
             /*
              * нужно отбросить некоторые символы (crc заметит это, но старые реализации
              * кадрирования slip не заметят)
              */
             goto bad;

        if ((int) cp & 3) {
             if (len > 0)
                  OVBCOPY(cp, (int) cp & ~3, len);
             cp = (u_char *) ((int) cp & ~3);
        }
        cp -= cs->cs_hlen;
        len += cs->cs_hlen;
        cs->cs_ip.ip_len = htons(len);
        BCOPY(&cs->cs_ip, cp, cs->cs_hlen);

        /* пересчитаем контрольную сумму заголовка ip */
        {
             register u_short *bp = (u_short *) cp;
             for (changes = 0; hlen > 0; hlen -= 2)
                  changes += *bp++;
             changes = (changes & 0xffff) + (changes >> 16);
             changes = (changes & 0xffff) + (changes >> 16);
             ((struct ip *) cp)->ip_sum = ~changes;
        }
        return (cp);

   bad:
        comp->flags |= SLF_TOSS;
        return ((u_char *) 0);
   }
```

A.4. Инициализация

Эта процедура служит для инициализации структур данных о состояниях приемной и передающей сторон последовательной линии. Процедура должна вызываться при каждой активизации линии.

```
   void
   sl_compress_init(comp)
        struct slcompress *comp;
   {
        register u_int i;
```

```
        register struct cstate *tstate = comp->tstate;

        /*
         * Сброс всей информации, которая могла сохраниться от прежнего использования линии.
         */
        bzero((char *) comp, sizeof(*comp));
        /*
         * Связывание состояний передачи в кольцевой список.
         */
        for (i = MAX_STATES - 1; i > 0; --i) {
             tstate[i].cs_id = i;
             tstate[i].cs_next = &tstate[i - 1];
        }
        tstate[0].cs_next = &tstate[MAX_STATES - 1];
        tstate[0].cs_id = 0;
        comp->last_cs = &tstate[0];
        /*
         * Убедимся в том, что случайно не была выполнена компрессия CID
         * (предполагается MAX_STATES < 255).
         */
        comp->last_recv = 255;
        comp->last_xmit = 255;
}
```

A.5. Зависимости Berkeley Unix

Примечание: Приведенная ниже информация относится только к случаям переноса кода в системы, не основанные на 4BSD (Berkeley Unix).

В коде используются обычные заголовочные файлы Berkeley Unix (из /usr/include/netinet) для определения структуры заголовков IP и TCP. Теги структур достаточно точно соответствуют протокольным RFC и должны быть понятны, даже если вы не работаете с системами 4BSD[1].

Макрос BCOPY(src, dst, amt) вызывается для копирования amt байтов из src в dst. В BSD этот макрос транслируется в вызов bcopy. Если вы работаете в System-V Unix, макрос можно транслировать в вызов memcpy. Макрос OVBCOPY(src, dst, amt) служит для копирования в тех случаях, когда области src и dst перекрываются (например, при копировании с выравниванием по 4-байтовой границе). В ядре BSD этот макрос транслируется в вызов ovbcopy. Поскольку в AT&T используется неудачное определение memcpy, возможно лучше будет транслировать макрос в цикл копирования System-V.

Макрос BCMP(src, dst, amt) служит для сравнения amt из src и dst на предмет совпадения. В BSD макрос транслируется в вызов bcmp. В System-V его можно транслировать в вызов memcmp или написать процедуру сравнения, которая должна возвращать 0 при совпадении всех байтов src и dst и отличное от нуля значение в остальных случаях.

Процедура ntohl(dat) преобразует (4-байтовое) значение dat из сетевого порядка байтов в хостовый. На некоторых процессорах это можно реализовать путем:

```
#define ntohl(dat) (dat)
```

На Vax или IBM PC (а также других системах с порядком байтов от Intel) потребуется определить макрос или подпрограмму для изменения порядка байтов.

Процедура ntohs(dat) похожа на ntohl но применяется для 2-байтовых значений. Процедуры htonl(dat) и htons(dat) выполняют обратные преобразования (хостовый порядок в сетевой) для 4- и 2-байтовых значений.

Структура mbuf используется при вызове sl_compress_tcp, поскольку этой процедуре нужно менять начальный адрес и размер, если входящий пакет был сжат. В BSD структура mbuf определена в ядре. В других системах достаточно будет приведенного ниже определения:

```
struct mbuf {
        u_char  *m_off; /* указатель на начало данных */
        int     m_len;  /* размер данных */
};

#define mtod(m, t) ((t)(m->m_off))
```

B. Совместимость с прошлыми ошибками

При использовании с современным протоколом PPP [9] компрессия выполняется автоматически и не заметна для пользователя. К сожалению на многих сайтах имеются пользователи SLIP [12], которые не могут отличать сжатые пакеты от обычных, а также поддерживать согласование номеров версий или обмен опциями для автоматического согласования компрессии заголовков.

Автор с помощью описанных ниже хитростей смог обеспечить взаимодействие SLIP с компрессией заголовков с другими имеющимися клиентами и серверами. Отметим, что эти действия по обеспечению совместимости с прошлыми версиями, содержащими ошибки, могут показаться странными любому здравомыслящему человеку. Они предлагаются здесь исключительно с целью обеспечения совместимости с имеющимися реализациями SLIP, пока разработчики на выпустят PPP.

B.1. Работа без байта типа кадрирования

В Приложении A.1 были выбраны причудливые номера для типов пакетов для того, чтобы обеспечить возможность передачи информации о типе через последовательные линии, на которых нежелательно или невозможно явно добавить байт типа. Отметим, что в четырех младших битах первого байта пакета IP всегда передается значение 4 (номер версии протокола IP). Старшие 4 бита первого байта в сжатом заголовке игнорируются. С помощью

[1]Заголовочные файлы (и остальной сетевой код Berkeley) можно загрузить по протоколу ftp с анонимным доступом с сайта ucbarpa.berkeley.edu (файлы pub/4.3/tcp.tar и pub/4.3/inet.tar).

предложенной в параграфе A.1 схемы эти биты можно использовать для передачи типа в исходящих пакетах, воспользовавшись кодом

```
p->dat[0] |= sl_compress_tcp(p, comp);
```

и декодировать их на приемной стороне с помощью процедуры

```
if (p->dat[0] & 0x80)
        type = TYPE_COMPRESSED_TCP;
else if (p->dat[0] >= 0x70) {
        type = TYPE_UNCOMPRESSED_TCP;
        p->dat[0] &=~ 0x30;
} else
        type = TYPE_IP;
status = sl_uncompress_tcp(p, type, comp);
```

B.2. Совместимость со старыми серверами SLIP

Протокол SLIP, описанный в [12] не включает каких-либо механизмов, которые могли бы использоваться для автоматического согласования компрессии заголовков. Было бы хорошо обеспечить пользователям такого протокола SLIP возможность сжатия заголовков, но когда пользователи разных вариантов SLIP подключаются к одному серверу, становится затруднительной настройка вручную обеих сторон каждого соединения для поддержки компрессии. Для выполнения автоматической настройки можно использовать описанную ниже процедуру.

Поскольку имеется два типа коммутируемых (dial-in) клиентов (поддерживающие и не поддерживающие компрессию), которые могут работать с одним сервером, очевидно, что сервер должен настраивать конфигурацию для каждой новой клиентской сессии, но сами клиенты редко меняют свою конфигурацию или совсем не меняют ее. При настройке конфигурации вручную ее следует выполнять на той стороне, где изменения редки — у клиента. Это предполагает, что сервер имеет возможность как-то узнать об использовании клиентом компрессии заголовков. Предполагая симметрию (т. е., использование или отказ от компрессии для обоих направлений), сервер может использовать факт получения сжатого пакета от того или иного клиента, как сигнал о возможности использования сжатия при передаче пакетов этому клиенту. Это позволяет реализовать описанные ниже алгоритм.

В линии имеется два бита для управления компрессией заголовков — разрешено (allowed) и включено (on). Если бит **on** установлен, передаются сжатые пакеты, при сброшенном флаге сжатие не используется. Если установлен бит **allowed**, сжатые пакеты могут приниматься и при получении пакета UNCOMPRESSED_TCP и сброшенном бите **on** этот бит будет установлен[1]. При получении сжатого пакета и сброшенном флаге **allowed** пакет будет игнорироваться.

На клиентах настраивается установка обоих битов (бит allowed всегда устанавливается при установленном флаге on), а сервер начинает каждую сессию с установки бита allowed и сброса бита on. Первый сжатый пакет от клиента (это должен быть пакет UNCOMPRESSED_TCP) обеспечит включение компрессии на сервере.

C. Более агрессивное сжатие

Как отмечено в параграфе 3.2.2, в потоке сжатых заголовков имеются легко детектируемые последовательности, говорящие о возможности дополнительного сжатия. Нужно ли это сделать?

Средняя сжатая дейтаграмма имеет размер заголовка 7 битов[2]. Кадрирование должно занимать хотя бы 1 бит (для представления типа) и может вероятно занимать до 2-3 байтов. В наиболее интересных случаях будет присутствовать хотя бы один байт данных. И, наконец, контрольная сумма TCP (сквозная проверка) должна передаваться неизменной[3].

Кадрирование, данные и контрольная сумма должны передаваться даже при полном сжатии заголовка. Поэтому в лучшем случае удастся уменьшить размер пакета с 4 байтов до 3 байтов и одного бита (снижение задержки на 25%[4]). Это может показаться существенным, но на линии 2400 бит/с время отклика изменится с 29 мсек до 25. Современный человек просто не заметит этой разницы.

Автор призывает не менять предложенную схему компрессии за исключением каких-то специальных случаев передачи данных. В качестве примера рассмотрим сутацию с высоковольтным и программой управления, доступ к которым осуществляется по коммуникационному каналу. В силу многих причин (мультиплексирование, каналы, корректировка ошибок, доступность проверенных реализаций и т. п.) был удобен обмен данными с программой по протоколу TCP/IP. Однако по причине того, что основной задачей телеметрического канала был сбор данных, этот канал был асимметричным с отношением скоростей более 200. С учетом требований по задержкам данные собирались в пакеты размером 100 байтов и, поскольку TCP подтверждал каждый второй пакет, относительная полоса «восходящего» канала составляла a/200, где a — общий размер пакетов подтверждения. При использовании описанной здесь схемы минимальный пакет подтверждения составлял 4 байта, что позволяло сделать полосу восходящего канала в 2% от нисходящего. Это оказалось невозможным и была применена схема, описанная на стр. 6. Если первый бит кадра имел значение 1, это означало «такой же сжатый заголовок, как в предыдущий раз». В противном случае два последующих бита определяли один из типов, описанных в параграфе 3.2 Детальное рассмотрение. Поскольку в канале была организована отличная коррекция ошибок и трафик проходил только через один интервал (hop) контрольную сумму TCP тоже подвергали сжатию (!). В течение нескольких месяцев использования этой схемы более 99% 40-байтовых заголовков TCP/IP сжимались до 1 бита[5].

---

[1]Поскольку кадрирование [12] не включает детектирования ошибок, нужно быть аккуратным, чтобы не было ложного включения компрессии на сервере в результате ошибок в линии. Пакет UNCOMPRESSED_TCP следует проверить на согласованность (например, корректность контрольной суммы IP) до включения компрессии. Получение пакета COMPRESSED_TCP не следует использовать в качестве сигнала для включения компрессии.

[2]Проверка на нескольких миллионах пакетов смешанного трафика (многолетняя статистика между домом и офисом автора) показывает, что 80% пакетов использует 1 из двух специальных типов кодирования и, таким образом, только заголовок задает маску изменений.

[3]Если кто-то предложит вам решение с компрессией контрольной суммы TCP, просто скажите: «Нет!». Опыт показывает, что доверять можно только при сквозном контроле. Отказавшись от неизменности контрольных сумм, вы можете просто не заметить ошибок. Стоят ли два сэкономленных байта душевного покоя?

[4]Еще раз отметим, что речь идет о задержке в интерактивных приложениях. При передаче больших объемов данных разница между 3 и четырьмя байтами заголовка с учетом значительного объема данных просто не будет заметна.

D. Вопросы безопасности

Вопросы безопасности не рассматриваются в этом документе.

E. Адрес автора

Van Jacobson
Real Time Systems Group
Mail Stop 46A
Lawrence Berkeley Laboratory
Berkeley, CA 94720
Телефон: используйте электронную почту (автор не пользуется своим телефоном)
Электронная почта: van@helios.ee.lbl.gov

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru

[5]Приходилось слышать, что для приложений в реальном масштабе времени TCP/IP не подходит и требуется «облегченный» протокол с меньшими заголовками. Однако протоколов с заголовком 1 бит видеть не пришлось.