

## Transmission Control Protocol

### Протокол управления передачей Программная спецификация протокола DARPA INTERNET

Подготовлена для Defense Advanced Research Projects Agency  
Information Processing Techniques Office  
1400 Wilson Boulevard  
Arlington, Virginia 22209

Information Sciences Institute  
University of Southern California  
4676 Admiralty Way  
Marina del Rey, California 90291

RFC: 793

Заменяет RFC 761

ИЕН: 129, 124, 112, 81, 55, 44, 40, 27, 21, 5

## Оглавление

Предисловие.....	1
1. Введение.....	2
1.1. Мотивация.....	2
1.2. Сфера применимости протокола.....	2
1.3. Структура документа.....	3
1.4. Интерфейсы.....	3
1.5. Работа протокола.....	3
2. Концепции.....	4
2.1. Элементы межсетевого взаимодействия.....	4
2.2. Модель работы протокола.....	4
2.3. Среда хоста.....	5
2.4. Интерфейсы.....	5
2.5. Связи с другими протоколами.....	5
2.6. Надёжная связь.....	5
2.7. Организация и разрыв соединений.....	5
2.8. Обмен данными.....	6
2.9. Предпочтения и безопасность.....	7
2.10. Принцип устойчивости.....	7
3. Функциональная спецификация.....	7
3.1. Формат заголовка.....	7
3.2. Терминология.....	9
3.3. Порядковые номера.....	10
3.4. Организация соединения.....	13
3.5. Завершение соединения.....	15
3.6. Предпочтения и безопасность.....	16
3.7. Обмен данными.....	16
3.8. Интерфейсы.....	18
3.9. Обработка событий.....	21
Глоссарий.....	29
Литература.....	32

## Предисловие

В этом документе описан стандарт Министерства обороны США для протокола управления передачей TCP (Transmission Control Protocol). До этого было выпущено девять предварительных редакций спецификации ARPA TCP, на которых основан данный стандарт, и текст документа тесно связан с предварительными спецификациями. В разработке концепций и подготовке текста документа принимало участие много людей. В данной редакции более чётко изложены некоторые детали, удалён механизм изменения размера буфера end-of-letter и заново описан механизм letter как функция push<sup>1</sup>.

*Jon Postel, редактор*

<sup>1</sup>Тем не менее в документе осталось некоторое количество ошибок, информацию о которых можно найти в [RFC 1122](#). Прим. перев.

Этот документ в какой-то мере уже стал достоянием истории, несмотря на то, что большинство рассмотренных здесь вопросов сохраняют актуальность. Протокол TCP является основой работы сети Internet в её современном состоянии. Долгие годы использования TCP и связанных с ним сетевых протоколов не могли не отразиться на терминологии, определениях и т. п. Поэтому буквальный перевод исходного документа современным читателям мог бы показаться непонятным. Исходя из этого, в переводе используется современная терминология. Кроме того, в RFC 793 за время использования протокола TCP был обнаружен целый ряд неточностей и ошибок, исправленных в более поздних документах и спецификациях. Нет смысла сохранять неточности при переводе, поэтому соответствующие фрагменты RFC 793 при переводе были скорректированы с учётом действующих спецификаций. В переводе такие места отмечены сносками с указанием источника корректных сведений.

Николай Малых, переводчик

## 1. Введение

Протокол TCP предназначен для надёжной и гарантированной доставки данных между хостами в компьютерных сетях с коммутацией пакетов и между такими сетями через промежуточные системы.

В этом документе описаны функции, выполняемые протоколом TCP, программные реализации протокола и интерфейсы с приложениями и пользователями, которым требуется сервис TCP.

### 1.1. Мотивация

Компьютерные коммуникационные системы играют все более важную роль в государственных, военных и деловых средах. В этом документе основное внимание сфокусировано на требованиях военных систем компьютерных коммуникаций и, особенно, на вопросах устойчивости при наличии коммуникационных сбоев и доступности связи при возникновении перегрузок, но многие из этих проблем встречаются не только в военных приложениях.

Разработаны и реализованы стратегические и тактические компьютерные коммуникационные системы и очень важно обеспечить связь между такими системами, а также стандартный протокол взаимодействия между процессами, который будет поддерживать широкий спектр приложений. Предполагая необходимость такого стандарта, исследовательская группа Deputy Undersecretary of Defense for Research and Engineering подготовила описываемый здесь протокол TCP<sup>1</sup> в качестве стандартного протокола обмена информацией между процессами для сетей Министерства обороны США.

TCP представляет собой ориентированный на соединения протокол сквозной гарантированной доставки, который предназначен для использования в многоуровневой иерархии протоколов, поддерживающей разнородные сетевые приложения. TCP обеспечивает надёжный обмен данными между парами процессов на хостах, подключённых к разным, но связанным между собой сетям. Протокол исходит из незначительного числа предположений о надёжности коммуникационных протоколов, расположенных ниже уровня TCP. Протокол TCP предполагает, что он может использовать простой и потенциально ненадёжный сервис доставки дейтаграмм протоколов нижележащих уровней. В принципе протокол TCP должен работать в широком классе коммуникационных систем, имеющих электрические соединения с сетями коммутации пакетов (packet-switched) или каналов (circuit-switched).

#### Уровни протоколов

Вышележащий уровень
TCP
Протокол Internet (IP)
Коммуникационная сеть

Рисунок 1.

Протокол TCP основан на концепциях, впервые изложенных Серфом и Каном в работе [1]. TCP располагается в многоуровневой модели непосредственно над базовым протоколом Internet (IP) [2], который обеспечивает для TCP возможность приёма и передачи сегментов информации переменной длины, «вложенных» в дейтаграммы. Дейтаграммы обеспечивают способ адресации отправителя и получателя TCP, расположенных в разных сетях. Протокол IP обеспечивает также фрагментацию и сборку сегментов TCP, требуемую для доставки информации через множество сетей и соединяющих их шлюзов (маршрутизаторов). Кроме того, протокол IP передаёт информацию о предпочтениях (precedence) и безопасности, а также о делении сегментов TCP на части - эта информация передаётся между отправителем и получателем через множество разнородных сетей.

Большая часть этого документа написана в контексте реализаций TCP, работающих на хостах вместе с протоколами вышележащего уровня. Некоторые компьютерные системы соединяются с сетью через специальные периферийные (front-end) компьютеры, которые обеспечивают поддержку уровней TCP и IP, а также сетевые программы. Спецификация TCP описывает интерфейс с протоколами вышележащего уровня, который представляется реализуемым и для случаев front-end (с помощью специального протокола взаимодействия между хостом и периферийным компьютером).

### 1.2. Сфера применимости протокола

Протокол TCP предназначен для организации надёжного обмена данными между процессами в разнородных сетевых средах. TCP является протоколом общего пользования для обмена информацией между хостами.

<sup>1</sup>Transmission Control Protocol - протокол управления передачей

### 1.3. Структура документа

В этом документе содержатся спецификации функций, требуемых от любой реализации протокола TCP, а также спецификации взаимодействия с протоколами соседних уровней и протоколом TCP на других хостах. В оставшейся части данного раздела приведено краткое описание работы протокола и интерфейсов. В разделе 2 рассмотрены базовые концепции архитектуры TCP. Раздел 3 содержит детальные описания действий, выполняемых протоколом TCP в ответ на те или иные события (приём новых сегментов, пользовательские вызовы, ошибки и т. п.), и описание форматов данных в сегментах TCP.

### 1.4. Интерфейсы

Протокол TCP с одной стороны взаимодействует с пользовательскими или прикладными процессами, а с другой стороны - с протоколом нижележащего уровня (таким, как IP).

Интерфейс между прикладными процессами и TCP проиллюстрирован достаточно детально. Этот интерфейс содержит набор процедур, подобных процедурам операционной системы, используемым для работы с файлами. Например, существуют процедуры открытия и закрытия соединений, а также приёма данных через организованное соединение. Предполагается, что TCP может обмениваться данными с прикладными программами в асинхронном режиме. Хотя разработчикам TCP предоставляется разумная свобода выбора реализации, существует минимальный набор функций для интерфейса между TCP и пользовательским уровнем, который должен присутствовать во всех реализациях протокола.

Интерфейс между TCP и протоколом нижележащего уровня задаётся менее детально за исключением предположения о наличии механизма асинхронного обмена информацией между уровнями. Предполагается также, что спецификация взаимодействия будет задаваться стандартом для протокола нижележащего уровня. Протокол TCP разработан для использования в разнотипных сетевых средах. В качестве протокола нижележащего уровня в данном документе предполагается протокол IP [2].

### 1.5. Работа протокола

Как было отмечено выше, задачей протокола TCP является обеспечение надёжных и потенциально безопасных логических соединений между парами процессов. Для обеспечения такого сервиса на базе менее надёжных коммуникационных систем Internet требуется поддержка следующих функций:

- базовый обмен данными (Basic Data Transfer);
- надёжность (Reliability);
- управление потоком данных (Flow Control);
- мультиплексирование (Multiplexing);
- поддержка соединений (Connection);
- предпочтения и безопасность (Precedence and Security).

Далее рассмотрены основные аспекты TCP для каждой из перечисленных функций.

#### **Базовая передача данных**

Протокол TCP способен поддерживать непрерывный поток октетов данных в каждом направлении между двумя точками соединения, упаковывая некоторое количество октетов в сегменты для передачи через межсетевую среду. В общем случае TCP решает, когда блокировать или пересылать данные по своему усмотрению.

Иногда пользователям нужна уверенность в том, что все данные, которые были направлены протоколу TCP, будут переданы. Для решения такой задачи определена функция выталкивания данных. Для обеспечения гарантии передачи отправленных протоколу TCP данных пользователь на передающей стороне должен «протолкнуть» данные удалённому пользователю. Функция push заставляет TCP незамедлительно переслать и доставить данные в точку приёма. Точка «выталкивания» может быть невидима для получателя и функция push не устанавливает маркер границы записи.

#### **Надёжность**

Протокол TCP должен восстанавливать данные в случае их повреждения, потери, дублирования или доставки с нарушением порядка. Это обеспечивается с помощью порядковых номеров, присваиваемых каждому передаваемому октету и подтверждением доставки данных (ACK<sup>1</sup>), отправляемым приёмной стороной TCP. Если подтверждение ACK не было получено в течение заданного времени (тайм-аут), данные передаются заново. На приёмной стороне порядковые номера используются для корректной расстановки сегментов, которые могут быть доставлены с нарушением порядка, и обнаружения дубликатов. Для обнаружения ошибок при передаче данных используются контрольные суммы, которые добавляются в каждый передаваемый сегмент. Контрольная сумма рассчитывается на передающей стороне и включается в сегмент. На приёмной стороне контрольная сумма вычисляется заново и полученное значение сравнивается с указанной в сегменте контрольной суммой. При обнаружении расхождений сегмент отбрасывается.

Пока протокол TCP работает нормально и среда Internet не разорвана на изолированные одна от другой части, никакие ошибки передачи не будут влиять на корректность доставки данных. TCP восстанавливает данные после ошибок в среде Internet.

#### **Управление потоком данных**

TCP предоставляет получателю способ управления количеством данных, передаваемых отправителем. Это достигается путём возврата с каждым подтверждением ACK "окна" (window), показывающего диапазон допустимых порядковых номеров за пределами последнего доставленного сегмента. Окно показывает допустимые номера октетов, которые отправитель может передать до получения следующего разрешения.

#### **Мультиплексирование**

Чтобы позволить множеству процессов на одном хосте использовать коммуникационные возможности TCP, этот протокол поддерживает набор адресов (портов) на каждом хосте. Вкупе с адресами сетей и номерами хостов

<sup>1</sup>Acknowledgment - подтверждение.

(уровень internet), номера портов формируют сокет (socket). Пара сокетов позволяет уникально идентифицировать каждое соединение. Таким образом, сокет может использоваться несколькими соединениями одновременно.

Связывание портов с процессами устанавливается каждым хостом независимо. Однако, для наиболее часто используемых процессов (например, протоколирование или работа в режиме разделения времени) целесообразно применять predetermined (фиксированные) порты, номера которых известны всем. Доступ к таким службам обеспечивается по известным адресам. Организация и определение номеров портов для других процессов может включать более динамичные механизмы.

### **Соединения**

Механизмы обеспечения надёжности и управления потоком данных, описанные выше, требуют от TCP поддержки информации о состоянии каждого потока данных. Комбинация этих сведений, включающая сокет, порядковые номера и размеры окна, называется соединением. Каждое соединение может быть однозначно указано парой сокетов, используемых по разные стороны соединения.

Когда два процесса хотят обмениваться информацией, протоколы TCP на хостах сначала должны организовать соединение (инициализировать информацию о состоянии на каждой стороне). Когда обмен данными завершается, соединение разрывается (terminate или close) для освобождения использованных им ресурсов.

Поскольку соединения организуются между хостами, не обеспечивающими надёжность, через ненадёжную сетевую среду, применяется механизм согласования параметров с порядковыми номерами на основе текущего времени для предотвращения ошибочной инициализации соединений.

### **Предпочтения и безопасность**

Пользователи TCP могут указывать режим безопасности и предпочтения для своих соединений. Когда эти возможности не используются, для соответствующих параметров устанавливаются принятые по умолчанию значения.

## **2. Концепции**

### **2.1. Элементы межсетевого взаимодействия**

Межсетевая среда состоит из хостов, подключённых к сетям, которые соединены между собой маршрутизаторами. Здесь термином «сеть» обозначаются локальные (например, ETHERNET) или распределённые (например, ARPANET<sup>1</sup>) сети, но в любом случае эти сети работают на основе коммутации пакетов. Активными агентами, которые производят и потребляют сообщения, являются процессы. Протоколы различного уровня в сети, на хостах и маршрутизаторах образуют систему обмена информацией между процессами, которая обеспечивает двухсторонние потоки данных и логические соединения между портами процессов.

Термин пакет в данном документе используется в основном для обозначения одной транзакции между хостом и его сетью. Формат блоков данных, передаваемых через сеть, в общем случае не будет рассматриваться в контексте этого документа.

Хостами называют компьютеры, подключённые к сети и (с коммуникационной точки зрения) являющиеся отправителями и получателями пакетов. Процессы рассматриваются, как активные элементы в хостах (в соответствии с наиболее общим определением процесса, как программы на стадии выполнения). Терминалы и файлы (или иные устройства ввода-вывода) рассматриваются, как взаимодействующие между собой с помощью процессов. Таким образом, весь обмен информацией рассматривается как связь между процессами.

Поскольку процессу может потребоваться возможность различать коммуникационные потоки между ним и разными другими процессами, мы предполагаем, что каждый процесс может иметь номер порта, через который он осуществляет взаимодействие с портами других процессов.

### **2.2. Модель работы протокола**

Процесс передаёт данные, вызывая функции TCP с передачей им буферов данных в качестве аргументов. TCP упаковывает данные из таких буферов в сегменты и вызывает модуль IP для передачи каждого сегмента адресату TCP. На приёмной стороне TCP помещает данные из сегмента в приёмный пользовательский буфер и уведомляет принимающего пользователя. TCP включает в сегменты управляющую информацию, которая служит для обеспечения гарантированной доставки с сохранением порядка сегментов.

Модель обмена информацией в IP-сетях построена на базе связанного с каждым модулем TCP модуля протокола IP, обеспечивающего интерфейс с локальной сетью. Модуль IP упаковывает сегменты TCP в дейтаграммы IP и маршрутизирует дейтаграммы модулю IP хоста-получателя или промежуточного шлюза. Для передачи дейтаграмм через локальную сеть они помещаются в кадры ЛВС.

Коммутация пакетов может включать дополнительное разбиение на пакеты, фрагментирование или иные операции, обеспечивающие доставку локальных пакетов модулю IP хоста-адресата.

В шлюзах между сетями дейтаграммы IP «разворачиваются» из локальных пакетов и проверяются не предмет определения сети, в которую их следует переслать. После этого дейтаграмма IP снова помещается в локальный пакет, подходящий для следующей сети и направляется в следующий шлюз или конечному получателю.

Шлюз может разбивать дейтаграммы IP на меньшие дейтаграммы (фрагменты), если это требуется для передачи в следующую сеть. Для этого дейтаграмма делится на несколько более мелких дейтаграмм IP, каждая из которых содержит фрагмент исходной дейтаграммы. На следующих шлюзах эти фрагменты могут дополнительно фрагментироваться. Формат фрагментов организован таким способом, чтобы модуль IP конечного получателя мог заново собрать из фрагментов исходную дейтаграмму.

Модуль IP конечного получателя разворачивает сегменты из дейтаграмм (после сборки фрагментов, если это необходимо) и передаёт их конечному модулю TCP.

В рассмотренной модели опущено множество деталей. Одним из важных аспектов является тип обслуживания - он говорит маршрутизаторам (или модулю IP), какие параметры обслуживания следует выбрать для передачи в

<sup>1</sup>Прообраз сети Internet. Прим. перев.

следующую сеть. Включённая в тип обслуживания информация задаёт преимущества (приоритет) дейтаграммы. Дейтаграмма может содержать также информацию о безопасности, позволяющую хостам и шлюзам, работающим в многоуровневой защищённой среде корректно разделять дейтаграммы с точки зрения требований безопасности.

## 2.3. Среда хоста

Предполагается, что TCP является модулем операционной системы. Пользователи обращаются к TCP подобно обращению к другим функциям ОС. TCP может вызывать другие функции ОС (например, для управления структурами данных). Предполагается, что реальный интерфейс с сетью управляется драйвером сетевого адаптера. TCP не может непосредственно обращаться к драйверу сетевого устройства и взамен этого обращается к модулю протокола IP, который, в свою очередь, обращается к драйверу устройства напрямую.

Механизмы TCP не мешают реализации протокола TCP на периферийных процессорах. Однако в таких реализациях должен обеспечиваться протокол взаимодействия между хостом и периферийным процессором для поддержки интерфейса TCP - пользователь, описанного в этом документе.

## 2.4. Интерфейсы

Интерфейс между TCP и пользователем обеспечивается с помощью функций TCP для организации (OPEN) и разрыва (CLOSE) соединений, передачи (SEND) и приёма (RECEIVE) данных, а также получения информации о соединении (STATUS). Вызовы этих функций осуществляются из пользовательских программ, подобно вызовам функций ОС (например, функции открытия, чтения и закрытия файла).

Интерфейс TCP - IP обеспечивается с помощью функций передачи и приёма дейтаграмм, адресованных модулям TCP на любых хостах сети IP. Эти функции используют параметры для передачи адресов, типа обслуживания, предпочтений, безопасности и другой управляющей информации.

## 2.5. Связи с другими протоколами

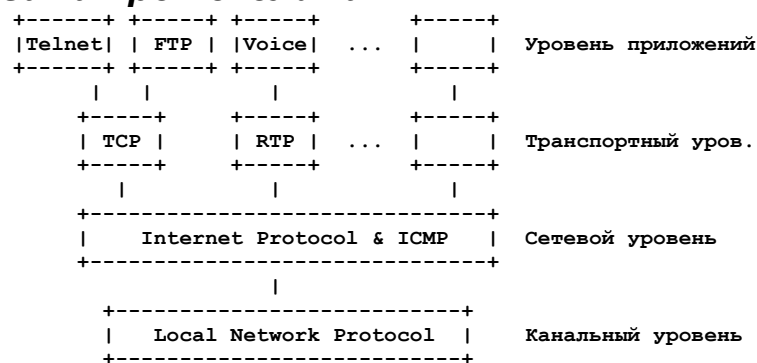


Рисунок 2. Отношения между протоколами.

На рисунке 2 показано местоположение TCP в иерархии протоколов.

Ожидается, что TCP будет эффективно поддерживать протоколы вышележащего уровня. Должен обеспечиваться простой интерфейс с протоколами вышележащего уровня типа Telnet или AUTODIN II THP.

## 2.6. Надёжная связь

Переданный модулю TCP поток данных гарантированно и с соблюдением порядка доставляется адресату.

Надёжность передачи обеспечивается за счёт использования порядковых номеров и передачи подтверждений. Концептуально, каждому октету данных присваивается порядковый номер. Номер первого октета данных в сегменте передаётся вместе с этим сегментом и называется порядковым номером сегмента. Сегмент содержит также номер подтверждения, являющийся порядковым номером следующего ожидаемого октета данных для обратного направления. Когда TCP передаёт сегмент, содержащий данные, копия этого сегмента помещается в очередь повторной передачи и включается таймер. При получении подтверждения доставки переданного сегмента его копия удаляется из очереди и таймер сбрасывается. Если подтверждение не поступило в течение заданного времени (тайм-аут), сегмент из очереди повтора передаётся снова.

Подтверждение TCP не даёт гарантии доставки данных конечному пользователю, оно лишь удостоверяет, что сегмент доставлен модулю TCP на удалённой стороне, который обеспечивает дальнейшую доставку клиенту.

Для управления потоками данных между хостами TCP используется специальный механизм управления потоком. TCP на приёмной стороне сообщает размер "окна" передающему модулю TCP. Этот размер задаёт число октетов (начиная с номера подтверждения), которые модуль TCP на приёмной стороне готов получить в настоящий момент.

## 2.7. Организация и разрыв соединений

Для идентификации отдельных потоков данных, которыми может управлять TCP, этот протокол поддерживает идентификаторы портов. Поскольку идентификаторы портов выбираются независимо разработчиками реализаций TCP, их значения могут оказаться неуникальными<sup>1</sup>. Для обеспечения уникальной адресации каждого TCP идентификатор порта объединяется с адресом IP для хоста TCP - такая комбинация порта и адреса называется сокетом. Для сокетов обеспечивается уникальность в масштабах Internet.

<sup>1</sup>В настоящее время выделение номеров для портов TCP/UDP достаточно жёстко регламентировано. Значения выделенных номеров можно найти по ссылке <http://www.iana.org/assignments/port-numbers>. Прим. перев.



Соединение однозначно идентифицируется парой сокетов на обеих сторонах соединения. Локальный сокет может использоваться во множестве соединений с различными внешними сокетами. Соединение может использоваться для передачи данных в обоих направлениях, т. е. является «полнодуплексным».

TCP распределяет номера портов между процессами по своему усмотрению, однако существуют базовые концепции, которым необходимо следовать в каждой реализации. Должны поддерживаться «общеизвестные» (well-known) номера портов, которые TCP будет связывать только с определёнными процессами. Предполагается, что такими процессами могут быть собственные процессы TCP и эти процессы могут инициировать соединения только через такие «привилегированные» порты. Применительно к реализации распределение портов является локальным вопросом, но предполагается наличие пользовательской команды Request Port или иного метода выделения уникальной группы портов для данного процесса (например, путём связывания старших битов в номере порта с данным процессом).

При вызове OPEN соединение задаётся номером локального порта и удалённого сокета, передаваемыми в качестве аргументов. В ответ TCP возвращает (короткое) локальное имя соединения, которое пользователь может указывать в последующих вызовах функций. Есть несколько важных аспектов организации соединений, о которых следует помнить. Предполагается, что информация о соединении хранится в структуре данных TCB<sup>1</sup>. В некоторых реализациях локальное имя соединения просто указывает на структуру TCB для этого соединения. При вызове OPEN также указывается тип вызова - активный (соединение организуется сразу) или пассивный (ожидание).

Пассивный запрос OPEN означает, что процесс будет ждать входящего соединения, не делая попыток организовать соединение самому. Зачастую процесс, использовавший пассивный вызов OPEN, будет воспринимать входящие запросы соединений от любого хоста. В таких случаях внешний сокет с номером, содержащим только нули, будет использоваться для обозначения незаданного сокета. Использование незаданных внешних сокетов допускается только для пассивных вызовов OPEN.

Сервисный процесс, который намерен обслуживать вызовы от неизвестных процессов, должен использовать пассивный вызов OPEN с неуказанным внешним сокетом. В таких случаях соединение может быть организовано с любым процессом, который запросит соединение с данным локальным сокетом. Такое решение полезно в тех случаях, когда известно, что локальный сокет связан с известным сервисом.

Общеизвестные сокеты обеспечивают удобный способ связывания адресов со стандартными службами. Например, серверный процесс Telnet постоянно связан с конкретным сокетом, а другие известные сокеты зарезервированы для таких служб как File Transfer, Remote Job Entry, Text Generator, Echoer, Sink (последние три используются в тестовых целях). Адрес сокета может быть зарезервирован для доступа к службе Look-Up, возвращающей номер сокета, по которому предоставляется новый сервис. Концепция общеизвестных сокетов является частью спецификации TCP, но выделение номеров для сокетов выходит за рамки этого документа (см. [4]<sup>1</sup>).

Процесс может использовать пассивный вызов OPEN и ждать соответствующего активного вызова OPEN другим процессом - после организации нужного соединения TCP будет информировать процесс об этом. Два процесса, одновременно использовавшие активные вызовы OPEN для связи друг с другом, будут корректно соединены. Такая гибкость очень важна для поддержки распределённых систем, в которых компоненты работают асинхронно.

Существует два важных случая соответствия между локальным пассивным вызовом OPEN и внешним активным вызовом OPEN. В первом случае локальный пассивный вызов OPEN полностью задаёт внешний сокет - соответствие является точным. Во втором случае локальный пассивный вызов OPEN оставляет внешний сокет неуказанным - приемлемы соединения с любыми внешними сокетами, соответствующими локальному сокету. Остальные случаи являются промежуточными вариантами.

При наличии нескольких ожидающих пассивных вызовов OPEN (записанных в TCB) с одинаковым локальным сокетом внешний активный вызов OPEN будет соответствовать TCB с указанным внешним сокетом, который совпадает с внешним активным вызовом OPEN (при наличии такого TCB); в остальных случаях выбор осуществляется среди TCB с незаданным внешним сокетом.

Процедуры организации соединений используют флаг контроля синхронизации (SYN) и включают обмен тремя сообщениями, известный, как трехэтапное согласование<sup>2</sup> [3].

Соединение инициируется при наличии входящего сегмента с флагом SYN и ожидающей записи TCB, которая была создана с помощью функции OPEN. Соответствие локального и внешнего сокета определяет возможность организации соединения. Соединение переходит в состояние established (организовано) после синхронизации порядковых номеров для обоих направлений.

Разрыв соединения также включает обмен сегментами - в этом случае они содержат флаг завершения FIN.

## 2.8. Обмен данными

Данные, передаваемые через соединение, можно трактовать как поток октетов. Передающая сторона показывает при каждом вызове SEND наличие в этом вызове (и любых предшествующих вызовах) данных, которые должны быть незамедлительно отправлены на принимающую сторону, с помощью флага PUSH.

TCP на передающей стороне разрешается собирать данные от пользователей-отправителей и передавать эти данные в сегменты по своему усмотрению, если не используется функция выталкивания. После вызова этой функции все неотправленные данные должны быть переданы. Когда TCP на приёмной стороне видит флаг PUSH, модуль больше не должен ждать данных от передающего модуля TCP до передачи их принимающему процессу.

Нет необходимости задавать связи между функцией push и границами сегментов. Данные в любом сегменте могут быть результатом одного вызова SEND (все или часть результатов вызова) или множества обращений к SEND.

Назначением функции push и флага PUSH является «проталкивание» данных от отправителя к получателю. Функция не обеспечивает сервис записи.

<sup>1</sup>Transmission Control Block - блок управления передачей.

<sup>2</sup>Three-way handshake.

Между функцией push и использованием буферов данных, проходящих через интерфейс TCP - пользователь, существует связь. Всякий раз флаг PUSH связывается с данными, помещёнными в буфер на приёмной стороне, и содержимое этого буфера передаётся пользователю для обработки, даже если буфер ещё не полон. Если данные заполняют пользовательский буфер до получения флага PUSH, содержимое буфера передаётся пользователю целиком.

TCP также обеспечивает способ информирования принимающей стороны о наличии в потоке, который принимается в настоящее время, данных, требующих срочной обработки. TCP не пытается задать, что конкретно должен делать пользователь при получении уведомления о срочных данных, но в общем случае принимающий процесс будет предпринимать попытки максимально быстрой обработки таких данных.

## 2.9. Предпочтения и безопасность

TCP позволяет использовать поля типа обслуживания и безопасности в заголовке IP для задания предпочтений и обеспечения защиты на уровне соединений. Не все модули TCP могут использоваться в средах с многоуровневой системой защиты - некоторые функции могут применяться для неклассифицированного<sup>1</sup> использования, другие могут работать только для определённого уровня защиты. Следовательно, некоторые реализации и службы TCP могут иметь ограниченную функциональность в средах с многоуровневой системой защиты.

Модули TCP, работающие в среде с многоуровневой системой защиты, должны корректно помечать уровень защиты и предпочтения для исходящих сегментов. Такие модули должны также обеспечивать для своих пользователей и протоколов вышележащих уровней (типа Telnet или TNP) интерфейс, позволяющий задать желаемый уровень защиты и предпочтения для своих соединений.

## 2.10. Принцип устойчивости

Реализации TCP должны следовать общему принципу устойчивости - быть консервативным по отношению к себе и либеральным с другими.

# 3. Функциональная спецификация

## 3.1. Формат заголовка

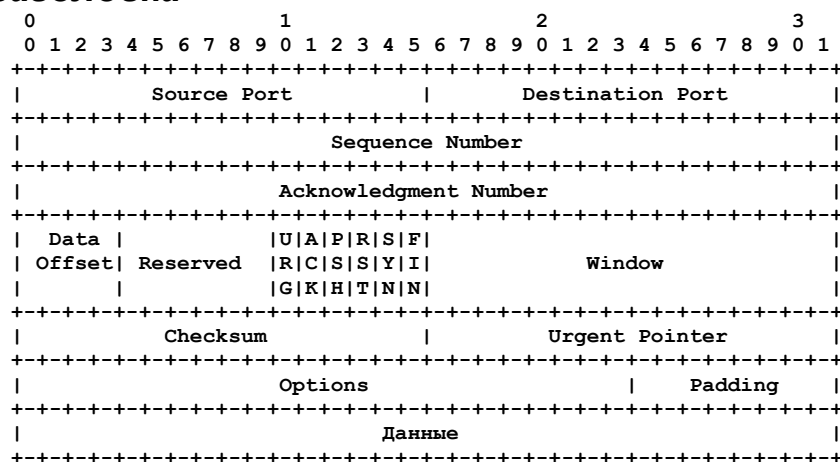


Рисунок 3. Формат заголовка TCP.

Сегменты TCP передаются в дейтаграммах IP. Заголовок протокола IP содержит несколько информационных полей, включая адреса хостов отправителя и получателя [2]. Заголовок TCP размещается вслед за заголовком IP и содержит информацию, относящуюся к протоколу TCP. Такое разделение позволяет использовать на уровне хоста<sup>2</sup> протоколы, отличные от TCP.

### Формат заголовка TCP

#### Source Port - порт отправителя: 16 битов

Номер порта отправителя.

#### Destination Port - порт назначения: 16 битов

Номер порта получателя.

#### Sequence Number - порядковый номер: 32 бита

Порядковый номер первого октета данных в сегменте при отсутствии флага SYN. Если в сегменте присутствует бит SYN, поле номера содержит значение начального порядкового номера (ISN), а первый октет данных имеет номер ISN+1.

#### Acknowledgment Number - номер подтверждения: 32 бита

Если бит ACK установлен, это поле содержит значение следующего порядкового номера, который отправитель сегмента ожидает получить. После организации соединения это значение передаётся всегда.

#### Data Offset - смещение данных: 4 бита

Число 32-битовых слов в заголовке TCP. Это значение указывает начало данных в сегменте. Заголовок TCP (даже при наличии опций) имеет длину, кратную 32 битам.

#### Reserved - резервное поле: 6 битов

Зарезервировано для использования в будущем и должно иметь нулевое значение.

#### Control Bits - биты управления: 6 битов (слева направо):

<sup>1</sup>Без требований по безопасности. Прим. перев.

<sup>2</sup>Транспортном уровне в современной терминологии. Прим. перев.

URG: указывает на значимость поля Urgent Pointer (указатель срочности);  
 ACK: указывает на значимость поля Acknowledgment Number (номер подтверждения);  
 PSH: функция Push (выталкивание);  
 RST: сброс (Reset) соединения;  
 SYN: синхронизация порядковых номеров;  
 FIN: у отправителя больше нет данных.

**Window - окно: 16 битов**

Число октетов данных, начиная с указанного в поле подтверждения, которые отправитель данного сегмента ожидает принять.

**Checksum - контрольная сумма: 16 битов**

Контрольная сумма представляет собой 16-битовое поразрядное дополнение суммы всех 16-битовых слов заголовка и данных. Если сегмент содержит в заголовке и поле данных нечётное число октетов, при подсчёте контрольной суммы справа добавляется октет нулей для выравнивания по 16-битовой границе. Биты заполнения не передаются как часть сегмента и используются только для расчёта контрольной суммы. При расчёте контрольной суммы значение поля Checksum принимается нулевым.

Адрес отправителя		
Адрес получателя		
0	PTCL	TCP Length

Контрольная сумма учитывает также 96-битовый псевдозаголовок, предшествующий заголовку TCP. Этот псевдозаголовок содержит адреса отправителя и получателя, тип протокола и размер опций TCP. Перечисленные поля помогают защитить TCP от сегментов с ошибочной маршрутизацией. Эта информация транспортируется протоколом IP и передаётся через интерфейс TCP - сетевой уровень в качестве аргументов или результатов вызовов из TCP на уровень IP.

Поле TCP Length содержит размер заголовка TCP и поля данных в октетах (это не явно передаваемое, а расчётное значение); 12-октетный псевдозаголовок при расчёте размера не учитывается.

**Urgent Pointer - указатель срочности: 16 битов**

Это поле содержит указатель на срочные данные - позитивное смещение начала таких данных от порядкового номера данного сегмента. Этот указатель задаёт номер октета, следующего за важными данными. Поле интерпретируется лишь для сегментов с установленным флагом URG.

**Options - опции: переменный размер**

Опции размещаются в конце заголовка TCP и могут занимать целое число октетов. Все опции учитываются при расчёте контрольной суммы. Опции могут начинаться на любой границе октета. Существует два варианта форматирования опций:

1. однооктетное поле вида опции;
2. однооктетное поле вида опции, поле размера опции (1 октет) и собственно опции.

Поле размера опции учитывает и 2 октета полей вида опции и самого поля размера, а также размер опции, как таковой.

Отметим, что поле опций может быть короче, чем указывает поле смещения данных. Неиспользуемые биты поля опций после опции End-of-Option должны заполняться нулями.

Протокол TCP должен поддерживать все опции.

Определённые к настоящему моменту опции включают (виды опций указаны восьмеричными значениями):

Вид	Размер	Значение	
0	-	End of option list	Конец списка опций
1	-	No-Operation	Нет операции
2	4	Maximum Segment Size	Максимальный размер сегмента

**Определения опций****End of Option List**

```
+-----+
|00000000|
+-----+
```

вид=0

Этот код говорит о завершении списка опций. Конец списка опций может не совпадать с концом заголовка TCP, заданным полем Data Offset. Код используется как индикатор завершения всех опций, а не какой-то конкретной и использование его требуется лишь в тех случаях, когда конец опций не совпадает с концом заголовка TCP.

**No-Operation**

```
+-----+
|00000001|
+-----+
```

вид=1

Этот код может использоваться между опциями (например, для их выравнивания по границе слова). Не существует гарантий использования этой опции отправителем, поэтому получатель должен быть готов к обработке опций, начало которых не совпадает с границей слова.

**Maximum Segment Size**

```
+-----+-----+-----+-----+
|00000010|00000100| max seg size |
+-----+-----+-----+-----+
```

вид=2 размер=4

**Max Seg Size - максимальный размер сегмента: 16 битов**



Если эта опция присутствует, она задаёт максимальный размер принимаемого сегмента для той стороны TCP, которая передаёт данный сегмент. Это поле должно передаваться только с начальным запросом организации соединения (сегмент с флагом SYN). Если эта опция не задана, допускается использование сегментов любого размера.

#### **Padding - заполнение: переменный размер**

Заполнение заголовка TCP используется для выравнивания размера заголовка по 32-битовой границе. Для заполнения служит значение 0.

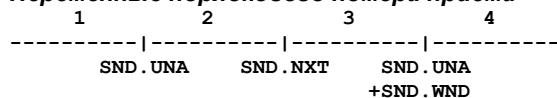
## 3.2. Терминология

Прежде, чем начать обсуждение деталей TCP, дадим определения некоторых терминов. Поддержка соединений TCP требует запоминания нескольких переменных. Предполагается, что эти переменные хранятся в специальном блоке управления передачей - TCB. Записи TCB включают номера локального и удалённого сокетов, опции безопасности и предпочтения для сегмента, указатели на пользовательские буферы приёма и передачи, указатели на очередь повторной передачи и текущий сегмент. Кроме того, в TCB хранится информация о порядковых номерах для приёма и передачи.

#### **Переменные порядкового номера передачи**

- SND.UNA - передать не подтверждённый;
- SND.NXT - передать следующий;
- SND.WND - окно передачи;
- SND.UP - передать указатель срочности;
- SND.WL1 - порядковый номер сегмента, использованный при последнем обновлении окна;
- SND.WL2 - порядковый номер подтверждения, использованный при последнем обновлении окна;
- ISS - начальный порядковый номер для передачи.

#### **Переменные порядкового номера приёма**



1. старые порядковые номера, которые были подтверждены
2. порядковые номера неподтвержденных данных
3. порядковые номера, допустимые для новой передачи данных
4. будущие порядковые номера, которые ещё не разрешены для использования

*Рисунок 4. Пространство номеров для передачи.*

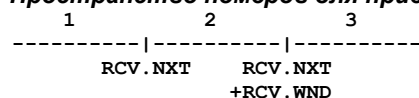
- RCV.NXT - принять следующий;
- RCV.WND - окно приёма;
- RCV.UP - приём указателя срочности;
- IRS - начальный порядковый номер для приёма;

Приведённые ниже рисунки помогут понять соотношения между переменными в пространстве порядковых номеров.

#### **Пространство номеров для передачи**

Окно передачи является частью области 3 на рисунке 4.

#### **Пространство номеров для приёма**



1. старые порядковые номера, которые были подтверждены
2. порядковые номера, допустимые для нового приёма данных
3. будущие порядковые номера, которые ещё не разрешены для использования

*Рисунок 5. Пространство номеров для приёма.*

Окно приёма является частью области 2 на рисунке 5.

Существует также ряд переменных, часто используемых при обсуждении, которые берут свои значения из полей текущего сегмента.

#### **Переменные текущего сегмента**

- SEG.SEQ - порядковый номер сегмента
- SEG.ACK - порядковый номер сегмента подтверждения
- SEG.LEN - длина сегмента
- SEG.WND - окно сегмента
- SEG.UP - указатель срочности сегмента
- SEG.PRC - значение предпочтений для сегмента

Соединение в процессе своего существования может находиться в нескольких состояниях - LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT и CLOSED (это состояние фиктивно, поскольку оно представляет ситуацию, когда уже нет TCB и, следовательно, соединения). Ниже кратко описаны все эти состояния.

- LISTEN - ожидание запроса на соединение от любого удалённого TCP и порта.
- SYN-SENT - ожидание соответствующего запроса на соединение после передачи своего запроса.

- SYN-RECEIVED - ожидание подтверждения соединения после передачи и приёма запросов на организацию соединения.
- ESTABLISHED - соединение действует и принятые данные могут быть доставлены пользователю. Это нормальное состояние для процесса обмена данными через соединение.
- FIN-WAIT-1 - ожидание запроса на разрыв соединения от удалённого TCP или подтверждения для ранее переданного запроса на разрыв соединения.
- FIN-WAIT-2 - ожидание запроса на разрыв соединения от удалённого TCP.
- CLOSE-WAIT - ожидание запроса на разрыв соединения от локального пользователя.
- CLOSING - ожидание подтверждения от удалённого TCP для запроса на разрыв соединения.
- LAST-ACK - ожидание подтверждения для запроса на разрыв соединения, переданного удалённому TCP (это подтверждение включается в запрос на разрыв соединения от удалённой стороны).
- TIME-WAIT - ожидание пока пройдёт достаточно времени, чтобы быть уверенным в приёме удалённым TCP подтверждения для его запроса на разрыв соединения.
- CLOSED - соединения уже нет (разорвано).

Соединение TCP переходит от одного состояния к другому в ответ на события, к числу которых относятся пользовательские вызовы OPEN, SEND, RECEIVE, CLOSE, ABORT и STATUS, входящие сегменты (в частности те, которые включают флаги SYN, ACK, RST, FIN) и тайм-ауты.

Диаграмма состояний на рисунке 6 иллюстрирует смену состояний в результате тех или иных событий и выполняемые действия, но не содержит информации о возможных ошибках и действиях, не связанных с изменением состояния. В последующих параграфах будет приведено более детальное описание реакций TCP на те или иные события.

### 3.3. Порядковые номера

Одним из фундаментальных аспектов TCP является нумерация данных - каждый октет, передаваемый через соединение TCP, имеет свой порядковый номер. Поскольку каждый октет пронумерован, для любого из октетов может быть передано подтверждение. Механизм подтверждений является накопительным<sup>1</sup>, поэтому подтверждение для порядкового номера X показывает, что все октеты до X (но не включая сам октет с номером X) были получены. Этот механизм позволяет обнаруживать дубликаты данных при использовании повторной передачи. Нумерация октетов в сегменте начинается от заголовка, т. е. октет, следующий сразу после заголовка, имеет наименьший порядковый номер, а номера следующих октетов последовательно возрастают.

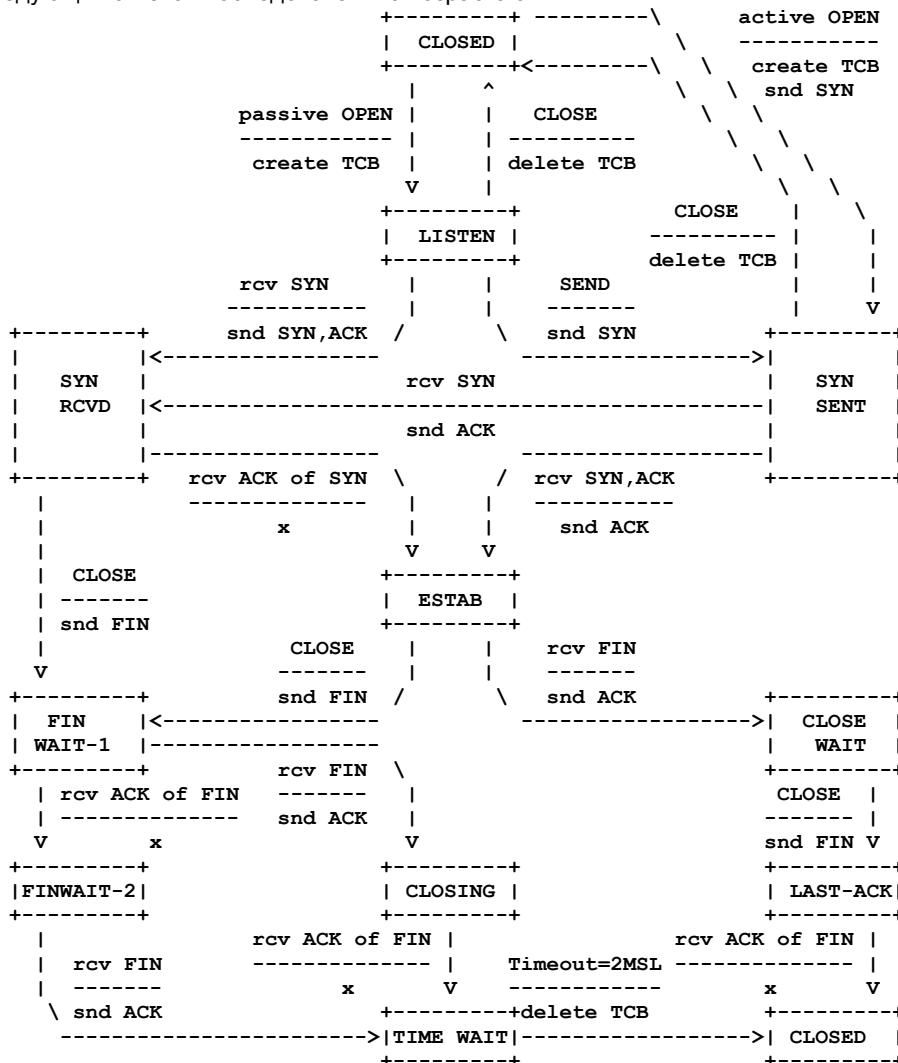


Рисунок 6. Состояния соединений TCP.

<sup>1</sup>В RFC 2883 был предложен в качестве расширения механизм селективных подтверждений. Прим. перев.

Важно помнить, что реальное пространство порядковых номеров имеет ограниченные размеры, хотя и достаточно велико (от 0 до  $2^{32} - 1$ ). Поскольку число порядковых номеров конечно, все арифметические операции с порядковыми номерами выполняются по модулю  $2^{32}$ . Такая беззнаковая арифметика сохраняет соотношения между порядковыми номерами при переходе номера от  $2^{32} - 1$  к нулю. В такой арифметике существуют некоторые тонкости, которые следует принимать во внимание при разработке программ, использующих сравнение значений. Символ  $=<$  означает «меньше или равно» (модуль  $2^{32}$ ).

Типичные операции сравнения порядковых номеров, используемые TCP, включают:

- Проверка того, что подтверждение указывает на некоторый порядковый номер для отправленных, но ещё не подтверждённых данных.
- Проверка того, что все порядковые номера, занимаемые сегментом, имеют подтверждение (например, для удаления сегмента из очереди повторной передачи).
- Проверка того, что входящий сегмент содержит ожидаемые порядковые номера (например, чтобы убедиться в том, что сегмент «перекрывается» с окном приёма).

В ответ на передачу данных TCP будет принимать подтверждения. При обработке подтверждений также выполняются операции сравнения порядковых номеров.

SND.UNA = самый старый неподтвержденный номер  
 SND.NXT = порядковый номер для следующей передачи  
 SEG.ACK = подтверждение от принимающего TCP (следующий порядковый номер, ожидаемый TCP)  
 SEG.SEQ = первый порядковый номер для сегмента  
 SEG.LEN = число октетов в сегменте, занятых данными (с учётом SYN и FIN)  
 SEG.SEQ + SEG.LEN - 1 = последний порядковый номер для сегмента

Для новых подтверждений (acceptable ack - приемлемое подтверждение) должно выполняться неравенство:

$SND.UNA < SEG.ACK \Rightarrow SND.NXT$

Сегмент в очереди на повторную передачу считается полностью подтверждённым, если сумма его порядкового номера и размера не превышает порядковый номер во входящем подтверждении.

Для приёма данных должны выполняться проверки следующих параметров:

RCV.NXT = следующий порядковый номер, ожидаемый во входящем сегменте и находящийся в левой (или нижней) части окна приёма  
 RCV.NXT+RCV.WND-1 = последний порядковый номер, ожидаемый во входящем сегменте и находящийся в правой (или верхней) части окна приёма  
 SEG.SEQ = первый порядковый номер, занимаемый входящим сегментом  
 SEG.SEQ+SEG.LEN-1 = последний порядковый номер, занимаемый входящим сегментом

Сегмент считается занимающим часть допустимого пространства порядковых номеров при условии:

$RCV.NXT \Rightarrow SEG.SEQ < RCV.NXT+RCV.WND$

или

$RCV.NXT \Rightarrow SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND$

Первое неравенство проверяет попадание в окно начала сегмента, второе относится к окончанию сегмента. Если выполняется хотя бы одно из условий, это говорит о том, что сегмент содержит данные из окна.

Реальные проверки несколько сложнее описанных. В результате использования нулевых окон и сегментов нулевой длины могут возникать четыре разных ситуации при восприятии входящих сегментов:

Размер сегмента	Приёмное окно	Проверка
0	0	$SEG.SEQ = RCV.NXT$
0	>0	$RCV.NXT \Rightarrow SEG.SEQ < RCV.NXT+RCV.WND$
>0	0	неприемлемо
>0	>0	$RCV.NXT \Rightarrow SEG.SEQ < RCV.NXT+RCV.WND$ или $RCV.NXT \Rightarrow SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND$

Отметим, что при нулевом размере приёмного окна никакие сегменты не принимаются, за исключением подтверждений ACK. Таким образом, TCP может поддерживать нулевое окно приёма во время повторной передачи, сохраняя возможность приёма сегментов ACK. Однако даже при нулевом приёмном окне протокол TCP должен обрабатывать поля RST и URG во всех входящих сегментах.

Используемая схема нумерации обеспечивает также защиту управляющей информации. Это достигается за счёт включения некоторых флагов управления в пространство порядковых номеров так, что они могут быть повторно переданы и подтверждены без конфликтов (т. е., используется одна и только одна копия флагов управления). Управляющая информация не переносится физически в пространстве данных сегмента. Следовательно, требуется адаптация правил выделения порядковых номеров с учётом флагов управления. Такая защита требуется только для флагов SYN и FIN, используемых лишь при организации и разрыве соединений. С учётом процессов порядковой нумерации флаг SYN размещается перед первым октетом данных в сегменте, а флаг FIN - вслед за последним октетом данных. Размер сегмента (SEG.LEN) учитывает поля данных и флагов управления. При наличии флага SYN переменная SEG.SEQ содержит порядковый номер SYN.

### Выбор начального номера

Протокол не задаёт каких-либо ограничений на повторное использование соединений. Соединение определяется парой сокетов. Новые экземпляры соединения будут рассматриваться как «реинкарнации». Однако в этом случае возникает проблема различий между инкарнации - «как протокол TCP сможет определить дубликаты от предыдущей

<sup>1</sup>В RFC 1323 была предложена опция Windows Scale, позволяющая масштабировать размер окна. *Прим. перев.*

инкарнации соединения?» Эта проблема возникает в результате кратковременных успешных соединений или при разрывах соединений с «потерей памяти» и повторной организацией.

Во избежание конфликтов с сегментами прежних инкарнаций нужно предотвратить использование тех порядковых номеров сегментов, которые могут оставаться в сети от предыдущей инкарнации. Такая возможность должна обеспечиваться даже после краха TCP и потери информации об использованных порядковых номерах. При организации нового соединения генерируется начальный порядковый номер ISN<sup>1</sup>. Генерация номера основана на текущем (возможно, фиктивном) 32-битовом значении времени, в котором младший бит инкрементируется приблизительно каждые 4 микросекунды. Таким образом, цикл номеров ISN занимает около 4,55 часа. Поскольку мы предполагаем, что сегмент сохраняется в сети в течение времени, не превышающего MSL<sup>2</sup>, и значение MSL < 4,55 час., можно считать значения ISN уникальными.

Для каждого соединения существует порядковый номер для приёма и передачи. Начальный порядковый номер для передачи (ISS) выбирается передающим модулем TCP, а начальный номер для приёма (IRS) определяется во время процедуры организации соединения.

Для организованных и инициализированных соединений два модуля TCP должны синхронизировать между собой порядковые номера. Это осуществляется в процессе обмена сегментами организации соединения, содержащими управляющий бит SYN (для синхронизации) и начальные порядковые номера. Для краткости сегменты с флагом SYN просто называют сегментами SYN. Следовательно, решение требует подходящего механизма для подбора начальных порядковых номеров и включает процедуру согласования параметров для обмена значениями ISN.

Для синхронизации требуется, чтобы каждая из сторон передала свой начальный порядковый номер удалённой стороне и получила от той подтверждение. Каждая из сторон должна получить от удалённой стороны её начальный порядковый номер и передать обратно подтверждение.

- 1) A → B SYN - мой порядковый номер X
- 2) A ← B ACK - ваш порядковый номер X
- 3) A ← B SYN - мой порядковый номер Y
- 4) A → B ACK - ваш порядковый номер Y

Поскольку пп. 2 и 3 могут быть объединены в одном сообщении, такую процедуру называют 3-этапным согласованием.

Трёхэтапное согласование необходимо, поскольку порядковые номера не связаны с глобальным сетевым временем и TCP может использовать различные механизмы выбора ISN. Получатель первого SYN не может проверить, какой это сегмент (не является ли он старой задержавшейся копией), пока не узнает последнего порядкового номера, использованного для соединения (это не всегда возможно), и поэтому он должен запросить у отправителя проверку SYN. Трёхэтапное согласование и преимущества схемы выделения номеров на основе времени рассмотрены в работе [3].

#### **Когда нужно сохранять паузу**

Для обеспечения уверенности в том, что TCP не создаёт сегментов с порядковыми номерами, которые могут дублироваться в старых сегментах, остающихся в сети, TCP должен сохранять молчание в течение максимального времени жизни сегмента MSL перед выделением каких-либо порядковых номеров для организации нового соединения или восстановления после сбоя с потерей информации об использованных порядковых номерах. Для данной спецификации значение MSL выбирается равным 2 минутам и может быть изменено, если практика покажет необходимость такого изменения. Отметим, что если по каким-то причинам происходит повторная инициализация TCP информация об использованных порядковых номерах сохраняется в памяти и ожидания не требуется, нужно лишь быть уверенным в том, что новые порядковые номера превышают использованные недавно.

#### **Концепция паузы TCP**

В соответствии с данной спецификацией хост после «краха» без сохранения каких-либо сведений о последних порядковых номерах, переданных через каждое активное (т. е., незакрытое) соединение, будет задерживать передачу сегментов TCP по крайней мере в течение максимального времени жизни сегмента MSL в сети, частью которой является «упавший» хост. В следующих параграфах приведены пояснения для этой спецификации. Разработчики TCP могут пренебречь паузой, но тогда будет возникать риск восприятия некоторых старых сегментов вместо новых или отбрасывания новых данных некоторыми получателями.

TCP потребляет часть пространства порядковых номеров всякий раз при формировании сегмента и передаче его в выходную очередь хоста-отправителя. Обнаружение дубликатов и механизм нумерации в протоколе TCP полагаются на уникальное связывание сегмента данных с пространством порядковых номеров, при котором не могут быть использованы все  $2^{32}$  значений порядковых номеров, пока сегменты с выделенными номерами не будут доставлены и подтверждены получателем. Все копии сегментов удаляются из сети. Без этого два различных сегмента TCP могут получить одинаковые или перекрывающиеся порядковые номера, что приведёт к конфликту на приёмной стороне, поскольку невозможно будет отличить новые данные от старой копии. Напомним, что каждый сегмент ограничен в пространстве порядковых номеров числом октетов данных в этом сегменте.

При нормальных условиях TCP сохраняет следующий порядковый номер для передачи и самый старый номер из ожидающих подтверждения, чтобы избежать ошибочного использования порядкового номера снова до того момента, как будет подтверждено его первое использование. Однако это не обеспечивает гарантии того, что старые дубликаты будут удалены из сети, поэтому пространство порядковых номеров сделано очень большим, чтобы снизить вероятность конфликта при одновременной доставке сегментов с перекрывающимися номерами. При скорости 2 Мбит/с затрачивается 4,5 часа на использование  $2^{32}$  порядковых номеров. Поскольку максимальное время жизни сегмента в сети точно не превышает нескольких десятков секунд, это обеспечивает достаточную защиту для недетерминированных сетей даже при скорости 10 Мбит/с. При скорости 100 Мбит/с цикл порядковых номеров занимает 5,4 мин. - это немного, но вполне достаточно<sup>3</sup>.

<sup>1</sup>Initial sequence number

<sup>2</sup>Maximum Segment Lifetime - максимальное время жизни сегмента.

<sup>3</sup>В высокоскоростных сетях ситуация несколько осложняется. См. [RFC 1323](#). Прим. перев.

Однако работа механизмов нумерации и обнаружения дубликатов может быть расстроена, если TCP на передающей стороне не помнит порядковых номеров, которые последними были использованы для данного соединения. Например, если TCP будет инициировать все соединения с порядковым номером 0, тогда при «крахе» и повторном запуске TCP может восстановить прежнее соединение (возможно после полудуплексного разрешения соединения) и передать пакеты с порядковыми номерами, которые будут совпадать или перекрываться с номерами пакетов, находящимися в сети (переданными в предыдущей инкарнации того же соединения). При отсутствии информации о порядковых номерах, используемых для конкретного соединения, спецификация TCP рекомендует отправителю сделать задержку на время MSL перед передачей сегментов в соединение, чтобы дать время на исчезновение из сети сегментов от предыдущей инкарнации. Эта проблема сохраняет актуальность даже на хостах, использующих текущее время для генерации начальных порядковых номеров.

Предположим, что при организации соединения нумерация начинается со значения S. Предположим также, что соединение используется не очень интенсивно и функция генерации начального порядкового номера ISN(t) берет значение порядкового номера (скажем, S1) последнего сегмента, переданного этим TCP для конкретного соединения. Далее предположим, что хост «упал», загрузился снова и организовал новую инкарнацию соединения. В качестве начального выбран порядковый номер S1 = ISN(t) — последний, использованный предыдущей инкарнацией соединения номер! Если восстановление происходит достаточно быстро, все старые дубликаты в сети, имеющие порядковые номера по соседству с S1, могут быть доставлены и восприняты, как новые пакеты получателем новой инкарнации соединения.

Проблема состоит в том, что хост при восстановлении может не знать, как долго продолжался «крах» и существуют ли в системе старые дубликаты от прежней инкарнации соединения.

Одним из способов решения этой проблемы является задержка передачи сегментов на время MSL при восстановлении после «краха» - *quite time*. Хосты, которые предпочитают не ждать, рискуют спровоцировать конфликт между старыми и новыми пакетами для получателя. Разработчики могут предоставить пользователям TCP возможность выбора для каждого соединения режима ожидания после краха или реализовать режим *quite time* для всех соединений. Обычно, даже при включённом механизме ожидания последнее становится ненужным по истечении времени MSL после загрузки хоста.

Каждый переданный сегмент занимает не менее одного порядкового номера в пространстве номеров. Используемые сегментом номера считаются занятыми в течение времени MSL. Если после «краха» новое соединение организуется слишком быстро и будет использоваться порядковые номера, выделенные для сегментов предыдущей инкарнации соединения, возможно перекрытие порядковых номеров и возникновение проблем на приёмной стороне.

### 3.4. Организация соединения

Для организации соединений используется процедура трехэтапного согласования. Эта процедура обычно инициируется одним TCP, а TCP на удалённой стороне даёт отклик. Трехэтапное согласование будет работать и при одновременном инициировании соединения с обеих сторон. При одновременной попытке каждый модуль TCP получает сегмент SYN, не содержащий подтверждения для переданного этой стороной сегмента SYN. Потенциально возможно прибытие старого дубликата SYN в процессе одновременной организации соединения. Для решения таких проблем можно использовать сегменты Reset.

Ниже рассмотрены несколько примеров организации соединений. Хотя в этих примерах не рассматривается синхронизация с использованием сегментов данных, они полностью легитимны, поскольку принимающая сторона TCP не будет доставлять данные пользователю до проверки их корректности (т. е., данные должны буферизоваться на приёмной стороне, пока соединение не перейдёт в состояние ESTABLISHED). Трехэтапное согласование снижает вероятность организации ложных соединений. Это согласование является компромиссом между расходом памяти и передачей сообщений, обеспечивающих информацию для проверки соединения.

Простейший вариант трехэтапного согласования показан на рисунке 7. Стрелка вправо (-->) на рисунке показывает отправку сегмента от TCP A к TCP B или прибытие сегмента в B из A. Левая стрелка (<-->) показывает передачу сегментов в обратном направлении. Трехточки (...) показывают сегменты, которые ещё остаются в сети (задержаны). XXX обозначает потерянные или отброшенные сегменты. В скобках приведены комментарии. Состояния TCP представлены **после** отправки или прибытия сегментов (содержимое сегментов показано в центральной части каждой строки). Содержимое сегментов представлено в сокращённом виде - порядковый номер, флаги управления и поле ACK. Остальные поля (размер окна, адреса, длина сегмента) для краткости и простоты опущены.

TCP A		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 7. Базовое 3-этапное согласование для синхронизации соединения.

В строке 2 на рисунке 7 TCP A начинает передачу сегмента SYN, говорящего об использовании порядковых номеров, начиная со 100. В строке 3 TCP B передаёт SYN и подтверждение для принятого SYN в адрес TCP A. Отметим, что поле подтверждения показывает ожидание TCP B приёма порядкового номера 101, подтверждающего SYN с номером 100.

В строке 4 TCP A отвечает пустым сегментом с подтверждением ACK для сегмента SYN от TCP B; в строке 5 TCP A передаёт некоторые данные. Отметим, что порядковый номер сегмента в строке 5 совпадает с номером в строке 4, поскольку ACK не занимает пространства порядковых номеров (если это сделать, придётся подтверждать подтверждения - ACK для ACK!).

Одновременное инициирование соединения лишь незначительно сложнее (см. рисунок 8) Каждое соединение TCP проходит от состояния CLOSED через SYN-SENT и SYN-RECEIVED в состояние ESTABLISHED.

TCP A		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> --> ESTABLISHED1

```

Рисунок 8. Одновременная синхронизация соединения.

Принципиальная необходимость использования трехэтапного согласования обусловлена стремлением избавиться от недоразумений, связанных со старыми дубликатами инициирования соединений. Для решения таких проблем используется специальное управляющее сообщение - reset (сброс). Если принимающая сторона TCP ещё находится в несинхронизированном состоянии (т. е., SYN-SENT, SYN-RECEIVED), она возвращается в состояние LISTEN при получении reset. Если TCP находится с засинхронизированном состоянии (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), соединение после reset разрывается и пользователю передаётся уведомление об этом. Позднее мы обсудим этот вопрос при рассмотрении полуоткрытых соединений.

TCP A	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. SYN-SENT <-- <SEQ=400><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
8. ESTABLISHED --> <SEQ=101><ACK=401><CTL=ACK>	--> ESTABLISHED

Рисунок 9. Действия при получении старого дубликата SYN.

На рисунке 9 показан простой пример решения проблемы со старым дубликатом. В строке 3 старый дубликат SYN принимается TCP B, который не знает об этом и даёт нормальный отклик (строка 4). TCP A детектирует некорректность поля ACK и возвращает RST (reset) с полем SEQ, делающим сегмент правдоподобным. TCP B, получив RST, возвращается в состояние LISTEN. Когда оригинальный сегмент SYN наконец принимается (строка 6), происходит нормальная синхронизация. Если SYN в строке 6 приходит до RST, может потребоваться более сложный обмен сообщениями с передачей RST в обоих направлениях.

### Полуоткрытые соединения и другие аномалии

Организованное соединение называют полуоткрытым, если одна из сторон TCP закрыла или прервала соединение от себя, а другая сторона не знает об этом или обе стороны находятся в рассинхронизированном состоянии в результате краха с потерей памяти. Такие соединения будут автоматически сбрасываться при попытке передачи данных в любом направлении. Однако, полуоткрытые соединения являются не совсем обычными и процедура восстановления для них будет отличаться.

Если на сайте A соединение больше не существует, тогда попытка пользователя с сайта B передать какие-либо данные через это соединение будет приводить к тому, что TCP B получит управляющее сообщение reset. Такое сообщение говорит TCP B о некорректности соединения и необходимости его разрыва.

Предположим, что пользователи A и B обменивались данными в момент краха TCP с потерей памяти на хосте A. В зависимости от операционной системы хоста A запускается тот или иной механизм повторной загрузки TCP. После восстановления TCP хост A будет пытаться организовать соединение заново или восстановить его. В результате хост может попытаться снова вызвать функцию OPEN для восстановления соединения или попытается вызвать SEND, надеясь, что соединение существует. Во втором случае будет получено сообщение об ошибке connection not open<sup>2</sup> от локального (A) TCP. При попытке организации нового соединения TCP A будет передавать сегмент, содержащий SYN (этот сценарий показан на рисунке 10). После краха TCP A пользователь пытается заново организовать соединение, а TCP B предполагает, что соединение существует по-прежнему.

TCP A	TCP B
1. (CRASH)	(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 10. Обнаружение полуоткрытого соединения.

При получении SYN (строка 3) TCP B, будучи в синхронизированном состоянии, видит входящий сегмент за пределами окна и отвечает подтверждением с порядковым номером следующего сегмента, который ожидается (ACK 100). TCP A видит, что этот сегмент (подтверждение) не подтверждает ничего из переданного им и, будучи в рассинхронизированном состоянии, передаёт RST, поскольку обнаружено полуоткрытое соединение. TCP B прерывает соединение (строка 5). TCP A будет продолжать попытки организовать соединение, используя стандартную трехэтапную процедуру согласования (см. рисунок 7).

Интересный случай наблюдается при крахе TCP A, когда TCP B пытается передать данные, предполагая наличие синхронизированного соединения (см. рисунок 11). В этом случае данные, приходящие TCP A от TCP B (строка 2), не могут быть восприняты по причине отсутствия соединения, поэтому TCP A будет слать RST. Сообщение RST воспринимается TCP B, после чего соединение будет разорвано.

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

Рисунок 11. Активная сторона пытается использовать полуоткрытое соединение.

На рисунке 12 показаны TCP A и TCP B в пассивном состоянии, ожидающие SYN. Старый дубликат, принимаемый TCP B (строка 2), заставляет B начать действия. Возвращается сегмент SYN-ACK (строка 3), который заставляет TCP A генерировать RST (ACK в строке 3 не может быть принят). TCP B принимает сигнал сброса и возвращается в своё пассивное состояние LISTEN.

<sup>1</sup>В исходном документе указан пакет <SEQ=101><ACK=301><CTL=ACK>, но в RFC 1122 эта ошибка исправлена. Прим. перев.

<sup>2</sup>Соединения не существует.

TCP A		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>	-->	(возврат в LISTEN!)
5. LISTEN		LISTEN

Рисунок 12. Старый дубликат SYN иницирует Reset при двух пассивных сокетах.

Возможно множество других ситуаций, каждая из которых разрешается при соблюдении рассмотренных ниже правил генерации и обработки RST.

### Генерация Reset

Как общее правило RST следует передавать всякий раз при получении сегмента, который представляется не предназначенным для данного соединения. В сомнительных случаях сигнал сброса передавать не следует.

Существует три группы состояний:

1. Если соединения не существует (CLOSED), тогда RST передаётся в ответ на любой входящий сегмент, за исключением другого RST. В частности, сегменты SYN, адресованные в несуществующее соединение, отбрасываются описанным способом.

Если входящий сегмент содержит поле ACK, RST берет порядковый номер из поля ACK, в остальных случаях для RST используется нулевой номер, а для поля ACK устанавливается значение, равное сумме порядкового номера и размера входящего сегмента. Соединение остаётся в состоянии CLOSED.

2. Если соединение находится в любом несинхронизированном состоянии (LISTEN, SYN-SENT, SYN-RECEIVED) и входящий сегмент подтверждает что-то, ещё не переданное (содержит неприемлемое подтверждение ACK), или входящий сегмент имеет уровень безопасности или запрошенное разделение (compartment), которые не соответствуют точно безопасности и разделению для соединения, передаётся reset.

Если наш сегмент SYN не был подтверждён и уровень предпочтения для входящего сегмента выше запрошенного уровня, повышается локальный уровень предпочтения (если это разрешено пользователем и системой) или передаётся RST; если же уровень предпочтения для входящего сегмента ниже запрошенного уровня, обработка продолжается как при соответствии уровней предпочтения (если удалённый TCP не может поднять уровень предпочтения до соответствия локальному уровню, это будет обнаружено в следующем полученном сегменте и соединение будет разорвано). Если наш сегмент SYN подтверждён (возможно в данном входящем сегменте), уровень предпочтения для входящего сегмента должен точно соответствовать локальному уровню; если это не так, нужно передать RST.

Если входящий сегмент содержит ACK, RST принимает порядковый номер из поля ACK; в остальных случаях используется нулевой номер, а в поле ACK помещается сумма порядкового номера и размера входящего сегмента. Состояние соединения не изменяется.

3. Если соединение синхронизировано (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), из любого неподходящего сегмента (с порядковым номером за пределами окна или неприемлемым номером подтверждения) следует извлекать только пустой сегмент подтверждения, содержащий текущий порядковый номер передачи, и подтверждение, показывающее следующий ожидаемый порядковый номер. Состояние соединения не изменяется.

Если входящий сегмент имеет уровень безопасности, разделение или уровень предпочтения, отличные от одноимённых параметров соединения, передаётся RST и соединение переводится в состояние CLOSED. Для RST берётся порядковый номер из поля ACK входящего сегмента.

### Обработка RST

Во всех состояниях, за исключением SYN-SENT, сегменты RST проверяются по их полям SEQ. Сброс считается корректным, если его порядковый номер попадает в окно. В состоянии SYN-SENT (сегмент RST принимается в ответ на изначальный сегмент SYN) RST считается приемлемым, если поле ACK подтверждает SYN.

Получатель RST сначала проверяет сегмент, а потом меняет состояние. Если получатель находился в состоянии LISTEN, сигнал сброса игнорируется. Если получатель находится в состоянии SYN-RECEIVED, а перед этим был в состоянии LISTEN, он возвращается в состояние LISTEN; в остальных случаях получатель разрывает соединение и переходит в состояние CLOSED. Если получатель был в любом другом состоянии, он прерывает соединение, уведомляя пользователя, и переходит в состояние CLOSED.

## 3.5. Завершение соединения

Операция CLOSE означает отсутствие данных для передачи. Уведомление о закрытии полнодуплексного соединения может породить недоразумения, поскольку принимающая сторона может не знать, как трактовать эту операцию. Предлагается трактовать CLOSE следующим образом - передавший CLOSE пользователь может продолжать использовать RECEIVE, пока не получит информацию о том, что другая сторона также использовала CLOSE. Таким образом, программа может инициировать несколько вызовов SEND, потом вызвать CLOSE и продолжать использовать RECEIVE, пока не будет получено уведомления о сбое RECEIVE по причине использования CLOSE на удалённой стороне. Предполагается, что TCP будет информировать пользователя о закрытии соединения удалённой стороной даже в тех случаях, когда все вызовы RECEIVE успешно обработаны, позволяя пользователю корректно завершить работу программы. TCP гарантированно доставит содержимое всех буферов SENT до того, как соединение будет закрыто, поэтому пользователь, который не ждёт возврата каких-либо данных, должен лишь дождаться успешного перехода соединения в состояние CLOSED, чтобы быть уверенным в передаче всей своей информации на удалённую сторону. Пользователь должен сохранять для чтения соединение, которое он закрыл для записи, пока TCP не сообщит об отсутствии каких-либо данных.

Существует три различных варианта:

- 1) Пользователь сообщает TCP о необходимости использовать CLOSE для завершения соединения.

2) Удалённый модуль TCP инициирует разрыв путём передачи управляющего сигнала FIN.

3) Обе стороны одновременно вызывают CLOSE.

#### 1: Завершение инициирует локальный пользователь

В этом случае сегмент FIN может быть сгенерирован и помещён в выходную очередь. После этого вызовы пользователем функции SEND уже не будут приниматься TCP и соединение перейдёт в состояние FIN-WAIT-1. Вызовы функции RECEIVE в этом состоянии допустимы. Все предшествующие сегменты и сегмент FIN будут передаваться, пока не будут получены для них подтверждения. Когда удалённая сторона TCP имеет подтверждённый сегмент FIN и передала свой FIN, первая сторона TCP может передать подтверждение ACK для этого FIN. Отметим, что TCP после получения FIN будет передавать ACK, но не будет слать свой сегмент FIN, пока локальный пользователь также не закроет соединение.

#### 2: TCP получает FIN из сети

При получении незапрошенного сегмента FIN из сети принимающая сторона может передать ACK и запросить у пользователя закрытие соединения. Пользователь будет вызывать CLOSE, а TCP в ответ - передавать FIN на удалённую сторону после передачи остающихся данных. TCP после этого будет ждать подтверждения своего сегмента FIN, а потом удалит соединение. Если ACK не приходит, соединение разрывается по истечении тайм-аута и пользователю передаётся уведомление.

#### 3: Оба пользователя закрывают соединение одновременно

Одновременный вызов CLOSE на обеих сторонах соединения приводит к обмену сегментами FIN. Когда все сегменты, предшествующие FIN, будут обработаны и подтверждены, каждая из сторон TCP может передать ACK для принятого сегмента FIN. После получения ACK обе стороны закрывают соединение.

TCP A		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 13. Нормальный порядок закрытия соединения

TCP A		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 14. Одновременное закрытия соединения.

## 3.6. Предпочтения и безопасность

Задача организации соединений заключается и в том, чтобы допускались только соединения между портами, работающими с одинаковым уровнем безопасности и разделения (compartment) с обеих сторон и с высшим из двух указанных уровней предпочтения.

Предпочтения и параметры безопасности, используемые TCP, в точности совпадают с одноимёнными параметрами протокола IP [2]. В данной спецификации термины security/compartment служат для обозначения параметров безопасности, используемых в IP, включая безопасность, разделение, группы пользователей и ограничения на обслуживание.

Попытка соединения с несоответствующими параметрами безопасности или более низким уровнем предпочтения приводит к сбросу соединения. Отказ от соединения при слишком низком уровне предпочтения происходит только после того, как будет получено подтверждение для сегмента SYN.

Отметим, что модули TCP, работающие только с принятым по умолчанию уровнем предпочтения, все равно будут проверять этот уровень для входящих сегментов и, возможно, повышать уровень предпочтения для данного соединения.

Параметры защиты могут использоваться даже в незащищённых средах (значения будут задавать неклассифицированные данные), поэтому хосты в незащищённой среде должны быть готовы к приёму параметров безопасности, хотя им нет нужды передавать эти параметры.

## 3.7. Обмен данными

После организации соединения передача данных осуществляется путём обмена сегментами. Поскольку сегменты могут теряться в результате ошибок (некорректная контрольная сумма) или насыщения сети, TCP использует механизм повторной передачи (после тайм-аута) для обеспечения доставки каждого сегмента. В результате этого могут возникать дубликаты сегментов. Как было сказано при обсуждении порядковых номеров, TCP выполняет ряд проверок для порядковых номеров и номеров подтверждений перед восприятием сегментов.

Отправитель данных сохраняет следующий порядковый номер (который будет использован) в переменной SND.NXT. Получатель сохраняет следующий порядковый номер (который предполагается получить) в переменной RCV.NXT. Отправитель данных сохраняет порядковый номер последнего неподтвержденного сегмента в переменной SND.UNA. Если поток данных сейчас не передаётся и все отправленные ранее данные подтверждены, значения всех трёх переменных будут совпадать.

Когда отправитель создаёт сегмент и передаёт его, значение SND.NXT увеличивается. Получатель, принимая сегмент, увеличивает значение RCV.NXT и передаёт подтверждение. Когда отправитель данных получает для них подтверждение, он увеличивает значение SND.UNA. Различия в значениях указанных переменных связаны с задержками при передаче сегментов через сеть. Величина, на которую изменяются значения переменных, определяется размером данных в сегменте. Отметим, что в состоянии ESTABLISHED все сегменты должны передавать текущую информацию для подтверждений.

При вызове пользователем функции CLOSE выполняется выталкивание данных (функция push), как это делает управляющий флаг FIN во входящем сегменте.

### **Тайм-аут повторной передачи**

В результате постоянно происходящих в сети изменений и использования различных соединений TCP значение тайм-аута для повторной передачи должно динамически изменяться. Ниже приведён пример определения значения тайм-аута для повторной передачи.

### **Пример процедуры тайм-аута**

Измеряется задержка между передачей октета данных с конкретным порядковым номером и получением подтверждения, покрывающего этот порядковый номер (переданные сегменты не точно соответствуют принятым), в единицах RTT<sup>1</sup>. После этого рассчитывается значение SRTT<sup>2</sup>, как:

$$SRTT = ( \text{ALPHA} * SRTT ) + ((1-\text{ALPHA}) * RTT)$$

и на основе этого рассчитывается тайм-аут для повторной передачи (RTO):

$$RTO = \min[UBOUND, \max[LBOUND, (\text{BETA} * SRTT)]]^3$$

где UBOUND задаёт верхний предел значения тайм-аута (например, 1 минута), LBOUND - нижний предел (например, 1 секунда), ALPHA - весовой фактор (например, 0.8 - 0.9), а BETA - коэффициент вариаций задержки (например, 1.3 - 2.0).

### **Обмен срочной информацией**

Механизм разделения информации по срочности позволяет передающей стороне стимулировать принимающую сторону на восприятие некоторых срочных данных и даёт приёмной стороне TCP возможность проинформировать пользователя, когда все полученные данные особой срочности будут переданы ему.

Этот механизм позволяет задать в потоке данных точку завершения данных особой срочности. Когда значение этого указателя превышает текущий порядковый номер для приёма (RCV.NXT), модуль TCP должен сообщить пользователю о необходимости перехода в режим urgent; когда принимаемый порядковый номер превысит значение указателя срочности модуль TCP должен сказать пользователю о том, что нужно перейти в нормальный режим. Если указатель срочности обновляется, когда пользователь находится в режиме urgent, этого изменения пользователь не увидит.

Для реализации этого метода используется флаг срочности (URG), передаваемый в каждом сегменте. Флаг URG показывает, что поле urgent имеет смысл и его значение должно быть добавлено к порядковому номеру сегмента для получения указателя на срочные данные. Отсутствие флага срочности говорит, что в сегменте нет срочных данных.

Передача указателя срочности говорит пользователю, что он также должен передать по крайней мере один октет данных. Если передающий пользователь применяет также функцию выталкивания, доставка срочной информации получателю ускоряется.

### **Управление окном**

Окно, передаваемое в каждом сегменте, показывает диапазон порядковых номеров отправителя окна (получателя данных), которые он готов принять. Предполагается, что это значение связано с доступным в настоящее время буферным пространством для данного соединения.

Указание большого окна стимулирует передачу. Если данных приходит больше, чем может быть воспринято, они отбрасываются, в результате чего возникают повторы передачи, добавляющие ненужную нагрузку на сеть и TCP. Задание маленького окна будет ограничивать передачу данных - это связано с задержкой обхода между передачей каждого нового сегмента.

Этот механизм позволяет TCP анонсировать большое окно и впоследствии многократно снижать размер окна без восприятия большого объёма данных. Такое явление называется «сжатием окна» и вносит серьёзные препятствия. Принцип устойчивости требует от TCP «не сжимать своё окно, но быть готовым к такому сжатию со стороны других TCP».

Передающая сторона TCP должна быть готова принять от пользователя и передать по крайней мере один октет новых данных, даже если она передала нулевое окно. Передающая сторона TCP должна регулярно повторять передачу приёмной стороне даже при установке нулевого окна. При установке нулевого окна рекомендуется повторять передачу каждые 2 минуты. Такой повтор важен для обеспечения гарантии, что при использовании другой стороной TCP нулевого размера окна при повторном открытии окна другая сторона гарантированно узнает об этом.

Когда принимающая сторона TCP имеет нулевое окно и получает сегмент данных, она сначала должна передать подтверждение, показывающее следующий ожидаемый номер и текущий размер окна (0).

Передающая сторона TCP пакует данные для передачи в сегменты, которые помещаются в текущее окно, и возможна также переупаковка сегментов в очереди повторной передачи. Такая переупаковка не является обязательной, но может оказаться полезной.

В соединении с односторонним потоком данных информация об окне будет передаваться в сегментах подтверждений, которые имеют те же порядковые номера, поэтому при некорректном порядке доставки порядок подтверждений также изменится. Эта проблема не очень серьёзна, но может приводить к временному использованию устаревшей информации об окне. Во избежание такой проблемы следует брать информацию об окне из сегмента с максимальным номером подтверждения (т. е., сегмента с номером подтверждения не меньшим, чем любой из полученных ранее).

<sup>1</sup>Round Trip Time - время кругового обхода.

<sup>2</sup>Smoothed Round Trip Time - взвешенное время кругового обхода.

<sup>3</sup>В параграфах 4.2.2.15 и 4.2.3.1 RFC [1122](#) внесены поправки в алгоритм расчёта тайм-аута. *Прим. перев.*



Процедура управления окном оказывает существенное влияние на производительность обмена данными. Ниже приведены рекомендации по реализации таких процедур.

### **Предложения по управлению окном**

Выделение слишком маленького окна заставляет передавать данные очень мелкими сегментами, снижая производительность обмена информацией.

Для того, чтобы избежать использования слишком мелких окон, приёмной стороне следует воздерживаться от обновления окна, пока не будет возможно выделить по крайней мере X% от максимального размера (X может принимать значение от 20 до 40).

Другим способом является предотвращение передачи слишком мелких сегментов отправителем за счёт ожидания перед отправкой данных расширения окна до подходящего размера. Если пользователь вызывает функцию push, данные должны передаваться даже в виде небольших сегментов.

Отметим, что передача подтверждений не должна слишком задерживаться, поскольку такая задержка приведёт к повторам передачи. Одним из вариантов является передача подтверждения при получении мелкого сегмента (без информации об изменении окна) и передача другого подтверждения с новой информацией об увеличении окна.

Сегмент, передаваемый для проверки нулевого окна, может инициировать разбиение данных на все более мелкие сегменты. Если сегмент с одним октетом данных передать для проверки нулевого окна, этот сегмент потребит один октет доступной части окна. Если передающая сторона TCP просто будет передавать сегменты, как будто окно имеет ненулевой размер, передача данных будет вестись в форме больших и мелких сегментов поочередно. В таких ситуациях случайная пауза при выделении окна на приёмной стороне будет приводить к разбиению крупных сегментов на мелкие и средние. В конечном итоге практически все данные станут передаваться в мелких сегментах.

Основным предложением для реализации TCP является необходимость активных попыток объединения мелких окон в окна большего размера, поскольку механизмы управления окном имеют тенденцию к снижению размера окна во многих простых реализациях.

## **3.8. Интерфейсы**

Протокол TCP поддерживает два интерфейса - с пользовательским уровнем и протоколом нижележащего уровня. Здесь будет подробно рассмотрена спецификация интерфейса между TCP и пользовательским уровнем, а взаимодействие с протоколами нижележащего уровня будет описано в спецификациях соответствующих протоколов, поэтому здесь мы не будем надолго останавливаться на этом интерфейсе. Для случая взаимодействия с протоколом IP мы рассмотрим некоторые параметры, которые могут использоваться в TCP.

### **Интерфейс TCP - пользователь**

Ниже приведено функциональное описание пользовательских команд TCP, которые могут существенно отличаться в различных операционных системах. Поэтому мы должны предупредить читателей о том, что различные реализации TCP могут иметь существенно отличающиеся пользовательские интерфейсы. Однако, все варианты TCP должны обеспечивать минимальный набор служб, гарантирующий поддержку всеми реализациями одинаковой иерархии протоколов. В этом разделе приведено функциональное описание компонент интерфейса, требуемых для любой реализации TCP.

#### **Пользовательские команды TCP**

В следующих параграфах приведены функциональные характеристики интерфейса между TCP и пользовательским уровнем. Используемая здесь нотация похожа на обозначения процедур и функций в программных языках высокого уровня, но это не означает запрета на использование сервиса типа ловушек (прерываний) - trap (например, SVC, UUC, EMT).

Описанные ниже пользовательские команды задают базовую функциональность TCP для поддержки обмена информацией между процессами. Каждая реализация должна точно определять свой формат и может объединять комбинации или подмножества базовых функций в одном вызове. В частности, некоторые реализации могут автоматически организовывать (OPEN) соединение при первом вызове SEND или RECEIVE, сделанном пользователем для данного соединения.

Для обеспечения взаимодействия между процессами от TCP требуется обеспечить не только восприятие команд, но и возврат информации от обслуживающих эти команды процессов. Эта информация включает:

- (а) общие сведения о соединении (например, прерывания, удалённое закрытие, связывание с неуказанным внешним сокетом).
- (б) отклики на конкретные пользовательские команды, содержащие результат выполнения (например, код ошибки).

#### **Open**

**формат:** OPEN (локальный порт, внешний сокет, активный/пассивный [, тайм-аут] [, предпочтение] [, безопасность/разделение] [, опции]) -> локальное имя соединения

Будем предполагать, что локальный модуль TCP способен идентифицировать обслуживаемые процессы и проверить их полномочия в части доступа к указанному соединению. В зависимости от реализации TCP идентификаторы локальной сети и TCP для адресов отправителей будут предоставляться TCP или протоколом нижележащего уровня (например, IP). Такое предположение является результатом учёта вопросов безопасности и предназначено для предотвращения маскировки программ TCP под другие программы и т. п. Точно так же никакой процесс не может маскироваться под другой процесс без конфликта с TCP.

Если флаг активности установлен в пассивное состояние, это означает вызов для перехода в состояние LISTEN (прослушивание входящих соединений). При пассивной организации внешний сокет может быть задан полностью (ждать вызова только от этого сокета) или не задан совсем (принимать любые вызовы). Пассивный вызов с указанным внешним сокетом можно впоследствии активизировать с помощью вызова SEND.

При организации соединения создаётся блок управления передачей TCB, который частично заполняется на основе параметров команды OPEN.

При активном вызове OPEN модуль TCP начинает процедуру синхронизации (организации) соединения.



Значение тайм-аута (если оно задано) позволяет вызвавшему процессу установить время ожидания для всех данных, направленных TCP. Если в течение заданного времени данные не были доставлены получателю, TCP будет разрывать соединение. В настоящее время принято значение тайм-аута 5 мин.

TCP или компоненты операционной системы будут проверять полномочия пользователя в части организации соединений с заданным уровнем безопасности. Отсутствие параметров безопасности и уровня предпочтения при вызове OPEN говорит об использовании принятых по умолчанию значений.

TCP будет воспринимать входящие запросы только при условии точного соответствия параметров безопасности и уровне предпочтения не ниже, чем был задан при вызове OPEN.

Уровнем предпочтения для соединения выбирается большее из значений, заданных при вызове OPEN и полученных во входящем запросе. Выбранное значение фиксируется на все время существования соединения. Разработчики могут пожелать предоставить пользователю контроль за установкой уровня предпочтения. Например, пользователь может указывать необходимость точного соответствия уровней предпочтения или требуется подтверждение пользователя при попытке повысить уровень предпочтения.

Локальное имя соединения возвращается пользователю модулем TCP. Это имя может применяться в качестве краткого обозначения соединений вместо пары <локальный сокет, внешний сокет>.

### Send

**Формат:** SEND (локальное имя соединения, адрес буфера, счётчик байтов, флаг PUSH, флаг URGENT [, тайм-аут])

Этот вызов заставляет передать в заданное соединение данные из указанного пользовательского буфера. Если соединение ещё не организовано, вызов SEND трактуется как ошибка. Некоторые реализации могут позволять вызовы SEND до организации соединения; в таких случаях автоматически вызывается функция OPEN. Если вызывающий процесс не имеет полномочий на использование указанного соединения, возвращается сообщение об ошибке.

Если установлен флаг PUSH, данные должны быть быстро переданы получателю и флаг PUSH устанавливается для последнего сегмента TCP, созданного из буфера. При отсутствии флага PUSH данные могут объединяться с данными от последующих вызовов SEND для повышения эффективности передачи.

При установке флага URG сегмент передаётся получателю с указателем срочности. Принимающая сторона TCP будет сообщать о наличии срочных данных принимающему процессу, если указатель срочности ещё не попадает в число обработанных этим процессом данных. Использование флага и указателя срочности позволяет стимулировать обработку срочных данных и информировать принимающий процесс о завершении приёма срочной информации. Число флагов срочности на передающей стороне TCP не обязано совпадать с числом фактов информирования принимающего процесса о присутствии срочных данных.

Если при вызове OPEN внешний сокет не был указан, но соединение организовано (например, в состоянии LISTEN принят вызов для данного локального сокета), указанный буфер передаётся косвенно заданному (подразумеваемому) внешнему сокету. Пользователи, вызывающие OPEN с незадаанным сокетом, могут вызывать SEND, даже не зная адрес внешнего сокета.

Однако вызов SEND до того, как внешний сокет определится, будет приводить к возврату сообщения об ошибке. Пользователь может вызвать функцию STATUS для проверки состояния соединения. В некоторых реализациях TCP может уведомлять пользователя при отсутствии указаний на внешний сокет.

Если задан тайм-аут, текущее время ожидания для данного соединения будет заменено новым значением.

В простейших реализациях SEND может не возвращать управления вызвавшему процессу, пока не будет завершена передача или не истечёт заданное время ожидания. Однако, такой подход может приводить к простоям (например, обе стороны соединения могут попытаться вызвать SEND до получения каких-либо данных с помощью RECEIVE) и не обеспечивает достаточной производительности, поэтому не рекомендуется к использованию. Более сложные реализации будут незамедлительно возвращать управление, позволяя процессу работать одновременно с сетевыми операциями ввода-вывода. В таких случаях может одновременно использоваться множество экземпляров SEND, обслуживание выполняется в порядке вызовов и TCP будет выстраивать данные в очередь.

Предполагается, что взаимодействие с пользовательским уровнем происходит асинхронно и вызов SEND сопровождается генерацией некоего сигнала (или псевдопрерывания) со стороны обслуживающего вызов TCP. Другим вариантом является незамедлительный отклик. Например, SEND может незамедлительно возвращать локальное подтверждение даже при отсутствии подтверждения доставки от удалённого TCP. Будем оптимистически предполагать успешную передачу. Если это не так, соединение будет разрываться по тайм-ауту. Реализации такого типа (синхронные) продолжают использовать асинхронные сигналы, но эти сигналы относятся к соединениям, а не сегментам или буферам.

Для того, чтобы процесс мог различать сообщения (код результата) от множества SEND вместе с результатом может возвращаться адрес использованного при вызове SEND буфера. Сигналы от TCP к пользователю, обсуждаемые ниже, содержат информацию, которая должна быть возвращена вызвавшему функцию процессу.

### Receive

**Формат:** RECEIVE (локальное имя соединения, адрес буфера, счётчик байтов) -> счётчик байтов, флаг срочности, флаг выталкивания

Эта команда выделяет приёмный буфер, связывая его с указанным соединением. Если до этого не было вызова OPEN или процесс не имеет полномочий на использование данного соединения, возвращается сообщение об ошибке.

В простейших реализациях управление может не возвращаться вызвавшему функцию процессу, пока не будет заполнен буфер или не произойдёт ошибка. Такое решение неэффективно, поскольку приводит к простоям. Более сложные реализации обеспечивают возможность использования нескольких экземпляров RECEIVE. Выделенные функциями буферы будут заполняться по мере прибытия сегментов. Такая стратегия

позволяет повысить производительность, но требует организации механизма (возможно, асинхронного) уведомления вызывающей программы о получении флага PUSH или заполнении буфера.

Если буфер заполняется до прихода PUSH, функция не возвращает флага выталкивания PUSH. При получении флага PUSH до заполнения буфера, функция будет возвращать содержимое частично заполненного буфера и устанавливать при возврате флаг PUSH.

При получении срочных данных пользователю будет передаваться уведомление с помощью сигнала пользовательского интерфейса TCP. Принимающий пользователь должен находиться в режиме urgent. Если флаг URG установлен, это говорит о наличии дополнительных срочных данных. Отсутствие флага URG говорит о том, что функция RECEIVE возвратила все срочные данные и пользователь может перейти в нормальный режим. Отметим, что данные после указателя срочности (несрочные данные) не могут быть доставлены пользователю в том же буфере, где содержатся срочные данные, если граница не указана для пользователем явно.

Для того, чтобы можно было различать буферы разных экземпляров RECEIVE и видеть полноту заполнения буфера, функция возвращает указатель на буфер и число байтов, которые были реально приняты в буфер.

Некоторые реализации RECEIVE могут использовать буферное пространство TCP или TCP может использовать кольцевой буфер совместно с пользовательскими процессами.

### Close

**Формат:** CLOSE (локальное имя соединения)

Эта команда закрывает указанное соединение. Если соединение не открыто или вызывающий процесс не имеет полномочий для работы с указанным соединением, функция возвращает сообщение об ошибке. Закрытие соединений осуществляется корректно и все данные незавершённых вызовов SEND будут переданы (возможно, повторно) с сохранением управления потоком данных. Таким образом, можно сделать несколько вызовов SEND, после которых будет следовать вызов CLOSE и быть уверенным в передаче всех данных. Следует помнить, что пользователи должны продолжать вызовы RECEIVE для закрываемого соединения, пока другая сторона пытается передать остающиеся у неё данные. Таким образом, вызов функции CLOSE означает, что больше нет данных для передачи, но вовсе не говорит о том, что данные больше не принимаются. Может случиться (если протокол пользовательского уровня реализован некорректно), что закрывающая соединение сторона не сможет передать все имеющиеся у неё данные до завершения тайм-аута. В этом случае CLOSE транслируется в ABORT и соединение TCP рвётся.

Пользователь может закрыть соединение с помощью CLOSE в любой момент по своей инициативе или в ответ на предложение со стороны TCP (например, закрытие соединения удалённой стороной, тайм-аут при передаче, недоступность получателя).

Поскольку закрытие соединения требует обмена информацией с внешним TCP, соединение в течение короткого времени может сохраняться в закрытом состоянии. Попытки открыть соединение заново до того, как TCP вернёт отклик на команду CLOSE будет приводить к ошибке.

CLOSE также косвенно выполняет функцию выталкивания данных.

### Status

**Формат:** STATUS (локальное имя соединения) -> информация о состоянии

Эта пользовательская команда зависит от реализации и может быть исключена без негативного влияния на работу TCP. Возвращаемая командой информация обычно является копией содержимого TCB для указанного соединения.

Команда возвращает блок данных, содержащий:

- локальный сокет,
- внешний сокет,
- локальное имя соединения,
- окно приёма,
- окно передачи,
- состояние соединения,
- число буферов, ожидающих подтверждения,
- число буферов, ожидающих приёма,
- состояние срочности,
- предпочтения,
- безопасность/разделение,
- тайм-аут для передачи.

В зависимости от состояния соединения и/или реализации некоторые поля могут быть недоступными или неоднозначными. Если вызвавший функцию процесс не имеет доступа к соединению, возвращается сообщение об ошибке.

### Abort

**Формат:** ABORT (локальное имя соединения)

Эта команда прерывает работу всех ожидающих вызовов SEND и RECEIVE, удаляет TCB и передаёт специальное сообщение RESET на удалённую сторону TCP-соединения. В зависимости от реализации пользователь может получить уведомления о прерывании для всех оставшихся незавершёнными вызовов SEND и RECEIVE или подтверждение операции прерывания.

### Сообщения от TCP к пользователю

Предполагается, что операционная система обеспечивает возможность асинхронной передачи сигналов от TCP к пользовательским программам. Когда TCP подаёт сигнал пользователю, с этим сигналом передаётся некая

информация (например, сообщение об ошибке). В некоторых случаях может передаваться информация о состоянии обработки SEND, RECEIVE или других пользовательских вызовов.

В сигналах содержатся следующие сведения:

Локальное имя соединения	всегда
Строка отклика	всегда
Адрес буфера	Send и Receive
Счётчик принятых байтов	Receive
Флаг Push	Receive
Флаг Urgent	Receive

### Интерфейс TCP - нижележащий уровень

TCP обращается к модулю протокола нижележащего уровня для реальной передачи или приёма данных через сеть. Одним из примеров такой реализации является система ARPA, использующая на нижележащем уровне модуль IP [2].

При использовании на нижележащем уровне протокола IP последний обеспечивает аргументы для указания типа обслуживания и времени жизни. TCP может использовать для этих параметров следующие значения:

Тип обслуживания (Type of Service) = Предпочтения (Precedence): routine, Задержка (Delay): normal, Пропускная способность (Throughput): normal, Надежность (Reliability): normal (эти значения задаются числом 00000000).

Время жизни (Time to Live) = 1 минута или 00111100<sup>1</sup>.

Отметим, что принятое максимальное время жизни пакетов составляет две минуты и мы здесь явно указываем необходимость уничтожения пакетов, которые не были доставлены в течение 1 минуты.

Если на нижележащем уровне работает протокол IP (или иной протокол, обеспечивающий такие возможности) и используется маршрутизация, заданная отправителем (source routing), интерфейс должен разрешать обмен маршрутной информацией. Это особенно важно потому, что адреса отправителя и получателя используются при расчёте контрольной суммы TCP на приёмной и передающей стороне. Важно также сохранить маршрут возврата для ответа на запрос организации соединения.

Любой протокол нижележащего уровня должен предоставлять информацию об адресах отправителя и получателя, поле протокола, а также тот или иной способ определения размера TCP. Эти данные обеспечивают функциональный эквивалент служб протокола IP и используются при расчёте контрольных сумм TCP.

## 3.9. Обработка событий

Рассмотренный ниже пример обработки событий является лишь одним из возможных вариантов реализации. В конкретных реализациях порядок обработки может несколько отличаться, но эти различия должны быть только в деталях, а не в сути.

Действия TCP можно рассматривать как отклики на события. Происходящие события можно разбить на три категории - пользовательские вызовы, доставка сегментов и тайм-ауты. В этом разделе описаны действия TCP в ответ на события каждого из перечисленных типов. Во многих случаях требуемая в ответ на событие обработка зависит от состояния соединения.

События:

Пользовательские вызовы

OPEN

SEND

RECEIVE

CLOSE

ABORT

STATUS

Доставка сегментов

SEGMENT ARRIVES

Тайм-ауты

TIMEOUT

RETRANSMISSION TIMEOUT

TIME-WAIT TIMEOUT

Модель пользовательского интерфейса TCP базируется на немедленном возврате из пользовательских вызовов и возможно задержанных<sup>2</sup> откликах на вызов с помощью события или псевдопрерывания. В последующих описаниях термин сигнал будет указывать причину асинхронного отклика.

Сообщения об ошибках приводятся в форме символьных строк. Например, при вызове команды, которая обращается к несуществующему соединению, будет возвращаться сообщение «errno: connection not open» (ошибка: соединение не открыто).

Отметим также, что все арифметические операции с порядковыми номерами, номерами подтверждений, окнами и т. п. основаны на модуле  $2^{32}$  (размер пространства порядковых номеров). Значок =< означает «меньше или равно» (по модулю  $2^{32}$ ).

Естественным вариантом процесса обработки входящих сегментов является сначала проверка корректности порядкового номера (т. е., его «попадания» в окно приёма), размещение в очереди и последующая обработка в порядке возрастания номеров.

Когда сегмент перекрывается с полученным ранее сегментом, он реконструируется таким образом, чтобы в сегменте содержались только новые данные (поля заголовков изменяются в соответствии с новым содержимым).

Отметим, что если изменение состояния TCP не указано, это говорит о сохранении прежнего состояния.

<sup>1</sup>Это требование устарело и время жизни не обязано составлять 60 секунд (см. RFC-1122, 4.2.2.19). Прим. перев.

<sup>2</sup>Асинхронных (прим. перев.)

**OPEN****Состояние CLOSED (TCB не существует)**

Создаётся новый блок управления передачей (TCB) для хранения данных о состоянии соединения. Заполняются поля идентификаторов локального и внешнего сокета, предпочтений, безопасности, а также пользовательский тайм-аут. Отметим, что при пассивном вызове OPEN внешний сокет может быть не указан полностью и это поле задаётся после приёма входящего сегмента SYN. Осуществляется проверка корректности задания пользователем уровня безопасности и предпочтения - при недопустимом для пользователя уровне возвращается сообщение об ошибке «error: precedence not allowed» (недопустимый уровень предпочтения) или «error: security/compartiment not allowed» (недопустимый уровень безопасности,разделения). При пассивном вызове осуществляется переход в состояние LISTEN и возврат. Если вызов активный и внешний сокет не указан, возвращается сообщение об ошибке «error: foreign socket unspecified» (внешний сокет не задан); если внешний сокет при активном вызове указан полностью, генерируется сегмент SYN. Выбирается значение начального порядкового номера ISS. Передаётся сегмент SYN в форме <SEQ=ISS><CTL=SYN>. Устанавливаются значения полей SND.UNA = ISS, SND.NXT = ISS+1, осуществляется переход в состояние SYN-SENT и возврат управления.

Если вызывающий процесс не имеет доступа к указанному локальному сокету, выдаётся сообщение об ошибке «error: connection illegal for this process» (некорректное соединение для данного процесса). Если нет возможности организовать новое соединение, возвращается сообщение «error: insufficient resources» (нехватка ресурсов).

**Состояние LISTEN**

Если вызов активный и внешний сокет полностью задан, соединение переходит из пассивного состояния в активное и задаётся значение ISS. Передаётся сегмент SYN, устанавливаются значения SND.UNA = ISS, SND.NXT + ISS+1. Соединение переводится в состояние SYN-SENT. Данные, связанные с вызовом SEND, могут быть переданы в сегменте SYN или помещены в очередь на передачу для их отправки после перехода в состояние ESTABLISHED. Если при вызове команды был запрошен флаг срочности, он передаётся с сегментом данных, отправляемым в результате этой команды. Если в очереди запросов нет свободного места, возвращается сообщение «error: insufficient resources». Если внешний сокет не был указан, выдаётся сообщение «error: foreign socket unspecified».

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

Возвращается сообщение об ошибке "error: connection already exists" (соединение уже существует).

**SEND****Состояние CLOSED (TCB не существует)**

Если у пользователя нет прав доступа к соединению, возвращается сообщение «error: connection illegal for this process» (некорректное соединение для данного процесса).

В остальных случаях возвращается сообщение «error: connection does not exist» (соединения не существует).

**Состояние LISTEN**

Если указан внешний сокет, соединение переводится из пассивного состояния в активное и выбирается значение ISS. Передаётся сегмент SYN, устанавливаются значения SND.UNA = ISS, SND.NXT + ISS+1. Соединение переводится в состояние SYN-SENT. Данные, связанные с вызовом SEND, могут быть переданы в сегменте SYN или помещены в очередь на передачу для их отправки после перехода в состояние ESTABLISHED. Если при вызове команды был запрошен флаг срочности, он передаётся с сегментом данных, отправляемым в результате этой команды. Если в очереди запросов нет свободного места, возвращается сообщение «error: insufficient resources». Если внешний сокет не был указан, выдаётся сообщение «error: foreign socket unspecified».

**Состояния SYN-SENT****SYN-RECEIVED**

Данные помещаются в очередь для передачи после перехода в состояние ESTABLISHED. Если в очереди нет свободного места, возвращается сообщение «error: insufficient resources».

**Состояния ESTABLISHED****CLOSE-WAIT**

Буфер сегментируется и передаётся с вложенным подтверждением (RCV.NXT). При нехватке памяти для запоминания заданного буфера просто возвращается сообщение «error: insufficient resources».

Если задан флаг срочности, тогда SND.UP <- SND.NXT-1 и для исходящих сегментов задаётся указатель срочности.

**Состояния FIN-WAIT-1**

**FIN-WAIT-2**  
**CLOSING**  
**LAST-ACK**  
**TIME-WAIT**

Возвращается сообщение «error: connection closing» (соединение закрывается) без запроса обслуживания.

## RECEIVE

### Состояние CLOSED (TCB не существует)

Если у пользователя нет прав доступа к соединению, возвращается сообщение «error: connection illegal for this process» (некорректное соединение для данного процесса).

В остальных случаях возвращается сообщение «error: connection does not exist» (соединения не существует).

**Состояния**    **LISTEN**  
                   **SYN-SENT**  
                   **SYN-RECEIVED**

Данные помещаются в очередь для передачи после перехода в состояние ESTABLISHED. Если в очереди нет свободного места, возвращается сообщение «error: insufficient resources».

**Состояния**    **ESTABLISHED**  
                   **FIN-WAIT-1**  
                   **FIN-WAIT-2**

Если для выполнения запроса не хватает входящих сегментов из очереди, запрашивается очередь. Если в очереди нет свободного места для запоминания RECEIVE, выдаётся сообщение «error: insufficient resources».

Осуществляется сборка входящих сегментов из очереди в приёмный буфер и передача буфера пользователю. При наличии флага выталкивания функция возвращает пользователю флаг PUSH.

Если передаваемым пользователю данным предшествует RCV.UP, пользователь получает уведомление о наличии срочных данных.

Когда TCP принимает на себя ответственность за доставку данных пользователю, это фактически означает необходимость передачи отправителю подтверждений. Формирование подтверждений рассматривается ниже при обсуждении обработки входящих сегментов.

### Состояние CLOSE-WAIT

Поскольку удалённая сторона уже передала FIN, вызов RECEIVE должен ограничиться данными, которые уже приняты, но не доставлены пользователю. Если ожидающих доставки данных уже нет, функция RECEIVE будет возвращать сообщение «error: connection closing» (соединение закрывается). В остальных случаях RECEIVE будет использовать все оставшиеся данные.

**Состояния**    **CLOSING**  
                   **LAST-ACK**  
                   **TIME-WAIT**

Возвращается сообщение "error: connection closing".

## CLOSE

### Состояние CLOSED (TCB не существует)

Если у пользователя нет прав доступа к соединению, возвращается сообщение «error: connection illegal for this process». В остальных случаях возвращается сообщение «error: connection does not exist».

### Состояние LISTEN

Все незавершённые вызовы RECEIVE завершаются с сообщением «error: closing» (соединение закрывается). Удаляется TCB и соединение переводится в состояние CLOSED.

### Состояние SYN-SENT

Удаляется TCB и возвращаются сообщения «error: closing» для всех поставленных в очередь вызовов SEND и RECEIVE.

### Состояние SYN-RECEIVED

Если нет очереди SEND и данных, ожидающих передачи, формируется сегмент FIN и соединение переходит в состояние FIN-WAIT-1. В противном случае вызов помещается в очередь для обработки после завершения состояния ESTABLISHED.

### Состояние ESTABLISHED

Запрос сохраняется в очереди, пока не будут сегментированы все предыдущие вызовы SEND. После этого формируется сегмент FIN и соединение переходит в состояние FIN-WAIT-1.

**Состояния**    **FIN-WAIT-1**  
                   **FIN-WAIT-2**

Строго говоря, такой вызов является для данных состояний ошибкой и должен возвращать сообщение «error: connection closing» (закрытие соединения). Однако отклик «ok» (нет ошибок) также допускается, пока не передан повторно сегмент FIN (хотя первый сегмент FIN может быть передан повторно).



**Состояние CLOSE-WAIT**

Запрос сохраняется в очереди, пока не будут сегментированы все предыдущие вызовы SEND. После этого формируется сегмент FIN и соединение переходит в состояние LAST-ACK<sup>1</sup>.

**Состояния CLOSING  
LAST-ACK  
TIME-WAIT**

Возвращается сообщение «error: connection closing».

**ABORT****Состояние CLOSED (TCB не существует)**

Если у пользователя нет прав доступа к соединению, возвращается сообщение «error: connection illegal for this process». В остальных случаях возвращается сообщение "error: connection does not exist".

**Состояние LISTEN**

Все остающиеся вызовы RECEIVE должны быть завершены с возвратом сообщения «error: connection reset» (соединение разорвано). Удаляется TCB, соединение переходит в состояние CLOSED.

**Состояние SYN-SENT**

Удаляется TCB и возвращаются сообщения «error: connection reset» для всех поставленных в очередь вызовов SEND и RECEIVE, соединение переводится в состояние CLOSED.

**Состояния SYN-RECEIVED  
ESTABLISHED  
FIN-WAIT-1  
FIN-WAIT-2  
CLOSE-WAIT**

Передаётся сегмент <SEQ=SNД.NXT><CTL=RST>.

Возвращаются сообщения «error: connection reset» для всех поставленных в очередь вызовов SEND и RECEIVE; все сегменты из очереди на передачу (за исключением сформированного ранее RST) или повторную передачу уничтожаются, удаляется TCB и соединение переводится в состояние CLOSED.

**Состояния CLOSING  
LAST-ACK  
TIME-WAIT**

Функция возвращает «ок», удаляется TCB и соединение переводится в состояние CLOSED.

**STATUS****Состояние CLOSED (TCB не существует)**

Если у пользователя нет прав доступа к соединению, возвращается сообщение «error: connection illegal for this process». В остальных случаях возвращается сообщение «error: connection does not exist».

**Состояние LISTEN**

Возвращается сообщение «state = LISTEN» и указатель на TCB.

**Состояние SYN-SENT**

Возвращается сообщение «state = SYN-SENT» и указатель на TCB.

**Состояние SYN-RECEIVED**

Возвращается сообщение «state = SYN-RECEIVED» и указатель на TCB.

**Состояние ESTABLISHED**

Возвращается сообщение «state = ESTABLISHED» и указатель на TCB.

**Состояние FIN-WAIT-1**

Возвращается сообщение «state = FIN-WAIT-1» и указатель на TCB.

**Состояние FIN-WAIT-2**

Возвращается сообщение «state = FIN-WAIT-2» и указатель на TCB.

**Состояние CLOSE-WAIT**

Возвращается сообщение «state = CLOSE-WAIT» и указатель на TCB.

**Состояние CLOSING**

Возвращается сообщение «state = CLOSING» и указатель на TCB.

**Состояние LAST-ACK**

Возвращается сообщение «state = LAST-ACK» и указатель на TCB.

**Состояние TIME-WAIT**

Возвращается сообщение «state = TIME-WAIT» и указатель на TCB.

**Доставка сегментов**

<sup>1</sup> В исходном документе ошибочно указан переход в состояние CLOSING вместо LAST-ACK (см. RFC-1122, 4.2.2.20(a)). Прим. перев.

**Состояние CLOSED (TCB не существует)**

Все данные во входящем сегменте отбрасываются, отбрасывается также входящий сегмент, содержащий RST. В ответ на входящий сегмент без RST передаётся сегмент RST. Порядковый номер и номер подтверждения выбираются таким образом, чтобы сделать порядковый номер RST приемлемым для TCP, передавшего сегмент.

Если бит ACK не установлен, используется порядковый номер 0.

<SEQ=0><ACK=SEG . SEQ+SEG . LEN><CTL=RST , ACK>

При наличии бита ACK

<SEQ=SEG . ACK><CTL=RST>

**Состояние LISTEN**

- 1) Проверка RST - все входящие RST должны игнорироваться с возвратом управления.
- 2) Проверка ACK - любые подтверждения являются некорректными, если они поступают через соединение в состоянии LISTEN. В ответ на такие подтверждения должен формироваться приемлемый сегмент сброса в формате:

<SEQ=SEG . ACK><CTL=RST>

Возврат управления.

- 3) Проверка SYN

Если бит SYN установлен, проверяется уровень безопасности. Если уровень безопасности/разделения для входящего сегмента не соответствует в точности уровню безопасности в TCB, передаётся сигнал сброса и возвращается управление.

<SEQ=0><ACK=SEG . SEQ+SEG . LEN><CTL=RST , ACK><sup>1</sup>

Если  $SEG.PRC > TCB.PRC$ , то при наличии разрешения со стороны пользователя и системы устанавливается  $TCB.PRC < -SEG.PRC$ ; в противном случае передаётся сигнал сброса и возвращается управление.

<SEQ=0><ACK=SEG . SEQ+SEG . LEN><CTL=RST , ACK><sup>1</sup>

Если  $SEG.PRC < TCB.PRC$ , обработка вызова продолжается.

Устанавливаются значения  $RCV.NXT = SEG.SEQ+1$  и  $IRS = SEG.SEQ$ , все остальные данные и поля управления помещаются в очередь для последующей обработки. Должно быть выбрано значение ISS и передан сегмент SYN в форме:

<SEQ=ISS><ACK=RCV . NXT><CTL=SYN , ACK>

Устанавливаются значения  $SND.NXT = ISS+1$  и  $SND.UNA = ISS$ . Соединение должно быть переведено в состояние SYN-RECEIVED. Отметим, что любые другие входящие данные и поля управления (включённые в SYN) будут обрабатываться в состоянии SYN-RECEIVED, но обработка SYN и ACK не должна повторяться. Если прослушивающая сторона не задана полностью (т. е., не полностью указан внешний сокет), пропущенные поля должны быть заполнены на этом этапе обработки.

- 4) Проверка других полей

Любые другие сегменты управления и данных (не содержащие SYN) должны содержать ACK и, таким образом, будут отброшены при проверке ACK. Входящий сегмент RST не может быть корректным, поскольку он не может служить откликом на какую-либо передачу в данной инкарнации соединения. Вероятность получения такого сегмента мала, но если это произойдёт, нужно просто отбросить сегмент и вернуть управление.

**Состояние SYN-SENT**

- 1) Проверка бита ACK

Если бит ACK установлен, выполняются следующие операции:

Если  $SEG.ACK = < ISS$  или  $SEG.ACK > SND.NXT$ , передаётся сигнал сброса (если бит RST уже установлен, сегмент отбрасывается с возвратом управления)

<SEQ=SEG . ACK><CTL=RST>

и сегмент отбрасывается с возвратом управления.

Если  $SND.UNA = < SEG.ACK = < SND.NXT$ , бит ACK считается допустимым и обработка продолжается.

- 2) Проверка бита RST

При наличии бита RST выполняются следующие операции:

Если бит ACK был допустимым, пользователю передаётся сообщение «error: connection reset» (сброс соединения), сегмент отбрасывается и соединение переводится в состояние CLOSED с удалением TCB и возвратом управления. В противном случае (нет ACK) сегмент отбрасывается с возвратом управления.

- 3) Проверка уровня безопасности и предпочтения

Если уровень безопасности/разделения в принятом сегменте не соответствует в точности уровню безопасности TCB, передаётся сигнал сброса:

*бит ACK установлен*

<SEQ=SEG . ACK><CTL=RST>

*бит ACK не установлен*

<SEQ=0><ACK=SEG . SEQ+SEG . LEN><CTL=RST , ACK>

<sup>1</sup>В исходном документе ошибочно указана команда сброса <SEQ=SEG.ACK><CTL=RST>. Ошибка исправлена в [RFC 1122](https://www.rfc-editor.org/rfc/rfc1122), параграф 4.2.2.20(b). Прим. перев.

Если уровни безопасности совпадают, выполняются следующие операции:

*бит ACK установлен*

Если уровень предпочтения в сегменте не соответствует уровню в TCB, передаётся сигнал сброса

<SEQ=SEG.ACK><CTL=RST>

*бит ACK не установлен*

Если уровень предпочтения в сегменте выше уровня в TCB, с позволения пользователя и системы повышается уровень предпочтения в TCB до совпадения с уровнем в сегменте; при невозможности повысить уровень предпочтения передаётся сигнал сброса

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

Если уровень предпочтения в сегменте ниже уровня в TCB, продолжается обработка вызова.

Если был передан сигнал сброса, сегмент отбрасывается с возвратом управления.

#### 4) Проверка бита SYN

Этот этап выполняется только при корректном ACK или отсутствии ACK, если сегмент не содержит RST.

Если бит SYN установлен и параметры безопасности/предпочтения приемлемы, устанавливаются значения RCV.NXT = SEG.SEQ+1 и IRS = SEG.SEQ. Значение SND.UNA должно быть увеличено до SEG.ACK (если имеется ACK) и все сегменты из очереди повторной передачи, которые таким образом будут подтверждены, должны быть удалены из очереди.

Если SND.UNA > ISS (сегмент SYN был подтверждён), состояние соединения меняется на ESTABLISHED<sup>1</sup>, формируется и передаётся сегмент ACK

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

В этот сегмент могут быть включены данные и сигналы управления из очереди на повторную передачу. Если в сегменте присутствуют другие данные или поля управления, продолжается обработка этого сегмента с проверки бита URG (см. этап (6) ниже), в противном случае обработка завершается с возвратом управления.

Если бит SYN не установлен, соединение переходит в состояние SYN-RECEIVED, формируется и передаётся сегмент SYN,ACK

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

Если в сегменте присутствуют другие данные или поля управления, они помещаются в очередь для обработки после перехода соединения в состояние ESTABLISHED и возвращается управление.

#### 5) Если не установлен ни бит SYN, ни бит RST, сегмент отбрасывается с возвратом управления.

После перечисленных для состояний CLOSED, LISTEN, SYN-SENT проверок выполняются следующие операции:

##### (1) Проверка порядкового номера.

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

Начальные проверки с целью отбрасывания старых дубликатов проводятся в порядке прибытия, но дальнейшая обработка сегментов ведётся в соответствии с порядковыми номерами в поле SEG.SEQ. Если сегмент включает старые и новые данные, должна обрабатываться только новая информация.

При начальной проверке прибывающих сегментов возможны 4 варианта:

Размер сегмента	Окно приёма	Проверка
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	недопустимо
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND или RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

Если RCV.WND = 0, сегменты не могут приниматься за исключением корректных сегментов ACK, URG и RST.

Если входящий сегмент неприемлем, в ответ на него должно быть передано уведомление (если установлен бит RST, сегмент должен быть отброшен с возвратом управления):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

После передачи подтверждения происходит отбрасывание неприемлемого сегмента и возврат управления.

В дальнейшем сегменты предполагаются идеализированными - они начинаются с RCV.NXT и не выходят за пределы окна. Реальные сегменты могут быть подогнаны под такую идеализацию путём отрезания любой части, выходящей за пределы окна (включая SYN и FIN), и дальнейшей обработки только для сегментов, начинающихся с RCV.NXT. Сегменты с более высокими порядковыми номерами могут сохраняться для последующей обработки.

<sup>1</sup>В соответствии с RFC 1122 при переходе в состояние ESTABLISHED должны быть установлены следующие переменные SND.WND<-SEG.WND, SND.WL1<-SEG.SEQ, SND.WL2<-SEG.ACK. Прим. перев.

**(2) Проверка бита RST****Состояние SYN-RECEIVED**

Если установлен бит RST

Если соединение было открыто пассивным вызовом OPEN (т. е., перешло из состояния LISTEN), оно будет возвращено в состояние LISTEN с возвратом управления. Пользователю не будет передаваться никакого уведомления об этом. Если соединение организовано активным вызовом OPEN (т. е., перешло из состояния SYN-SENT), оно будет отвергнуто (refused) и пользователь получит сообщение "connection refused". В обоих случаях все сегменты из очереди на передачу должны быть удалены. В случае активного вызова OPEN соединение переводится в состояние CLOSED с удалением TCB и возвратом управления.

**Состояния ESTABLISHED****FIN-WAIT-1****FIN-WAIT-2****CLOSE-WAIT**

Если установлен бит RST, все остающиеся вызовы RECEIVE и SEND должны получить команду сброса. Все сегменты из очередей должны быть удалены. Пользователь должен получить сигнал «connection reset» (сброс соединения). Соединение переходит в состояние CLOSED с удалением TCB и возвратом управления.

**Состояния CLOSING****LAST-ACK****TIME-WAIT**

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

**(3) Проверка предпочтений и безопасности****Состояние SYN-RECEIVED**

Если уровни безопасности/разделения и предпочтения в сегменте не соответствуют в точности таким же уровням в TCB, передаётся сигнал сброса с возвратом управления.

**Состояния ESTABLISHED<sup>1</sup>****FIN-WAIT-1****FIN-WAIT-2****CLOSE-WAIT****CLOSING****LAST-ACK****TIME-WAIT**

Если уровни безопасности/разделения и предпочтения в сегменте не соответствуют в точности таким же уровням в TCB, передаётся сигнал сброса (reset), все остающиеся вызовы RECEIVE и SEND должны получить сигнал reset. Все очереди сегментов удаляются. Пользователь должен получить сигнал «connection reset». Соединение переходит в состояние CLOSED с удалением TCB и возвратом управления.

Отметим, что эта проверка проводится после проверки порядкового номера для предотвращения приёма сегментов из старых соединений между теми же портами, но с другим уровнем безопасности или предпочтений, которые могут вызвать разрыв текущего соединения.

Если соединение было инициировано пассивным вызовом OPEN, оно переходит в состояние LISTEN с возвратом управления. В остальных случаях выполняется дальнейшая обработка<sup>2</sup>.

**(4) Проверка бита SYN****Состояния SYN-RECEIVED****ESTABLISHED****FIN-WAIT-1****FIN-WAIT-2****CLOSE-WAIT****CLOSING****LAST-ACK****TIME-WAIT**

Если SYN находится в окне, это говорит об ошибке. Передаётся сигнал сброса, все незавершённые вызовы RECEIVE и SEND должны получить отклики reset. Все сегменты из очередей удаляются, пользователь должен получить сигнал «connection reset». Соединение переходит в состояние CLOSED с удалением TCB и возвратом управления.

Если SYN не находится в окне, этот этап не будет выполняться, поскольку будет передано подтверждение ещё на первом этапе (проверка порядкового номера).

**(5) Проверка поля ACK**

Если бит ACK не установлен, сегмент отбрасывается с возвратом управления.

<sup>1</sup>В исходном документе указано только состояние ESTABLISHED - [RFC 1122](#) (параграф 4.2.2.20(d)) содержит исправление. *Прим. перев.*

<sup>2</sup>В исходном документе этот абзац отсутствует. См. исправления в параграфе 4.2.2.20(e) [RFC 1122](#). *Прим. перев.*



При наличии бита ACK выполняется дальнейшая обработка.

#### Состояние SYN-RECEIVED

Если  $SND.UNA \leq SEG.ACK \leq SND.NXT$ , соединение переходит в состояние ESTABLISHED<sup>1</sup> и обработка продолжается.

Если сегмент подтверждения неприемлем, формируется и передаётся сегмент сброса (reset)

$\langle SEQ=SEG.ACK \rangle \langle CTL=RST \rangle$

#### Состояние ESTABLISHED

Если  $SND.UNA < SEG.ACK \leq SND.NXT$ , устанавливается  $SND.UNA \leftarrow SEG.ACK$ . Все, так или иначе, целиком подтверждённые сегменты удаляются из очереди повторной передачи. Пользователь должен получить позитивные подтверждения для буферов, которые были переданы и полностью подтверждены (т. е., в результате вызова SEND должно быть возвращено сообщение «ок»). Если подтверждение ACK является дубликатом ( $SEG.ACK \leq SND.UNA^1$ ), его можно игнорировать. Если ACK подтверждает что-то, ещё не переданное ( $SEG.ACK > SND.NXT$ ), посылается подтверждение ACK и сегмент отбрасывается с возвратом управления.

Если  $SND.UNA \leq SEG.ACK \leq SND.NXT^2$ , окно нужно обновить. Если выполняется условие

$(SND.WL1 < SEG.SEQ) \text{ OR } ((SND.WL1 = SEG.SEQ) \text{ AND } (SND.WL2 \leq SEG.ACK))$ ,  
нужно установить  $SND.WND \leftarrow SEG.WND$ ,  $SND.WL1 \leftarrow SEG.SEQ$  и  $SND.WL2 \leftarrow SEG.ACK$ .

Отметим, что  $SND.WND$  задаёт смещение от  $SND.UNA$ , поле  $SND.WL1$  содержит порядковый номер последнего сегмента, использованного для обновления  $SND.WND$ , а  $SND.WL2$  содержит номер подтверждения последнего сегмента, использованного для обновления  $SND.WND$ . Описанная проверка позволяет предотвратить использование старых сегментов для обновления окна.

#### Состояние FIN-WAIT-1

В дополнение к операциям, выполняемым в состоянии ESTABLISHED (см. выше), если наш запрос FIN уже подтверждён, соединение переводится в состояние FIN-WAIT-2 и продолжается обработка для этого состояния.

#### Состояние FIN-WAIT-2

В дополнение к операциям, выполняемым в состоянии ESTABLISHED (см. выше), если очередь повторной передачи пуста, на пользовательский запрос CLOSE может быть передан отклик «ок», но TCB не удаляется.

#### Состояние CLOSE-WAIT

Выполняются те же операции, что и для состояния ESTABLISHED.

#### Состояние CLOSING

В дополнение к операциям, выполняемым в состоянии ESTABLISHED (см. выше), если ACK подтверждает наш запрос FIN, соединение переводится в состояние TIME-WAIT, в остальных случаях сегмент игнорируется.

#### Состояние LAST-ACK

В этом состоянии могут доставляться только подтверждения для запроса FIN. Если запрос FIN уже подтверждён, удаляется TCB и соединение переводится в состояние CLOSED с возвратом управления.

#### Состояние TIME-WAIT

В этом состоянии могут доставляться только повторы сегмента FIN с удалённой стороны. Подтверждается доставка и запускается заново отсчёт тайм-аута 2 MSL.

### (6) Проверка бита URG

Состояния     **ESTABLISHED**  
                  **FIN-WAIT-1**  
                  **FIN-WAIT-2**

При установленном бите URG, задаётся значение  $Rcv.UP \leftarrow \max(Rcv.UP, SEG.UP)$  и пользователю передаётся сигнал о том, что удалённая сторона имеет срочные данные, если указатель срочности ( $Rcv.UP$ ) превышает порядковые номера воспринятых данных. Если пользователю уже был передан такой сигнал (или пользователь находится в режиме «urgent») для продолжающейся последовательности срочных данных, повторять такой сигнал не нужно.

Состояния     **CLOSE-WAIT**  
                  **CLOSING**  
                  **LAST-ACK**  
                  **TIME-WAIT**

В этих состояниях флаг срочности не должен появляться, поскольку удалённая сторона уже передала запрос FIN. Флаг URG игнорируется.

### (7) Обработка содержимого сегмента

Состояния     **ESTABLISHED**  
                  **FIN-WAIT-1**  
                  **FIN-WAIT-2**

<sup>1</sup>В исходном документе ошибочно написано  $SEG.ACK < SND.UNA$ . Поправка внесена в параграфе 4.2.2.20 (g) [RFC 1122](#). Прим. перев.

<sup>2</sup>В исходном документе ошибочно написано  $SND.UNA < SEG.ACK \leq SND.NXT$ . Поправка внесена в параграфе 4.2.2.20 (g) [RFC 1122](#). Прим. перев.

В состоянии ESTABLISHED возможна доставка содержимого сегмента в пользовательские буферы RECEIVE. Содержимое сегмента может переноситься в буфер до его заполнения или до завершения сегмента. Если закончившийся сегмент содержит флаг PUSH, пользователю передаётся уведомление путём установки флага PUSH при возврате управления.

Когда TCP отвечает за доставку данных пользователю, требуется также подтверждать получение данных.

Когда TCP несёт ответственность за доставку данных, значение RCV.NXT увеличивается на размер принятых данных и уточняется значение RCV.WND с учётом текущего размера буфера. Сумма RCV.NXT и RCV.WND при этом не должна уменьшаться.

Пожалуйста, не забывайте о предложениях по управлению окном, рассмотренных в параграфе 3.7.

Передаётся подтверждение в форме:

`<SEQ=SNД . NXT><ACK=RCV . NXT><CTL=ACK>`

Это подтверждение должно включаться в сегмент, который будет передаваться с минимальной возможной задержкой.

**Состояния**      **CLOSE-WAIT**  
**CLOSING**  
**LAST-ACK**  
**TIME-WAIT**

В этих состояниях сегменты данных не должны приходить, поскольку с удалённой стороны уже получен сигнал FIN. Игнорируйте такие сегменты.

### (8) Проверка бита FIN

Флаг FIN для состояний CLOSED, LISTEN или SYN-SENT не обрабатывается, поскольку значение SEG.SEQ не может быть проверено; сегмент отбрасывается с возвратом управления.

Если бит FIN установлен, пользователю передаётся сигнал «connection closing» (соединение закрывается) и все ожидающие вызовы RECEIVE должны вернуть такое же сообщение, увеличив RCV.NXT на размер FIN и передав подтверждение для FIN. Отметим, что FIN вызывает PUSH для всех сегментов, которые ещё не доставлены пользователю.

**Состояния**      **SYN-RECEIVED**  
**ESTABLISHED**

Переход в состояние CLOSE-WAIT.

#### Состояние FIN-WAIT-1

Если наш запрос FIN уже подтверждён (возможно в данном сегменте), соединение переводится в состояние TIME-WAIT, начинается новый отсчёт для времени ожидания и выключаются другие таймеры. В противном случае соединение переводится в состояние CLOSING.

#### Состояние FIN-WAIT-2

Переход в состояние TIME-WAIT, начинается новый отсчёт для времени ожидания и выключаются другие таймеры.

**Состояния**      **CLOSE-WAIT**  
**CLOSING**  
**LAST-ACK**  
**TIME-WAIT**

Состояние сохраняется неизменным. В состоянии TIME-WAIT заново запускается отсчёт тайм-аута 2 MSL.

Завершение обработки и возврат управления.

### Тайм-ауты

#### Пользовательский тайм-аут

Если при любом состоянии соединения истекает время пользовательского тайм-аута, удаляются все буферы, пользователю передаётся сообщение «error: connection aborted due to user timeout» для всех остающихся вызовов, удаляется TCB и соединение переходит в состояние CLOSED с возвратом управления.

#### Повторная передача

Если при любом состоянии соединения истекает время тайм-аута повторной передачи, первый сегмент из очереди повторной передачи посылается заново, таймер повторной передачи сбрасывается и управление возвращается.

#### Время ожидания

При завершении времени ожидания для соединения удаляется TCB и соединение переходит в состояние CLOSED с возвратом управления.

## Глоссарий

### 1822

BBN Report 1822, «The Specification of the Interconnection of a Host and an IMP» - спецификация интерфейса между хостом и ARPANET.

#### ACK (флаг подтверждения)

Бит управления (подтверждение), не занимающий места в пространстве порядковых номеров и показывающий, что поле подтверждения в данном сегменте содержит следующий порядковый номер, который отправитель данного сегмента ожидает получить в подтверждение доставки всех предыдущих порядковых номеров.

**ARPANET message (сообщение ARPANET)**

Единица передачи информации между хостом и IMP в ARPANET<sup>1</sup>. Максимальный размер сообщения составляет 1012 октетов (8096 битов).

**ARPANET packet (пакет ARPANET)**

Единица передачи информации между IMP в сети ARPANET. Максимальный размер пакета составляет 126 октетов (1008 битов).

**Connection (соединение)**

Логический путь передачи данных, задаваемый парой сокетов.

**Datagram (дейтаграмма)**

Сообщение, передаваемое через компьютерную сеть с коммутацией пакетов.

**Destination Address (адрес получателя)**

Адрес получателя, обычно указывающий хост или сеть.

**FIN (флаг завершения)**

Бит управления, занимающий единицу пространства порядковых номеров, который показывает, что отправитель больше не будет передавать данные или сигналы управления, занимающие порядковые номера.

**Fragment (фрагмент)**

Часть логической единицы данных (в частности, фрагментом называют часть дейтаграммы IP).

**FTP**

Протокол передачи файлов.

**Header (заголовок)**

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

**Host (хост)**

Компьютер. В частности, отправитель и получатель сообщений с точки зрения коммуникационной сети.

**Identification (идентификация)**

Поле протокола IP, содержащее идентификатор, заданный отправителем для обеспечения сборки дейтаграммы из фрагментов.

**IMP**

Interface Message Processor - пакетный коммутатор ARPANET.

**internet address**

Адрес отправителя или получателя, используемый на уровне хостов.

**internet datagram**

Единица обмена данными между модулем IP и протоколом вышележащего уровня, содержащая заголовок IP и данные.

**internet fragment**

Часть данных из дейтаграммы IP с заголовком IP.

**IP**

Internet Protocol.

**IRS**

Initial Receive Sequence - начальный порядковый номер для приёма. Первый порядковый номер, используемый отправителем в данном соединении.

**ISN**

Initial Sequence Number - начальный порядковый номер. Первый порядковый номер (ISS или IRS), используемый для соединения. Выбирается на основе текущего времени.

**ISS**

Initial Send Sequence - начальный порядковый номер для передачи. Первый порядковый номер, используемый отправителем в данном соединении.

**left sequence**

Следующий порядковый номер, который будет подтверждён данными, принимаемыми TCP (или наименьший из ещё неподтвержденных порядковых номеров); в некоторых случаях трактуется как левый край окна передачи.

**local packet (локальный пакет)**

Единица передачи данных в локальной сети.

**Module (модуль)**

Реализация (обычно программная) протокола или другой процедуры.

**MSL**

Maximum Segment Lifetime - максимальное время жизни сегмента TCP в сети. Произвольно выбрано значение 2 минуты.

**Octet (октет)**

Восьмибитовый блок данных - байт.

**Options (опции)**

Поле Option может содержать несколько опций, каждая из которых может занимать несколько октетов. Опции используются в основном для проверки или задания тех или иных условий (например, содержат временные метки). Поля опций поддерживаются как IP, так и TCP.

**Packet (пакет)**

Единица данных с заголовком (может быть логически полным или неполным). Более часто термин пакет относят к физическим, а не логическим единицам данных.

**Port (порт)**

Часть сокета, задающая логический канал ввода или вывода данных для процесса.

**Process (процесс)**

Выполняемая программа. Источник или потребитель данных с точки зрения TCP или других протоколов обмена между хостами.

**PUSH**

Управляющий бит, не занимающий пространства порядковых номеров, который указывает, что сегмент содержит данные, требующие незамедлительной отправки получателю.

**RCV.NXT**

receive next sequence number - следующий порядковый номер для приёма.

<sup>1</sup>Прообраз сегодняшней сети Internet. Прим. перев.

**RCV.UP**

receive urgent pointer - указатель срочности для приёма.

**RCV.WND**

receive window - окно приёма.

**receive next sequence number (следующий порядковый номер для приёма)**

Следующий порядковый номер, который ожидает получить локальный модуль TCP.

**receive window (окно приёма)**

Представляет порядковый номер, который локальный (принимающий) модуль TCP ожидает получить. Таким образом, локальный модуль TCP предполагает, что сегменты, перекрывающие диапазон от RCV.NXT до RCV.NXT+RCV.WND-1, содержат приемлемые данные и поля управления. Сегмент, содержащий порядковые номера, целиком выходящие за эти пределы, рассматривается как дубликат и отбрасывается.

**RST**

Управляющий бит (reset - сброс), не занимающий пространства порядковых номеров, который показывает, что приёмная сторона должна удалить соединение без дальнейших вопросов. Приёмная сторона может определить, основываясь на порядковом номере и номере подтверждения во входящем сегменте, следует воспринять команду сброса или игнорировать её. Ни в коем случае в ответ на сегмент, содержащий RST, не следует передавать отклик с битом RST.

**RTP**

Real Time Protocol - протокол для передачи между хостами критичной к задержкам информации.

**SEG.ACK**

segment acknowledgment - подтверждение сегмента.

**SEG.LEN**

segment length - размер сегмента.

**SEG.PRC**

segment precedence value - уровень предпочтения для сегмента.

**SEG.SEQ**

segment sequence - порядковый номер сегмента.

**SEG.UP**

segment urgent pointer field - поле указателя срочности в сегменте.

**SEG.WND**

segment window field - поле окна в сегменте.

**segment (сегмент)**

Логическая единица данных. В частности, сегмент TCP представляет собой единицу данных, передаваемую между парой модулей TCP.

**segment acknowledgment (подтверждение сегмента)**

Порядковый номер в поле подтверждения прибывающего сегмента.

**segment length (размер сегмента)**

Число порядковых номеров, занимаемое сегментом, с учётом полей управления, занимающих пространство порядковых номеров.

**segment sequence**

Номер в поле sequence прибывающего сегмента.

**send sequence**

Следующий порядковый номер, который будет использован локальным (передающим) TCP для данного соединения. Нумерация начинается с начального порядкового номера ISN и далее увеличивается с каждым переданным октетом данных или учитываемым октетом управляющей информации.

**send window (окно передачи)**

Представляет порядковые номера, которые удалённый (принимающий) TCP готов получить. Это значение поля window, указанное в сегментах от удалённого (принимающего данные) TCP. Диапазон новых порядковых номеров, которые может передать TCP, лежит между SND.NXT и SND.UNA+SND.WND-1 (порядковые номера для повторной передачи лежат между SND.UNA и SND.NXT).

**SND.NXT**

Порядковый номер для передачи.

**SND.UNA**

Порядковый номер оставшегося.

**SND.UP**

Указатель срочности для передачи.

**SND.WL1**

Номер сегмента при последнем обновлении окна.

**SND.WL2**

Номер сегмента подтверждения при последнем обновлении окна.

**SND.WND**

Окно передачи.

**socket (сокет)**

Идентификатор, включающий номер порта TCP и адрес IP.

**Source Address**

Адрес отправителя, обычно идентифицирующий хост или сеть.

**SYN**

Бит управления во входящем сегменте (занимающий один порядковый номер), который используется для индикации начала отсчёта порядковых номеров.

**TCB**

Transmission control block - блок управления передачей, содержащий информацию о состоянии соединения.

**TCB.PRC**

Предпочтения для данного соединения.

**TCP**

Transmission Control Protocol - протокол управления передачей. Протокол обмена информацией между хостами, обеспечивающий гарантированную доставку в межсетевых средах.



**TOS**

Type of Service - тип обслуживания. Поле протокола IP.

**Type of Service**

Поле протокола IP, задающее тип обслуживания для данного фрагмента<sup>1</sup> IP.

**URG (флаг срочности)**

Бит управления (urgent - срочность), не включаемый в пространство порядковых номеров и служаций для индикации того, что получатель должен быть уведомлен о срочности обработки данных (эти данные должны быть восприняты сразу же после завершения работы с данными, порядковый номер которых меньше, нежели задано указателем срочности).

**urgent pointer (указатель срочности)**

Поле управления, имеющее смысл только при установке бита URG и содержащее значение указателя срочности, которое говорит о заданном пользователем уровне срочности данных.

**Литература**

- [1] Cerf, V., and R. Kahn, "A Protocol for Packet Network Intercommunication", IEEE Transactions on Communications, Vol. COM-22, No. 5, pp 637-648, May 1974.
- [2] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification", [RFC 791](#), USC/Information Sciences Institute, September 1981.
- [3] Dalal, Y. and C. Sunshine, "Connection Management in Transport Protocols", Computer Networks, Vol. 2, No. 6, pp. 454-473, December 1978.
- [4] Postel, J., "Assigned Numbers", RFC 790<sup>2</sup>, USC/Information Sciences Institute, September 1981.

**Перевод на русский язык**

Николай Малых

[nmalykh@protokols.ru](mailto:nmalykh@protokols.ru)

<sup>1</sup>Тип обслуживания может задаваться не только для фрагментов, но и для любых дейтаграмм IP. *Прим. перев.*

<sup>2</sup>В соответствии с [RFC 3232](#) документ STD 2 утратил силу. Значения Assigned Numbers следует искать в базе данных, доступной на сайте [www.iana.org/numbers.html](http://www.iana.org/numbers.html). *Прим. перев.*