Internet Engineering Task Force (IETF)W. Eddy, Ed.
STD: 7 MTI Systems
Request for Comments: 9293 August 2022
Obsoletes: 793, 879, 2873, 6093, 6429, 6528,
6691
Updates: 1011, 1122, 5961
Category: Standards Track
ISSN: 2070-1721

Transmission Control Protocol (TCP)

Протокол управления передачей (TCP)

Аннотация

Этот документ задаёт протокол управления передачей (Transmission Control Protocol или TCP). TCP является важным протоколом транспортного уровня в стеке протоколов Internet и непрерывно развивался в течение десятилетий использования и роста сети Internet. За это время было внесено много изменений в протокол TCP, заданный RFC 793, хотя они были документированы лишь частично. В этом документе такие изменения собраны и объединены со спецификацией RFC 793. Документ отменяет RFC 793, а также RFC 879, RFC 2873, RFC 6093, RFC 6429, RFC 6528 и RFC 6691, которые частично обновляли RFC 793. Документ обновляет RFC 1011 и RFC 1122 и его следует считать заменой тем частям указанных документов, которые относятся к требованиям TCP. Документ также обновляет RFC 5961, добавляя небольшое разъяснение в обработку сброса в состоянии SYN-RECEIVED. Биты управления в заголовке TCP из RFC 793 также обновлены на основе RFC 3168.

Статус документа

Документ относится к категории Internet Standards Track.

Документ является результатом работы IETF[1] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[2]. Дополнительную информацию о стандартах Internet можно найти в разделе 2 в RFC 7841.

Информация о текущем статусе документа, найденных ошибках и способах обратной связи доступна по ссылке https://www.rfc-editor.org/info/rfc9293.

Оглавление

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

1. Назначение и область действия

В 1981 году был выпущен документ RFC 793 [16], описывающий протокол TCP и отменяющий опубликованные ранее спецификации TCP. С тех порт TCP получил широкое распространение и применяется в качестве транспортного протокола во множестве приложений Internet.

В течение нескольких десятилетий RFC 793 и ряд других документов служили основной спецификацией для TCP [49]. За это время в RFC 793 был обнаружен ряд ошибок, а также были обнаружены и устранены недостатки в защите, производительности и других аспектах. Со временем число документов с изменениями значительно выросло, но они никогда не объединялись в комплексное обновление базовой спецификации. Цель этого документа состоит в объединении всех изменений IETF Standards Track и других разъяснений к базовой функциональной спецификации TCP (RFC 793) в одну обновлённую версию спецификации.

В некоторых сопутствующих документах упоминаются важные алгоритмы, применяемые в TCP (например, для контроля перегрузки), но они не включены полностью в этот документ. Это осознанный выбор, поскольку базовая спецификация может применяться со множеством дополнительных алгоритмов, разрабатываемых и включаемых независимо. Этот документ фокусируется на общей для всех реализаций TCP основе, которая должна поддерживаться для совместимости взаимодействия. Поскольку некоторые дополнительные свойства TCP сами по себе достаточно сложны (например, усовершенствованное восстановление потерь и контроль перегрузок), они могут быть объединены независимо в будущих сопутствующих документах.

В дополнение к спецификации протокола, описывающей формат сегмента TCP, правила генерации и обработки, реализуемые в кода, RFC 793 и его обновления содержат сведения и описания, помогающие понять различные аспекты устройства и работы протокола. Данный документ не пытается изменить или обновить эти сведения и сосредоточен лишь на обновлении нормативной спецификации протокола. В документе сохранены ссылки на источники с важными разъяснениями и обоснованиями, где это уместно.

Этот документ предназначен как для проверки имеющихся реализаций TCP, так и для создания новых.

2. Введение

В RFC 793 рассмотрены цели проектирования TCP и даны примеры работы, включая организацию и завершение соединений, повтор передачи пакетов и восстановление потерь.

Этот документ описывает базовую функциональность, ожидаемую от современных реализаций TCP, и заменяет спецификацию протокола в RFC 793. Здесь не повторяются и не обновляются введение и концепции из разделов 1 и 2 в RFC 793. Даны ссылки на другие документы, содержащие разъяснения теории работы, обоснования и обсуждение проектных решений. Данные документ сосредоточен исключительно на нормативном поведении протокола.

В работе [49] представлено детальное руководство по RFC, определяющим TCP и описывающим различные важные алгоритмы. Этот документ содержит разделы, посвящённые настоятельно рекомендуемым усовершенствованиям,повышающим производительность и улучшающим другие аспекты TCP сверх описанных в этом документе базовых операций. Например, реализация контроля перегрузок (такая как [8]) является требованием TCP, но это сложная тема сама по себе и не рассматривается подробно в этом документе, поскольку имеется много вариантов и возможностей, которые не влияют на базовую совместимость. Точно так же большинство современных реализаций TCP включает высокопроизводительные расширения из [47], но они не являются обязательными и не рассматриваются в этом документе. Вопросы работы TCP по нескольким путям также рассматриваются отдельно [59].

Список отличий от RFC 793 приведён в разделе 5. Отличия от RFC 793.

2.1. Уровни требований

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не нужно** (SHALL NOT), **следует** (SHOULD), **не следует** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **не рекомендуется** (NOT RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе должны интерпретироваться в соответствии с BCP 14[3] [12] тогда и только тогда, когда они выделены шрифтом, как показано здесь.

Каждое применение ключевых слов RFC 2119 в документе помечено ссылкой на Приложение B. Сводка требований TCP. Предложения с ключевым словом MUST содержат метку MUST-X, где X указывает числовой идентификатор требования в Приложении B. Аналогично помечаются и требования уровней SHOULD, MAY, RECOMMENDED. Требования уровня SHOULD NOT и MUST NOT помечены как SHOULD и MUST.

2.2. Основные концепции TCP

TCP обеспечивает приложениям надёжное упорядоченное обслеживание потоков байтов. Потоки байтов приложений доставляются через сеть в сегментах TCP, каждый из которых передаётся как дейтаграмма IP (Internet Protocol).

Надёжность (гарантии доставки) TCP включает обнаружение потери пакетов (по номерам) и ошибок (по контрольным суммам сегментов), а также их исправление путём повторной передачи.

TCP поддерживает индивидуальную (unicast) доставку данных. Имеются anycast-приложения, способные применять TCP без изменений, хотя это связано с риском нестабильности при смене поведения пересылки на нижележащих уровнях [46].

Протокол TCP ориентирован на (прямые) соединения, хотя, по сути, не включает возможность проверки их живучести.

Поток данных в соединении TCP поддерживается в обоих направлениях, хотя приложения могут по своему усмотрению передавать данные лишь в одном направлении.

TCP использует номера портов для идентификации прикладных служб и мультиплексирования разных потоков между хостами.

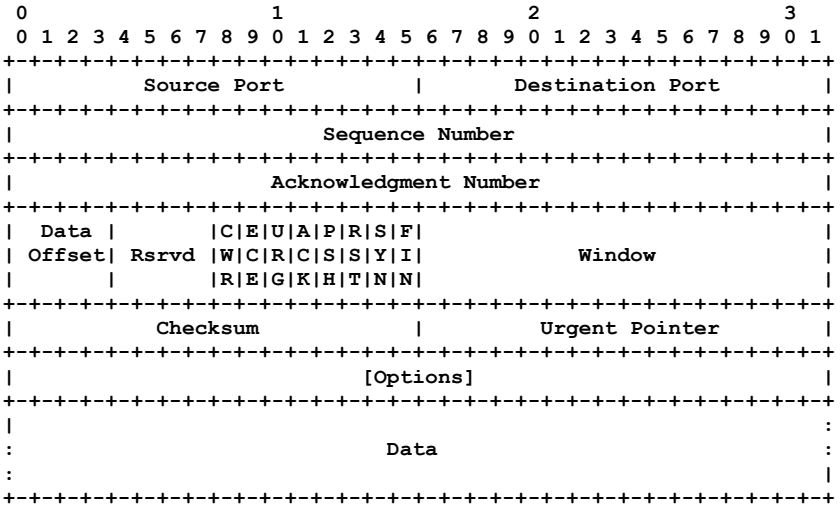
Более подробное описание свойств TCP по сравнению с другими транспортными протоколами представлено в параграфе 3.1 [52]. Дополнительное описание мотивов разработки TCP и роли протокола в стеке протоколов Internet можно найти в разделе Section 2 [16] и ранних версиях спецификации TCP.

3. Функциональная спецификация

3.1. Формат заголовка

Сегменты TCP передаются как дейтаграммы IP. Заголовок протокола IP содержит несколько полей, включая адреса хостов источника и получателя [1] [13]. Заголовок TCP размещается вслед за заголовками IP и содержит относящиеся к TCP сведения. Такое деление позволяет использовать на хосте протоколы, отличные от TCP. В ранних реализациях стека протоколов Internet поля заголовков IP были частью TCP.

Этот документ описывает протокол TCP, использующий заголовки TCP. Формат заголовка TCP, за которым могут следовать любые пользовательские данные, показан на рисунке 1 с использованием стиля из [66].



Каждая «ячейка» на рисунке соответствует 1 биту.

Рисунок 1. Формат заголовка TCP.

Source Port - 16 битов
Номер порта у отправителя.

Destination Port - 16 битов
Номер порта у получателя.

Sequence Number - 32 бита
Порядковый номер первого октета данных в этом сегменте (за исключением случаев, когда установлен флаг SYN). При установленном флаге SYN порядковым номером является исходный порядковый номер (initial sequence number или ISN) и первый октет данных имеет номер ISN+1.

Acknowledgment Number - 32 бита
При устновленном бите ACK это поле содержит значение следующего порядкового номера, который отправитель этого сегмента оджидает получить. Когда соединение установлено, это поле передаётся всегда.

Data Offset (DOffset) - 4 бита
Число 32-битовых слов в заголовке TCP (указывает начало данных). Размер заголовка TCP (даже при наличии опций) всегда содержит целое число 32-битовых слов.

Reserved (Rsrvd) - 4 бита
Набор битов, зарезервированных на будущее. Эти биты должны сбрасываться (0) при генерации сегментов и игнорироваться при получении, если соответствующие (будущие) свойства не реализованы отправителем или получателем.

Control bits
Биты управления, называемые также флагами (flag). Назначение битов контролируется IANA через реестр TCP Header Flags [62]. В настоящее время выделены флаги CWR, ECE, URG, ACK, PSH, RST, SYN, FIN.

CWR - 1 бит
Сокрещено окно перегрузки (насыщения) [6].

ECE - 1 бит
ECN-Echo [6].

URG - 1 бит
Указатель срочности (важности) в поле Urgent Pointer имеет значение.

ACK - 1 бит
Поле Acknowledgment имеет значение.

PSH - 1 бит
Функция выталкивания (Push), см. 3.9.1.2. Send.

RST - 1 бит
Сброс соединения (Reset).

SYN - 1 бит
Синхронизация порядковых номеров.

FIN - 1 бит
У отправителя больше нет данных.

Window - 16 битов
Число октетов данных, начиная с указанного в поле Acknowledgment, которые отправитель этого сегмента готов воспринять. Значение смещается при использовании расширения для масштабирования окна [47].
Размер окна должен трактоваться как целое число без знака, иначе окна большего размера будут рассматриваться как имеющие отрицательный размер и TCP не будет работать (MUST-1). Реализациям рекомендуется резервировать 32-битовые поля для размеров окон передачи и приёма в записи соединения и выполнять все расчёты для окон с использованием 32 битов (REC-1).

Checksum - 16 битов

Поле контрольной суммы содержит 16-битовое поразрядное дополнение суммы всех 16-битовых слов в заголовке и данных. Расчёт контрольной суммы выполняется с учётом размера, кратного 16-битам. Если сегмент содержит нечётное число октетов в заголовке и данных, к нему добавляется октет нулей для выравнивания размера, который применяется лишь при расчёте контрольной суммы и не передаётся в сегменте. При расчёте контрольной суммы значение поля Checksum принимается нулевым.

Контрольная сумма учитывает также псевдозаголовок (Рисунок 2), концептуально предшествующий заголовку TCP.Этот псевдозаголовок имеет размер 96 битов для IPv4 и 320 для IPv6. Включение псевдозаголовка в контрольную сумму обеспечивает соединению TCP защиту от некорректно маршрутизированных сегментов. Эта информация содержится в заголовках IP и передаётся через интерфейс TCP-сеть в аргументах или результатах вызовов реализацией TCP функций уровня IP.

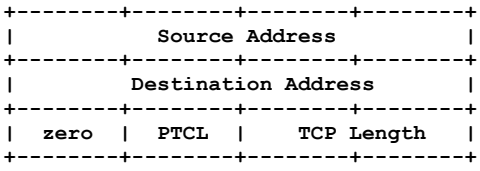


Рисунок 2. Псевдозаголовок IPv4.

Компоненты псевдозаголовка IPv4 приведены ниже.

Source Address

Адрес источника IPv4 в сетевом порядке байтов.

Destination Address

Адрес получателя IPv4 в сетевом порядке байтов.

zero

Биты, установленные в 0.

PTCL

Номер протокола из заголовка IP.

TCP Length

Размер заголовка и данных TCP в октетах (вычисляется, а не передаётся явно) без учёта 12 октетов псевдозаголовка.

Псевдозаголовок для IPv6 определён в параграфе 8.1 RFC 8200 [13] и содержит поля IPv6 Source Address и Destination Address, Upper-Layer Packet Length (32-битовое значение, в остальном эквивалентное TCP Length в псевдозаголовке IPv4), 3 нулевых байта заполнения и значение Next Header, которое отличается от заголовка IPv6 при наличии заголовков расширения между заголовками IPv6 и TCP.

Контрольная сумма TCP обязательна. Отправитель должен создавать её (MUST-2), а получатель должен проверять (MUST-3).

Urgent Pointer - 16 битов

Это поле передаёт текущее значение указателя важности (срочности) как положительное смещение от порядкового номера в этом сегменте. Этот указатель задаёт номер октета, следующего за важными данными. Поле интерпретируется лишь для сегментов с установленным флагом URG.

Options - [TCP Option]; size(Options) == (DOffset-5)*32;

Поле опций присутствует лишь при DOffset > 5. Отметим, что приведённое выше выражение включает помещенное после опций заполнение.

Опции могут размещаться в конце заголовка TCP и занимают целое число октетов. Все опции учитываются в контрольной сумме. Опция может начинаться на любой границе октета. Имеется два варианта формата опций:

1. одноoктетная опция;
2. октет вида опции (option-kind), октет размера опции и фактические данные (option-data) опции.

В поле размера (option-length) учитываются два октета option-kind и option-length, а также октеты option-data.

Отметим, что список опций может быть короче, нежели указывает поле Data Offset. Содержимое заголовка после опции End of Option List **должно** дополняться нулями (MUST-69).

Список определённых опций контролируется IANA [62] и каждая опция определена в RFC, как указано здесь. Набор включает экспериментальные опции, которые могут быть расширены для одновременной поддержки нескольких вариантов применения [45].

Реализация TCP может поддерживать любые определённые опции, но **должна** поддерживать опции, указанные ниже (MUST-4 - отметим, что поддержка опции Maximum Segment Size задана также MUST-14 в параграфе 3.7.1).

Таблица 1. Набор обязательных опций.

Тип	***Размер***	***Назначение***
0	-	Опция завершения опций (End of Option List).
1	-	Нет операции (No-Operation или NOP).
2	4	Максимальный размер сегмента (MSS).

Эти опции подробно описаны в параграфе 3.2. Определения конкретных опций.

Реализация TCP **должна** быть способна принимать опции TCP в любом сегменте (MUST-5).

Реализация TCP **должна** (MUST-6) игнорировать без ошибки либые опции TCP, которые она не реализует,предполагая, что опция имеет поле размера. Все опции TCP, кроме End of Option List (EOL) и No-Operation (NOP), включая все будущие опции, **должны** иметь поле размера (MUST-68). Реализация TCP **должна** быть готова обработать некорректный размер опции (например, 0), для чего рекомендуется процедура, сбрасывающая соединение и записывающая причину ошибки в системный журнал - log (MUST-7).

Отметим, что продолжаются работы по расширению пространства, доступного для опций TCP, например, [65].

Data - переменный размер

Пользовательские данные, передаваемые в сегменте TCP.

3.2. Определения конкретных опций

Набор обязательных опций TCP включает End of Option List (конец списка опций), No-Operation (нет операции) иMaximum Segment Size (максимальный размер сегмента или MSS).

Формат End of Option List Option показан ниже.

Kind - 1 байт; Kind == 0

Эта опция указывает завершение набора опций, которое может не совпадать с концом заголовка TCP в соответствии с полем Data Offset. Опция указывает завершение всех опций в заголовке, а не конкретной опции и её нужно применять лишь в тех случаях, когда конец опций не совпадает с завершением заголовка TCP.

Формат No-Operation Option показан ниже

Kind - 1 байт; Kind == 1

Эту опции можно помещать между любыми опциями, например, для выравнивания следующей опции по границе слова. Нет гарантии использования этой опции отправителем, поэтому получатели **должны** быть готовы к обработке опций, не начинающихся на границе слова (MUST-64).

Формат опции Maximum Segment Size Option показан ниже.

Kind - 1 байт; Kind == 2

При наличии этой опции она указывает максимальный размер сегмента, который готова воспринимать передавшая опцию сторона TCP. Значение опции ограничено пределом размера сборки фрагментов IP. Поле может передаваться в начальном запросе соединения (т. е. в сегменте с флагом SYN) и его **недопустимо** включать в другие сегменты (MUST-65). Если опция не используется, разрешён любой размер сегментов. Более полное описание опции дано в параграфе 3.7.1. Максимальный размер сегмента (MSS).

Length - 1 байт; Length == 4

Lразмер опции в байтах.

Maximum Segment Size (MSS) - 2 байта

Макимальный размер сегмента, который готова принимать передавшая опцию сторона TCP.

3.2.1. Другие опции общего назначения

В других RFC определены некоторые опции общего назначения, которые рекомендуется реализовать для повышения производительности, но они не требуются с точки зрения базовой совместимости TCP. Это опции селективных подтверждений (Selective Acknowledgment или SACK) [22] [26], временных меток (Timestamp или TS) [47], масштабирования окна (Window Scale или WS) [47].

3.2.2. Экспериментальные опции TCP

Экспериментальные опции TCP заданы в [30], а в [45] описано рекомендуемое применение этих опций.

3.3. Обзор терминологии TCP

В этом параграфе приведён обзор основных терминов, требуемых для понимания деталей работы протокола в оставшейся части документа. Глоссарий терминов приведён в разделе 4. Глоссарий.

3.3.1. Основные переменные состояния соединений

Перед обсуждением деталей работы реализации TCP введём некоторые термины и обозначения. Для поддержки соединения TCP нужно поддерживать состояния некоторых переменных. Предполагается, что эти переменные храняться в записи соединения, называемой блоком управления передачей (Transmission Control Block или TCB). В TCB хранятся локальные и удалённые адреса IP и номера портов, уровень защиты IP, назначение (compartment) соединения (см. Приложение A.1), указатели на пользовательские буферы приёма и передачи, указатели на очередь повтора передачи и текущий сегмент, а также некоторые переменные для порядковых номеров приёма и передачи.

Таблица 2. Переменные последовательности Send.

Переменная	***Описание***
SND.UNA	Передача не подтверждена
SND.NXT	Передать следующим
SND.WND	Окно передачи
SND.UP	Указатель важности для передачи
SND.WL1	Порядковый номер сегмента, использованный для последнего обновления окна
SND.WL2	Номер сегмента подтверждения, использованный для последнего обновления окна
ISS	Исходный порядковый номер для передачи

Таблица 3. Переменные последовательности Receive.

Переменная	***Описание***
RCV.NXT	Принять следующим
RCV.WND	Окно приема
RCV.UP	Указатель важности для приема
IRS	Исходный порядковый номер для приема

Приведённые ниже рисунки помогут связать некоторые из этих переменных с пространством номеров.

Окно передачи является частью пространства номеров, помеченной 3 на рисунке 3.

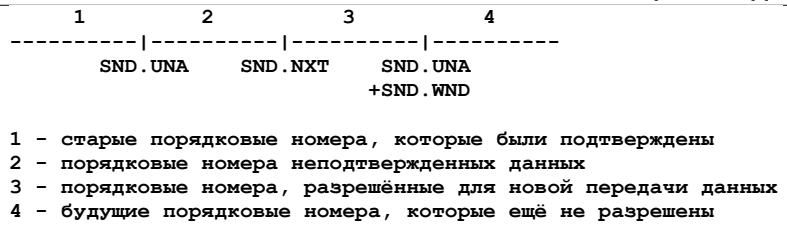


Рисунок 3. Пространство номеров передачи.

Окно приёма является частью пространства номеров, помеченной 2 на рисунке 4.

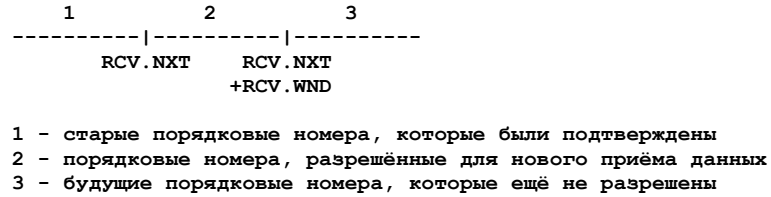


Рисунок 4. Пространство номеров приема.

В обсуждении часто применяются некоторые переменные, значения которых берутся из полей текущего сегмента.

Таблица 4. Переменные из текущего сегмента.

Переменная	***Описание***
SEG.SEQ	Порядковый номер сегмента
SEG.ACK	Номер подтверждения сегмента
SEG.LEN	Размер сегмента
SEG.WND	Окно сегмента
SEG.UP	Указатель важности сегмента

3.3.2. Обзор конечного автомата

В процессе своего существования соединение проходит через последовательность состояний LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT и фиктивное состояние CLOSED. Состояние CLOSED фиктивно, поскольку в нем нет TCB и, следовательно, соединения.

LISTEN
Ожидание запроса соединения от любого удалённого партнёра TCP и порта.

SYN-SENT
Ожидание соответствующего запроса соединения после отправки своего запроса.

SYN-RECEIVED
Ожидание подтверждения запроса соединения после отправки запросов обеими сторонами.

ESTABLISHED
Представляет созданное (открытое) соединение, в котором полученные данные могут быть доставлены пользователю. Это обычно состояние в фазе передачи данных через соединение.

FIN-WAIT-1
Ожидание запроса на завершение соединения от удалённого партнёра TCP или подтверждения переданного ранее запроса на завершение соединения.

FIN-WAIT-2
Ожидание запроса на завершение соединения от удалённого партнёра TCP.

CLOSE-WAIT
Ожидание запроса на завершение соединения от локального пользователя.

CLOSING
Ожидание подтверждения запроса на завершение соединения от удалённого партнёра TCP.

LAST-ACK
Ожидание подтверждения запроса на завершение соединения, переданного удалённому партнёру TCP (этот запрос уже включает подтверждение запроса на завершение соединения от удалённого партнёра TCP).

TIME-WAIT
Ожидание в течение времени, достаточного, чтобы убедиться в получении удаленным партнёром TCP подтверждения своего запроса на завершение соединения и избежать влияния задержанных сегментов прежних соединений на новые.

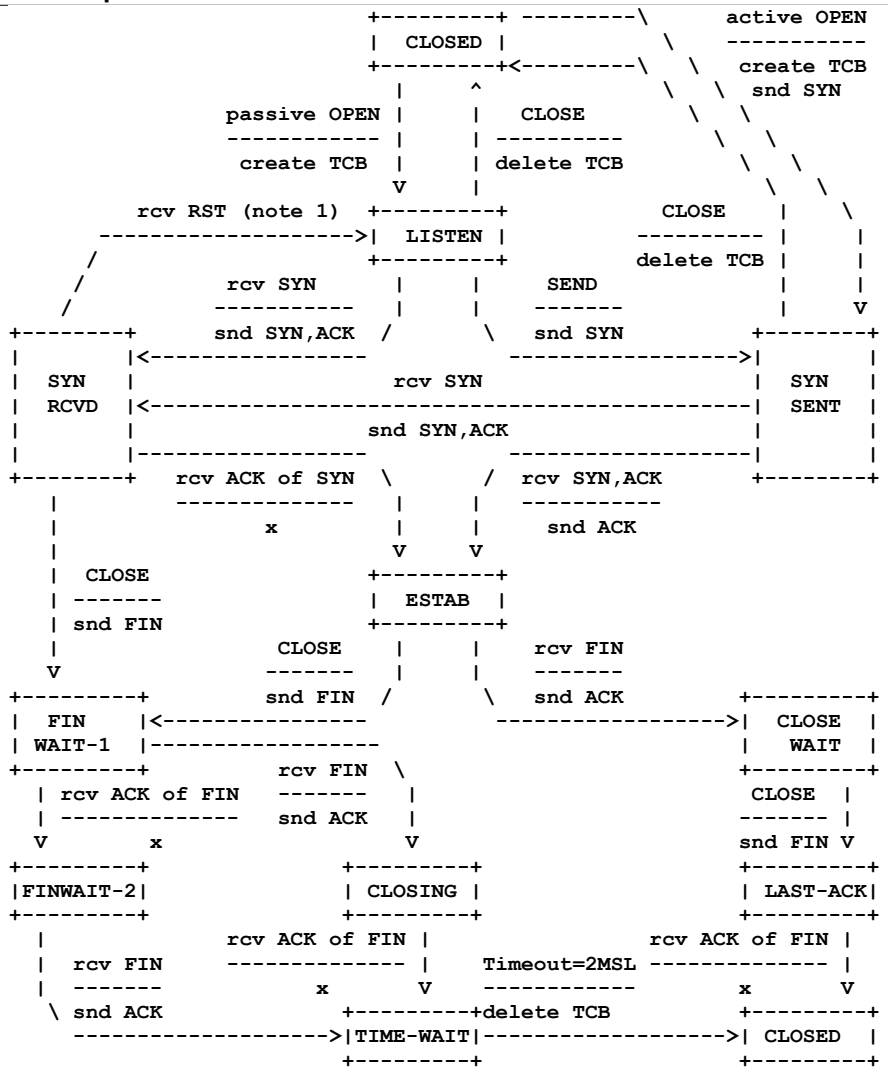
CLOSED
Представляет отсутствие соединения.

Соединение TCP меняет состояние по событиям, которые включают пользовательские вызовы, OPEN, SEND, RECEIVE, CLOSE, ABORT, STATUS, входящие сегменты, особенно с флагами SYN, ACK, RST, FIN и тайм-ауты.

Вызов OPEN указыват, следует создавать соединение активно или просто пассивно ждать. Пассивный запрос OPEN означает что процесс хочет воспринимать входящие запросы соединений, а при активном вызове OPEN пытается сам инициировать соединение.

В диаграмме состояний на рисунке 5 показаны лишь переходы вместе с вызвавшими их событиями и результирующие действия, но не рассматриваются ошибки и действия, не связанные со сменой состояния. В следующем параграфе более подробно рассматривается реакция реализации TCP на события. Имена некоторых состояний на рисунке сокращены для удобочитаемости.

Важно подчеркнуть, что диаграмма показывает лишь сводку и её не следует считать полной спецификацией. Многие детали на рисунке опущены.



1. *Переход из состояния SYN-RECEIVED в LISTEN при получении RST является условным по наличию принятого SYN-RECEIVED после пассивного вызова OPEN.*
2. *На рисунке опущен переход из FIN-WAIT-1 в TIME-WAIT, если получен сегмент FIN и локальный сегмент FIN подтверждён.*
3. *Сегмент RST можно передать из любого состояния с соответствующим переходом в состояние TIME-WAIT (см. обоснование в [70]). Эти переходы не показаны явно, чтобы не загромождать рисунок. Точно так же не показан переход в состояние LISTEN или CLOSED при получении RST в любом состоянии.*

Рисунок 5. Диаграмма состояний соединения TCP.

3.4. Порядковые номера

Фундаментальным свойством протокола является наличие порядкового номера у каждого октета данных, переданного через соединение TCP. Поскольку все октеты упорядочены, каждый из них можно подтвердить. Применяемый механизм подтверждений является кумулятивным и подтверждение порядкового номера X означает, что получены все предшествующие октеты (но не X). Этот механизм обеспечивает прямое обнаружение дубликатов при использовании повторной передачи. Схема нумерации октетов в сегменте проста - октет, следующий сразу после заголовка имеет наименьший номер, а номера следующих октетов возрастают по порядку.

Важно помнить, что пространство порядковых номеров конечно, хотя и достаточно велико - от 0 до 2^{32} - 1. Поскольку пространство номеров конечно, все арифметические операции с порядковыми номерами должны выполняться по модулю 2^{32}. Эта арифметика целых чисел без знака сохраняет отношения порядковых номеров при переходе от номера 2^{32} - 1 к 0. В компьютерной арифметике по модулю есть некоторые тонкости, поэтому при программировании нужно осторожно сравнивать значения. Символ =< означает «меньше или равно» (по модулю 2^{32}).

Типовые сравнения порядковых номеров, которые должна выполнять реализация TCP, включают:

(a) определение того, что подтверждение относится к порядковому номеру, который передан, но не подтверждён;

(b) определение того, что все порядковые номера из сегмента были подтверждены (например, для удаления сегмента из очереди повторной передачи);

(c) определение того, что входящий сегментсодержит ожидаемые номера (т. е. «перекрывается» с окном приема).

В ответ на передачу данных конечная точка TCP будет получать подтверждения, для обработки которых нужно выполнять перечисленные ниже сравнения.

SND.UNA = самый старый из неподтвержденных номеров.

SND.NXT = следующий номер для передачи.

SEG.ACK = подтверждение от принимающего партнёра TCP (следующий номер, ожидаемый им).

SEG.SEQ = первый порядковый номер в сегменте.

SEG.LEN = число октетов, занимаемых данными в сегменте (с учётом SYN и FIN).

SEG.SEQ+SEG.LEN-1 = последний порядковый номер в сегменте.

Новым подтверждением (его называют приемлемым - acceptable ack) является то, для которого выполняется условие

```
SND.UNA < SEG.ACK =< SND.NXT
```

Сегмент из очереди на повтор передачи считается полностью подтверждённым, если сумма его порядкового номера и размера не больше номера подтверждения во входящем сегменте.

При получении данных требуется выполнять указанные ниже сравнения.

RCV.NXT = следующий номер, ожидаемый во входящем сегменте, и является ли он левым (нижним) краем окна приёма.

RCV.NXT+RCV.WND-1 = последний номер, ожидаемый во входящем сегменте, и является ли он правым (верхним) краем окна приёма.

SEG.SEQ = первый порядковый номер во входящем сегменте.

SEG.SEQ+SEG.LEN-1 = последний порядковый номер во входящем сегменте.

Считается, что сегмент является частью допустимой последовательности приёма, если

```
RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
```

или

```
RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND
```

Первое выражение проверяет, попадает ли начало сегмента в окно, второе выполняет аналогичную проверку для конца сегмента. Если любая из проверок проходит, сегмент содержит данные из окна.

На деле все немного сложнее и из-за нулевых окон и сегментов нулевого размера возникают 4 случая приемлемости входящего сегмента, указанные в таблице 5.

Таблица 5. Проверка пригодности сегментов.

Размер сегмента	***Окно приема***	***Проверка***
0	0	`SEG.SEQ = RCV.NXT`
0	>0	`RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND`
>0	0	`Не поддерживается`

>0	>0	`RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND`
		или
		`RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND`

Отметим, что при нулевом размере окна приёма следует принимать лишь сегменты ACK. Это позволяет реализации TCP передавать данные и принимать подтверждения (ACK), установив нулевое окно приема. Получатель TCP должен обрабатывать поля RST и URG во всех входящих сегментах даже при нулевом размере окна приема (MUST-66).

Схема нумерации применяется также для защиты некоторой управляющей информации. Это достигается за счёт неявного включения некоторых флагов управления в пространство номеров, чтобы их можно было передать повторно и повторить без путаницы (т. е. действовать будет одна и только одна копия элемента управления). Управляющая информация не передаётся физически в пространстве сегмента данных, поэтому нужно принять правила для неявного назначения номеров элементам управления. Такая защита требуется лишь для флагов SYN и FIN, которые применяются только при создании и закрытии соединений. Для нумерации считается, что SYN присутствует перед первым октетом фактических данных в сегменте, тогда как FIN считается находящимся после последнего фактического элемента данных в сегменте с соответствующим флагом. Размер сегмента (SEG.LEN) учитывает как данные, так и занимающие пространство номеров элементы управления. При наличии SYN в переменной SEG.SEQ содержится порядковый номер SYN.

3.4.1. Выбор начального порядкового номера

Соединение определяется парой сокетов и может использоваться неоднократно. Новые экземпляры соединений называют инкарнациями соединения. В связи с этим возникает проблема - как реализация TCP идентифицирует дубликаты сегментов из предыдущих инкарнаций соединения? Эта проблема становится явной, когда соединение создаётся и завершается быстро или прерывается из-за потери памяти, а затем организуется снова. Для поддержки таких ситуаций состояние TIME-WAIT ограничивает темп повторного использования соединения, а описанный ниже выбор начального порядкового номера дополнительно защищает от неоднозначности принадлежности входящего пакета соединению.

Чтобы избежать путаницы, нужно предотвратить использование новой инкарнацией сегментов прежней инкарнации, которые все ещё могут присутствовать в сети и содержать те же порядковые номера. Нужно обеспечить гарантиюэтого даже в случаях, когда конечная точка TCP теряет все сведения об использованных номерах. При создании новых соединений применяется генератор исходного порядкового номера (initial sequence number или ISN), выбирающий новое 32-битовое значение ISN. Имеются проблемы безопасности, связанные с возможностью находящегося вне пути злоумышленника предсказать или угадать значение ISN [42].

Начальные порядковые номера TCP создаются из числовой последовательности, которая монотонно возрастает до достижения максимального значения (wrap), что обычно называют «часами». Эти часы представляют собой 32-битовый счётчик, который увеличивается не реже 1 раза приблизительно каждые 4 мксек, хотя не предполагается, что счётчик работает в реальном масштабе времени или точен и его не требуется сохранять при перезагрузке. Эти часы предназначены для того, чтобы гарантировать уникальность создаваемых ISN в течение максимального срока жизни сегмента (Maximum Segment Lifetime или MSL), который много меньше цикла счётчика, чоставляющего примерно 4,55 часа. Следует отметить, что в современных сетях с высокой скоростью передачи данных, где порядковые номера могут перекрываться в течение MSL, рекомендуется применять опцию Timestamp, описанную в параграфе 3.4.3.

Реализация TCP **должна** применять описанный выше тип «часов» для управления выбором начальных номеров (MUST-8) и **следует** генерировать эти номера на основе выражения

```
ISN = M + F(localip, localport, remoteip, remoteport, secretkey)
```

где M - значение 4-микросекундного таймера, F() - псевдослучайная функция (pseudorandom function или PRF) параметров идентификации соединения (localip, localport, remoteip, remoteport) и секретного ключа (secretkey) (SHLD-1).F() **недопустимо** рассчитывать вовне (MUST-9), иначе злоумышленник все равно сможет угадать порядковые номера по ISN, использованному в другом соединении. PRF можно реализовать как криптографических хэш конкатенации параметров соединения TCP и неких секретных данных. Выбор конкретного алгоритма хэширования и поддержка данных секретного ключа рассмотрены в разделе 3 [42].

Для каждого соединения имеется порядковый номер приёма и передачи. Исходный номер для передачи (initial send sequence number или ISS) выбирается передающим партнёром TCP, и исходный номер для приёма (initial receive sequence number или IRS) определяется в процедуре организации соединения.

Для организации и инициализации соединения два партнёра TCP должны синхронизировать свои начальные порядковые номера. Это выполняется путём обмена сегментами организации соединения с установленным флагом SYN (для синхронизации) и начальными номерами. Для краткости сегменты с флагом SYN называются просто SYN. Для решения задачи нужен подходящий механизм выбора начального порядкового номера и некоторое усложнение согласования (handshake) для обмена значениями ISN.

Для синхронизмции требуется отправка каждой стороной своего начального порядкового номера и приём подтверждения от удалённого партнёра TCP. Каждая сторона должна также получить начальный порядковый номер удалённого партнёра и передать соответствующее подтверждение.

1) A --> BSYN - мой порядковый номер X;
2) A <-- BACK - ваш порядковый номер X;
3) A <-- BSYN - мой порядковый номер Y;
4) A --> BACK - ваш порядковый номер Y.

Поскольку этапы 2 и 3 можно объединить в одно сообщение, эта процедура называется 3-этапным (three-way или three message) согласованием (handshake, 3WHS).

Согласование 3WHS требуется потому, что порядковые номера не привязаны к глобальным часам в сети и реализации TCP могут применять разные механизмы выбора ISN. Получатель первого SYN не может узнать, является ли сегмент старым, если только он не помнить последний номер, использованный в соединении (это возможно не всегда), поэтому должен запросить у отправителя проверку этого SYN. Трехэтапное согласование и преимущества схемы выбора ISN по часам рассмотрены в [69].

3.4.2. Когда нужно молчать

Существует теоретическая проблема, когда данные могут быть повреждены в результате путаницы между старыми сегментами в сети и новыми сегментами после перезагрузки хоста, если применяются те же номера портов и пространство номеров. Концепция «времени тишины» (quiet time), рассматриваемая ниже, решает эту проблему и её рассмотрение включено для ситуаций, где это уместно, хотя в большинстве современных реализаций этого не требуется. Проблема была актуальной на ранних этапах развития TCP. В современной практике Internet вероятность условия возникновения ошибок пренебрежимо мала. Причинами этого являются (a) ISS и случайное использование эфемерных портов, снижающие вероятность выбора того же номера порта и порядковых номеров после перезагрузки, (b) эффективное значение MSL в Internet снижается по мере роста скорости каналов, (c) перезагрузка зачастую длится дольше MSL.

Для гарантии того, что реализация TCP не создаст сегмент с порядковым номером, который может дублироваться в остающихся в сети старых сегментах, конечная точка TCP должна «сохранять молчание» в интервале MSL перед назначением каких-либо порядковых номеров при старте или восстановлении в ситуациях с потерей сведений о ранее использованных номерах. В этой спецификации принимается значение MSL 2 минуты. Это инженерный выбор и его можно изменить, если опыт покажет, что это желательно. Отметим, что при повторной инициализации конечной точки TCP с сохранением сведений об использованных номерах ждать не требуется, нужно лишь выбрать порядковые номера больше применяемых ранее.

3.4.3. Концепция TCP Quiet Time

Хостам, по какой-либо причине потерявшим сведения о последних переданных номерах в каждом активном (не закрытом) соединении, нужно перед отправкой какого-либо сегмента TCP выжать не менее интервала MSL согласованного в системе (сети), к которой хост относится. В последующих параграфах эта спецификация разъяснена. Разработчики TCP могут нарушать ограничение на «время тишины», но возникает риск принятия некоторых старых данных как новых или отклонения новых данных как дубликатов некоторыми получателями в системе.

Конечные точки TCP «расходуют» пространство номеров при каждом формировании сегмента и размещении его в выходной очереди передающего хоста. Алгоритм обнаружения дубликатов и упорядочения в TCP основан на однозначной привязке сегментов данных к пространству номеров так, чтобы порядковые номера не проходили весь цикл из 232 значений до того, как привязанные к номерам сегменты будут доставлены и подтверждены получателем, а все дубликаты сегментов будут «слиты» из сети. Без такого допущения два разных сегмента TCP могут получить однаковые или перекрывающиеся номера, вызывающие у получателя путаницу между старыми и новыми данными. Следует помнить, что с каждым сегментом связано множество последовательных номеров, число элементов которого совпадает с числом октетов данных и флагов SYN или FIN в сегменте.

При нормальных условиях реализация TCP отслеживает следующий порядковый номер для отправки и самый старых из ожидающих подтверждения номеров, чтобы избежать ошибочного повторного использования номера до того, как первая его передача будет подтверждена. Само по себе это не гарантирует удаления из сети старых сегментов, поэтому пространство номеров выбрано достаточно большим, чтобы снизить вероятность проблем из-за блуждающих в сети старых дубликатов. При скорости 2 Мбит/с пространство из 232 порядковых номеров расходуется за 4,5 часа. Поскольку максимальный срок жизни сегмента в сети вряд ли превысит несколько десятков секунд, это считается достаточной защитой для представимых сетей даже при росте скорости передачи данных до десятков Мбит/с. При

скорости 100 Мбит/с цикл расхода номеров составляет 5,4 мнут, что намного меньше, но в пределах разумного. Сегодня доступны более высокие скорости и последствия этого описаны в конце параграфа.

Базовый механизм обнаружения дубликатов и упорядочения в TCP можно обойти, если исходная конечная точка не запоминает порядковые номера, использованные последними в данном соединении. Например, если реализация TCP начинает все соединения с порядковым номером 0, после перезагрузки хоста партнёр TCP может восстановить прежнее соединение (возможно, после устранения полуоткрытого состояния) и передавать пакеты с номерами, которые идентичны или перекрываются с номерами остающихся в сети пакетов, отправленных прежней инкарнацией того же соединения. При отсутствии сведений о порядковых номерах, используемых в конкретном соединении, спецификация TCP рекомендует источнику задерживать отправку сегментов в соединении на MSL секунд, чтобы старые сегменты из прежнего соединения могли выйти из системы.

Даже хосты, способные помнить время суток и использовать его для выбора начальных порядковых номеров, не застрахованы от этой проблемы (даже при использовании времени суток при выборе начального номера для каждой новой инкарнации соединения).

Предположим, например, что соединение начинается с порядкового номера S. пусть это соединение используется мало и в конечном итоге функция начального порядкового номер (ISN(t)) принимает значение S1 последнего сегмента, переданного этой конечной точкой TCP в конкретном соединении. Предположим теперь, что в этот момент хост перезагружается и создаёт новую инкарнацию соединения. В качестве начального номера выбирается S1 = ISN(t) - последний номер, использованный в прежней инкарнации соединения! Если восстановление произошло достаточно быстро, старые дубликаты с номерами около S1 могут быть получены из сети и восприняты как новые пакеты получателем в новой инкарнации соединения.

Проблема состоит в том, что восстанавливающийся хост не может знать, как долго он не работал в процессе перезагрузки, а также у него не может быть сведений о наличии в системе старых дубликатов прежней инкарнации.

Одним из способов решения этой проблемы является преднамеренная задержка передачи сегментов на время MSL после восстановления при перезагрузке, т. е. «время тишины». Хосты, не желающие ждать и готовые к возможной путанице между старыми и новыми сегментами у данного адресата, могут отказаться от ожидания. Разработчики могут предоставить пользователям TCP возможность управления ожиданием на уровне каждого соединения или неформально реализовать «время ожидания» для всех соединений. Очевидно, что даже при выборе пользователем ожидания, в нем не будет необходимости после того, как хост проработает после загрузки хотя бы MSL секунд.

Подводя итог скажем, что каждый переданный сегмент занимает хотя бы 1 порядковый номер в пространстве номеров и использованные номера будут «заняты», пока не пройдёт MSL секунд. При перезагрузке блок, занимаемый октетами данных и флагом SYN или FIN может оставаться в находящихся в пути сегментах. Если новое соединение запускается слишком быстро и использует номера из находящихся в пути сегментов прежней инкарнации, возможно перекрытие порядковых номеров, способное вызвать путаницу у получателя.

В высокоскоростных системах циклы порядковых номеров будет короче, чем при мегабитных скоростях, которые учтены в описанном выше базовом устройстве TCP. При скорости 1 Гбит/с цикл занимает 34 секунды, при 10 Гбит/с - 3 секунды, а при 100 Гбит/с - лишь треть секунды. В таких скоростных системах опции TCP Timestamp и защита от перехода номеров через максимум (Protection Against Wrapped Sequences или PAWS) [47] обеспечивают требуемые возможности обнаружения и отбрасывания старых дубликатов.

3.5. Организация соединения

Процедура трехэтапного согласования (three-way handshake) служит для организации соединений. Обычно процедуру инициирует один партнёр TCP, а другой отвечает. Процедура работает и в случае одновременного инициирования обоими партнёрами. В этом случае каждый партнёр TCP получает сегмент SYN без подтверждения после отправки им своего SYN. Конечно, прибытие старого дубликата SYN может создать у получателя иллюзию одновременной организации соединения. Подобающее использование сегментов сброса (reset) может устранить неоднозначность.

Ниже приведено несколько примеров инициирования соединений. Хотя в примерах не показана синхронизация соединений с использованием сегментов с данными, это возможно, если принимающая сторона TCP не доставляет данные пользователю до проверки их действительности (например, данные хранятся в буфере, пока соединение не достигнет состояния ESTABLISHED с учётом того, что трехэтапное согласования снижает вероятность ложных соединений).

Простейший пример 3WHS показан на рисунке 6. каждая строка на рисунке пронумерована, стрелка вправо (-->) указывает прибытие сегмента TCP от партнёра A к партнёру B, а стрелка влево (<--) прибытие сегмента в обратном направлении. Многоточия (…) указывают сегменты, остающиеся в сети (задержанные). Комментарии приведены в скобках. Состояния соединения TCP показаны после отправки или прибытия сегмента, указанного в строке. Содержимое сегментов дано в сокращённой форме с указанием номера, флагов управления и поля ACK. Другие поля, такие как окно, адреса, данные для краткости не показаны на рисунках.

В строке 2 на рисунке 6 партнёр A начинает отправку сегмента SYN, указывающего использованием порядковых номеров начиная с 100. В строке 3 партнёр B передаёт SYN и подтверждение SYN, полученного от партнёра A. Отметим, что поле подтверждения указывает, что партнёр B ожидает порядковый номер 101, подтверждая SYN с номером 100. В строке 4 партнёр A отвечает пустым сегментом с ACK для SYN от партнёра B, а в строке 5 партнёр A передаёт данные. Отметим, что номера в строках 4 и 5 совпадают, поскольку ACK не занимает пространство порядковых номеров (если это делать, придётся подтверждать ACK).

```
    Партнёр TCP A                                          Партнёр TCP B
1.  CLOSED                                                 LISTEN
2.  SYN-SENT    --> <SEQ=100><CTL=SYN>               --> SYN-RECEIVED
3.  ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>  <-- SYN-RECEIVED
4.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>       --> ESTABLISHED
5.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA> --> ESTABLISHED
```

Рисунок 6. Базовое 3-этапное согласования для синхронизации соединения.

Одновременное инициирование несколько сложнее и показано на рисунке 7. Состояния каждого из партнёров TCP проходят цикл CLOSED → SYN-SENT → SYN-RECEIVED → ESTABLISHED. Реализация TCP **должна** поддерживать одновременные попытки организации соединения (MUST-10).

```
    Партнёр TCP A                                          Партнёр TCP B
1.  CLOSED                                                 CLOSED
2.  SYN-SENT     --> <SEQ=100><CTL=SYN>                    ...
3.  SYN-RECEIVED <-- <SEQ=300><CTL=SYN>                    <-- SYN-SENT
4.               ... <SEQ=100><CTL=SYN>                    --> SYN-RECEIVED
5.  SYN-RECEIVED --> <SEQ=100><ACK=301><CTL=SYN,ACK>       ...
6.  ESTABLISHED  <-- <SEQ=300><ACK=101><CTL=SYN,ACK>       <-- SYN-RECEIVED
7.               ... <SEQ=100><ACK=301><CTL=SYN,ACK>       --> ESTABLISHED
```

Рисунок 7. Одновременная синхронизация соединения.

Отметим, что реализация TCP **должна** отслеживать, перешло соединение в SYN-RECEIVED в результате пассивного или активного вызова OPEN (MUST-11).

Основной причиной использования трехэтапного согласования является предотвращение путаницы из-за старых дубликатов инициирования соединений. Для этого задано специальное управляющее соединение сброса (reset). Если принимающий партнёр TCP находится в несинхронизированном состоянии (SYN-SENT, SYN-RECEIVED), он возвращается в состояние LISTEN при получении приемлемого сброса. Если же партнёр TCP находится в одном из синхронизированных состояний (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), он прерывает соединение и информирует пользователя. Это описано более подробно ниже при рассмотрении полуоткрытых (half-open) соединений.

```
    Партнёр TCP A                                              Партнёр TCP B
1.  CLOSED                                                     LISTEN
2.  SYN-SENT    --> <SEQ=100><CTL=SYN>                     ...
3.  (duplicate) ... <SEQ=90><CTL=SYN>                  --> SYN-RECEIVED
4.  SYN-SENT    <-- <SEQ=300><ACK=91><CTL=SYN,ACK>  <-- SYN-RECEIVED
5.  SYN-SENT    --> <SEQ=91><CTL=RST>                  --> LISTEN
6.              ... <SEQ=100><CTL=SYN>                  --> SYN-RECEIVED
7.  ESTABLISHED <-- <SEQ=400><ACK=101><CTL=SYN,ACK>  <-- SYN-RECEIVED
8.  ESTABLISHED --> <SEQ=101><ACK=401><CTL=ACK>         --> ESTABLISHED
```

Рисунок 8. Восстановление при старом дубликате SYN.

Простой пример восстановления от старых дубликатов показан на рисунке 8. В строке 3 старый дубликат SYN приходит партнёру B. Тот не может сказать, что это старый дубликат и отвечает нормально (строка 4). Партнёр A обнаруживает некорректность поля ACK и возвращает RST (сброс) с полем SEQ, делающим сегмент правдоподобным. Партнёр B при получении RST возвращается в состояние LISTEN. При получении в конце концов исходного SYN (строка 6) синхронизация обрабатывается нормально. Если SYN из строки 6 приходит до RST, может происходить более сложный обмен с передачей RST в обоих направлениях.

3.5.1. Полуоткрытые соединения и другие аномалии

Организованное соединение называют полуоткрытым (half-open), если один из партнёров TCP закрыл или прервал его без ведома другого партера или стороны соединения рассинхронизированы в результате отказа или перезагрузки с потерей памяти. Такие соединения автоматически сбрасываются при попытке передачи данных в любом из направлений. Однако предполагается, что такие соединения будут необычными.

Если на сайте A соединения больше нет, попытка пользователя сайта B передать данные будет приводить к получению сайтом B управляющего сообщения reset, показывающего конечной точке сайта B наличие какой-то проблемы и ожидаемый разрыв соединения.

Предположим, что пользовательские процессы A и B взаимодействуют между собой и происходит отказ или перезагрузка с потерей памяти в реализации на стороне A. В зависимости от операционной системы, поддерживающей реализацию A, вполне вероятен некий механизм восстановления при ошибках. Когда конечная точка TCP заново активизируется, процесс A, скорей всего, запустится с начала или с точки восстановления. В результате A, вероятно, вызовет OPEN для соединения или попытается использовать для него SEND, считая соединение открытым. В последнем случае он получит сообщение об ошибке connection not open (соединение не открыто) от локальной реализации TCP A. При попытке организовать соединение реализация TCP A будет передавать сегмент с флагом SYN. Это ведёт к ситуации, показанной на рисунке 9. После перезагрузки партнёра A пользователь пытается заново организовать соединение, а партнёр B считает соединение открытым.

```
    Партнёр TCP A                                          Партнёр TCP B
1.  (REBOOT)                                       (send 300,receive 100)
2.  CLOSED                                                 ESTABLISHED
3.  SYN-SENT --> <SEQ=400><CTL=SYN>                    --> (??)
4.  (!!)     <-- <SEQ=300><ACK=100><CTL=ACK>           <-- ESTABLISHED
5.  SYN-SENT --> <SEQ=100><CTL=RST>                    --> (Abort!!)
6.  SYN-SENT                                               CLOSED
7.  SYN-SENT --> <SEQ=400><CTL=SYN>                    -->
```

Рисунок 9. Обнаружение полуоткрытого соединения.

Когда SYN приходит партнёру B (строка 3), тот находится в синхронизированном состоянии, но входящий сегмент не попадает в окно и в ответ на него передаётся подтверждение, указывающее следующий ожидаемый номер (ACK 100). Партнёр A видит, что этот сегмент подтверждает данные, которых он не передавал и, будучи несинхронизированным, передаёт сбров (RST), поскольку обнаруживает полуоткрытое соединение. Партнер B прерывает соединение в строке 5. Партнёр A продолжает попытку организовать соединение, используя базовое 3-этапное согласование (Рисунок 6).

Интересный вариант возникает, когда партнёр A перезагружается, а B пытается передать данные, считая соединение синхронизированным (Рисунок 10). В этом случае данные, прибывшие партнёру A от партнёра B (строка 2) не приемлемы, поскольку соединения нет, поэтому партнёр A передаёт RST. Сегмент RST приемлем для B и тот обрабатывает его, разрывая (abort) соединение.

На рисунке 11 показаны партнёры TCP A и B с пассивными соединениями, ожидающими SYN. Прибытие старого дубликата (строка 2) побуждает партнёра B действовать. Он возвращает SYN-ACK (сторока 3), который вызывает у

```
    Партнёр TCP A                                        Партнёр TCP B
1.  (REBOOT)                                        (send 300,receive 100)
2.  (??)    <-- <SEQ=300><ACK=100><DATA=10><CTL=ACK> <-- ESTABLISHED
3.          --> <SEQ=100><CTL=RST>                      --> (ABORT!!)
```

Рисунок 10. Активная сторона вызывает обнаружение полуоткрытого соединения.

партнера A генерацию RST (ACK в строке 3 не приемлем). Партнер Peer B воспринимает сброс и возвращается в состояние LISTEN.

```
    Партнёр TCP A                                    Партнёр TCP B
1.  LISTEN                                           LISTEN
2.        ... <SEQ=Z><CTL=SYN>                 -->   SYN-RECEIVED
3.  (??) <-- <SEQ=X><ACK=Z+1><CTL=SYN,ACK>     <--   SYN-RECEIVED
4.        --> <SEQ=Z+1><CTL=RST>               -->   (return to LISTEN!)
5.  LISTEN                                           LISTEN
```

Рисунок 11. Старый дубликат SYN вызывает Reset на двух пассивных сокетах.

Возможны другие варианты и все они обрабатываются по приведённым ниже правилам создания и обработки RST.

3.5.2. Генерация Reset

Пользователь или приложение TCP может инициировать сброс соединения в любой момент с помощью события reset, кроме того, такие события может генерировать сам протокол при возникновении ошибок, как указано ниже. Инициировавшей сброс стороне соединения следует перейти в состояние TIME-WAIT, поскольку это обычно помогает снизить нагрузку на загруженные серверы по причинам, описанным в [70].

Как правило сброс (RST) передаётся всякий раз при получении сегмента, который представляется не предназначенным для текущего соединения. Сброс недопустимо инициировать, если не очевидно, что это так.

Имеется три группы состояний, описанных ниже.

1. Если соединения не существует (CLOSED), сброс передаётся в ответ на любой входящий сегмент, за исключением reset. Таким образом, сегмент SYN не соответствующий имеющемуся соединению, отвергается.

   Если во входящем сегменте установлен флаг ACK, сброс принимает порядковый номер из поля ACK в сегменте, в иных случаях сброс имеет номер 0, а в поле ACK помещается сумма порядкового номера и размера входящего сегмента. Соединение сохраняет состояние CLOSED.

2. Если соединение находится в несинхронизированном состоянии (LISTEN, SYN-SENT, SYN-RECEIVED) и входящий сегмент подтверждает что-то ещё не отправленное (содержит неприемлемое значение ACK), имеет уровень защиты или изоляции (compartment, A.1. IP Security Compartment и Precedence) не соответствуют запрошенному для соединения уровню изоляции, передаётся сброс.

   Если во входящем сегменте установлен флаг ACK, сброс принимает порядковый номер из поля ACK в сегменте, в иных случаях сброс имеет номер 0, а в поле ACK помещается сумма порядкового номера и размера входящего сегмента. Соединение сохраняет своё состояние.

3. Если соединение находится в синхронизированном состоянии (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), на любой неприемлемый сегмент (номер за пределами окна или неприемлемый номер подтверждения) должен возвращаться пустой сегмент подтверждения (без пользовательских данных) с текущим порядковым номером передачи и подтверждением, указывающим ожидаемый следующим на приёме номер, а соединение остаётся в прежнем состоянии.

   Если входящий сегмент имеет уровень защиты или изоляцию, не соответствующие установленному для соединения, передаётся сброс и соединение переходит в состояние CLOSED. Для сброса применяется порядковый номер из поля ACK во входящем сегменте.

3.5.3. Обработка Reset

В любом состоянии, кроме SYN-SENT, все сегменты сброса (RST) проверяются по полям SEQ. Сброс действителен, если его номер попадает в окно. В состоянии SYN-SENT (RST принимается в ответ на начальный SYN) сегмент RST может быть воспринят, если поле ACK подтверждает SYN.

Получатель RST сначала проверяет сегмент, затем меняет состояние. Если получатель был в состоянии LISTEN, он игнорирует сброс. Если получатель был в состоянии SYN-RECEIVED, а ранее - в LISTEN, он возвращается в состояние LISTEN, в иных случаях - прерывает соединение и переходит в состояние CLOSED. Если получатель был в любом другом состоянии, он прерывает соединение, уведомляя пользователя, и переходит в состояние CLOSED.

Реализациям TCP **следует** разрешать включение данных в сегмент RST (SHLD-2). Высказаны предложение по включению в RST диагностических данных, объясняющих причину RST. Для таких данных ещё нет стандарта.

3.6. Закрытие соединения

Операция CLOSE означает: «У меня больше нет данных для передачи.» Понятие закрытия полнодуплексного соединения можно толковать по-разному, поскольку может быть не очевидно, как трактовать принимающую сторону соединения. Поэтому выбрана симплексная трактовка CLOSE. Пользователь, применяющий CLOSE, может продолжать RECEIVE, пока получателю TCP не будет сказано, что удалённый партнёр также перешёл в состояние CLOSED. Таким образом, программа может инициировать несколько SEND, за которыми следует CLOSE и после этого продолжать RECEIVE, пока не будет сигнала о невозможности RECEIVE по причине перехода удалённого партнёра в состояние CLOSED. Реализация TCP будет сигнализировать пользователю о закрытии удалённого партнёра даже при отсутствии остающихся RECEIVE, поэтому пользователь может аккуратно завершить работу на своей стороне. Реализация TCP будет гарантированно доставлять все буферы SENT до перехода в состояние CLOSED, поэтому пользователю, не ожидающему данных в ответ, достаточно лишь подождать сведений о переходе соединения в состояние CLOSED, чтобы знать о доставке всех данных удалённой точке TCP. Пользователи должны продолжать считывание соединения, пока реализация TCP не укажет, что данных больше нет.

Здесь следует рассмотреть три особых случая:

1) действие пользователя, просящего реализацию TCP закрыть (CLOSE) соединение (TCP Peer A на рисунке 12);

2) передача удалённой точкой TCP сигнала управления FIN (TCP Peer B на рисунке 12);

3) оба пользователя одновременно применяют CLOSE (Рисунок 13).

Инициатива пользователя по закрытию

В этом случае сегмент FIN может быть создан и помещён в очередь исходящих сегментов. Дальнейшие SEND от пользователя реализация TCP не воспринимает и переходит в состояние FIN-WAIT-1. В этом состоянии разрешены RECEIVE. Все прежние сегменты, включая FIN будут передаваться (повторно), пока не будут подтверждены. Когда удалённый партнёр TCP подтвердил FIN и передал свой сегмент FIN, первый партнёр TCP может подтвердить (ACK) этот сегмент FIN. Отметим, что получившая FIN конечная точка TCP будет передавать ACK но не будет передавать свой сегмент FIN, пока пользователь не закрыл соединение (CLOSED).

Получение конечной точкой TCP сегмента FIN из сети

Если из сети приходит незапрошенный сегмент FIN, принимающая конечная точка TCP может подтвердить его (ACK) и сообщить пользователю о закрытии соединения. Пользователь будет отвечать вызовом CLOSE, по которому конечная точка может передать FIN партнёру TCP после отправки остающихся данных. Конечная точка TCP может ждать подтверждения своего сегмента FIN, после чего удалит соединение. Если ACK не приходит, соединение прерывается по тайм-ауту пользователя и тот уведомляется об этом.

Одновременное закрытие соединения пользователями

Одновременный вызов CLOSE пользователями на обеих сторонах соединения вызывает обмен сегментами FIN (Рисунок 13). Когда все сегменты, предшествующие FIN обработаны и подтверждены, каждый из партнёров TCP может подтвердить (ACK) полученный сегмент FIN. При получении ACK оба будут удалять соединение.

```
    Партнёр TCP A                                          Партнёр TCP B
1.  ESTABLISHED                                            ESTABLISHED
2.  (Close)
    FIN-WAIT-1  --> <SEQ=100><ACK=300><CTL=FIN,ACK>  --> CLOSE-WAIT
3.  FIN-WAIT-2  <-- <SEQ=300><ACK=101><CTL=ACK>      <-- CLOSE-WAIT
4.                                                         (Close)
    TIME-WAIT   <-- <SEQ=300><ACK=101><CTL=FIN,ACK>  <-- LAST-ACK
5.  TIME-WAIT   --> <SEQ=101><ACK=301><CTL=ACK>      --> CLOSED
6.  (2 MSL)
    CLOSED
```

Рисунок 12. Последовательность нормального закрытия.

```
    Партнёр TCP A                                          Партнёр TCP B
1.  ESTABLISHED                                            ESTABLISHED
2.  (Close)                                                (Close)
    FIN-WAIT-1  --> <SEQ=100><ACK=300><CTL=FIN,ACK>  ... FIN-WAIT-1
                <-- <SEQ=300><ACK=100><CTL=FIN,ACK>  <--
                ... <SEQ=100><ACK=300><CTL=FIN,ACK>  -->
3.  CLOSING     --> <SEQ=101><ACK=301><CTL=ACK>      ... CLOSING
                <-- <SEQ=301><ACK=101><CTL=ACK>      <--
                ... <SEQ=101><ACK=301><CTL=ACK>      -->
4.  TIME-WAIT                                              TIME-WAIT
    (2 MSL)                                                (2 MSL)
    CLOSED                                                 CLOSED
```

Рисунок 13. Последовательность одновременного закрытия.

Соединение TCP может разрываться двумя способами: (1) нормальная последовательность закрытия TCP с применением обмена FIN (Рисунок 12) и (2) прерывание (abort) с передачей одного или нескольких сегментов RST и немедленным отбрасыванием состояния соединения. Если локальное соединение TCP закрыто удалённой точкой с помощью передачи FIN или RST, локальное приложение должно быть проинформировано о нормальном завершении или прерывании (MUST-12).

3.6.1. Полузакрытые соединения

При нормальной последовательности завершения TCP буферизованные данные гарантированно длставляются в обоих направлениях. Поскольку каждое направление в TCP закрывается независимо, созможны полузакрытые (half closed) соединения (т. е. закрыто лишь одно направление), когда хост может продолжать передачу данных в открытом направлении полузакрытого соединения.

Хост **может** реализовать полудуплексную последовательность закрытия TCP так, что вызвавшее CLOSE приложение может читать данные из соединения (MAY-1). Если такой хост получает вызов CLOSE и в соединении ещё остаются ожидающие приёма данные или новые данные получены после вызова CLOSE, реализации TCP на этом хосте **следует** передать RST для индикации потери данных (SHLD-3). Это рассмотрено в параграфе 2.17 [23].

Когда соединение закрывается активно, оно **должно** сохранять состояние TIME-WAIT в течение 2xMSL (максимальный срок жизни сегмента) (MUST-13). Однако оно **может** воспринять новый сегмент SYN от удалённой точки TCP для повторного открытия соединения напрямую из состояния TIME-WAIT (MAY-2)

(1) если в новом соединении задан начальный порядковый номер больше максимального номера в прежнем соединении, создаётся новое соединение;

(2) если SYN оказывается старым дубликатом, сохраняется состояние TIME-WAIT.

При доступности опций TCP Timestamp улучшенный алгоритм [40] повышает скорость организации соединений. Этот алгоритм сокращения TIME-WAIT, основанный на опыте (Best Current Practice), **следует** применять, поскольку опции Timestamp используются широко и это снижает TIME-WAIT, что важно для загруженных серверов Internet (SHLD-4).

3.7. Сегментация

Термин сегментация обозначает действия TCP, выполняемые при восприятии потока байтов от передающего приложения и пакетизации этого потока абйтов в сегменты TCP. Отдельные сегменты TCP зачастую не отображаются «один к одному» в отдельные вызовы send (или записи в сокет) из приложения. Приложения могут выполнять запись на уровне сообщений в протоколе вышележащего уровня, но TCP не гарантирует сопоставления границ передаваемых и принимаемых сегментов TCP с границами буферов чтения и записи данных пользовательского приложения. В некоторых протоколах, таких как удалённый прямой доступ к памяти (Remote Direct Memory Access или RDMA) с использованием прямого размещения данных (Direct Data Placement или DDP) и маркерного кадрирования с выравниванием PDU (Marker PDU Aligned Framing или MPA) [34], возможна оптимизация производительности при доступности контроля над сопоставлением сегментов TCP и блоков данных приложенийа MPA включает конкретный механизм обнаружения и проверки взаимосвязи между сегментами TCP и структурами данных пользовательских сообщений, но этот метод специфичен для таких приложений, как RDMA. В общем случае на размер сегментов, создаваемых реализацией TCP влияет множество целей и параметров.

Цели, ведущие к отправке больших сегментов, включают:

- снижение числа находящихся в сети пакетов;
- повышение эффективности обработки и возможной производительности за счёт меньшего числа прерываний и межуровневых взаимодействий;
- снижение издержек, связанных с заголовками TCP.

Отметим, что выигрыш в производительности за счёт роста размеров может сокращаться по мере увеличения и возможны границы, после которых рост размера будет снижать производительность. Например, в реализациях для некоторых вариантов архитектуры производительность при размере сегмента 1025 байтов может быть ниже, чем при 1024 байтах в результате выравнивания данных при операциях копирования.

Цели, ведущие к снижению размера сегментов, включают:

- предотвращение передачи сегментов TCP, которые ведут к размеру дейтаграмм IP больше минимального MTU на пути через сеть IP, что ведёт к потере или фрагментации дейтаграмм, кроме того, некоторые межсетевые экраны и промежуточные устройства могут отбрасывать фрагменты пакетов или связанные с ними сообщения ICMP;
- предотвращение задержки потока данных приложения, особенно в случае ожидания протоколом TCP от приложения генерации дополнительных данных или ожидании приложением события или входных данных от партнёра для генерации последующих данных;
- разрешение «общей судьбы» (fate sharing) для сегментов TCP и блоков данных нижележащего уровня (например, ниже IP для каналов с ячейками или кадрами меньше IP MTU).

Для согласования этих противоречивых целей в TCP включено несколько механизмов, таких как опция максимального размера сегмента (Maximum Segment Size или MSS), Path MTU Discovery, алгоритм Nagle и поддержка IPv6 Jumbogram, которые описаны в последующих параграфах.

3.7.1. Максимальный размер сегмента (MSS)

Конечные точки TCP **должны** реализовать приём и передачу опции MSS (MUST-14).

Реализациям TCP **следует** передавать опцию MSS в каждом сегменте SYN при MSS для приёма меньше принятого по умолчанию значения 536 для IPv4 и 1220 для IPv6 (SHLD-5) и **можно** передавать её всегда (MAY-3).

Если опция MSS не получена при организации соединения, реализация TCP **должна** предполагать принятое по умолчанию значение MSS 536 (576 - 40) для IPv4 и 1220 (1280 - 60) для IPv6 (MUST-15).

Максимальный размер передаваемого конечной точкой TCP сегмента или эффективное значение MSS для передачи (effective send MSS) **должно** быть меньшим из значений (MUST-16) SendMSS (отражает доступный размер буфера сборки на приёмной стороне EMTU_R [19]) и наибольшим размером передачи, разрешаемым уровнем IP (EMTU_S [19])

```
Eff.snd.MSS = min(SendMSS+20, MMS_S) - TCPhdrsize - IPoptionsize
```

с учётом приведённых ниже сведений.

- SendMSS - значение MSS, полученное от удалённого хоста или принятый по умолчанию размер 536 для IPv4 и 1220 для IPv6, если опция MSS не была получена.
- MMS_S - максимальный размер сообщения транспортного уровня, которое может передать TCP.
- TCPhdrsize - размер фиксированного заголовка TCP и всех опций. Это 20 в (редком) случае отсутствия опций и увеличивается при добавлении опций TCP. Отметим, что некоторые опции могут включаться не во все сегменты и при передаче каждого сегмента отправителю следует корректировать значение в Eff.snd.MSS.
- IPoptionsize - размер всех опций IPv4 или заголовков расширения IPv6, связанных с соединением TCP. Отметим, что некоторые опции и заголовки расширения могут включаться не во все сегменты и при передаче каждого сегмента отправителю следует корректировать значение в Eff.snd.MSS.

Значению MSS для передачи в опции MSS следует указывать эффективное значение MTU за вычетом фиксированных заголовков IP и TCP. При игнорировании опций IP и TCP в расчёте значения опции MSS и наличии в пакете каких-либо опций IP или TCP отправитель должен соответственно сократить размер данных TCP (см. RFC 6691 [43]).

Значение MSS для передачи в опции MSS должно быть не больше MMS_R - 20, где MMS_R - максимальный размер сообщения транспортного уровня, которое может быть принято (и собрано на уровне IP) (MUST-67). TCP получает значения MMS_R и MMS_S от уровня IP (см. базовый вызов GET_MAXSIZES в параграфе 3.4 RFC 1122). Эти значения определены на основе эквивалентов IP MTU - EMTU_R и EMTU_S [19].

При использовании TCP в ситуации с переменным размером заголовков IP или TCP отправитель должен уменьшить количество данных TCP в данном пакете на число октетов в опциях IP и TCP. С этим связана историческая путаница, рассмотренная в параграфе 3.1 RFC 6691.

3.7.2. Определение Path MTU

Реализация TCP может знать MTU подключённых напрямую каналов, но очень редко имеет сведения о MTU всех путей через сеть. Для IPv4 в RFC 1122 рекомендуется применять на уровне IP по умолчанию MTU не более 576 для адресатов, не подключённых напрямую, а для IPv6 значение составляет 1280. Использование этих фиксированных значений ограничивает производительность и эффективность соединения TCP. Взамен настоятельно рекомендуется реализовать механизм определения MTU для пути (Path MTU Discovery или PMTUD) и обнаружение MTU уровня пакетизации для пути (Packetization Layer Path MTU Discovery или PLPMTUD) в TCP, чтобы сделать сегментацию более эффективной. Оба механизма PMTUD и PLPMTUD помогают TCP выбрать размер сегментов так, чтобы избежать фрагментации в пути (IPv4) или у отправителя (IPv4 и IPv6).

PMTUD для IPv4 [2] или IPv6 [14] реализуется в сочетании TCP, IP и ICMP. Механизм основан на предотвращении фрагментации у отправителя и установке флага IPv4 DF (не фрагментировать), позволяющего предотвратить фрагментацию в пути. Механизм опирается на сообщения ICMP от маршрутизаторов в пути, передаваемые, когда сегмент слишком велик для передачи через канал. Некоторые настройки реализации TCP при использовании PMTUD описаны в RFC 2923 и позволяют решить возникающие на практике проблемы [27]. PLPMTUD [31] является стандартным (Standards Track) развитием PMTUD, смягчающим требования к поддержке ICMP на пути и повышающим производительность в случаях, когда сообщения ICMP не передаются согласованно, с сохранением передачи без фрагментации. Механизмы во всех 4 упомянутых RFC рекомендуется включать в реализации TCP.

Опция TCP MSS задаёт верхнюю границу размера пакетов, которые могут быть приняты (см. [43]). Поэтому установка соишком малого значения опции MSS может влиять на способность PMTUD и PLPMTUD найти большие значения path MTU. В RFC 1191 обсуждается последствие установки в старых реализациях TCP значения TCP MSS = 536 (соответствует IPv4 MTU в 576 байтов) для нелокальных получателей вместо вывода его из MTU подключённых интерфейсов, как это рекомендуется.

3.7.3. Интерфейсы с переменным значением MTU

Эффективное значение MTU может иногда меняться, например, при использовании переменного сжатия, такого как ROHC (RObust Header Compression) [37]. Для реализации TCP заманчиво объявить наибольшее возможное значение MSS, чтобы поддерживать наиболее эффективное использование сжатых данных. К сожалению, некоторым схемам сжатия иногда требуется передавать полные заголовки (и меньше полезных данных) при ресинхронизации состояния на компрессорах и декомпрессорах конечных точек. Если использовать наибольшее значение MTU при расчёте анонсируемого значения опции MSS, повторная передача TCP может помещать ресинхронизации компрессора.

В результате при смене MTU на интерфейсе от пакета к пакету реализации TCP **следует** применять наименьшее значение MTU интерфейса при расчёте значения опции MSS (SHLD-6).

3.7.4. Алгоритм Nagle

Алгоритм Nagle описан в RFC 896 [17] и был рекомендован в RFC 1122 [19] для смягчения проблемы создания слишком большого числа мелких пакетов. Алгоритм реализован с большинстве современных баз кода TCP, иногда с теми или иными вариациями (см. A.3. Изменение алгоритма Nagle).

При наличии неподтвержденных данных (SND.NXT > SND.UNA) передающая конечная точка TCP буферизует все данные пользователя (независимо от бита PSH) пока остающиеся данные не будут подтверждены или конечная точка сможет передать полный сегмент TCP (Eff.snd.MSS байт).

Реализации TCP **следует** включать алгоритм Nagle для объединения коротких сегментов (SHLD-7). Однако у приложения **должен** быть способ отключить алгоритм Nagle на отдельном соединении (MUST-17). В любом случае отправка данных происходит с учётом ограничений, задаваемых алгоритмом замедленного старта [8].

Поскольку при взаимодействии алгоритма Nagle с отложенными подтверждениями могут возникать проблемы, некоторые реализации несколько изменяют алгоритм Nagle, например, как описано в Приложении A.3.

3.7.5. Джамбограммы IPv6

Для поддержки TCP с использованием IPv6 Jumbogram реализация должна быть способна передавать сегменты TCP размером больше 64 Кбайт, которые позволяет указать опция MSS. В RFC 2675 [24] указано, что значение MSS = 65535 байт считается бесконечным и применяется Path MTU Discovery [14] для определения фактического MSS.

Опцию Jumbo Payload не обязаны реализовать и понимать узлы IPv6, не поддерживающие подключение к каналам с MTU больше 65575 [24], а действующие требования к узлам IPv6 не включают поддержку Jumbogram [55].

3.8. Обмен данными

После организации соединения данные передаются путём обмена сегментами. Поскольку сегменты могут теряться из-за ошибок (несовпадение контрольной суммы) или перегрузки в сети, TCP использует повтор передачи для гарантированной доставки каждого сегмента. По причинам, связанным с сетью или повторной передачей TCP, могут возникать дубликаты сегментов. Как указано в параграфе 3.4. Порядковые номера, реализация TCP проверяет порядковые номера и номера подтверждений в сегментах перед их восприятием.

Отправитель данных сохраняет порядковый номер для следующей передачи в переменной SND.NXT, а старейший из неподтвержденных порядковых номеров - в переменной SND.UNA. Получатель данных хранит следующий ожидаемый номер в переменной RCV.NXT. Если все данные сразу же подтверждаются, значения этих 3 переменных совпадают.

Когда отправитель создает сегмент и передаёт его, он увеличивает значение SND.NXT. Получатель при приёме сегмента увеличивает RCV.NXT и передаёт подтверждение. При получении подтверждения отправителем, тот увеличивает SND.UNA. Степень различия этих переменных является мерой задержки в коммуникациях. Размер

увеличения переменных определяется суммарным размером данных и флага SYN или FIN в сегменте. Отметим, что после достижения состояния ESTABLISHED все сегменты должны содержать данные подтверждения.

Пользовательский вызов CLOSE предполагает функцию выталкивания (push) (параграф 3.9.1. Интерфейс TCP - пользователь), как при флага FIN во входящем сегменте.

3.8.1. Тайм-аут повтора передачи

Из-за изменчивости сетей, образующих многосетевую (internetwork) систему и широкого спектра применения соединений TCP тайм-аут повтора передачи (retransmission timeout или RTO) должен задаваться динамически. Значение RTO **должно** рассчитываться по алгоритмам из [10], включая алгоритм Karn для выборки RTT (MUST-18).

В RFC 793 приведена процедура расчёта RTO на основе работы, упомянутой в IEN 177 [71]. Это было заменено алгоритмом, описанным в RFC 1122, который был обновлён в RFC 2988, а затем - в RFC 6298.

RFC 1122 говорит, что при повторной передаче пакета, идентичного исходному (это предполагает совпадение не только границ данных, но и полей заголовка), **можно** использовать то же поле IPv4 Identification (см. параграф 3.2.1.5 в RFC 1122) (MAY-4). Одно и то же поле IP Identification можно применять повторно в любом случае, поскольку оно имеет значение лишь при фрагментации дейтаграмм [44]. Реализациям TCP не следует полагаться на это поле заголовка и обычно даже взаимодействовать с ним. Это неразумный способ указания передачи и получения дубликатов.

3.8.2. Контроль перегрузок TCP

В RFC 2914 [5] разъяснена важность контроля перегрузок в Internet.

RFC 1122 требует реализовать алгоритмы Van Jacobson для контроля перегрузок - замедленный старт (slow start) и предотвращение перегрузки (congestion avoidance) вместе с экспоненциальной отсрочкой (backoff) в последовательных RTO для одного и того же сегмента. В RFC 2581 дано стандартизованное (IETF Standards Track) описание slow start и congestion avoidance, а также механизмов ускоренного повтора (fast retransmit) и быстрого восстановления (fast recovery). RFC 5681 содержит текущее описание этих алгоритмов и текущую спецификацию Standards Track с рекомендациями по контролю перегрузок в TCP. В RFC 6298 описана экспоненциальная отсрочка для значений RTO, включая сохранение отложенного значения, пока последующий сегмент с новыми данными не будет передан и подтверждён без повтора передачи.

Конечная точка TCP **должна** реализовать базовые алгоритмы контроля перегрузок slow start, congestion avoidance и экспоненциальной отсрочки RTO для предотвращения условий коллапса насыщения (MUST-19). В RFC 5681 и RFC 6298 описаны широко применяемые базовые алгоритмы (IETF Standards Track). Имеется много других подходящих алгоритмов, которые широко распространены. Многие реализации TCP поддерживают набор разных алгоритмов, которые можно настроить для применения на конечной точке. Такие дополнительные алгоритмы **можно** реализовать при условии их соответствия спецификациям TCP из IETF Standards Track, как описано в RFC 2914, RFC 5033 [7], RFC 8961 [15] (MAY-18).

Явное уведомление о перегрузке (Explicit Congestion Notification или ECN) было определено в RFC 3168 и является усовершенствованием IETF Standards Track, имеющим много преимуществ [51]. Конечной точке TCP **следует** реализовать ECN, как описано в RFC 3168 (SHLD-8).

3.8.3. Отказы соединений TCP

Чрезмерные повторы передачи одного сегмента конечной точкой TCP говорят о том или ином отказе на удалённом хосте или в пути через сеть. Этот отказ может быть краткосрочным или продолжительным. Описанные ниже процедуры **должны** применяться при обработке чрезмерных повторов передачи сегментов данных (MUST-20).

(a) Имеется два пороговых значения R1 и R2 при измерении числа повторов передачи одного сегмента. R1 и R2 можно измерять по времени или числу повторов (с текущим RTO и соответствующей экспоненциальной отсрочкой в качестве коэффициента преобразования, при необходимости).

(b) Когда число повторов передачи одного сегмента становится не меньше порога R1, передаётся рекомендация (см. параграф 3.3.1.4 в [19]) уровню IP для вызова диагностики «мертвого шлюза».

(c) Когда число повторов передачи одного сегмента достигает порога R2 (> R1), соединение закрывается.

(d) Приложение **должно** (MUST-21) быть способно устанавливать порог R2 для конкретного соединения. Например, интерактивное приложение может установить для R2 значение infinity (бесконечно), отдающее пользователю управления разрывом соединения.

(e) Реализациям TCP **следует** информировать приложение о проблемах доставки (если это не отключено приложением, см. параграф 3.9.1.8. Асинхронные отчёты), по достижении порога R1 и до R2 (SHLD-9). Это позволит информировать пользователя, например, в программе удалённой регистрации в системе (login).

Значению R1 **следует** соответствовать по меньшей мере 3 повторам передачи при текущем значении RTO (SHLD-10). Для R2 **следует** устанавливать значение не менее 100 секунд (SHLD-11).

Попытка организации соединения TCP может приводить к отказу из-за чрезмерного повтора передачи сегмента SYN, получения сегмента RST или сообщения ICMP Port Unreachable. Повторы передачи SYN **должны** обрабатываться в общем порядке, приведённом выше для повторов, включая уведомление прикладного уровня. Однако значения R1 и R2 для SYN и сегментов данных могут различаться. В частности значение R2 для сегмента SYN **должно** быть достаточно большим, чтобы обеспечивать повтор передачи сегмента в течение по меньшей мере 3 минут (MUST-23). Приложение, естественно, может закрыть соединение (отказаться от попытки соединиться) раньше.

3.8.4. TCP Keep-Alive

Соединение TCP считается бездействующим (idle), если в течение некоторого времени через него не было принято ни одного входящего сегмента и передано новых или неподтверждённых данных.

Разработчики **могут** включать сообщения keep-alive в свои реализации TCP (MAY-5), хотя такая практика не является общепринятой. Некоторые реализации TCP, тем не менее, применяют механизм keep-alive для сохранения активности бездействующего соединения, передавая пробный сегмент, предназначенный для получения ответа от партнёра TCP. Такие сегменты обычно включают SEG.SEQ = SND.NXT-1 и могут содержать «сорные» октеты данных. Если сообщения keep-alive применяются, приложение **должно** быть способно включать и отключать их передачу для каждого соединения TCP (MUST-24) и по умолчанию сообщения **должны** быть отключены (MUST-25).

Пакеты keep-alive **должны** передаваться лишь при отсутствии данных для передачи и отсутствии принятых пакетов данных или подтверждений в течение заданного интервала (MUST-26), который **должен** быть настраиваемым (MUST-27) и по умолчанию **должен** составлять не менее 2 часов (MUST-28).

Очень важно помнить, что сегменты ACK без данных не передаются TCP с гарантией. Поэтому при реализации механизма keep-alive **недопустимо** рассматривать отсутствие отклика на конкретный тестовый пакет признаком «метвого» соединения (MUST-29).

Реализации **следует** передавать сегменты keep-alive без данных (SHLD-12), однако **можно** поддерживать настраиваемую передачу сегментов keep-alive с 1 октетом «сорных» данных (MAY-6) для совместимости с ошибочными реализациями TCP.

3.8.5. Обмен важными сведениями

Из-за различий в реализациях и взаимодействии с промежуточными устройствами новым приложениям **не следует** реализовать механизм TCP urgent (SHLD-13). Однако реализации TCP **должны** включать поддержку механизма срочности (MUST-30). Сведения об интерпретации некоторыми реализациями TCP указателя важности (срочности) приведены в RFC 6093 [39].

Цель механизма TCP urgent состоит в предоставлении передающему пользователю возможности стимулировать принимающего пользователя к восприятию неких важных (срочных) данных, а приёмной точке TCP - указать пользователю, когда все известные в данный момент важные данные будут получены им. Этот механизм позволяет указать в потоке данных точку, задающую конец важных данных. Всякий раз, когда эта точка опережает порядковый номер приёма (RCV.NXT) на принимающей стороне TCP, реализация TCP должна указать пользователю переход в режим срочности (urgent mode). Когда порядковый номер приёма «догоняет» указатель важности, реализация TCP должна указать пользователю переход в обычный режим. Если указатель важности меняется во время пребывания пользователя в режиме urgent, это обновление будет незаметно для пользователя.

Метод использует поле Urgent во всех передаваемых сегментах, а флаг URG указывает значимость этого поля и должен устанавливаться в сегментах, фактически содержащих указатель важности. Отсутствие флага говорит, что важных (срочных) данных в сегменте нет.

Для отправки важной информации пользователь должен передать хотя бы 1 октет. Если передающий пользователь указывает также выталкивание (push), это улучшает своевременность доставки важных данных целевому процессу. Поскольку указатель важности соответствует данным, записанным передающим приложением, значение указателя важности не может «отступить» в пространстве порядковых номеров, но получателям TCP следует быть устойчивыми к некорректному поведению указателя важности.

Реализации TCP **должны** поддерживать последовательности важных данных любого размера (MUST-31) [19].

Указатель важности **должен** содержать порядковый номер октета, следующего за важными данными (MUST-62).

Реализация TCP **должна** (MUST-32) асинхронно информировать уровень приложения о получении указателя важности (если ранее важных данных не было) или перемещении указателя важности вперёд. Реализация TCP **должна** (MUST-33) обеспечивать приложениям способ узнать объем остающихся в соединении важных данных или хотя бы определить остаются ли ещё важные данные для считывания [19].

3.8.6. Управление окном

Окно, передаваемое в каждом сегменте, указывает диапазон порядковых номеров отправителя окна (получателя данных), которые он готов воспринять в данный момент. Предполагается, что это связано с размером буферного пространства, доступного для этого соединения.

Передающая сторона TCP пакует данные для передачи в сегменты, заполняющие текущее окно, и может перепаковывать сегменты в очереди повторной передачи. Такая перепаковка не требуется, но может быть полезной.

В соединении с односторонним потоком данных информация об окне передаётся в сегментах подтверждения с одним порядковым номером, поэтому их порядок невозможно поменять при нарушении порядка доставки. Это не является серьёзной проблемой, но может время от времени давать сведения об окне по старым «отчетам» от получателя. Позволяющее избежать этой проблемы усовершенствование состоит в том, чтобы брать информацию об окне из сегмента с наибольшим порядковым номером подтверждения (т. е. с номером не меньше наибольшего из принятых последними).

Указание большего окна стимулирует передачу. Если поступает больше данных, чем можно воспринять, они будут отбрасываться. Это ведёт к избыточным повторам передачи, повышая загрузку сети и конечных точек TCP. Указание малого окна ограничивает передачу данных до такой степени, что может добавляться задержка в интервал кругового обхода (round-trip) между передачами сегментов.

Имеющиеся механизмы позволяют конечной точке TCP анонсировать большое окно, а затем анонсировать его уменьшение, не принимая так много данных. Так называемое «сокращение окна» (shrinking the window) настоятельно не рекомендуются. Принцип отказоустойчивости [19] диктует партнёрам TCP не сокращать окно самостоятельно, но быть готовыми в такому поведению партнёра TCP.

Получателю TCP **не следует** сокращать окно, т. е. сдвигать его правый край влево (SHLD-14). Однако передающий партнёр TCP **должен** быть готов к такому сокращению, которое может сделать размер доступного окна (usable window, см. 3.8.6.2.1. Алгоритм отправителя - когда передавать данные) отрицательным (MUST-34). Если это происходит, отправителю **не следует** передавать новых данных (SHLD-15), но **следует** обычным способом повторить передачу

неподтвержденных данных из интервала SND.UNA - SND.UNA+SND.WND (SHLD-16). Отправитель **может** также повторить старые данные сверх SND.UNA+SND.WND (MAY-7), но **не следует** фиксировать тайм-аут соединения, если данные за пределами правого края окна не подтверждены (SHLD-17). Если окно сокращается до 0, реализация TCP **должна** проверять его стандартным способом, описанным в следующем параграфе (MUST-35).

3.8.6.1. Проверка нулевого окна

Передающий партнёр TCP должен регулярно отправлять хотя бы 1 октет новых данных (при наличии) или повторно передавать данные принимающему партнёру TCP даже при нулевом размере окна с целью зондирования (probe) окна. Эти повторы передачи важны для гарантии того, что при установке одним из партнёров TCP нулевого размера окна это будет надёжно сообщено другому партнёру. В других документах это называется зондированием нулевого окна (Zero-Window Probing или ZWP). Зондирование (предложенных) нулевых окон **должно** поддерживаться (MUST-36).

Реализация TCP **может** сохранять предлагаемое окно приёма закрытым неограниченно долго (MAY-8). Пока принимающий партнёр TCP продолжает передавать подтверждения в ответ на сегменты зондирования, передающий партнёр TCP **должен** позволять соединению оставаться открытым (MUST-37). Это позволяет TCP работать в таких ситуациях, как отсутствие бумаги в принтере, описанное в параграфе 4.2.2.17 [19]. Поведение зависит от концепций управления ресурсами в реализации, как указано в [41].

Когда сегмент приходит принимающему партнёру TCP с нулевым окном, тот должен передать подтверждение, показывающее следующий ожидаемый порядковый номер и текущее окно (0). Передающему хосту **следует** отправлять первую пробу (зонд) наличия нулевого окна, когда такое окно существует в течение тайм-аута повтора передачи (SHLD-29) (см. параграф 3.8.1. Тайм-аут повтора передачи), а интервал между пробами **следует** увеличивать экспоненциально (SHLD-30).

3.8.6.2. Предотвращение SWS

«Синдром глупого окна» (Silly Window Syndrome или SWS) - это устойчивая картина небольших пошаговых изменений окна, приводящая к крайне низкой производительности TCP. Алгоритмы предотвращения SWS приведены ниже для передающей и принимающей стороны. Более подробное рассмотрение проблемы приведено в RFC 1122. Отметим, что алгоритм Nagle и алгоритм предотвращения SWS у отправителя дополняют друг друга в повышении производительности. Алгоритм Nagle препятствует отправке крошечных сегментов, когда данные для передачи поступают мелкими порциями, а алгоритм предотвращения SWS препятствует отправке мелких сегментов, возникающих в результате небольшого расширения окна справа.

3.8.6.2.1. Алгоритм отправителя - когда передавать данные

Реализация TCP должна включать алгоритм предотвращения SWS у отправителя (MUST-38). Алгоритм Nagle из параграфа 3.7.4 позволяет объединять короткие сегменты.

Алгоритм предотвращения SWS у отправителя может оказаться сложнее, чем у получателя, поскольку отправитель не знает (напрямую) общее буферное пространство получателя (RCV.BUFF). Было обнаружено, что хорошо работает подход на основе расчёта отправителем значения Max(SND.WND), которое определяет максимальное окно передачи, наблюдавшееся до этого в соединении, и использовании этого значения как оценки RCV.BUFF. К сожалению это лишь оценка и получатель в любой момент может изменить RCV.BUFF. Чтобы избежать в результате тупиковой ситуации, требуется тайм-аут принудительной передачи данных, отменяющий алгоритм предотвращения SWS. На практике такой тайм-аут будет возникать редко.

Применимое окно определяется значением

```
U = SND.UNA + SND.WND - SND.NXT
```

т. е. предложенное окно меньше объёма переданных, но не подтверждённых данных. Если D - объем ещё не отправленных данных в очереди отправителя на передачу, рекомендуется приведённые ниже правила передачи:

(1) если можно отправить сегмент максимального размера, т. е. min(D,U) >= Eff.snd.MSS;

(2) если данные выталкиваются (push) и можно отправить все данные из очереди, т. е. [SND.NXT = SND.UNA], применяется PUSH и D <= U (выражение в скобках вносит алгоритм Nagle);

(3) если можно отправить хотя бы часть Fs максимального окна передачи, т. е. [SND.NXT = SND.UNA] и min(D,U) >= Fs * Max(SND.WND);

(4) если возникает тайм-аут переопределения.

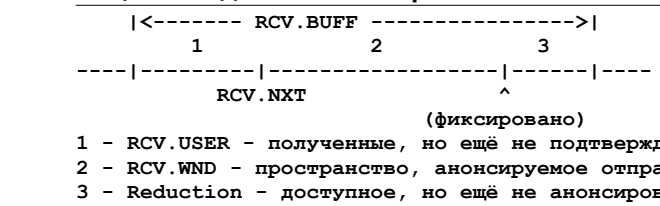
Здесь Fs - часть, для которой рекомендуемое значение составляет 1/2. Тайм-ауту переопределения следует быть в интервале 0,1 - 1,0 секунда. Может оказаться удобные объединение этого таймера с таймером для нулевого окна (3.8.6.1. Проверка нулевого окна).

3.8.6.2.2. Алгоритм получателя - когда передавать обновление окна

Реализация TCP должна включать алгоритм предотвращения SWS у получателя (MUST-39). Этот алгоритм у получателя определяет, когда можно сдвинуть вправо правый край окна, это обычно называют обновлением окна. Алгоритм сочетается с алгоритмом отложенных подтверждений (параграф 3.8.6.3) для определения, когда сегмент ACK с текущим окном можно действительно передать получателю.

Задачей алгоритма у получателя является предотвращение сдвига правого края окна RCV.NXT+RCV.WND малыми шагами даже при получении данных из сети в мелких сегментах. Предположим, что буфер приёма имеет размер RCV.BUFF. В любой момент RCV.USER октетов из этого общего объёма могут быть связаны с данными, которые приняты и подтверждены, но ещё не считаны пользовательским процессом. При находящемся в покое соединении RCV.WND = RCV.BUFF и RCV.USER = 0.

Сохранение правого края окна неизменным при поступлении и подтверждении данных требует, чтобы получатель предлагал не все своё буферное пространство, т. е. получатель должен указать значение RCV.WND, сохраняющее сумму RCV.NXT+RCV.WND постоянной при увеличении RCV.NXT. Таким образом все буферное пространство RCV.BUFF обычно делится на 3 части, как показано ниже.



1 - RCV.USER - полученные, но ещё не подтверждённые данные;
2 - RCV.WND - пространство, анонсируемое отправителю;
3 - Reduction - доступное, но ещё не анонсированное пространство.

Предложенный алгоритм предотвращения SWS для получателя состоит в удержании фиксированной суммы RCV.NXT+RCV.WND, пока не будет выполнено условие

RCV.BUFF - RCV.USER – RCV.WND >= min(Fr * RCV.BUFF, Eff.snd.MSS)

где Fr - часть (рекомендуемое значение 1/2), а Eff.snd.MSS - эффективное значение MSS для соединения (см. параграф 3.7.1. Максимальный размер сегмента (MSS)). При выполнении неравенства устанавливается RCV.WND = RCV.BUFF-RCV.USER.

Отметим, что общий эффект этого алгоритма заключается в увеличении RCV.WND с приращением Eff.snd.MSS (для реалистичных буферов приёма Eff.snd.MSS < RCV.BUFF/2). Отметим также, что получатель должен использовать своё значение Eff.snd.MSS, предполагая, что оно такое же, как у отправителя.

3.8.6.3. Отложенные подтверждения - когда передавать сегмент ACK

Хост, получающий поток сегментов данных TCP, может повысить эффективность сети и участвующих хостов, передавая подтверждения (ACK) не для каждого принятого сегмента. Это называется отложенным подтверждением (delayed ACK).

Конечной точке TCP **следует** реализовать delayed ACK (SHLD-18), но подтверждения не следует задерживать чрезмерно и задержка **должна** быть меньше 0,5 сек (MUST-40). ACK **следует** генерировать не реже, чем для каждого второго полноразмерного сегмента или получения 2*RMSS новых данных (RMSS - значение MSS, заданное конечной точкой TCP, принимающей сегменты, которые будут подтверждаться, или принятое по умолчанию значение MSS) (SHLD-19). Чрезмерная задержка ACK может нарушать синхронизацию приема-передачи и алгоритмы «тактирования» пакетов. Более подробно отложенные подтверждения рассмотрены в параграфе 4.2 RFC 5681 [8], включая рекомендации по незамедлительному подтверждению сегментов с нарушением порядка доставки, сегментов после пропуска в порядковых номерах или внутри пропущенного интервала для ускорения восстановления при потерях.

Отметим, что имеется несколько современных подходов к снижению числа ACK, включая общую разгрузку приёма (generic receive offload или GRO) [72], сжатие и прореживание ACK [28].

3.9. Интерфейсы

Здесь рассматриваются два интерфейса - с пользователем и нижележащим уровнем. Описана достаточно подробно модель интерфейса TCP - пользователь, а интерфейс с модулем протокола нижележащего уровня рассмотрен лишь кратко, поскольку его подробно описывает спецификация нижележащего протокола. Для случая IP отмечены некоторые значения параметров, которые реализации TCP могут использовать.

3.9.1. Интерфейс TCP - пользователь

Приведённые ниже функциональные описания пользовательских команд для реализации TCP являются в лучшем случае вымышленными, поскольку каждая операционная система имеет свои возможности. Поэтому следует предупредить читателей, что в разных реализациях TCP пользовательские интерфейсы могут отличаться. Однако все реализации TCP должны обеспечивать некий минимальный набор услуг, гарантирующий, что все реализации TCP поддерживают общую иерархию протоколов. В этом параграфе рассмотрены функциональные интерфейсы, требуемые в каждой реализации TCP. В параграфе 3.1 [53] указаны предоставляемые TCP примитивы, которые могут служить дополнительным справочником для разработчиков.

В последующих параграфах рассмотрены функциональные характеристики интерфейса TCP - пользователь. Используемые обозначения похожи на применяемые в вызовах процедур и функций в языках высокого уровня, но это не означает исключение вызовов прерываний (ловушек).

Описанные ниже пользовательские команды задают базовые функции, которые реализация TCP должна выполнять для поддержки взаимодействий между процессами. Реализации должны определять свой точный формат и могут обеспечивать комбинации или подмножества базовых функций в одном вызове. В частности, некоторые реализации могут захотеть автоматически создавать (OPEN) соединение при первом вызове пользователем SEND или RECEIVE.

Предоставляя средства межпроцессного взаимодействия, реализация TCP должна не только воспринимать команды, но и возвращать информацию обслуживаемому процессу. Такая информация включает:

(a) общие сведения о соединении (например, прерывания, удалённое закрытие, привязка к неуказанному удалённому сокету);

(b) отклики на конкретные команды пользователя, указывающие успех или соответствующий тип отказа.

3.9.1.1. Open

OPEN (local port, remote socket, active/passive [, timeout] [, Diffserv field] [, security/compartment] [, local IP address] [, options]) -> local connection name

Если флаг active/passive установлен для пассивного режима, это будет вызов LISTEN для входящих соединений. Пассивный вызов OPEN может указывать удалённый сокет полностью для восприятия конкретного соединения или не задавать его для восприятия любых соединений. Полностью заданный пассивный вызов можно сделать активным последующим исполнением SEND.

Блок управления передачей (TCB) создаётся и частично заполняется данными из параметров команды OPEN.

Каждый пассивный вызов OPEN создаёт новую запись для соединения в состоянии LISTEN или возвращает ошибку. Таким вызовам **недопустимо** оказывать влияние на созданные ранее записи соединений (MUST-41).

Поддерживающая одновременные соединения реализация TCP **должна** обеспечивать вызовы OPEN, функционально позволяющие приложению прослушивать (LISTEN) порт, связанный с локальным портом, находящимся в состоянии SYN-SENT или SYN-RECEIVED (MUST-42).

При активной команде OPEN конечная точка TCP сразу начинает процедуру синхронизации (создания) соединения.

При наличии тайм-аута он позволяет вызывающему установить время ожидания для всех данных, представленных TCP. Если данные не доставлены получателю в заданном интервале времени, конечная точка TCP будет разрывать соединение. В настоящее время по умолчанию глобально задано время ожидания 5 минут.

Реализация TCP или какая-либо компонента операционной системы (ОС) будет проверять полномочия пользователя на создание соединений с заданным значением поля Diffserv или security/compartment. Отсутствие Diffserv или спецификации security/compartment при вызове OPEN указывает использование принятых по умолчанию значений. TCP будет считать входящие запросы пригодными только при точном соответствии сведений security/compartment заданным при вызове OPEN.

Указанное пользователем значение Diffserv влияет лишь на исходящие пакеты, может быть изменено на пути через сеть и не имеет прямого отношения или влияния на принимаемые пакеты.

Локальное имя соединения реализация TCP возвращает пользователю и оно может служить в качестве краткого обозначения соединения, определяемого парой <локальный сокет, удалённый сокет>.

Необязательный параметр local IP address **должен** поддерживаться, чтобы можно было указать локальный адрес IP (MUST-43). Это позволяет приложениям выбирать локальный адрес IP на многодомных хостах. Пассивный вызов OPEN с параметром local IP address будет ждать запросы входящих соединений с этим адресом. Если параметр не задан, при пассивном вызове OPEN соединения будут ожидаться на всех адресах с последующей привязкой к использованному в запросе конкретному адресу.

При активном вызове OPEN заданный параметр local IP address будет использован при организации соединения. Если параметр не задан, хост будет выбирать подходящий локальный адрес IP (см. RFC 1122, параграф 3.3.4.2).

Если приложение на многодомном хосте не указывает локальный адрес IP при активном создании соединения TCP, реализация TCP **должна** запросить у уровня IP выбор локального адреса до отправки (первого) SYN (MUST-44). См. функцию GET_SRCADDR() в параграфе 3.4 RFC 1122.

В остальных случаях для этого соединения уже был принят или передан предыдущий сегмент и реализация TCP **должна** использовать локальный адрес из прежних сегментов (MUST-45).

Реализация TCP **должна** отвергать как ошибку локальный вызов OPEN с недействительным удаленным адресом IP (например, групповым или широковещательным) (MUST-46).

3.9.1.2. Send

```
SEND (local connection name, buffer address, byte count, URGENT flag [, PUSH flag] [, timeout])
```

Этот вызов ведёт к передаче данных из указанного пользователем буфера в заданное соединение. Если соединение не открыто вызов SEND считается ошибкой. Некоторые реализации могут разрешать пользователю сначала вызывать SEND, автоматически вызывая в этом случае OPEN. Это может быть, например, одним из способов включения данных приложения в сегменты SYN. Если вызывающий процесс не уполномочен использовать указанное соединение, возвращается ошибка.

Конечная точка TCP **может** реализовать флаг PUSH при вызове SEND (MAY-15). Если этот флаг не реализован, передающей точке TCP (1) **недопустимо** буферизовать данные на неопределённый срок (MUST-60) и (2) она **должна** установить бит PSH в последнем буферизованном сегменте (т. е. когда в очереди больше нет данных для передачи) (MUST-61). Далее в описании предполагается поддержка флага PUSH в вызовах SEND.

Если флаг PUSH установлен, приложение считает, что данные будут незамедлительно доставлены получателю, и в последнем сегменте TCP, созданном из буфера, будет установлен флаг PSH. Этот бит не является маркером записи и не зависит от границ сегмента. Передатчику **следует** при пакетизации данных исключать (collapse) последующие биты для отправки максимально возможного сегмента (SHLD-27).

Если флаг PUSH не установлен, данные могут объединяться с данными из последующих вызовов SEND для более эффективной передачи. Когда приложение делает серию вызовов SEND без установки флага PUSH, реализация TCP **может** агрегировать данные без их передачи (MAY-16). Отметим, что при использовании алгоритма Nagle реализации TCP могут буферизовать данные до передачи независимо от флага PUSH (см. параграф 3.7.4. Алгоритм Nagle).

Логически прикладная программа требует установки флага PUSH при вызове SEND всякий раз, когда ей нужно форсировать доставку данных, чтобы не возникало блокировки связи. Однако реализации TCP **следует** по возможности передавать сегменты максимального размера (SHLD-28) для повышения производительности (см. параграф 3.8.6.2.1. Алгоритм отправителя - когда передавать данные).

Новым приложениям **не следует** устанавливать флаг URGENT flag [39] из-за различий в реализациях и проблем с промежуточными устройствами (SHLD-13). Если флаг URGENT установлен, в сегментах, передаваемых получателю TCP будет задан указатель важности. Принимающий партнёр TCP будет сообщать о важности принимающему процессу, если данные, указатель важности задаёт, что предшествующие ему данные ещё не восприняты принимающим процессом. Целью флага URGENT служит для стимулирования получателя к обработке важных (срочных) данных при их получении. Число сигналов реализации TCP о важности данных не обязано совпадать с числом уведомлений принимающего пользователя о присутствии важных данных.

Если удалённый сокет не был задан при вызове OPEN, но соединение организовано (например, потому, что прослушиваемое соединение организовано в результате прибытия удалённого сегмента в локальный сокет), указанный буфер передаётся в подразумеваемый удалённый сокет. Пользователи, применяющие OPEN без указания удалённого сокета могут вызвать SEND без явного знания адреса этого сокета. Однако, если попытка вызвать SEND предшествует назначению удалённого сокета, возвращается ошибка. Пользователи могут определить статус соединения с помощью вызова STATUS. Некоторые реализации TCP могут уведомлять пользователя о привязке к незаданному сокету.

Если задан параметр timeout, он устанавливает текущий параметр тайм-аута для пользователя соединения.

В простейшей реализации SEND не будет возвращать управление передающему процессу, пока передача не будет завершена или не истечёт время ожидания. Однако в обоих случаях может возникать блокировка (например, обе стороны могут вызвать SEND, не вызвав RECEIVE) и производительность мала, поэтому такие простые решения не рекомендуются. Более изощрённая реализация будет сразу же возвращать управление, чтобы позволить процессу работать одновременно с сетевым вводом-выводом и обеспечить возможность одновременной работы нескольких SEND. Вызовы SEND обслуживаются в порядке поступления, поэтому конечная точка TCP помещает в очередь вызовы, которые она не может обработать сразу же.

Здесь неявно предполагается асинхронный пользовательский интерфейс, через который SEND позднее получает некий **сигнал** или псевдо-прерывание от обслуживающей конечной точки TCP. Другим вариантом является незамедлительный возврат отклика. Например, SEND может сразу возвращать локальное подтверждение, даже когда сегмент ещё не подтверждён удалённой конечной точкой TCP. Можно оптимистически предположить возможный успех, а при неудаче соединение все равно будет закрыто по тайм-ауту. В реализациях этого типа (синхронных) все равно будут какие-то асинхронные сигналы, но они будут иметь дело с самим соединением, а не с конкретными сегментами или буферами.

Чтобы различать индикацию ошибки или успеха разных вызовов SEND, может быть целесообразным возврат адреса буфера вместе с кодом отклика на запрос SEND. Сигналы TCP пользователю, рассматриваемые ниже, указывают сведения, которые следует возвращать вызвавшему процессу.

3.9.1.3. Receive

```
RECEIVE (local connection name, buffer address, byte count) -> byte count, URGENT flag [, PUSH flag]
```

Эта команда выделяет приёмный буфер, связанный с указанным соединением. Если перед командой не было вызова OPEN или вызывающему процессу не разрешено использовать это соединение, возвращается ошибка.

В простейшей реализации управление не возвращается вызвавшей программе, пока не будет заполнен буфер или не возникнет ошибка, но эта схема сильно подвержена блокировкам. Более сложная реализация будет разрешать одновременное выполнение нескольких операций RECEIVE, заполняющих буферы по мере поступления сегментов. Такая стратегия позволяет поднять производительность за счёт усложнения схемы (возможно, асинхронной) уведомления вызывающей программы о наличии флага PUSH или заполнении буфера.

Получатель TCP **может** передать принятый бит PSH прикладному уровню через флаг PUSH на интерфейсе (MAY-17), но это не требуется (см. разъяснения в RFC 1122, параграф 4.2.2.2). Далее в описании вызова RECEIVE предполагается поддержка передачи индикации PUSH.

Если в буфер поступило достаточно данных до того, как был замечен флаг PUSH, этот флаг не устанавливается на интерфейсе в ответ на RECEIVE. В буфер будет помещено столько данных, сколько он может вместить. Если флаг PUSH замечен до заполнения буфера, буфер будет возвращаться заполненным частично с индикацией PUSH.

При наличии важных (срочных) данных пользователь будет проинформирован сразу же по их прибытии через сигнал TCP - пользователь. Принимающему пользователю следует перейти в режим urgent. Если установлен флаг URGENT, это говорит о наличии дополнительных важных данных. Сброшенный флаг URGENT указывает, что этот вызов RECEIVE вернул все важные данные и пользователь может выйти из режима urgent. Отметим, что данные после указателя важности (обычные данные) не могут быть доставлены пользователю в одном буфере с предшествующими важными данными, пока пользователю чётко не указана граница.

Чтобы различать несколько необработанных вызовов RECEIVE и обрабатывать ситуации, когда буфер ещё не заполнен, код возврата сопровождается указателем на буфер и счётчиком фактически принятых данных.

В реализациях RECEIVE конечная точка TCP может выделять буфер для вызова или использовать кольцевой буфер, созданный для пользователя.

3.9.1.4. Close

```
CLOSE (local connection name)
```

Эта команда закрывает указанное соединение. Если соединение не открыто или вызывающий процесс не имеет полномочий использовать соединение, возвращается ошибка. Закрытие соединения должно быть аккуратным в том смысле, что остающиеся вызовы SEND будут обработаны (переданы, включая повторы), насколько позволяет управление потоком данных. Таким образом, следует разрешать несколько вызовов SEND, за которыми следует CLOSE, и ожидать отправки получателю всех данных. Также нужно понимать, что пользователям следует продолжать обработку RECEIVE на закрываемых соединениях, поскольку удалённый партнёр может пытаться передать остаток своих данных. Таким образом, вызов CLOSE означает: «мне нечего больше передать», но не подразумевает: «я больше не принимаю». Может случиться так (если протокол приложения не продуман), что закрывающая сторона не способна справиться со всеми своими данными до возникновения тайм-аута. В таком случае CLOSE превращается в ABORT.

Пользователь может в любой момент вызвать CLOSE по своему усмотрению или по рекомендации (подсказке) реализации TCP (например, закрытие удалённой стороны, тайм-аут, недоступность получателя).

Поскольку для закрытия соединения требуется взаимодействие с удаленным партнёром TCP, процесс закрытия занимает некоторое время. Попытки заново открыть соединение до отклика партнёра TCP на команду CLOSE будут приводить к ошибкам.

Закрытие также предполагает функцию выталкивания (push).

3.9.1.5. Status

```
STATUS (local connection name) -> status data
```

Это зависимая от приложения пользовательская команда, которую можно исключить без негативных последствий. Возвращаемые командой сведения обычно поступают из связанного с соединением блока TCB и включают:

локальный сокет,

удалённый сокет,
локальное имя соединения,
окно приёма,
окно передачи,
статус соединения,
число буферов, ожидающих подтверждения,
число буферов, ожидающих приёма,
состояние важности (urgent),
значение поля Diffserv,
security/compartment (безопасности, изоляция),
тайм-аут передачи.

В зависимости от состояния соединения или самой реализации некоторая часть сведений может быть недоступной или бессмысленной. Если вызывающий процесс не имеет доступа к данному соединению, возвращается ошибка. Это предотвращает доступ сторонних процессов к информации о соединении.

3.9.1.6. Abort

`ABORT (local connection name)`

Эта команда вызывает прерывание всех ожидающих SEND и RECEIVE, удаление TCB и передачу специального сообщения RST удалённому партнёру TCP. В зависимости от реализации пользователь может получить индикацию прерывания для каждого остающегося вызова SEND и RECEIVE или просто подтверждение ABORT.

3.9.1.7. Flush

Некоторые реализации TCP включают вызов FLUSH, очищающий очередь передачи TCP от всех данных, которые были включены вызовами SEND, но остаются справа от текущего окна передачи. Т. е. из очереди удаляется как можно больше данных, но без потери синхронизации порядковых номеров. Вызовы FLUSH **можно** реализовать (MAY-14).

3.9.1.8. Асинхронные отчёты

Должен быть механизм информирования приложения о программных (soft) ошибках TCP (MUST-47). Обычно предполагается, что это предоставляемая приложением процедура ERROR_REPORT которую можно асинхронно вызвать с транспортного уровня

`ERROR_REPORT(local connection name, reason, subreason)`

Точное кодирование параметров причины (reason и subreason) здесь не задаётся, однако асинхронно сообщаемые приложению условия ошибок **должны** включать:

- получение сообщения ICMP об ошибке (см. параграф 3.9.2.2. Сообщения ICMP, где описана обработка каждого типа сообщений ICMP, поскольку для некоторых типов нужно отключать уведомление приложения);
- избыточные повторы передачи (см. параграф 3.8.3. Отказы соединений TCP);
- продвижение указателя важности (см. параграф 3.8.5. Обмен важными сведениями).

Прикладной программе, не желающей получать такие вызовы ERROR_REPORT, **следует** обеспечивать возможность их эффективного отключения (SHLD-20).

3.9.1.9. Установка поля дифференцированного обслуживания (IPv4 TOS или IPv6 Traffic Class)

Уровень приложения **должен** быть способен установить поле дифференцированного обслуживания (DiffServ) для передаваемых в соединение сегментов (MUST-48). Это поле включает 6 битов кода дифференцированного обслуживания (Differentiated Services Codepoint или DSCP). Это не требуется, но приложению **следует** поддерживать возможность изменить поле в процессе работы соединения (SHLD-21). Реализациям TCP **следует** передавать текущее значение поля без изменения на уровень IP при передаче сегментов в соединение (SHLD-22).

Поле дифференцированных услуг может задаваться независимо для каждого направления в соединении и принимающее приложение будет задавать это поле для сегментов ACK.

Реализация TCP **может** передавать полученное последним значение поля приложению (MAY-9).

3.9.2. Интерфейс TCP - нижележащий уровень

Конечная точка TCP вызывает модуль протокола нижележащего уровня для фактической передачи или приёма данных через сеть. Двумя стандартными версиями протокола Internet (IP), расположенного ниже TCP, в настоящее время являются IPv4 [1] и IPv6 [13].

Если нижележащим протоколом является IPv4, он обеспечивает аргументы для типа обслуживания (указывается в поле DiffServ) и времени жизни (TTL). TCP использует для этих параметров приведённые ниже установки

Diffserv

Значение поля Diffserv в заголовке IP указывает пользователь и оно включает биты кода обслуживания DSCP.

Time to Live (TTL)

Значение TTL для передачи сегментов TCP **должно** быть настраиваемым (MUST-49).

RFC 793 задаёт 1 минуту (60 секунд) как константу для TTL, поскольку предполагается максимальный срок действия сегмента 2 минуты. Это было предназначено для явного запроса уничтожения сегмента, который не может быть доставлен системой internet в течение 1 минуты. RFC 1122 обновляет RFC 793 требованием настраиваемого поля TTL.

Поле Diffserv может меняться в процессе работы соединения (параграф 4.2.4.2 в RFC 1122), однако интерфейс с приложением может не поддерживать эту возможность и приложение не знает об отдельных сегментах TCP, поэтому в лучшем случае это поле можно установить лишь грубо. Это ограничение рассмотрено в RFC 7657 (параграфы 5.1, 5.3, 6) [50]. Обычно приложению **не следует** менять поле Diffserv в течение срока действия соединения (SHLD-23).

Любой протокол нижележащего уровня будет предоставлять адреса отправителя и получателя, протокол и способ определения размера TCP для обеспечения функционального эквивалента IP и учёта в контрольной сумме TCP.

При передаче TCP принятых опций от уровня IP, реализация TCP **должна** игнорировать непонятные опции (MUST-50).

Реализация TCP **может** поддерживать опции Timestamp (MAY-10) и Record Route (MAY-11).

3.9.2.1. Source Routing

Если нижележащим уровнем является IP (или иной протокол, поддерживающий эту функцию) и применяется маршрутизация от источника (source routing), интерфейс должен обеспечивать обмен маршрутной информацией. Особенно важно, чтобы адреса отправителя и получателя, учитываемые в контрольной сумме TCP, указывали исходного отправителя и окончательного получателя. Важно также сохранение обратного маршрута для ответов на запросы соединения.

Приложение **должно** быть способно задать source route при активном создании соединения TCP (MUST-51) и этот маршрут **должен** быть предпочтительней source route, полученного в дейтаграмме (MUST-52).

При пассивном открытии соединения TCP и прибытии пакета с заполненной опцией IP Source Route (содержащей маршрут возврата), реализация TCP **должна** сохранять обратный маршрут и использовать его для всех сегментов, передаваемых в это соединение (MUST-53). Если позднее в сегменте приходит другой маршрут source route, **следует** использовать его для замены имеющегося (SHLD-24).

3.9.2.2. Сообщения ICMP

Реализации TCP **должны** реагировать на сообщения ICMP об ошибках, полученные от уровня IP, направляя их соединению, вызвавшему ошибку (MUST-54). Необходимые для демультиплексирования сведения можно получить из заголовка IP, содержащегося в сообщении ICMP. Это относится к ICMPv6 в дополнение к IPv4 ICMP.

В [35] обсуждаются конкретные сообщения ICMP и ICMPv6, поделенные на мягкие (soft) и жёсткие (hard) ошибки, которые могут вызывать разные отклики. Трактовка сообщений ICMP описана ниже.

Source Quench - притормозить источник

Реализация TCP **должна** без уведомления отбрасывать любые полученные сообщения ICMP Source Quench (MUST-55). Обсуждение этого вопроса приведено в работе [11].

Мягкие ошибки

Для IPv4 ICMP к таким ошибкам относятся Destination Unreachable с кодами 0, 1, 5, Time Exceeded с кодами 0, 1, Parameter Problem.

Для ICMPv6 это Destination Unreachable с кодами 0, 3, Time Exceeded с 0, 1, Parameter Problem с 0, 1, 2.

Поскольку эти сообщения о недоступности (Unreachable) указывают мягкие ошибки, реализации TCP **недопустимо** прерывать соединение (MUST-56), а информацию об ошибке **следует** делать доступной приложению (SHLD-25).

Жёсткие ошибки

Для ICMP к этому относятся Destination Unreachable с кодами 2 - 4.

Это жёсткие ошибки, поэтому реализации TCP **следует** прерывать соединение (SHLD-26). К [35] отмечено, что некоторые реализации не разрывают соединение при жёсткой ошибке ICMP для соединения, находящегося в каком-либо из синхронизированных состояний.

Отметим, что в разделе 4 [35] описано широко распространённое поведение, при котором мягкие ошибки считаются жёсткими во время организации соединений.

3.9.2.3. Проверка адреса отправителя

RFC 1122 требует проверки адресов во входящих пакетах SYN.

Входящие сегменты SYN с недействительным адресом отправителя **должны** игнорироваться уровнем TCP или IP [(MUST-63)] (см. параграф 3.2.1.3).

Реализация TCP **должна** отбрасывать входящие сегменты SYN, направленные по групповому или широковещательному адресу [(MUST-57)].

Это предотвращает ошибочное создание состояний и откликов и разработчикам следует учитывать, что ти рекомендации применимы ко всем входящим сообщениям, а не только к SYN, как указано в RFC 1122.

3.10. Обработка событий

Описанная в этом параграфе обработка является примером возможной реализации. Фактические реализации могут применять иную последовательность, которая может отличаться от представленной в деталях, но не по существу.

Действия конечной точки TCP можно считать откликами на события, которые можно разделить на 3 категории: вызовы пользователя, прибытие сегментов и тайм-ауты. В данном параграфе описывается работа конечной точки TCP для каждого из таких событий. Во многих случаях обработка зависит от текущего состояния. События приведены ниже.

Пользовательские вызовы

OPEN
SEND
RECEIVE
CLOSE
ABORT
STATUS

Прибытие сегментов

прибытие сегмента

Тайм-ауты

пользовательский тайм-аут
таймаут повтора передачи
таймаут TIME-WAIT.

Модель интерфейса TCP - пользователь подразумевает незамедлительный возврат из пользовательских вызовов и возможность отложенных откликов в виде событий или псевдо-прерываний. Далее причина отложенного отклика называется сигналом.

Отклики на ошибки указываются в документе символьными строками, например, команда пользователя для отсутствующего соединения возвращает ошибку error: connection not open.

Следует отметить, что в последующем обсуждении вся арифметимка порядковых номеров, номеров подтверждений, окна и т. п. использует модуль 2^{32} (размер пространства номеров), а знак =< обозначает «меньше или равно» (по модулю 2^{32}).

Естественным способом представления обработки входящих сегментов является сначала проверка порядкового номера (т. е. его попадание в пространство номеров окна приёма), затем сегменты обычно помещаются в очередь и обрабатываются в порядке роста номеров.

Когда сегмент перекрывается с принятым ранее, из него восстанавливается сегмент, содержащий только новые данные, и соответственно корректируются поля заголовка.

Если смена состояния соединения не указана, предполагается его сохранение.

3.10.1. Вызов OPEN

CLOSED (TCB не существует)

Создаётся новый блок TCB для данных о состоянии соединения с включением идентификатора локального сокета, удалённого сокета, поля Diffserv, security/compartment и пользовательского тайм-аута. Отметим, что удалённый сокет может быть указан не полностью при пассивном вызове OPEN и заполняется параметрами входящего сегмента SYN. Проверяется, что запрошенная защита и Diffserv разрешены для пользователя и в противном случае возвращается error: Diffserv value not allowed или error: security/compartment not allowed. При пассивном вызове соединение переходит в состояние LISTEN с возвратом управления. Если вызов активный и удалённый сокет не задан, возвращается error: remote socket unspecified, а при заданном сокете создаётся сегмент SYN. Выбирается начальный порядковый номер (ISS). Передаётся сегмент SYN вида <SEQ=ISS><CTL=SYN>. Устанавливается SND.UNA = ISS, SND.NXT = ISS+1, соединение переходит в состояние SYN-SENT и управление возвращается.

Если вызывающий не имеет доуступа к указанному локальному сокету, возвращается error: connection illegal for this process, а при отсутствии места для нового соединения - error: insufficient resources.

LISTEN

Если вызов OPEN был активным и удалённый сокет указан, состояние соединения меняется на активное и выбирается ISS. Передаётся сегмент SYN с SND.UNA = ISS, SND.NXT = ISS+1 и соединение переходит в состояние SYN-SENT. Данные, связанные с SEND могут передаваться в сегменте SYN или помещаться в очередь для передачи в состоянии ESTABLISHED. Если команда запрашивает флаг важности, но должен передаваться с сегментом данных как результат этой команды. Если нет места в очереди для запроса, возвращается error: insufficient resources. Если не задан удалённый сокет, возвращается error: remote socket unspecified.

SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT

Возвращается error: connection already exists.

3.10.2. Вызов SEND

CLOSED (TCB не существует)

Если у пользователя нет доступа к соединению, возвращается error: connection illegal for this process, иначе - error: connection does not exist.

LISTEN

Если задан удалённый сокет, соединение переходит в активно состояние и выбирается ISS. Передаётся сегмент SYN с SND.UNA = ISS, SND.NXT = ISS+1 и устанавливается состояние SYN-SENT. Данные, связанные с SEND могут передаваться в сегменте SYN или помещаться в очередь для передачи в состоянии ESTABLISHED. Если команда запрашивает флаг важности, но должен передаваться с сегментом данных как результат этой команды. Если нет места в очереди для запроса, возвращается error: insufficient resources. Если не задан удалённый сокет, возвращается error: remote socket unspecified.

SYN-SENT, SYN-RECEIVED

Данные помещаются в очередь для передачи в состоянии ESTABLISHED. Если нет места в очереди, возвращается error: insufficient resources.

ESTABLISHED, CLOSE-WAIT

Содержимое буфера сегментируется и передаётся с прицепленным подтверждением (RCV.NXT). Если недостаточно пространства для запоминания этого буфера, возвращается error: insufficient resources. При установленном флаге URGENT выполняется SND.UP <- SND.NXT и устанавливается указатель важности в исходящих сегментах.

FIN-WAIT-1, FIN-WAIT-2, CLOSING, LAST-ACK, TIME-WAIT

Возвращается error: connection closing и сервис не запрашивается.

3.10.3. Вызов RECEIVE

CLOSED (TCB не существует)

Если у пользователя нет доступа к соединению, возвращается error: connection illegal for this process, иначе - error: connection does not exist.

LISTEN, SYN-SENT, SYN-RECEIVED

Данные помещаются в очередь для передачи в состоянии ESTABLISHED. Если нет места в очереди, возвращается error: insufficient resources.

ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2

Если число входящих сегментов в очереди недостаточно для выполнения запроса, тот помещается в очередь. Если нет места в очереди, возвращается error: insufficient resources.

Выполняется сборка входящих сегментов из очереди в буфер приёма и управление возвращается пользователю. Если замечен флаг PUSH, устанавливается соответствующий маркер.

Если RCV.UP указывает вперёд передаваемых в данный момент пользователю данных, пользователь уведомляется о наличии важных данных.

Когда конечная точка TCP принимает ответственность за доставку данных пользователю, она должна взаимодействовать с отправителем через подтверждение. Формирование подтверждений описано ниже.

CLOSE-WAIT

Поскольку удалённая сторона уже передала FIN, вызовы RECEIVE должны быть удовлетворены уже полученными данными, которые ещё не доставлены пользователю. Если ожидающих доставки данных нет, RECEIVE возвращает "error: connection closing", иначе остающиеся данные используются для выполнения RECEIVE.

CLOSING, LAST-ACK, TIME-WAIT

Возвращается error: connection closing.

3.10.4. Вызов CLOSE

CLOSED (TCB не существует)

Если у пользователя нет доступа к соединению, возвращается error: connection illegal for this process, иначе - error: connection does not exist.

LISTEN

Все остающиеся RECEIVE завершаются с error: closing, удаляется TCB, устанавливается состояние CLOSED и возвращается управление.

SYN-SENT

Удаляется TCB и возвращаются отклики error: closing на все находящиеся в очереди запросы SEND и RECEIVE.

SYN-RECEIVED

Если не было вызовов SEND и нет ожидающих передачи данных, формируется и передаётся сегмент FIN, устанавливается состояние FIN-WAIT-1. В иных случаях постановка в очередь для обработки в состоянии ESTABLISHED.

ESTABLISHED

Постановка в очередь всех предшествующих SEND, которые были сегментированы, формирование и передача сегмента FIN, затем переход в состояние FIN-WAIT-1.

FIN-WAIT-1, FIN-WAIT-2

Строго говоря, это ошибка и следует получать отклик error: connection closing. Отклие ok тоже может быть приемлем, если не передан второй сегмент FIN (хотя первый может быть передан повторно).

CLOSE-WAIT

Запрос помещается в очередь, пока все предшествующие SEND не будут сегментированы. Затем передаётся сегмент FIN и устанавливается состояние LAST-ACK.

CLOSING, LAST-ACK, TIME-WAIT

Возвращается error: connection closing.

3.10.5. Вызов ABORT

CLOSED (TCB не существует)

Если у пользователя нет доступа к соединению, возвращается error: connection illegal for this process, иначе - error: connection does not exist.

LISTEN

Всем остающимся вызовам RECEIVE следует возвращать отклие error: connection reset. Удаляется TCB, устанавливается состояние CLOSED и управление возвращается.

SYN-SENT

Всем находящимся в очереди вызовам SEND и RECEIVE следует дать уведомление connection reset. Удаляется TCB, устанавливается состояние CLOSED и управление возвращается.

SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT

Передаётся сегмент сброса <SEQ=SND.NXT><CTL=RST>.

Всем находящимся в очереди вызовам SEND и RECEIVE следует дать уведомление connection reset. Все сегменты, помещённые в очередь на передачу (кроме указанного выше RST) или повтор передачи следует очистить. Удаляется TCB, устанавливается состояние CLOSED и управление возвращается.

CLOSING, LAST-ACK, TIME-WAIT

Возвращается отклие ok, удаляется TCB, устанавливается состояние CLOSED и управление возвращается.

3.10.6. Вызов STATUS

CLOSED (TCB не существует)

Если у пользователя нет доступа к соединению, возвращается error: connection illegal for this process, иначе - error: connection does not exist.

LISTEN

Возврат state = LISTEN и указателя TCB.

SYN-SENT

Возврат state = SYN-SENT и указателя TCB.

SYN-RECEIVED

Возврат state = SYN-RECEIVED и указателя TCB.

ESTABLISHED

Возврат state = ESTABLISHED и указателя TCB.

FIN-WAIT-1

Возврат state = FIN-WAIT-1 и указателя TCB.

FIN-WAIT-2

Возврат state = FIN-WAIT-2 и указателя TCB.

CLOSE-WAIT

Возврат state = CLOSE-WAIT и указателя TCB.

CLOSING

Возврат state = CLOSING и указателя TCB.

LAST-ACK

Возврат state = LAST-ACK и указателя TCB.

TIME-WAIT

Возврат state = TIME-WAIT и указателя TCB.

3.10.7. Прибытие сегмента

3.10.7.1. Состояние CLOSED

В состоянии CLOSED (TCB не существует) все данные входящего сегмента отбрасываются. Сегмент с RST отбрасывается, а входящий сегмент без RST вызывает передачу RST в ответ. Выбираются значения полей подтверждения и порядкового номера, чтобы сделать последовательность сброса приемлемой для конечной точки TCP, отправившей проблемный сегмент.

Если бит ACK сброшен (0), используется порядковый номер 0 - <SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>. При установленном (1) бите ACK передаётся <SEQ=SEG.ACK><CTL=RST>. Управление возвращается.

3.10.7.2. Состояние LISTEN

1. Проверка наличия бита RST.

   Входящий сегмент RST не может быть действительным, поскольку он не может являться ответом на передачу данной инкарнации соединения. Сегмент следует игнорировать и возвращать управление.

2. Проверка наличия бита ACK.

   Любое подтверждение, прибывшее с состоянии LISTEN, является недействительным. В ответ следует передать подходящий сегмент сброса <SEQ=SEG.ACK><CTL=RST>. Возврат управления.

3. Проверка наличия бита SYN.

   При установленном флаге SYN проверяются установки безопасности и если security/compartment во входящем сегменте не совпадает с security/compartment в TCB передаётся сброс и управление возвращается <SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>.

   Устанавливается RCV.NXT = SEG.SEQ+1, IRS = SEG.SEQ, все остальные элементы управления и данные следует поместить в очередь для последующей обработки. Следует выбрать значение ISS и передать сегмент SYN вида <SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>.

   Устанавливается SND.NXT = ISS+1, SND.UNA = ISS. Состояние соединения меняется на SYN-RECEIVED. Отметим, что другие элементы управления и данные (в комбинации с SYN) будут обрабатываться в состоянии SYN-RECEIVED, но обработку SYN и ACK повторять не следует. Если вызов LISTEN был задан не полностью (удалённый сокет не указан целиком), следует заполнить не заданные поля.

4. Проверка других элементов управления и данных.

   Этого не должно происходить. Сегмент отбрасывается и возвращается управление. Любой другой сегмент с элементами управления или данными (без SYN) должен иметь поле ACK и, таким образом, будет отброшен при проверке ACK на этапе 2, если не будет отброшен на этапе 1 при проверке RST.

3.10.7.3. Состояние SYN-SENT

1. Проверка наличия бита ACK.

   При установленном бите ACK выполняются указанные ниже проверки.

   - Если SEG.ACK =< ISS или SEG.ACK > SND.NXT, передаётся сброс (если нет флага RST, при котором сегмент отбрасывается с возвратом управления) <SEQ=SEG.ACK><CTL=RST> и сегмент отбрасывается с возвратом управления.

   - Если SND.UNA < SEG.ACK =< SND.NXT, проверяется пригодность ACK. Некоторые реализации TCP используют проверку SEG.ACK == SND.NXT (== вместо =<), но это не применимо, когда стек может передавать данные по SYN, так как партнёр TCP не может воспринять и подтвердить все данные по SYN.

2. Проверка наличия бита RST.

   При установленном бите RST выполняются указанные ниже проверки.

   - Возможная атака со сбросом вслепую описана в RFC 5961 [9]. Описанное в этом документе смягчение имеет описанную в нем конкретную применимость и не является заменой криптографической защиты (например, IPsec или TCP-AO). Реализации TCP с поддержкой описанного в RFC 5961 смягчения **следует** сначала проверить точное совпадение порядкового номера значению RCV.NXT и лишь потом выполнять действия следующего абзаца.

   - Если ACK был приемлем, пользователю передаётся error: connection reset, сегмент отбрасывается, устанавливается состояние CLOSED с удалением TCB и возвратом управления. В ином случае (нет ACK) сегмент просто отбрасывается с возвратом управления.

3. Проверяются установки безопасности.

   Если security/compartment в сегменте не совпадает с security/compartment в TCB, передаётся сброс:

   - При наличии в сегменте ACK передаётся <SEQ=SEG.ACK><CTL=RST>, в противном случае - <SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>.

Если передан сброс, сегмент отбрасывается с возвратом управления.

4. Проверка наличия бита SYN.

   Этого не должно происходить при корректно ACK или его отсутствии в сегменте без RST.

   Если бит SYN установлен и параметры security/compartment приемлемы, устанавливается RCV.NXT = SEG.SEQ+1, IRS = SEG.SEQ. Значению SND.UNA следует быть не меньше SEG.ACK (если это ACK) и все сегменты из очереди повторной передачи, подтверждённые таким путём, должны быть удалены.

Если SND.UNA > ISS (SYN был подтверждён), соединение переходит в состояние ESTABLISHED, формируется и передаётся сегмент ACK <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>, в который **можно** включить данные и элементы управления из очереди. Некоторые реализации TCP подавляют передачу этого сегмента, когда принятый сегмент содержит данные, которые в любом случае создают подтверждение на последующих этапах обработки. Если в сегменте есть другие элементы управления или данные, обработка продолжается с п. 6 в параграфе 3.10.7.4, где проверяется бит URG. В ином случае управление возвращается.

В противном случае устанавливается состояние SYN-RECEIVED, формируется и передаётся сегмент SYN,ACK <SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>. Устанавливаются переменные

```
SND.WND <- SEG.WND
SND.WL1 <- SEG.SEQ
SND.WL2 <- SEG.ACK
```

Если в сегменте есть другие элементы управления или данные, они помещаются в очередь для обработки в состоянии ESTABLISHED и управление возвращается.

Отметим допустимость передачи и приёма данных приложения в сегментах SYN, как отмечено выше. Исторически сложились существенные искажения и неверная интерпретация этого. Некоторые межсетевые экраны и устройства защиты считают такие сегменты подозрительными. Однако эта возможность была использована в T/TCP [21] и применяется в TCP Fast Open (TFO) [48], поэтому важна её поддержка в реализациях и сетевых устройствах.

5. Если биты SYN и RST не установлены, сегмент отбрасывается с возвратом управления.

3.10.7.4. Другие состояния

Первым делом проверяется порядковый номер.

SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT

- Сегменты обрабатываются по порядку. Первые проверки по прибытии служат для отбрасывания старых дубликатов, а дополнительные проверки выполняются в порядке SEG.SEQ. Если содержимое сегмента пересекает границу между старой и новой частью, обрабатывается только новая.
- В общем случае обработка принятых сегментов **должна** быть реализована для агрегирования сегментов ACK, когда это возможно (MUST-58). Например, если конечная точка TCP обрабатывает последовательность сегментов из очереди, все они **должны** быть обработаны до отправки какого-либо ACK (MUST-59).
- Имеется 4 случая для проверки пригодности входящих сегментов, показанные в таблице 6.

Таблица 6. Проверка пригодности сегмента.

Размер сегмента	***Окно приема***	***Проверка***
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND или RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

- При реализации описанной здесь проверки порядковых номеров следует учитывать Приложение A.2. Проверка порядковых номеров.
- Если RCV.WND = 0, сегменты не будут приемлемы, но следует воспринимать действительные ACK, URG, RST.
- Если входящий сегмент неприемлем, следует передать в ответ подтверждение (если не установлен флаг RST, когда сегмент отбрасывается с возвратом управления) вида <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>. После передачи подтверждения неприемлемый сегмент отбрасывается с возвратом управления.
- Отметим, что для состояния TIME-WAIT в [40] представлен улучшенный алгоритм обработки входящих сегментов SYN с использованием временных меток вместо описанной здесь проверки порядковых номеров. При реализации такого алгоритма описанная выше логика не применяется для входящих сегментов SYN с опцией Timestamp, принятых в состоянии TIME-WAIT.
- Далее сегменты считаются идеализированными, начинающимися с RCV.NXT и не выходящими за пределы окна. Фактические сегменты можно привести к этому допущению, отрезав выходящие за пределы окна части (включая SYN и FIN) и обрабатывая лишь оставшееся, как будто сегмент начинается с RCV.NXT. Сегменты, начинающиеся с больших порядковых номеров **следует** оставлять для последующей обработки (SHLD-31).

Вторым этапом служит проверка бита RST.

В разделе 3 RFC 5961 [9] описана возможная атака со сбросом вслепую и предложения по смягчению. Это не обеспечивает криптографической защиты *например, как IPsec или TCP-AO), но применимо в случае RFC 5961. Для стеков, реализующих описанную в RFC 5961 защиту, применимы три проверки, указанных ниже. В остальных случаях проверки выполняются в зависимости от состояния, как описано далее.

1) При установленном бите RST и порядковым номером вне текущего окна приёма сегмент отбрасывается.
2) Если бит RST установлен и порядковый номер точно соответствует ожидаемому (RCV.NXT), конечная точка TCP **должна** сбросит соединение, как описано ниже для соответствующего состояния.
3) Если бит RST установлен и порядковый номер не соответствует ожидаемому, но попадает в текущее окно приёма, конечная точка TCP **должна** передать подтверждение (challenge ACK) <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>.

   После отправки challenge ACK конечные точки TCP **должны** отбрасывать неприемлемые сегменты и останавливать дальнейшую обработку входящих пакетов. Отметим, что в RFC 5961 и Errata ID 4772 [99] приведены дополнительные соображения по contain дросселированию (throttling) ACK в реализации.

SYN-RECEIVED

При установленном бите RST.

- Если соединение инициировано пассивным вызовом OPEN (т. е. перешло из состояния LISTEN), оно возвращается в состояние LISTEN и управление возвращается вызвавшему процессу без необходимости информировать пользователя. Если вызов OPEN был активным (т. е. переход из состояния SYN-SENT), соединение отвергается с передачей пользователю ошибки connection refused. Соединение переходит в состояние CLOSED с удалением TCB и возвратом управления.
  В любом случае очередь повторной передачи следует очистить.

ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT

Если бит RST установлен, оставшимся вызовам RECEIVE и SEND следует получить отклики reset с очисткой всех сегментов из очереди. Пользователю следует получить незапрошенный общий сигнал connection reset. Соединение переходит в состояние CLOSED с удалением TCB и возвратом управления.

CLOSING, LAST-ACK, TIME-WAIT

Если бит RST установлен, соединение переходит в состояние CLOSED с удалением TCB и возвратом управления.

На третьем этапе проверяется безопасность.

SYN-RECEIVED

Если security/compartment в сегменте не соответствует точно security/compartment в TCB, передаётся сброс и управление возвращается.

ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT

- Если security/compartment в сегменте не соответствует точно security/compartment в TCB, передаётся сброс, оставшимся вызовам RECEIVE и SEND следует получить отклики reset с очисткой всех сегментов из очереди. Пользователю следует получить незапрошенный общий сигнал connection reset. Соединение переходит в состояние CLOSED с удалением TCB и возвратом управления.

- Отметим, что эта проверка выполняется вслед за проверкой порядкового номера, чтобы предотвратить прерывание сегмента старым сегментом между теми же номерами портов с другим уровнем безопасности, вызывающего прерывание текущего соединения.

На четвёртом этапе проверяется бит SYN.

SYN-RECEIVED

- Если соединение инициировано пассивным вызовом OPEN, оно переходит в состояние LISTEN с возвратом управления. В ином случае выполняется обработка в соответствии с синхронизированными состояниями.

ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT

- Если установлен бит SYN, это может быть легитимная попытка нового соединения (например, для TIME-WAIT), ошибка, при которой нужно сбросить соединение или результат попытки атаки, как описано в RFC 5961 [9]. Для TIME-WAIT новое соединение может быть воспринято, если применяется опция Timestamp и метка соответствует ожиданиям (в соответствии с [40]). Для других случаев RFC 5961 предоставляет смягчение атаки, применимое в некоторых ситуациях, но имеются также варианты криптографической защиты (см. 7. Вопросы безопасности и приватности). В соответствии с рекомендациями RFC 5961 для этих синхронизированных состояний при установленном флаге SYN независимо от порядкового номера конечная точка TCP **должна** передать удалённому партнёру challenge ACK <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>.
- После отправки подтверждения реализация TCP **должна** отбросить неприемлемый сегмент и прекратить обработку. Отметим, что в RFC 5961 и Errata ID 4772 [99] приведены дополнительные замечанию по дросселированию ACK для реализации.
- Для реализаций, не следующих RFC 5961, применяется поведение, описанное в RFC 793. Если SYN попадает в окно, это является ошибкой - передаётся сброс, остающимся RECEIVE и SEND следует получить отклик reset, все сегменты из очереди следует очистить, а пользователю следует передать незапрошенный сигнал connection reset. Соединение переводится в состояние CLOSED с удалением TCB и возвратом управления. Если SYN не попадает в окно, до этого этапа дело не доходит и сегмент ACK передаётся на первом этапе проверки порядкового номера.

Пятым этапом является проверка поля ACK.

- Если бит ACK сброшен (0) сегмент отбрасывается с возвратом управления.
- Если бит ACK установлен (1),выполняются указанные ниже действия.
  - В разделе 5 RFC 5961 [9] описаны возможные атаки с внедрением вслепую и смягчение этих атак, которое **можно** включить в реализацию (MAY-12). Стеки TCP, реализующие RFC 5961, **должны** добавлять входную проверку приемлемости значения ACK, если оно находится в диапазоне (SND.UNA - MAX.SND.WND) =< SEG.ACK =< SND.NXT. Все входящие сегменты, не удовлетворяющие этому условию, **должны** отбрасываться с передачей ACK. Новое состояние переменной MAX.SND.WND определяется наибольшим значением размера окна, полученного локальным отправителем от его партнёра (может изменяться) или может быть жёстко закодировано максимально допустимым размером окна. Когда значение ACK приемлемо, выполняются указанные ниже проверки в зависимости от состояния.

SYN-RECEIVED

- Если SND.UNA < SEG.ACK =< SND.NXT, соединение переходит в состояние ESTABLISHED и продолжается обработка с установкой значений
  ```
  SND.WND <- SEG.WND
  SND.WL1 <- SEG.SEQ
  SND.WL2 <- SEG.ACK
  ```
- Если подтверждение сегмента не приемлемо, формируется и передаётся сегмент сброса <SEQ=SEG.ACK><CTL=RST>.

ESTABLISHED

- Если SND.UNA < SEG.ACK =< SND.NXT, устанавливается SND.UNA <- SEG.ACK. Все подтверждённый таким образом сегменты из очереди повторной передачи удаляются. Пользователям следует отправить позитивные подтверждения для буферов, которые переданы (SENT) и полностью подтверждены (т. е. буфер SEND вернул отклик ok). Если ACK является дубликатом (SEG.ACK =< SND.UNA), его можно игнорировать. Если ACK

подтверждает ещё не переданное (SEG.ACK > SND.NXT), передаётся ACK и сегмент отбрасывается с возвратом управления.

- Если SND.UNA =< SEG.ACK =< SND.NXT, следует обновить окно передачи. Если (SND.WL1 < SEG.SEQ или (SND.WL1 = SEG.SEQ и SND.WL2 =< SEG.ACK)), устанавливается SND.WND <- SEG.WND, SND.WL1 <- SEG.SEQ, SND.WL2 <- SEG.ACK. Отметим, что SND.WND - это смещение от SND.UNA, SND.WL1 содержит порядковый номер последнего сегмента, использованного для обновления SND.WND, а SND.WL2 - номер подтверждения последнего сегмента, использованного для обновления SND.WND. Эта проверка предотвращает использование старых сегментов для обновления окна.

FIN-WAIT-1

В дополнение к обработке для состояния ESTABLISHED, если сегмент FIN подтверждён, состояние меняется на FIN-WAIT-2 и продолжается обработка там.

FIN-WAIT-2

В дополнение к обработке для состояния ESTABLISHED, если очередь передачи пуста, пользовательский вызов CLOSE можно подтвердить (ok), но без удаления TCB.

CLOSE-WAIT

Та же обработка, что и для состояния ESTABLISHED.

CLOSING

В дополнение к обработке для состояния ESTABLISHED, если ACK подтверждает наш FIN, сообщение переходит в состояние TIME-WAIT, в противном случае сегмент игнорируется.

LAST-ACK

В этом состоянии может приходить лишь подтверждение нашего FIN. Если сегмент FIN подтверждён, удаляется TCB, соединение переходит в состоянии CLOSED и возвращается управление.

TIME-WAIT

В этом состоянии может приходить лишь подтверждение нашего FIN, которое подтверждается и выполняется перезапуск с тайм-аутом 2 MSL.

Шестым этапом является проверка бита URG.

ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2

При установленном бите URG устанавливается RCV.UP <- max(RCV.UP,SEG.UP) и пользователю подаётся сигнал о наличии важных данных, если указатель важности (RCV.UP) указывает дальше (вперёд) воспринятых данных. Если пользователь уже уведомлен (или находится в режиме urgent) об этой непрерывной последовательности важных данных, новое уведомление не передается.

CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT

Этого не должно быть, поскольку от удалённой стороны уже получен сегмент FIN. Флаг URG игнорируется.

Седьмым этаком является обработка данных (текста).

ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2

- В состоянии ESTABLISHED можно доставлять сегмент данных в пользовательский буфер RECEIVE. Данные из сегментов можно перемещать в буферы до их заполнения или до завершения сегмента. Если сегмент использован полностью (пуст) и содержит флаг PUSH, пользователь информируется о получении PUSH.
- Когда конечная точка TCP принимает на себя ответственность за доставку данных пользователю, она должна подтвердить получение данных.
- Как только конечная точка TCP приняла ответственность за данные, она продвигает RCV.NXT за полученные данные и корректирует RCV.WND должным образом в соответствии с текущей доступностью буфера. Сумму RCV.NXT и RCV.WND не следует снижать.
- Реализация TCP **может** передать сегмент ACK, подтверждающий RCV.NXT, когда прибывает пригодный сегмент, попадающий в окно, но не находящийся на левом краю окна (MAY-13).
- Отметим предложения по управлению окном из параграфа 3.8. Обмен данными.
- Передаётся сегмент подтверждения в форме <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>. Это подтверждение следует цеплять (piggyback) к сегменту, который будет передаваться, по возможности, без неоправданной задержки.

CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT

Этого не может быть, поскольку от удалённой стороны получен сегмент FIN. Данные из сегмента игнорируются.

На восьмом этапе проверяется бит FIN.

- В состоянии CLOSED, LISTEN или SYN-SENT бит FIN не обрабатывается, поскольку нет возможности проверить SEG.SEQ. Сегмент отбрасывается с возвратом управления.

- Если бит FIN установлен, пользователю передаётся сигнал connection closing, возвращаются все ожидающие RECEIVE с тем же сообщение, значение RCV.NXT продвигается на FIN и передаётся подтверждение FIN. Отметим, что FIN предполагает PUSH для любых данных сегмента, ещё не доставленных пользователю.

SYN-RECEIVED, ESTABLISHED

Переход в состояние CLOSE-WAIT.

FIN-WAIT-1

Если наш сегмент FIN был подтверждён (возможно, в этом сегменте), соединение переходит в состояние TIME-WAIT, запускается таймер ожидания с отключением других таймеров. В ином случае переход в состояние CLOSING.

FIN-WAIT-2

Переход в состояние TIME-WAIT. Запуск таймера ожидания с отключением других таймеров.

CLOSE-WAIT

Сохраняется состояние CLOSE-WAIT.

CLOSING

Сохраняется состояние CLOSING.

LAST-ACK

Сохраняется состояние LAST-ACK.

TIME-WAIT

Сохраняется состояние TIME-WAIT и перезапускается таймер ожидания 2 MSL.

Далее управление возвращается вызвавшему процессу.

3.10.8. Тайм-ауты

USER TIMEOUT

По завершении отсчёта пользовательского таймера в любом состоянии очищаются все очереди, пользователю обычно передаётся error: connection aborted due to user timeout, удаляется TCB и соединение переходит в состояние CLOSED с возвратом управления.

RETRANSMISSION TIMEOUT

По завершению отсчёта таймера повторной передачи для сегмента из очереди в любом состоянии повторно передаётся сегмент из начала очереди, таймер запускается снова и управление возвращается.

TIME-WAIT TIMEOUT

По тайм-ауту ожидания (time-wait) в соединении удаляется TCB и соединение переходит в состояние CLOSED с возвратом управления.

4. Глоссарий

ACK - подтверждение

Бит управления (acknowledge) подтверждение, на занимающий пространства порядковых номеров, который указывает, что поле Acknowledgment в этом сегменте указывает следующий порядковый номер, который отправитель этого сегмента ждёт получить, тем самым подтверждая все предыдущие номера.

Connection - соединение

Логический коммуникационный путь, определяемый парой сокетов.

Datagram - дейтаграмма

Сообщение, переданное в компьютерной коммуникационной сети с коммутацией пакетов.

Destination Address - адрес получателя

Адрес сетевого уровня конечной точки, для которой предназначен сегмент.

FIN - завершение

Бит управления (финиш), занимающий 1 порядковый номер и указывающий, что отправитель больше не будет передавать данные или элементы управления, занимающие пространство номеров.

Flush - очистка

Удаление всего содержимого (данные или сегменты) из хранилища (буфер или очередь).

Fragment - фрагмент

Часть логического блока данных. Например, фрагмент internet является частью дейтаграммы internet.

Header - заголовок

Управляющая информация в начале сообщения, сегмента, фрагмента, пакета или блока данных.

Host - хост

Компьютер. В частности, это источник или получатель сообщений с точки зрения коммуникационной сети.

Identification - идентификация

Поле протокола IP, указываемое отправителем и помогающее при сборке фрагментов.

internet address - адрес internet

Адрес сетевого уровня.

internet datagram - дейтаграмма internet

Блок данных, передаваемых между хостами internet вместе с заголовком internet так, чтобы дейтаграмму можно было маршрутизировать от источника к получателю.

internet fragment - фрагмент internet

Часть дейтаграммы internet с заголовком internet.

IP

Протокол Internet, см. [1] и [13].

IRS

Начальный порядковый номер приёма - первый номер, использованный отправителем в соединении.

ISN

Начальный порядковый номер - первый номер, используемый в соединении (ISS или IRS). Выбирается так, чтобы быть уникальным в данный период времени и непредсказуемым для злоумышленников.

ISS

Начальный порядковый номер приёма - первый номер, используемый отправителем в соединении.

left sequence - левый край

Следующий порядковый номер, который должен быть подтверждён принимающей данные стороной TCP (или наименьший ещё не подтверждённый номер), иногда его называют левым краем окна передачи.

Module - модуль

Реализация (обычно программная) протокола или иной процедуры.

MSL (Maximum Segment Lifetime)

Максимальное время жизни сегмента - время, в течение которого сегмент TCP может существовать в межсетевой системе (internetwork). Сейчас выбрано произвольное значение 2 минуты.

Octet - октет

8-битовый байт.

Options - опции

Поле Option может включать несколько опций, каждая из которых может занимать несколько октетов.

Packet - пакет

Пакет данных с заголовком, который может быть логически полным или неполным. Пакетирование данных чаще бывает физическим, чем логическим.

Port - порт

Часть идентификатора соединения, служащая для демультиплексирования соединений в конечной точке.

Process - процесс

Исполняемая программа. Источник или получатель данных с точки зрения конечной точки TCP или иного протокола «хост-хост».

PUSH - выталкивание

Бит управления, не занимающий пространство порядковых номеров и указывающий, что сегмент содержит данные, которые должны быть «вытолкнуты» принимающему пользователю.

RCV.NXT

Следующий номер на приёме.

RCV.UP
Принятый указатель важности.

RCV.WND
Окно приёма.

receive next sequence number
Следующий порядковый номер, который локальная конечная точка TCP ожидает получить.

receive window - окно приёма
Представляет порядковые номера, которые локальная (принимающая) конечная точка TCP готова получить. Таким образом, конечная точка TCP считает, что сегменты из диапазона от RCV.NXT до RCV.NXT + RCV.WND - 1 содержат приемлемые данные или управление. Сегменты, содержащие порядковые номера полностью за пределами этого диапазона, считаются дубликатами или вставками при атаке и отбрасываются.

RST
Бит управления (сброс), не занимающий пространства порядковых номеров и указывающий, что получателю следует удалить соединение без дальнейшего взаимодействия. Получатель на основе порядкового номера и поля подтверждения во входящем сегменте может определить, нужно выполнить команду сброса или игнорировать её. Получение сегмента RST никогда не ведёт к отправке ответного RST.

SEG.ACK
Подтверждение сегмента.

SEG.LEN
Размер сегмента.

SEG.SEQ
Порядковый номер сегмента.

SEG.UP
Поле указателя важности в сегменте.

SEG.WND
Поле окна в сегменте.

Segment - сегмент
Логический блок данных. В частности, сегмент данных TCP является блоком данных, передаваемых между парой модулей TCP.

segment acknowledgment - подтверждение сегмента
Порядковый номер в поле подтверждения прибывающего сегмента.

segment length - размер сегмента
Количество порядковый номеров, занятых сегментом, включая элементы управления с номерами.

segment sequence - порядковый номер сегмента
Значение поля порядкового номера в прибывающем сегменте.

send sequence
Следующий порядковый номер, который локальная (передающая) конечная точка TCP будет использовать в соединении. Первоначально номер выбирается из кривой исходных порядковых номеров (ISN), а затем инкрементируется для каждого передаваемого октета данных или элемента управления.

send window - окно передачи
Представляет порядковые номера, которые удалённая (принимающая) конечная точка TCP готова принять. Это значение поля окна в сегментах от удалённой (принимающей данные) конечной точки TCP. Диапазон новых порядковых номеров, которые могут быть переданы реализацией TCP находится между SND.NXT и SND.UNA + SND.WND - 1 (естественно, предполагается возможность повторной передачи из диапазона SND.UNA - SND.NXT).

SND.NXT
Номер для передачи.

SND.UNA
Порядковый номер слева (окна).

SND.UP
Указатель важности для передачи.

SND.WL1
Порядковый номер сегмента при последнем обновлении окна.

SND.WL2
Номер подтверждения при последнем обновлении окна.

SND.WND
Окно передачи.

socket (socket number, socket address, socket identifier) - сокет (номер, идентификатор, адрес сокета)
Адрес, включающий номер порта, т. е. конкатенация адреса Internet (IP) и порта TCP.

Source Address
Адрес канального уровня передающей конечной точки.

SYN
Бит управления во входящем сегменте, занимающий 1 порядковый номер и служащий для инициирования соединения путём указания начального порядкового номера.

TCB
Блок управления передачей - структура данных, содержащая состояние соединения.

TCP
Протокол управления передачей - протокол коммуникаций между хостами для надёжного взаимодействия в межсетевых средах.

TOS
Тип обслуживания - устаревшее поле IPv4. Эти биты заголовка в настоящее время применяются для поля дифференцированных услуг (Differentiated Services) [4], содержащего код дифференцированного обслуживания (Differentiated Services Codepoint или DSCP) и 2-битовый код ECN [6].

Type of Service - тип обслуживания
См. TOS.

URG

Бит управления (urgent), не занимающий пространства номеров и служащий для указания принимающему партнёру необходимости уведомить пользователя о необходимости срочной обработки, пока есть данные с номерами меньше, чем значение указателя важности.

urgent pointer - указатель важности

Поле управления, имеющее смысл лишь при установленном флаге URG. Это поле содержит значение указателя важности, который показывает октет данных, связанных со срочным вызовом передающего пользователя.

5. Отличия от RFC 793

Этот документ отменяет RFC 793, а также обновляющие его RFC 6093 и RFC 6528. Во всех случаях в документ были включены лишь нормативная спецификация и требования, а информационный текст с обоснованиями не переносился. Информационная часть отменённых документов сохраняет интерес для изучения и понимания TCP, несмотря на перенос нормативных частей в этот документ.

Основная часть этого документа адаптирована из раздела 3 в RFC 793 (Функциональная спецификация) с попыткой максимального сохранения форматирования и макета.

В документе учтены сведения об ошибках (RFC errata), принятые или отложенные до обновления RFC 793 (Errata ID: 573 [73], 574 [74], 700 [75], 701 [76], 1283 [77], 1561 [78], 1562 [79], 1564 [80], 1571 [81], 1572 [82], 2297 [83], 2298 [84], 2748 [85], 2749 [86], 2934 [87], 3213 [88], 3300 [89], 3301 [90], 6222 [91]). Некоторые сообщения не были включены, поскольку их учли в других изменениях (Errata ID: 572 [92], 575 [93], 1565 [94], 1569 [95], 2296 [96], 3305 [97], 3602 [98]).

Внесены изменения спецификации указателя важности (срочности), описанные в RFC 1011, RFC 1122, RFC 6093. Подробное обсуждение необходимости этих изменений представлено в RFC 6093.

Обсуждение RTO в RFC 793 было обновлено ссылкой на RFC 6298. Связанный с RTO текст RFC 1122 в своё время заменил текст RFC 793, однако позднее RFC 2988 обновил RFC 1122, а сам был потом отменён RFC 6298.

В RFC 1011 [18] содержится много комментариев к RFC 793, включая некоторые изменения спецификации TCP. Они были расширены в RFC 1122, где содержится набор изменений и разъяснений к RFC 793. Влияющие на протокол нормативные элементы включены в этот документ, хотя часть исторически полезных советов по реализации и обсуждений из RFC 1122 не была включена. Настоящий документ, который сейчас является спецификацией TCP вместо RFC 793, обновляет RFC 1011 и комментарии из RFC 1011 включены здесь.

В RFC 1122 содержатся не просто требования к TCP, поэтому данный документ не может отменить RFC 1122 целиком. Он лишь обновляет RFC 1122, однако следует понимать, что это фактически отменяет все относящиеся к TCP сведения в RFC 1122.

Включён более защищённый алгоритм генерации начального порядкового номера из RFC 6528, где подробно рассмотрены атаки на него и их смягчение, а также даны рекомендации по выбору алгоритмов PRF и управления секретными ключами.

Добавлено основанное на RFC 6429 примечание для явного разъяснение того, что управление ресурсами системы позволяет высвобождать ресурсы соединения. RFC 6429 отменён в том смысле, что это разъяснение отражено в этой базовой спецификации.

Добавлено описание реализации контроля перегрузок на основе документов IETF BCP и Standards Track, соответствующих задачам и текущему состоянию распространённых реализаций.

6. Взаимодействие с IANA

В реестр Transmission Control Protocol (TCP) Header Flags агентство IANA внесло описанные здесь изменения.

В RFC 3168 был исходно создан этот реестр с указанием лишь заданных в RFC 3168 битов без учёта битов, описанных ранее в RFC 793 и других документах. После этого бит 7 был обновлён в RFC 8311 [54].

Столбец Bit был переименован в Bit Offset (Смещение), поскольку он указывает смещение каждого флага заголовка в 16 битовом слове заголовка TCP на рисунке 1. Биты со смещением 0-3 являются полем TCP Data Offset, а не флагами.

Агентство IANA добавило столбец Assignment Notes (замечанию к назначению).

Выделенные IANA значения представлены в таблице 7.

Таблица 7. Флаги заголовков TCP.

Смещение	***Имя***	***Документ***	***Замечания к назначению***
4	Резерв на будущее	RFC 9293	
5	Резерв на будущее	RFC 9293	
6	Резерв на будущее	RFC 9293	
7	Резерв на будущее	RFC 8311	Ранее применялся RFC 3540 как NS (Nonce Sum).
8	CWR (Congestion Window Reduced)	RFC 3168	
9	ECE (ECN-Echo)	RFC 3168	
10	Поле Urgent pointer значимо (URG)	RFC 9293	
11	Поле Acknowledgment значимо (ACK)	RFC 9293	
12	Функция Push (PSH)	RFC 9293	
13	Сброс соединения (RST)	RFC 9293	
14	Синхронизация порядковых номеров (SYN)	RFC 9293	
15	У отправителя больше нет данных (FIN)	RFC 9293	

Реестр TCP Header Flags перенесён в субреестр реестра Transmission Control Protocol (TCP) Parameters <https://www.iana.org/assignments/tcp-parameters/>. Процедурой регистрации для реестра остаётся Standards Action, но ссылка (Reference) обновлена с указанием данного документа, а колонка Note удалена.

7. Вопросы безопасности и приватности

TCP включает лишь рудиментарные свойства защиты, повышающие отказоустойчивость и надёжность соединений и доставки данных приложений, но не имеет криптографических средств поддержки проверки подлинности, защиты конфиденциальности и других функций защиты. Были разработаны некриптографические улучшения (например, [9]) для повышения устойчивости соединений TCP к отдельным типам атак, но применимость и защита некриптографических средств ограничены (см., например, параграф 1.1 в [9]). Приложения обычно применяют протокол нижележащего (например, IPsec) и вышележащего (например, TLS) уровня для обеспечения защиты и приватности соединений TCP и передаваемых по ним данных приложений. Были разработаны и методы на основе опций TCP для поддержки некоторых защитных возможностей.

Для полного обеспечения конфиденциальности, защиты целостности и аутентификации соединений TCP (включая флаги управления) единственным методом сейчас является IPsec. Доступна защита целостности и контроль подлинности с помощью опции TCP Authentication (TCP-AO) [38], а предложенное расширение обеспечивает конфиденциальность содержимого сегментов. Другие методы, описанные здесь, могут обеспечить конфиденциальность или целостность для содержимого, но включают лишь часть полей заголовка (например, tcpcrypt [57]) или не защищают их совсем (например, TLS). Другие средства защиты, добавленные в TCP (например, генерация ISN, проверка порядковых номеров и пр.) способны лишь частично отражать атаки.

Приложения с длительными потоками TCP были уязвимы для атак, использующих обработку флагов управления, описанную в ранних спецификациях TCP[33]. Опция TCP-MD5 широко реализована для поддержки аутентификации некоторых из этих соединений, но имеет недостатки и признана устаревшей. TCP-AO обеспечивает возможность защиты длительных соединений TCP от атак и превосходит по своим свойствам TCP-MD5. Однако она на защищает приватность данных приложений и заголовков TCP.

Экспериментальное расширение tcpcrypt [57] обеспечивает возможность криптографической защиты данных соединения. Аспекты метаданных потока по-прежнему видны, но поток приложения хорошо защищён. В заголовке TCP обеспечивается лишь защита указателя важности и флага FIN.

TCP Roadmap [49] включает замечания о нескольких RFC, связанных с безопасностью TCP. В этом документе объединены многие из предложенных в этих RFC улучшений, включая генерацию ISN, смягчение атак вслепую в окне, улучшение обработки мягких ошибок и пакетов ICMP. Все это подробно рассматривается в упомянутых RFC, исходно описывающих изменения, требуемые в ранних спецификациях TCP. В RFC 6093 [39] рассмотрены вопросы безопасности, относящиеся к полю указателя важности, а также рекомендуется не применять его в новых реализациях.

Поскольку TCP часто применяется для передачи больших объёмов данных, возможны атаки, злоупотребляющие логикой управления контролем перегрузок, например, атаки ACK-division. В спецификации механизмов контроля перегрузок TCP были внесены изменения для смягчения таких атак, например, подходящий поток байтов (Appropriate Byte Counting или ABC) [29].

Другие атаки нацелены на истощение ресурсов сервера TCP. Примеры включают лавинные атаки SYN [32] или расход ресурсов на неактивные соединения [41]. Операционные системы реализуют смягчение таких атак. В число распространённых средств защиты входят прокси, межсетевые экраны с учётом состояния и другие методы за пределами реализации TCP на конечном хосте.

Концепция образа протокола в линии (wire image), описанная в RFC 8546 [56], указывает, как открытые заголовки TCP раскрывают узлам на пути больше метаданных, нежели нужно для маршрутизации пакетов адресатам. Злоумышленники на пути могут использовать такие метаданные. Извлечённые из этого уроки для TCP были применены при разработке нового транспорта, такого как QUIC [60]. Кроме того, имеются соображения, частично основанные на опыте работы с TCP и расширениями протокола, которые можно применить при разработке новых транспортных расширения TCP и иного транспорта, опубликованные IETF в RFC 9065 [61], наряду с рекомендациями IAB в RFC 8558 [58] и [67].

Существуют также методы отпечатков (fingerprinting), которые можно применять для извлечения версии реализации TCP на хосте (в ОС) или сведений о платформе. Они собирают наблюдения по нескольким аспектам, таким как опции в сегментах, их порядок, специфическое поведение в разных условиях, синхронизация пакетов и другие аспекты протокола, оставленные для задания разработчикам, и могут по этим сведениям определять хост и реализацию.

Поскольку обработка сообщений ICMP также может взаимодействовать с соединениями TCP, имеются возможные атаки на основе ICMP. Они рассмотрены в RFC 5927 [100] вместе с реализованными мерами по смягчению.

8. Литература

8.1. Нормативные документы

[1] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[2] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <https://www.rfc-editor.org/info/rfc1191>.

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

[4] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <https://www.rfc-editor.org/info/rfc2474>.

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

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

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

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

[9] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <https://www.rfc-editor.org/info/rfc5961>.

[10] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[11] Gont, F., "Deprecation of ICMP Source Quench Messages", RFC 6633, DOI 10.17487/RFC6633, May 2012, <https://www.rfc-editor.org/info/rfc6633>.

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

[13] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[14] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <https://www.rfc-editor.org/info/rfc8201>.

[15] Allman, M., "Requirements for Time-Based Loss Detection", BCP 233, RFC 8961, DOI 10.17487/RFC8961, November 2020, <https://www.rfc-editor.org/info/rfc8961>.

8.2. Дополнительная литература

[16] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[17] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <https://www.rfc-editor.org/info/rfc896>.

[18] Reynolds, J. and J. Postel, "Official Internet protocols", RFC 1011, DOI 10.17487/RFC1011, May 1987, <https://www.rfc-editor.org/info/rfc1011>.

[19] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[20] Almquist, P., "Type of Service in the Internet Protocol Suite", RFC 1349, DOI 10.17487/RFC1349, July 1992, <https://www.rfc-editor.org/info/rfc1349>.

[21] Braden, R., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, DOI 10.17487/RFC1644, July 1994, <https://www.rfc-editor.org/info/rfc1644>.

[22] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <https://www.rfc-editor.org/info/rfc2018>.

[23] Paxson, V., Allman, M., Dawson, S., Fenner, W., Griner, J., Heavens, I., Lahey, K., Semke, J., and B. Volz, "Known TCP Implementation Problems", RFC 2525, DOI 10.17487/RFC2525, March 1999, <https://www.rfc-editor.org/info/rfc2525>.

[24] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms", RFC 2675, DOI 10.17487/RFC2675, August 1999, <https://www.rfc-editor.org/info/rfc2675>.

[25] Xiao, X., Hannan, A., Paxson, V., and E. Crabbe, "TCP Processing of the IPv4 Precedence Field", RFC 2873, DOI 10.17487/RFC2873, June 2000, <https://www.rfc-editor.org/info/rfc2873>.

[26] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, DOI 10.17487/RFC2883, July 2000, <https://www.rfc-editor.org/info/rfc2883>.

[27] Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, DOI 10.17487/RFC2923, September 2000, <https://www.rfc-editor.org/info/rfc2923>.

[28] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <https://www.rfc-editor.org/info/rfc3449>.

[29] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, DOI 10.17487/RFC3465, February 2003, <https://www.rfc-editor.org/info/rfc3465>.

[30] Fenner, B., "Experimental Values In IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers", RFC 4727, DOI 10.17487/RFC4727, November 2006, <https://www.rfc-editor.org/info/rfc4727>.

[31] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <https://www.rfc-editor.org/info/rfc4821>.

[32] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <https://www.rfc-editor.org/info/rfc4987>.

[33] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, DOI 10.17487/RFC4953, July 2007, <https://www.rfc-editor.org/info/rfc4953>.

[34] Culley, P., Elzur, U., Recio, R., Bailey, S., and J. Carrier, "Marker PDU Aligned Framing for TCP Specification", RFC 5044, DOI 10.17487/RFC5044, October 2007, <https://www.rfc-editor.org/info/rfc5044>.

[35] Gont, F., "TCP's Reaction to Soft Errors", RFC 5461, DOI 10.17487/RFC5461, February 2009, <https://www.rfc-editor.org/info/rfc5461>.

[36] StJohns, M., Atkinson, R., and G. Thomas, "Common Architecture Label IPv6 Security Option (CALIPSO)", RFC 5570, DOI 10.17487/RFC5570, July 2009, <https://www.rfc-editor.org/info/rfc5570>.

[37] Sandlund, K., Pelletier, G., and L-E. Jonsson, "The Robust Header Compression (ROHC) Framework", RFC 5795, DOI 10.17487/RFC5795, March 2010, <https://www.rfc-editor.org/info/rfc5795>.

[38] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <https://www.rfc-editor.org/info/rfc5925>.

[39] Gont, F. and A. Yourtchenko, "On the Implementation of the TCP Urgent Mechanism", RFC 6093, DOI 10.17487/RFC6093, January 2011, <https://www.rfc-editor.org/info/rfc6093>.

[40] Gont, F., "Reducing the TIME-WAIT State Using TCP Timestamps", BCP 159, RFC 6191, DOI 10.17487/RFC6191, April 2011, <https://www.rfc-editor.org/info/rfc6191>.

[41] Bashyam, M., Jethanandani, M., and A. Ramaiah, "TCP Sender Clarification for Persist Condition", RFC 6429, DOI 10.17487/RFC6429, December 2011, <https://www.rfc-editor.org/info/rfc6429>.

[42] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, DOI 10.17487/RFC6528, February 2012, <https://www.rfc-editor.org/info/rfc6528>.

[43] Borman, D., "TCP Options and Maximum Segment Size (MSS)", RFC 6691, DOI 10.17487/RFC6691, July 2012, <https://www.rfc-editor.org/info/rfc6691>.

[44] Touch, J., "Updated Specification of the IPv4 ID Field", RFC 6864, DOI 10.17487/RFC6864, February 2013, <https://www.rfc-editor.org/info/rfc6864>.

[45] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <https://www.rfc-editor.org/info/rfc6994>.

[46] McPherson, D., Oran, D., Thaler, D., and E. Osterweil, "Architectural Considerations of IP Anycast", RFC 7094, DOI 10.17487/RFC7094, January 2014, <https://www.rfc-editor.org/info/rfc7094>.

[47] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <https://www.rfc-editor.org/info/rfc7323>.

[48] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[49] Duke, M., Braden, R., Eddy, W., Blanton, E., and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", RFC 7414, DOI 10.17487/RFC7414, February 2015, <https://www.rfc-editor.org/info/rfc7414>.

[50] Black, D., Ed. and P. Jones, "Differentiated Services (Diffserv) and Real-Time Communication", RFC 7657, DOI 10.17487/RFC7657, November 2015, <https://www.rfc-editor.org/info/rfc7657>.

[51] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <https://www.rfc-editor.org/info/rfc8087>.

[52] Fairhurst, G., Ed., Trammell, B., Ed., and M. Kuehlewind, Ed., "Services Provided by IETF Transport Protocols and Congestion Control Mechanisms", RFC 8095, DOI 10.17487/RFC8095, March 2017, <https://www.rfc-editor.org/info/rfc8095>.

[53] Welzl, M., Tuexen, M., and N. Khademi, "On the Usage of Transport Features Provided by IETF Transport Protocols", RFC 8303, DOI 10.17487/RFC8303, February 2018, <https://www.rfc-editor.org/info/rfc8303>.

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

[55] Chown, T., Loughney, J., and T. Winters, "IPv6 Node Requirements", BCP 220, RFC 8504, DOI 10.17487/RFC8504, January 2019, <https://www.rfc-editor.org/info/rfc8504>.

[56] Trammell, B. and M. Kuehlewind, "The Wire Image of a Network Protocol", RFC 8546, DOI 10.17487/RFC8546, April 2019, <https://www.rfc-editor.org/info/rfc8546>.

[57] Bittau, A., Giffin, D., Handley, M., Mazieres, D., Slack, Q., and E. Smith, "Cryptographic Protection of TCP Streams (tcpcrypt)", RFC 8548, DOI 10.17487/RFC8548, May 2019, <https://www.rfc-editor.org/info/rfc8548>.

[58] Hardie, T., Ed., "Transport Protocol Path Signals", RFC 8558, DOI 10.17487/RFC8558, April 2019, <https://www.rfc-editor.org/info/rfc8558>.

[59] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <https://www.rfc-editor.org/info/rfc8684>.

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

[61] Fairhurst, G. and C. Perkins, "Considerations around Transport Header Confidentiality, Network Operations, and the Evolution of Internet Transport Protocols", RFC 9065, DOI 10.17487/RFC9065, July 2021, <https://www.rfc-editor.org/info/rfc9065>.

[62] IANA, "Transmission Control Protocol (TCP) Parameters", <https://www.iana.org/assignments/tcp-parameters/>.

[63] Gont, F., "Processing of IP Security/Compartment and Precedence Information by TCP", Work in Progress, Internet-Draft, draft-gont-tcpm-tcp-seccomp-prec-00, 29 March 2012, <https://datatracker.ietf.org/doc/html/draft-gont-tcpm-tcp-seccomp-prec-00>.

[64] Gont, F. and D. Borman, "On the Validation of TCP Sequence Numbers", Work in Progress, Internet-Draft, draft-gont-tcpm-tcp-seq-validation-04, 11 March 2019, <https://datatracker.ietf.org/doc/html/draft-gont-tcpm-tcp-seq-validation-04>.

[65] Touch, J. and W. M. Eddy, "TCP Extended Data Offset Option", Work in Progress, Internet-Draft, draft-ietf-tcpm-tcp-edo-12, 15 April 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-tcp-edo-12>.

[66] McQuistin, S., Band, V., Jacob, D., and C. Perkins, "Describing Protocol Data Units with Augmented Packet Header Diagrams", Work in Progress, Internet-Draft, draft-mcquistin-augmented-ascii-diagrams-10, 7 March 2022, <https://datatracker.ietf.org/doc/html/draft-mcquistin-augmented-ascii-diagrams-10>.

[67] Thomson, M. and T. Pauly, "Long-Term Viability of Protocol Extension Mechanisms", RFC 9170, DOI 10.17487/RFC9170, December 2021, <https://www.rfc-editor.org/info/rfc9170>.

[68] Minshall, G., "A Suggested Modification to Nagle's Algorithm", Work in Progress, Internet-Draft, draft-minshall-nagle-01, 18 June 1999, <https://datatracker.ietf.org/doc/html/draft-minshall-nagle-01>.

[69] Dalal, Y. and C. Sunshine, "Connection Management in Transport Protocols", Computer Networks, Vol. 2, No. 6, pp. 454-473, DOI 10.1016/0376-5075(78)90053-3, December 1978, <https://doi.org/10.1016/0376-5075(78)90053-3>.

[70] Faber, T., Touch, J., and W. Yui, "The TIME-WAIT state in TCP and Its Effect on Busy Servers", Proceedings of IEEE INFOCOM, pp. 1573-1583, DOI 10.1109/INFCOM.1999.752180, March 1999, <https://doi.org/10.1109/INFCOM.1999.752180>.

[71] Postel, J., "Comments on Action Items from the January Meeting", IEN 177, March 1981, <https://www.rfc-editor.org/ien/ien177.txt>.

[72] "Segmentation Offloads", The Linux Kernel Documentation, <https://www.kernel.org/doc/html/latest/networking/segmentation-offloads.html>.

[73] RFC Errata, Erratum ID 573, RFC 793, <https://www.rfc-editor.org/errata/eid573>.

[74] RFC Errata, Erratum ID 574, RFC 793, <https://www.rfc-editor.org/errata/eid574>.

[75] RFC Errata, Erratum ID 700, RFC 793, <https://www.rfc-editor.org/errata/eid700>.

[76] RFC Errata, Erratum ID 701, RFC 793, <https://www.rfc-editor.org/errata/eid701>.

[77] RFC Errata, Erratum ID 1283, RFC 793, <https://www.rfc-editor.org/errata/eid1283>.

[78] RFC Errata, Erratum ID 1561, RFC 793, <https://www.rfc-editor.org/errata/eid1561>.

[79] RFC Errata, Erratum ID 1562, RFC 793, <https://www.rfc-editor.org/errata/eid1562>.

[80] RFC Errata, Erratum ID 1564, RFC 793, <https://www.rfc-editor.org/errata/eid1564>.

[81] RFC Errata, Erratum ID 1571, RFC 793, <https://www.rfc-editor.org/errata/eid1571>.

[82] RFC Errata, Erratum ID 1572, RFC 793, <https://www.rfc-editor.org/errata/eid1572>.

[83] RFC Errata, Erratum ID 2297, RFC 793, <https://www.rfc-editor.org/errata/eid2297>.

[84] RFC Errata, Erratum ID 2298, RFC 793, <https://www.rfc-editor.org/errata/eid2298>.

[85] RFC Errata, Erratum ID 2748, RFC 793, <https://www.rfc-editor.org/errata/eid2748>.

[86] RFC Errata, Erratum ID 2749, RFC 793, <https://www.rfc-editor.org/errata/eid2749>.

[87] RFC Errata, Erratum ID 2934, RFC 793, <https://www.rfc-editor.org/errata/eid2934>.

[88] RFC Errata, Erratum ID 3213, RFC 793, <https://www.rfc-editor.org/errata/eid3213>.

[89] RFC Errata, Erratum ID 3300, RFC 793, <https://www.rfc-editor.org/errata/eid3300>.

[90] RFC Errata, Erratum ID 3301, RFC 793, <https://www.rfc-editor.org/errata/eid3301>.

[91] RFC Errata, Erratum ID 6222, RFC 793, <https://www.rfc-editor.org/errata/eid6222>.

[92] RFC Errata, Erratum ID 572, RFC 793, <https://www.rfc-editor.org/errata/eid572>.

[93] RFC Errata, Erratum ID 575, RFC 793, <https://www.rfc-editor.org/errata/eid575>.

[94] RFC Errata, Erratum ID 1565, RFC 793, <https://www.rfc-editor.org/errata/eid1565>.

[95] RFC Errata, Erratum ID 1569, RFC 793, <https://www.rfc-editor.org/errata/eid1569>.

[96] RFC Errata, Erratum ID 2296, RFC 793, <https://www.rfc-editor.org/errata/eid2296>.

[97] RFC Errata, Erratum ID 3305, RFC 793, <https://www.rfc-editor.org/errata/eid3305>.

[98] RFC Errata, Erratum ID 3602, RFC 793, <https://www.rfc-editor.org/errata/eid3602>.

[99] RFC Errata, Erratum ID 4772, RFC 5961, <https://www.rfc-editor.org/errata/eid4772>.

[100] Gont, F., "ICMP Attacks against TCP", RFC 5927, DOI 10.17487/RFC5927, July 2010, <https://www.rfc-editor.org/info/rfc5927>.

Приложение A. Замечания по реализации

В этом приложении приведены дополнительные примечания и ссылки на решения по реализации TCP, которые в настоящее время не включены в RFC и не являются частью стандарта TCP. Эти элементы могли быть рассмотрены разработчиками, но ещё не включены в стандарт.

A.1. IP Security Compartment и Precedence

Спецификация IPv4 [1] включает значение предпочтений в ныне отмененное поле типа обслуживания (Type of Service или TOS). Оно было изменено в [20], а сейчас отменено определением дифференцированного обслуживания (Differentiated Services или Diffserv) [4]. установка и передача поля TOS между сетевым уровнем, реализацией TCP и приложениями устарела и заменена Diffserv в текущей спецификации TCP.

RFC 793 требует проверки изоляции и предпочтений безопасности IP во входящих сегментах TCP на предмет согласованности с соединением и запросами приложений. Каждый из этих аспектов IP устарел, но в RFC 793 не внесено соответствующих обновлений. Проблема с предпочтениями решена в [25] со статусом Standards Track, поэтому действующая спецификация TCP включает эти изменения. Однако состояние опций безопасности IP, которые могут применяться в многоуровневых системах защиты (Multi-Level Secure или MLS) не столь очевидно в IETF.

Сброс соединений при несоответствии входящий пакетов ожиданиям в части изоляции или предпочтений был сочтён возможным вектором атак [63] и обсуждались поправки к спецификации TCP для предотвращения разрыва соединений из-за несоответствия изоляции безопасности IP и кодов Diffserv.

A.1.1. Предпочтение

В Diffserv прежнее значение precedence, считается кодом селектора класса (Class Selector) и совместимые методы обработки описаны в архитектуре Diffserv. Спецификация TCP,заданная в RFC 793 и RFC 1122 включает логику, предназначенную для использования приложением наивысшего уровня предпочтения или сохранения предпочтений, согласованных для соединения. Эта логика для устаревших TOS не применима к Diffserv и её не следует включать в реализации TCP, хотя смена значений Diffserv в рамках соединения не рекомендуется. Обсуждение этих вопросов приведено в RFC 7657 (параграфы 5.1, 5.3, 6) [50].

Устаревшие правила обработки TOS в TCP предполагали двухсторонние (симметричные) значения предпочтений в соединении, а архитектура Diffserv асимметрична. Проблемы прежней логики TCP в этой части описаны в [25] и предложено игнорировать IP precedence в TCP. Поскольку RFC 2873 задаёт стандарт (Standards Track), хотя и не указывает обновление RFC 793, предполагается устойчивость текущих реализаций к этим условиям. Отметим, что применяемое для каждого направления значение поля Diffserv является частью интерфейса между TCP и сетевым уровнем, а значения могут быть заданы между TCP и приложением в обоих направлениях.

A.1.2. Системы MLS

Опция IP Security (IPSO) и изоляция, определённые в [1], пересмотрены в RFC 1038, который был отменен RFC 1108. Опция Commercial IP Security (CIPSO) определена в FIPS-188 (отозван NIST в 2015 г.) и поддерживается некоторыми производителями и операционными системами. RFC 1108 сейчас считается устаревшим (Historic), хотя RFC 791 не был обновлён с удалением опции IP Security. Для IPv6 похожая опция (Common Architecture Label IPv6 Security Option или CALIPSO) определена в [36]. RFC 793 включает логику, применяющую сведения security/compartment при трактовке сегментов TCP. Упоминания IP security/compartment в этом документе могут иметь отношение к разработкам систем многоуровневой защиты, но могут не приниматься во внимание системами, не включающими MLS, с работающим в Internet кодом (см. A.1. IP Security Compartment и Precedence). Отметим, что в RFC 5570 описаны некоторые межсетевые системы MLS, где может применяться IPSO, CIPSO или CALIPSO. В этих особых случаях разработчикам TCP следует ознакомиться с параграфом 7.3.1 RFC 5570 и выполнять рекомендации этого документа.

A.2. Проверка порядковых номеров

В некоторых случаях правила проверки порядковых номеров TCP могут препятствовать обработке полей ACK. Это может приводить к проблемам соединений, описанным в [64], где рассмотрены проблемы, возможные при одновременной организации соединений, самоподключении, одновременном закрытии и одновременном зондировании окна. В документе также описаны возможные изменения спецификации TCP для смягчения проблемы путём расширения приемлемых порядковых номеров.

При использовании TCP в Internet эти условия возникают редко, Основные операционные системы включают различные дополнительные варианты смягчения, а стандарт ещё не обновлён с учётом этого, но разработчикам следует учитывать проблемы, описанные в [64].

A.3. Изменение алгоритма Nagle

В распространённых ОС алгоритм Nagle и отложенные подтверждения реализованы и по умолчанию включены. TCP применяется во многих приложениях «запрос-отклик», где сочетание алгоритма Nagle с отложенными подтверждениями может снижать производительность приложения. В работе [68] описано изменение алгоритма Nagle , решающего эту проблему. Это изменение реализовано в основных ОС и не влияет на совместимость TCP. Кроме того, многие приложения просто отключают алгоритм Nagle, поскольку он обычно поддерживается опцией сокета. Стандарт TCP не изменён в части включения этой модификации алгоритма Nagle, но разработчики могут счесть её полезной.

A.4. Настройки Low Watermark

Некоторые реализации TCP в ядре ОС включают опции сокета, позволяющие задать число байтов в буфере, пока уровень сокета передаёт отправляемые данные TCP (SO_SNDLOWAT) или принятые - приложению (SO_RCVLOWAT).

Кроме того, опция сокета TCP_NOTSENT_LOWAT может применяться для управления число не переданных байтов в очереди записи. Это может помочь передающему приложению TCP избежать создания большого объёма буферизованных данных (м соответствующей задержки). Например, это может быть полезно для приложений, мультиплексирующих данные из нескольких потоков вышележащего уровня в одно соединение, особенно при чередовании в потоков интерактивных (реальный масштаб времени) обменов и передачи больших объёмов.

Приложение B. Сводка требований TCP

Это приложение является адаптацией RFC 1122.

Отметим, что требования, связанные с PLPMTUD в таблице не приведены, но реализация PLPMTUD рекомендуется.

Таблица 8. Сводка требований TCP.

Свойство	*ReqID*	*MUST*	*SHOULD*	*MAY*	*SHOULD NOT*	*MUST NOT*
Флаг PUSH						
Агрегирование или размещение в очереди невыталкиваемых данных	MAY-16			X		
Отправитель исключает последовательные биты PSH	SHLD-27		X			
Вызов SEND может включать флаг PUSH	MAY-15			X		
- при невозможности буферизация на неопределённый срок	MUST-60					X
- при невозможности устанавливать бит PSH в последнем сегменте	MUST-61	X				
Уведомление принимающего ALP[1] о флаге PSH	MAY-17			X		
Передача сегмента максимального размера, когда это возможно	SHLD-28		X			
Размер окна						
Трактовка значения как целого числа без знака	MUST-1	X				
Обработка как 32-битового числа	REC-1		X			
Сокращение окна справа	SHLD-14				X	
- передача новых данных при сокращении окна	SHLD-15				X	
- повторная передача не подтверждённых данных из окна	SHLD-16		X			
- тайм-аут соединения для данных после правого края	SHLD-17				X	
Устойчивость к сокращению окна	MUST-34	X				
Закрытие приёмного окна на неопределённый срок	MAY-8			X		
Использование стандартной логики зондирования	MUST-35	X				
Зондирование нулевого окна отправителем	MUST-36	X				
- первое зондирование после интервала RTO	SHLD-29		X			
- экспоненциальное снижение частоты проб	SHLD-30		X			
Разрешение нулевого окна в неопределённый срок	MUST-37	X				
Повтор старых данных сверх SND.UNA+SND.WND	MAY-7			X		
Обработка RST и URG даже при нулевом окне	MUST-66	X				
Важные данные						
Наличие поддержки для указателя важности (urgent)	MUST-30	X				
Указатель задаёт первый октет после важных данных	MUST-62	X				
Произвольный размер последовательности важных данных	MUST-31	X				
Асинхронное информирование ALP о поступлении важных данных	MUST-32	X				
ALP может узнать, как много важных данных нужно прочесть	MUST-33	X				
Реализация механизма важности (срочности) в ALP	SHLD-13				X	
Опции TCP						
Поддержка обязательных опций	MUST-4	X				
Приём опций TCP в любом сегменте	MUST-5	X				
Игнорирование не поддерживаемых опций	MUST-6	X				
Включение размера во все опции, кроме EOL+NOP	MUST-68	X				
Устойчивость к опциям некорректного размера	MUST-7	X				
Обработка опций, не выравненных по границе слова	MUST-64	X				
Приём и передача опции MSS	MUST-14	X				
IPv4 передаёт опцию MSS пока не 536	SHLD-5		X			
IPv6 передаёт опцию MSS пока не 1220	SHLD-5		X			
Всегда передавать опцию MSS	MAY-3			X		
IPv4 Send-MSS по умолчанию 536	MUST-15	X				
IPv6 Send-MSS по умолчанию 1220	MUST-15	X				
Расчёт эффективного размера передаваемого сегмента	MUST-16	X				
MSS учитывает изменение MTU	SHLD-6		X			
MSS не передаётся в сегментах без флага SYN	MUST-65					
Значение MSS на основе MMS_R	MUST-67	X				
Заполнение нулями	MUST-69	X				
Контрольная сумма TCP						
Расчёт контрольной суммы отправителем	MUST-2	X				
Проверка контрольной суммы получателем	MUST-3	X				
Выбор ISN						
Использование основанного на «часах» генератораISN	MUST-8	X				
Защищённый генератор ISN с применением PRF	SHLD-1		X			
Расчёт PRF вне хоста	MUST-9					X
Создание соединений						
SYN-RECEIVED запоминает последнее состояние	MUST-11	X				
Пассивные вызовы OPEN могут влиять на другие записи (TCB)	MUST-41					X
Одновременно несколько LISTEN на одном порту	MUST-42	X				
Спрашивать при необходимости адрес отправителя у уровня IP	MUST-44	X				
- иначе использовать локальный адрес соединения	MUST-45	X				
Вызов OPEN с групповым или широковещательным адресом IP	MUST-46					X
Отбрасывание сегментов, переданных по групповому или широковещательному адресу	MUST-57	X				
Закрытие соединений						
RST может включать данные	SHLD-2		X			
Информирование приложения о прерванном (abort) соединении	MUST-12	X				
Полудуплексное закрытие соединений	MAY-1			X		
передача RST для индикации потери данных	SHLD-3		X			
Состояние TIME-WAIT в течение 2MSL	MUST-13	X				
- восприятие SYN в состоянии	MAY-2			X		

[1]Application-Layer Program - программа прикладного уровня.

- применение Timestamp для сокращения состояния TIME-WAIT	SHLD-4		X			
Повторная передача						
Реализация экспоненциальной отсрочки, замедленного старта и предотвращения перегрузки	MUST-19	X				
Повтор передачи с тем же полем IP Identification	MAY-4			X		
Алгоритм Karn	MUST-18	X				
Генерация ACK						
Агрегирование при наличии возможности	MUST-58	X				
Очередь сегментов с нарушением порядка	SHLD-31		X			
Обработка всей очереди перед отправкой ACK	MUST-59	X				
Передача ACK для сегментов с нарушением порядка	MAY-13			X		
Отложенные ACK	SHLD-18		X			
- задержка меньше 0,5 сек	MUST-40	X				
- подтверждать каждый второй полноразмерный сегмент или 2*RMSS принятых данных	SHLD-19		X			
Алгоритм предотвращения SWS у получателя	MUST-39	X				
Передача данных						
Настраиваемое значение TTL	MUST-49	X				
Алгоритм предотвращения SWS у отправителя	MUST-38	X				
Алгоритм Nagle	SHLD-7		X			
- приложение может отключать алгоритм Nagle	MUST-17	X				
Отказы соединений						
Рекомендация уровню IP при достижении порога R1	MUST-20	X				
Закрывать соединение при достижении порога R2	MUST-20	X				
ALP может устанавливать порог R2	MUST-21	X				
Информировать ALP об условии R1 <= retxs < R2	SHLD-9		X			
Рекомендовать значение для R1	SHLD-10		X			
Рекомендовать значение для R2	SHLD-11		X			
Тот же механизм для SYN	MUST-22	X				
- R2 для SYN не менее 3 минут	MUST-23	X				
Передача пакетов Keep-alive						
Передача пакетов Keep-alive:	MAY-5			X		
- приложение может включить и отключить передачу	MUST-24	X				
- по умолчанию передача отключена	MUST-25	X				
- передача только в интервале бездействия	MUST-26	X				
- настраиваемый интервал передачи	MUST-27	X				
- интервал по умолчанию не менее 2 часов	MUST-28	X				
- устойчивость к потере ACK	MUST-29	X				
- передача без данных	SHLD-12		X			
- настраиваемая передача «сорного» октета данных	MAY-6			X		
Опции IP						
Игнорировать опции, не понятные TCP	MUST-50	X				
Поддержка временных меток	MAY-10			X		
Поддержка записи маршрута	MAY-11			X		
Source Route						
- ALP может задавать маршрут	MUST-51	X				
- переписывать Source Route в дейтаграмме	MUST-52	X				
- построение обратного пути из Source Route	MUST-53	X				
- переписывание маршрута по более новым данным	SHLD-24		X			
Приём сообщений ICMP от IP						
Приём сообщений ICMP от IP	MUST-54	X				
- Destination Unreachable (0,1,5) передаётся ALP	SHLD-25		X			
- прерывание соединения по Destination Unreachable (0,1,5)	MUST-56					
- Destination Unreachable (2 - 4) ведёт к разрыву соединения	SHLD-26		X			
- отбрасывание сообщений Source Quench	MUST-55	X				
- прерывание соединения по сообщению Time Exceeded	MUST-56	X				X
- прерывание соединения по сообщению Parameter Problem	MUST-56					X
Проверка пригодности адресов						
Отклонять вызовов OPEN для недействительного адреса IP	MUST-46	X				
Отклонять SYN с недействительного адреса IP	MUST-63	X				
Отбрасывать SYN с групповым или широковещательным адресом IP	MUST-57	X				
Службы интерфейса TCP - ALP						
Механизм отчётов об ошибках	MUST-47	X				
ALP может отключать процедуру отчётов об ошибках	SHLD-20		X			
ALP способна задавать поле Diffserv для передачи	MUST-48	X				
- передача Diffserv уровню IP без изменений	SHLD-22		X			
ALP может менять поле Diffserv в процессе работы соединения	SHLD-21		X			
ALP меняет поле Diffserv в процессе работы соединения	SHLD-23				X	
Передача ALP полученного поля Diffserv	MAY-9			X		
Вызов FLUSH	MAY-14			X		
Поддержка локального адреса IP в параметре OPEN	MUST-43	X				
Поддержка RFC 5961						
Реализация защиты от внедрения данных	MAY-12			X		
Явное уведомление о перегрузке						
Поддержка ECN	SHLD-8		X			
Дополнительный контроль перегрузки						
Реализация дополнительных совместимых алгоритмов	MAY-18			X		

Благодарности

Этот документ в значительной части является пересмотром RFC 793, редактором которого был Jon Postel. Благодаря его превосходной работе документ просуществовал три десятилетия без пересмотра.

Andre Oppermann участвовал в редактировании первой редакции этого документа.

Авторы признательны руководителям рабочей группы IETF TCPM за помощь в подготовке этого документа: Michael Scharf, Yoshifumi Nishida, Pasi Sarolahti, Michael Tüxen.

При обсуждении этой работы в почтовой конференции TCPM, на встречах рабочей группы и обзорах в рамках направления полезные комментарии, критику и рецензии предоставили (в алфавитном порядке фамилий) Praveen Balasubramanian, David Borman, Mohamed Boucadair, Bob Briscoe, Neal Cardwell, Yuchung Cheng, Martin Duke, Francis Dupont, Ted Faber, Gorry Fairhurst, Fernando Gont, Rodney Grimes, Yi Huang, Rahul Jadhav, Markku Kojo, Mike Kosek, Juhamatti Kuusisaari, Kevin Lahey, Kevin Mason, Matt Mathis, Stephen McQuistin, Jonathan Morton, Matt Olson, Tommy Pauly, Tom Petch, Hagen Paul Pfeifer, Kyle Rose, Anthony Sabatini, Michael Scharf, Greg Skinner, Joe Touch, Michael Tüxen, Reji Varghese, Bernie Volz, Tim Wicinski, Lloyd Wood, Alex Zimmermann.

Joe Touch предоставил дополнительную помощь в прояснении описания параметров размера сегмента и рекомендации для PMTUD/PLPMTUD. Markku Kojo помог собрать воедино текст раздела по контролю перегрузок TCP.

Этот документ включает содержимое присланных замечаний об ошибках, присланных (в хронологическом порядке) Yin Shuming, Bob Braden, Morris M. Keesan, Pei-chun Cheng, Constantin Hagemeier, Vishwas Manral, Mykyta Yevstifeyev, EungJun Yi, Botong Huang, Charles Deng, Merlin Buge.

Адрес автора

Wesley M. Eddy (editor)
MTI Systems
United States of America
Email: wes@mti-systems.com

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru