Network Working Group
Request for Comments: 2960
Category: Standards Track

R. Stewart
Q. Xie
Motorola
K. Morneault
C. Sharp
Cisco
H. Schwarzbauer
Siemens
T. Taylor
Nortel Networks
I. Rytina
Ericsson
M. Kalla
Telcordia
L. Zhang
UCLA
V. Paxson
ACIRI

October 2000

# Протокол SCTP

Stream Control Transmission Protocol

**Статус документа**

Этот документ является спецификацией стандарта Internet, предназначенного для сообщества Internet, и служит приглашением к дискуссии в целях развития протокола. Сведения о текущем состоянии стандартизации протокола можно найти в документе "Internet Official Protocol Standards" (STD 1). Документ может распространяться без ограничений.

**Авторские права**



**Аннотация**

Данный документ описывает протокол управления потоковой передачей SCTP[1]. Протокол SCTP предназначен для передачи сигнальных сообщений PSTN[2] через сети IP, но может использоваться и для других приложений.

SCTP представляет собой протокол транспортного уровня с гарантированной доставкой, работающий в пакетных сетях без явной организации соединений таких, как IP. Протокол обеспечивает пользователям следующие типы сервиса:

- передача пользовательских данных с корректировкой ошибок, подтверждением доставки и отсутствием дубликатов;
- фрагментирование данных в соответствии с определенным для пути значением MTU[3];
- упорядоченная доставка пользовательских сообщений внутри множества потоков с возможностью управления порядком доставки отдельных пользовательских сообщений;
- возможность группировки пользовательских сообщений в один пакет SCTP;
- устойчивость к отказам на сетевом уровне за счёт поддержки многодомных хостов на обеих сторонах соединения.

Протокол SCTP включает механизмы предотвращения насыщения и устойчивости к атакам с использованием лавины пакетов (flooding) или маскированием адресов (masquerade).

## Оглавление



[1]Stream Control Transmission Protocol.
[2]Телефонные сети общего назначения. *Прим. перев.*
[3]Максимальный размер передаваемых пакетов. *Прим. перев.*

# *1. Введение*

В этом разделе рассматриваются причины, побудившие к разработке протокола SCTP, обеспечиваемые протоколом типы сервиса, а также базовые концепции, требуемые для детального описания протокола.

## 1.1 Мотивация

Протокол TCP [RFC793] обеспечивает, прежде всего, гарантированную доставку данных в сетях IP. Однако многие современные приложения сталкиваются с ограниченными возможностями TCP и вынуждены реализовать свои средства обеспечения гарантированной доставки на основе транспорта UDP [RFC768]. Ниже перечислены основные ограничения TCP.

- Протокол TCP обеспечивает гарантию доставки данных с сохранением их порядка. Некоторым приложениям требуется лишь гарантированная доставка, независимо от порядка, а другим приложениям может быть достаточно частичного сохранения порядка доставки. Оба типа приложений не устраивают дополнительные задержки TCP, возникающие при нарушении порядка доставки пакетов в сети.

- Потоковая природа протокола TCP зачастую не подходит для приложений, которым приходится вводить свои средства маркировки записей для делинеаризации передаваемых сообщений. Кроме того, приложениям приходиться явно использовать средства «выталкивания» данных (push) для того, чтобы сообщение было передано полностью за разумное время.

- Ограниченные возможности сокетов TCP усложняют для приложений задачу обеспечения высокого уровня доступности при обмене данными за счёт использования многодомных[1] хостов.

- Протокол TCP достаточно слабо защищён от атак на службы[2], в частности, от SYN-атак.

Передача сигнальных сообщений PSTN через сети IP является примером приложения, которое сталкивается со всеми ограничениями протокола TCP. Это послужило основным мотивом разработки протокола SCTP, но можно найти и другие приложения, для которых транспорт SCTP будет предпочтительным.

[1]Имеющих множество сетевых интерфейсов. *Прим. перев.*
[2]Denial of service attack.

## 1.2 Архитектура SCTP

Протокол SCTP размещается в многоуровневой модели между уровнем пользовательских приложений SCTP и сетевым сервисом без организации явных соединений (например, IP). В остальной части этого документа предполагается, что SCTP работает «поверх» протокола IP. Основным типом сервиса, обеспечиваемого протоколом SCTP, является гарантированная передача сообщений между пользователями SCTP. Этот сервис обеспечивается в контексте ассоциаций между парами конечных точек SCTP. В параграфе 10 данного документа приводится схематическое рассмотрение интерфейса API, который должен существовать на границе между протоколом SCTP и пользовательскими приложениями SCTP.

Протокол SCTP работает на основе явных соединений[1], но ассоциация SCTP представляет собой более широкое понятие, нежели соединение TCP. Протокол SCTP обеспечивает для каждой конечной точки SCTP (см. параграф 1.4) способ обеспечения другой конечной точки (в процессе создания ассоциации) списком транспортных адресов (т. е., множеством адресов IP в комбинации с номером порта SCTP), которые эта конечная точка может использовать для связи и откуда она будет получать пакеты SCTP. Ассоциация может передавать данные с использованием всех возможных пар адресов отправителя и получателя, которые могут быть созданы на основе списков адресов конечных точек.

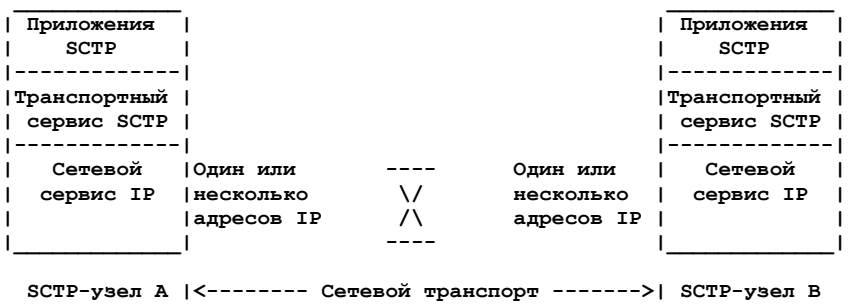



*Рисунок 1. SCTP-ассоциация.*

## 1.3 Функциональность SCTP

Транспортный сервис SCTP можно разделить на несколько функциональных групп, показанных на рисунке 2.

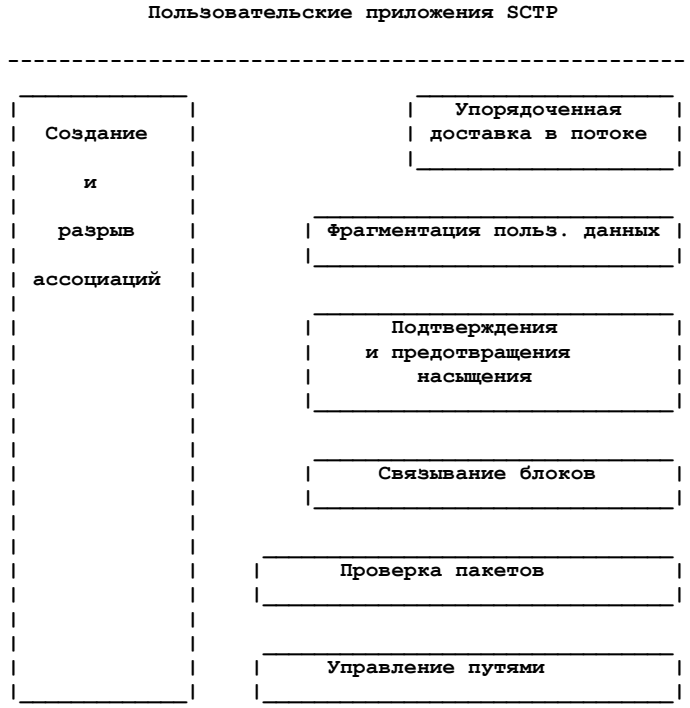



*Рисунок 2. Функциональное представление сервиса SCTP.*

### *1.3.1 Создание и разрыв ассоциаций*

Ассоциации создаются по запросам пользователей SCTP (см. описание примитива ASSOCIATE на стр. 44 или SEND на стр. 45).

В процессе создания ассоциаций используется механизм cookie, подобный предложенному Кэрном (Karn) и Симпсоном (Simpson) в [RFC2522], для обеспечения защиты от атак. Этот механизм использует четырехэтапное согласование (four-way handshake), в котором два последних этапа могут использоваться для передачи пользовательских данных в целях ускорения процедуры соединения. Стартовые последовательности описаны в главе 5 данного документа.

Протокол SCTP обеспечивает аккуратное завершение работы активных ассоциаций (shutdown) по запросу пользователя SCTP (см. описание примитива SHUTDOWN на стр. 44). Протокол SCTP позволяет также разрывать ассоциации (abort) по запросу пользователя (примитив ABORT) или в результате обнаружения ошибок на уровне SCTP. В главе 9 описаны оба варианта завершения работы ассоциаций.

Протокол SCTP не поддерживает полуоткрытых состояний (как в TCP), когда одна сторона может передавать данные после того, как другая сторона уже закрыла соединение. Когда любая из конечных точек выполняет процедуру

[1]Connection-oriented.

завершения shutdown, на обеих сторонах ассоциации прекращается приём новых данных и доставляются лишь данные из очереди (см. главу 9).

### *1.3.2 Упорядоченная доставка в потоках*
Термин поток (stream) используется в протоколе SCTP для обозначения последовательности пользовательских сообщений, которые доставляются протоколам вышележащих уровней в том же порядке, который сообщения имели в самом потоке. Это отличается от значения термина поток в контексте TCP, где потоком называют последовательности байтов (в этом документе предполагается, что размер байта составляет 8 битов).

Пользователь SCTP может во время создания ассоциации задать число потоков, поддерживаемых для этой ассоциации. Это значение согласуется с удалённой стороной (см. параграф 5.1.1). Пользовательские сообщения связываются с номерами потоков (см. примитивы SEND на стр. 45 и RECEIVE на стр. 45). Протокол SCTP присваивает порядковые номера в потоке каждому сообщению, полученному протоколом от пользователя SCTP. На приёмной стороне SCTP обеспечивает доставку сообщений пользователю с сохранением их последовательности в данном потоке. В силу того, что один поток может быть заблокирован ожиданием следующего по порядку пользовательского сообщения, в это время возможна доставка сообщений из других потоков.

Протокол SCTP обеспечивает механизм обхода упорядоченной доставки. Пользовательские сообщения, переданные с использованием такого механизма, доставляются пользователю по мере их приёма.

### *1.3.3 Фрагментация пользовательских данных*
При необходимости протокол SCTP фрагментирует пользовательские сообщения, чтобы пакеты SCTP, передаваемые нижележащему уровню, соответствовали значению MTU для пути. При получении фрагменты собираются протоколом SCTP до их доставки пользователю.

### *1.3.4 Подтверждения и предотвращение насыщения*
Протокол SCTP присваивает номер TSN каждому фрагменту и нефрагментированному пользовательскому сообщению. Нумерация TSN не зависит от порядковых номеров в потоках, присваиваемых на уровне потока. Принимающая сторона подтверждает все полученные TSN даже при обнаружении пропуска в номерах. Такой подход обеспечивает независимую функциональность для гарантии доставки и сохранения порядка сообщений.

Функция подтверждения и предотвращения насыщения отвечает за повторную передачу пакетов, когда подтверждение о доставке не приходит от получателя вовремя. Повтор передачи связан с предотвращением насыщения подобно тому, как это сделано для протокола TCP. В главах 6 и 7 приводится подробное описание протокольных процедур, связанных с выполнением этой функции.

### *1.3.5 Группировка блоков (Chunk Bundling)*
Как описано в главе 3, пакет SCTP, передаваемый на нижележащий уровень, содержит общий заголовок, за которым следует один или несколько информационных блоков (chunk). Каждый блок может содержать пользовательские данные или управляющую информацию SCTP. Пользователь SCTP может запросить группировку нескольких сообщений в один пакет SCTP. Функция группировки блоков (chunk bundling) протокола SCTP отвечает за сборку таких пакетов и их последующую разборку на приёмной стороне.

В периоды насыщения реализация SCTP может продолжать группировку сообщений даже в тех случаях, когда пользователь просил SCTP не группировать сообщения. Пользовательский запрет группировки влияет лишь на реализации SCTP, которые могут вносить незначительную задержку перед отправкой пакета в сеть. Когда пользовательский уровень запрещает группировку, этой незначительной задержки не возникает, но группировка по-прежнему будет использоваться при насыщении и повторных передачах.

### *1.3.6 Проверка пакетов*
Обязательное поле Verification Tag и 32-битовая контрольная сумма (см. Приложение B, в котором приведено описание алгоритма Adler-32), передаются в общем заголовке SCTP. Значение Verification Tag выбирается каждой стороной в процессе создания ассоциации. Пакеты, полученные без ожидаемого значения Verification Tag, отбрасываются в целях защиты от атак вслепую с маскированием адресов и избавления от старых пакетов SCTP, оставшихся от предыдущей ассоциации. Контрольная сумма Adler-32 помещается отправителем в каждый пакет SCTP для обеспечения дополнительной защиты от повреждения данных в сети. Получатель пакета SCTP с некорректной контрольной суммой Adler-32 отбрасывает такие пакеты без уведомления.

### *1.3.7 Управление путями*
Передающий пакеты SCTP пользователь может манипулировать набором транспортных адресов, используемых для указания получателя пакетов SCTP с помощью примитивов, описанных в главе 10. Функция управления путями (SCTP path management) выбирает транспортный адрес получателя для каждого исходящего пакета SCTP на основе пользовательских инструкций и доступной информации о достижимости желанного для пользователя адреса. Функция управления путями ведёт мониторинг доступности с помощью блоков heartbeat, когда другой трафик не позволяет получить достоверных данных о доступности адресов, и сообщает пользователю об изменениях состояния доступности любых адресов транспортного уровня на удалённой стороне. Функция управления путями отвечает также за передачу информации о доступных локальных адресах транспортного уровня на удалённую сторону в процессе создания ассоциации, а также за передачу пользователю таких сведений, полученных от удалённой стороны.

При создании ассоциации определяется основной путь (primary path) для каждой конечной точки SCTP. Этот путь в дальнейшем используется при нормальных условиях работы.

На приёмной стороне функция управления путями отвечает за проверку существования корректной ассоциации SCTP, к которой относятся входящие пакеты SCTP, до передачи таких пакетов на дальнейшую обработку.

Отметим, что Path Management и Packet Validation выполняются одновременно, хотя и описаны по отдельности (в реальности эти не могут выполняться независимо одна от другой).

## 1.4 Терминология

Некоторые термины, используемые в контексте SCTP, уже были упомянуты в предыдущих параграфах. Здесь даются определения основных терминов, связанных с протоколом.

- **Active destination transport address - активный транспортный адрес получателя**: адрес транспортного уровня конечной точки-партнера, который отправитель рассматривает как доступный для приёма. пользовательских сообщений.
- **Bundling - группировка**: дополнительная операция мультиплексирования, позволяющая передать в одном пакете несколько сообщений SCTP. Каждое пользовательское сообщение помещается в отдельный блок DATA.
- **Chunk - блок**: единица информации в пакете SCTP, содержащая заголовок блока (chunk header) и данные.
- **Congestion Window (cwnd) - окно насыщения**: переменная SCTP, ограничивающая объем данных (число байтов), которые отправитель может передать в один активный адрес получателя до получения от последнего подтверждения.
- **Cumulative TSN Ack Point - указатель на кумулятивный номер Ack**: значение TSN для последнего блока DATA, подтверждённого с использованием поля Cumulative TSN Ack в SACK.
- **Idle destination address - неиспользуемый адрес получателя**: адрес, который не используется для передачи пользовательских сообщений в течение некоторого периода (обычно ≥ HEARTBEAT).
- **Inactive destination transport address - неактивный транспортный адрес получателя**: адрес, который считается неактивным в результате обнаружения ошибки, и не используется для доставки пользовательских сообщений.
- **Message = user message - пользовательское сообщение**: данные, полученные протоколом SCTP от протокола вышележащего уровня (Upper Layer Protocol или ULP).
- **Message Authentication Code (MAC) - код аутентификации сообщения**: механизм проверки целостности, основанный на криптографической хэш-функции с секретным ключом. Обычно коды аутентификации сообщений используются между партнёрами, использующими общий секретный ключ для проверки целостности информации, передаваемой между системами. В SCTP этот код применяется конечными точками для проверки информации State Cookie, полученной от удалённой стороны в блоке COOKIE ECHO. Термин MAC может иметь различные трактовки в разном контексте. В SCTP этот термин используется в том же значении, которое принято в [RFC2104].
- **Network Byte Order - сетевой порядок байтов**: порядок передачи байтов, при котором старший байт следует первым. Синоним Big Endian.
- **Ordered Message - упорядоченные сообщения**: пользовательские сообщения, доставленные в том же порядке, в котором они были помещены в поток.
- **Outstanding TSN (at an SCTP endpoint) - остающийся в сети TSN (для конечной точки SCTP)**: номер TSN (и связанный с ним блок DATA), который был передан конечной точкой, но его получение ещё не подтверждено.
- **Path - путь**: маршрут, используемый пакетами SCTP, переданными одной конечной точкой SCTP по определённому транспортному адресу другой конечной точки SCTP. Передача пакетов по различным транспортным адресам удалённой точки не обязательно ведёт к изменению пути.
- **Primary Path - основной путь**: комбинация адресов отправителя и получателя которая будет помещаться в исходящие пакеты SCTP по умолчанию. Адрес отправителя включён в определение потому, что реализации протокола **могут** указывать оба адреса (получателя и отправителя) для обеспечения контроля пути возврата блоков с откликами и выбора интерфейса для передачи пакетов многодомными хостами.
- **Receiver Window (rwnd) - приёмное окно**: переменная SCTP, которую отправитель использует для хранения последнего рассчитанного значения размера окна приёма. своего партнёра по ассоциации. Размер окна приёма. задаётся количеством байтов. Значение размера позволяет отправителю определить величину свободного пространства в приёмном буфере получателя.
- **SCTP association - ассоциация SCTP**: протокольные отношения между конечными точками SCTP, включающие пару узлов SCTP и информацию о состоянии протокола (теги Verification, активные значения TSN и т. п.). Для уникальной идентификации ассоциаций SCTP могут использоваться пары транспортных адресов конечных точек ассоциации. Между любой парой конечных точек SCTP **недопустимо** одновременное существование нескольких ассоциаций.
- **SCTP endpoint - конечная точка SCTP**: логический приёмник/передатчик пакетов SCTP. Для многодомных хостов конечная точка SCTP представляется своему партнёру как комбинация набора допустимых транспортных адресов, по которым можно передавать пакеты SCTP и набора допустимых адресов транспортного уровня, с которых могут приниматься пакеты SCTP. Все транспортные адреса, используемые конечной точкой SCTP, должны быть связаны с одним номером порта, но могут иметь различные адреса IP. Транспортные адреса, используемые конечной точкой SCTP, недопустимо устанавливать для другой конечной точки SCTP (транспортный адрес каждой конечной точки SCTP должен быть уникальным).
- **SCTP packet (packet) - пакет SCTP**: элемент данных, передаваемый через интерфейс между SCTP и пакетной сетью без организации соединений (например, IP). Пакет SCTP включает общий заголовок SCTP, а также блок пользовательских данных (DATA chunk) и может включать блок управления SCTP.
- **SCTP user application (SCTP user) - пользовательское приложение SCTP (пользователь SCTP)**: логический объект вышележащего уровня, который использует сервис SCTP. Используется также термин Upper-layer Protocol (ULP) - протокол вышележащего уровня.

- **Slow Start Threshold (ssthresh) - порог замедленного старта**: переменная SCTP, определяющая пороговое значение, по которому конечная точка будет определять необходимость использования для конкретного транспортного адреса процедуры slow start или congestion avoidance. Значение ssthresh указывается в байтах.

- **Stream - поток**: однонаправленный логический канал между двумя участниками ассоциации SCTP, через который передаются все пользовательские сообщения. При доставке сообщений их порядок сохраняется, если не была задана неупорядоченная доставка.
  Отметим, что соотношения между номерами потоков в противоположных направления сильно зависят от того, как приложения используют эти потоки. Ответственность за поддержку корреляции между порядковыми номерами (если она нужна) ложится на пользовательское приложение SCTP.

- **Stream Sequence Number - порядковый номер в потоке**: 16-битовый порядковый номер, используемый SCTP для обеспечения упорядоченной доставки пользовательских сообщений в данном потоке. Каждому пользовательскому сообщению в потоке присваивается один порядковый номер.

- **Tie-Tags**: теги Verification из предыдущей ассоциации. Эти теги используются в State Cookie для того, чтобы создаваемую ассоциацию можно было связать с предыдущей ассоциацией в той конечной точке где ассоциация не создаётся заново.

- **Transmission Control Block (TCB) - блок управления передачей**: внутренняя структура данных, создаваемая конечной точкой SCTP для каждой из существующих ассоциаций SCTP с другими конечными точками SCTP. TCB содержит всю информацию о состоянии и режиме работы для конечной точки, чтобы поддерживать соответствующую ассоциацию и управлять ею.

- **Transmission Sequence Number (TSN) - порядковый номер при передаче**: 32-битовый порядковый номер, используемый протоколом SCTP для нумерации передаваемых блоков. Значение TSN связывается с каждым блоком, который содержит пользовательские данные, чтобы дать возможность конечной точке SCTP на приёмной стороне подтвердить приём блока и обнаружить дубликаты.

- **Transport address - транспортный адрес**: адрес транспортного уровня обычно определяется комбинацией адреса сетевого уровня, транспортным протоколом и номером порта на транспортном уровне. При использовании SCTP в сетях IP транспортный адрес представляет собой комбинацию адреса IP и номера порта SCTP (транспортным протоколом является SCTP).

- **Unacknowledged TSN (at an SCTP endpoint) - неподтвержденный TSN (в конечной точке SCTP)**: номер TSN (и связанный с ним блок DATA), который был получен конечной точкой, но приём этого блока ещё не был подтверждён удалённой стороне. С точки зрения передающей стороны это случай, когда пакет был передан, но подтверждение ещё не получено.

- **Unordered Message - неупорядоченные сообщения**: неупорядоченные сообщения могут передаваться с изменением их местоположения в потоке относительно других сообщений. Неупорядоченные сообщения могут размещаться в потоке перед упорядоченными или после них.

- **User message - пользовательское сообщение**: элемент данных, передаваемый через интерфейс между пользовательским приложением и уровнем SCTP.

- **Verification Tag - тег верификации**: 32-битовое беззнаковое целое значение, которое обычно выбирается с использованием генератора случайных чисел. Verification Tag позволяет получателю проверить принадлежность полученного пакета SCTP к ассоциации и избавиться от старых пакетов из прежних ассоциаций.

## 1.5. Используемые сокращения

MAC - Message Authentication Code [RFC2104] (код аутентификации сообщения).

RTO - Retransmission Time-out (тайм-аут повтора передачи).

RTT - Round-trip Time (время кругового обхода).

RTTVAR - Round-trip Time Variation (вариации времени кругового обхода).

SCTP - Stream Control Transmission Protocol (протокол управления потоковой передачей).

SRTT - Smoothed RTT (сглаженное значение RTT).

TCB - Transmission Control Block (блок управления передачей).

TLV - Type-Length-Value Coding Format (формат представления тип-размер-значение).

TSN - Transmission Sequence Number (порядковый номер при передаче).

ULP - Upper-layer Protocol (протокол вышележащего уровня).

## 1.6 Порядковые номера

Важно помнить, что пространство пространство порядковых номеров TSN ограничено, хотя и достаточно велико. Значения порядковых номеров могут находиться в диапазоне от 0 до $2^{32}$ - 1. Ввиду конечных размеров пространства номеров TSN все арифметические операции с такими номерами должны осуществляться по модулю $2^{32}$ - это обеспечит возможность после достижения максимального значения TSN снова перейти к номеру 0. При сравнении значений порядковых номеров следует принимать во внимание цикличность их изменения. Символ =< применительно к TSN означает «меньше или равно» (модуль $2^{32}$).

При арифметических операциях и сравнении номеров TSN для протокола SCTP **следует** пользоваться арифметикой порядковых номеров, определённой в [RFC1982] для случая SERIAL_BITS = 32.

Конечным точкам **не следует** передавать блоки DATA со значением TSN, которое более чем на $2^{31}$ - 1 превышает начальное значение TSN для текущего окна передачи. Использование таких значений может привести к возникновению проблем при сравнении TSN.

Порядковые номера TSN сбрасываются в 0 при достижении границы $2^{32}$ - 1. Т. е., следующий блок DATA после блока с TSN = $2^{32}$ - 1 **должен** иметь TSN = 0.

Во всех арифметических операциях с порядковыми номерами в потоках (Stream Sequence Number) следует использовать арифметику порядковых номеров, определённую в [RFC1982] для случая SERIAL_BITS = 16. Все прочие операции с порядковыми номерами в данном документе используют обычную арифметику.

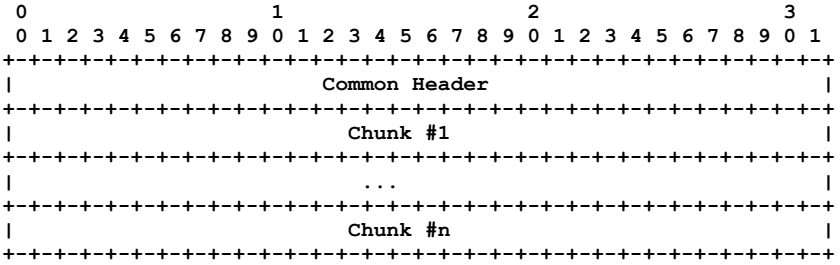
# *2. Соглашения о терминах*

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не нужно** (SHALL NOT), **следует** (SHOULD), **не следует** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе должны интерпретироваться в соответствии с [RFC 2119].

# *3. Формат пакетов SCTP*

Пакет SCTP состоит из общего заголовка (common header) и блоков (chunk), содержащих пользовательские сообщения или управляющую информацию.

Формат пакетов SCTP показан на рисунке.



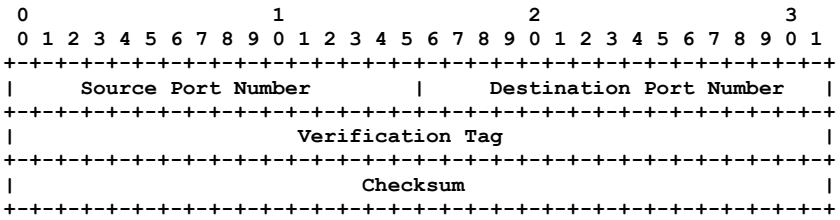

В одном пакете SCTP может содержаться множество блоков, пока размер пакета не превышает значение MTU. Исключение составляют блоки INIT, INIT ACK, и SHUTDOWN COMPLETE, которые **недопустимо** группировать с другими блоками в один пакет. Более подробное описание группировки блоков приводится в параграфе 6.10.

Если пользовательское сообщение не помещается в пакет SCTP, оно может быть разделено на фрагменты с использованием процедуры, описанной в параграфе 6.9.

Все целочисленные поля пакетов SCTP **должны** передаваться с использованием сетевого порядка байтов, если явно не указан другой порядок.

## 3.1 Описание полей общего заголовка SCTP

Формат SCTP Common Header




**Source Port Number**: 16 битов (целое без знака)

Номер порта SCTP, используемого отправителем. Это значение, вместе с IP-адресом отправителя, номером порта получателя и (возможно) IP-адресом получателя, может использоваться на приёмной стороне для идентификации ассоциации, к которой относится пакет.

**Destination Port Number**: 16 битов (целое без знака)

Номер порта SCTP, в который данный пакет адресован. На приёмной стороне это значение будет использоваться для демультиплексирования пакета SCTP соответствующей конечной точке или приложению.

**Verification Tag**: 32 бита (целое без знака)

Принимающая сторона использует Verification Tag для проверки отправителя пакета SCTP. На передающей стороне значение поля Verification Tag **должно** устанавливаться в соответствии со значением Initiate Tag, полученным от партнёра при создании ассоциации, за исключением перечисленных ниже случаев.

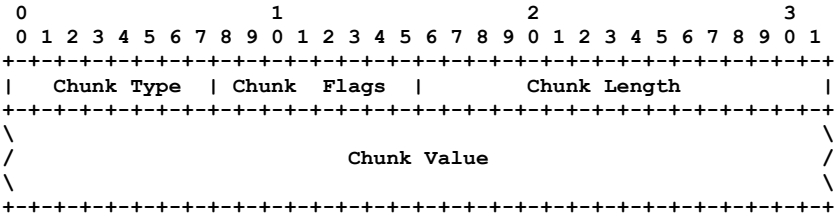
- В пакетах, содержащих блок INIT, **должно** быть Verification Tag = 0.
- В пакетах, содержащих блок SHUTDOWN-COMPLETE с установленным флагом T, значение тега Verification **должно** копироваться из пакета с блоком SHUTDOWN-ACK.
- В пакетах, содержащих блок ABORT, тег верификации может быть копией значения тега из пакета, вызвавшего передачу блока ABORT. Более подробное описание приведено в параграфах 8.4 и 8.5.

Блок INIT **должен** быть единственным блоком в пакете SCTP.

**Checksum**: 32 бита (целое без знака)

Это поле содержит контрольную сумму для данного пакета SCTP. Расчёт контрольных сумм описан в параграфе 6.8. Протокол SCTP использует алгоритм Adler-32, описанный в Приложении B.

## 3.2 Описание поля Chunk




На рисунке справа показан формат блоков (chunk), передаваемых в пакетах SCTP. Каждый блок содержит поле Chunk Type, зависящие от типа флаги (Flags), поле размера Chunk Length и поле данных Value.

**Chunk Type**: 8 битов (целое без знака)

Это поле указывает тип блока, содержащегося в поле Chunk Value. Поле типа может содержать значения от 0 до 254. Значение 255 зарезервировано для использования в качестве расширения.

Значения идентификаторов типа приведены в таблице.

| *ID* | *Тип блока* | *ID* | *Тип блока* |
|---|---|---|---|
| 0 | Пользовательские данные (DATA) | 12 | Зарезервировано для Explicit Congestion Notification Echo (ECNE) |
| 1 | Инициализация (INIT) | 13 | Зарезервировано для Congestion Window Reduced (CWR) |
| 2 | Подтверждение инициирования (INIT ACK) | 14 | Процедура окончания работы завершена (SHUTDOWN COMPLETE) |
| 3 | Выборочное подтверждение (SACK) | 15 - 62 | Резерв IETF |
| 4 | Запрос Heartbeat (HEARTBEAT) | 63 | Определённый IETF дополнительный блок |
| 5 | Подтверждение Heartbeat (HEARTBEAT ACK) | 64 - 126 | Резерв IETF |
| 6 | Прерывание (ABORT) | 127 | Определённый IETF дополнительный блок |
| 7 | Завершение работы (SHUTDOWN) | 128 - 190 | Резерв IETF |
| 8 | Подтверждение завершения (SHUTDOWN ACK) | 191 | Определённый IETF дополнительный блок |
| 9 | Ошибка при операции (ERROR) | 192 - 254 | Резерв IETF |
| 10 | State Cookie (COOKIE ECHO) | 255 | Определённый IETF дополнительный блок |
| 11 | Подтверждение Cookie (COOKIE ACK) | | |

Коды Chunk Type выбраны таким образом, чтобы два старших бита определяли действие, которое следует предпринять конечной точке на приёмной стороне, если Chunk Type не удаётся распознать.

00 - прекратить обработку данного пакета SCTP и отбросить его, не выполняя никаких действий по отношению к содержащейся в нем информации.

01 - прекратить обработку пакета SCTP и отбросить его без выполнения каких либо действий с содержащимися в пакете данными; указать неопознанный параметр в поле Unrecognized Parameter Type блока ERROR или INIT ACK.

10 - пропустить данный блок и продолжить обработку пакета.

11 - пропустить данный блок и продолжить обработку пакета с генерацией блока ERROR, содержащего в поле Unrecognized Chunk Type сведения о причине ошибки.

Отметим, что типы ECNE и CWR зарезервированы с целью использования в будущем для явного уведомления о насыщении (ECN[1]).

**Chunk Flags**: 8 битов

Использование этих битов определяется типом блока, указанным в поле Chunk Type. Если в документе явно не указано иное, на передающей стороне все биты следует устанавливать в 0, а на приёмной - игнорировать.

**Chunk Length**: 16 битов (целое без знака)

Это поле указывает размер блока в байтах с учётом полей Chunk Type, Chunk Flags, Chunk Length и Chunk Value. Следовательно, для блоков с пустым Chunk Value поле Length имеет значение 4. Байты заполнения в поле Chunk Length не учитываются.

**Chunk Value**: переменный размер

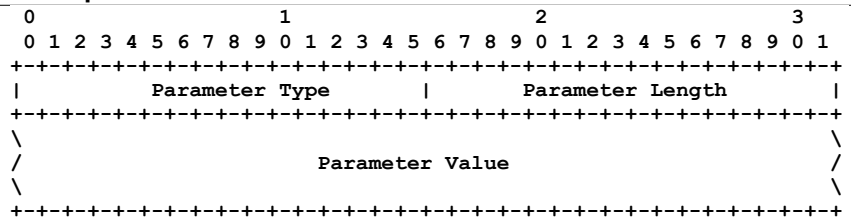
Поле Chunk Value содержит передаваемую в этом блоке информацию. Формат этого поля и способы его использования зависят от значения Chunk Type.

Общий размер блока (включая поля Type, Length и Value) **должен** быть кратным 4. Если размер блока не кратен 4, отправитель **должен** дополнить блок байтами со значением 0, не учитывая эти байты в поле Chunk Length. Заполнение размером долее 3 байтов **недопустимо**. Получатель **должен** игнорировать байты заполнения.

Подробное описание используемых в SCTP блоков приводится в параграфе 3.3. Руководства для создания расширений, определяемых IETF, приведены в параграфе 13.1.

### *3.2.1 Необязательные параметры и поля переменного размера*

[1]Explicit Congestion Notification.




Блоки управления SCTP включают зависящий от типа блока заголовок с набором полей, за которым может следовать то или иное количество параметров. Необязательные параметры и поля переменной длины указываются в формате TLV[1] как описано ниже.

**Parameter Type**: 16 битов (целое без знака)

Поле Type содержит 16-битовый идентификатор типа параметра и может принимать значения от 0 до 65534. Значение 65535 зарезервировано для определяемых IETF расширений. Все значения, кроме тех которые перечислены в данном документе при описании конкретных блоков SCTP зарезервированы для использования IETF.

**Parameter Length**: 16 битов (целое без знака)

Поле Parameter Length указывает размер параметра в байтах с учётом полей Parameter Type, Parameter Length и Parameter Value. Следовательно, для параметров с полем Parameter Value данное поле имеет значение 4. Поле размера не учитывает байты заполнения.

**Parameter Value**: переменный размер.

Поле Parameter Value содержит информацию, передаваемую с помощью данного параметра.

Общий размер параметра (включая поля Type, Length и Value) должен быть кратным 4. Если размер параметра не кратен 4, отправитель добавляет в конце (т. е., после Parameter Value) байты с нулевым значением. Байты заполнения не учитываются в поле размера. Для отправителя **недопустимо** использование более 3 байтов заполнения. Получатель **должен** игнорировать байты заполнения.

Поле Parameter Type кодируется так, чтобы два старших бита определяли действия, предпринимаемые при обнаружении неизвестного параметра.

00 - прекращение обработки пакета SCTP и его отбрасывание без учёта содержащихся в нем данных.

01 - прекращение обработки и отбрасывание пакета SCTP без обработки содержащихся в нем данных, но с генерацией сообщения об ошибке (блок ERROR или INIT ACK) с полем Unrecognized Parameter Type.

10 - пропуск параметра и продолжение обработки пакета.

11 - пропуск параметра и продолжение обработки пакета с генерацией сообщения об ошибке (блок ERROR или INIT ACK) с полем Unrecognized Parameter Type.

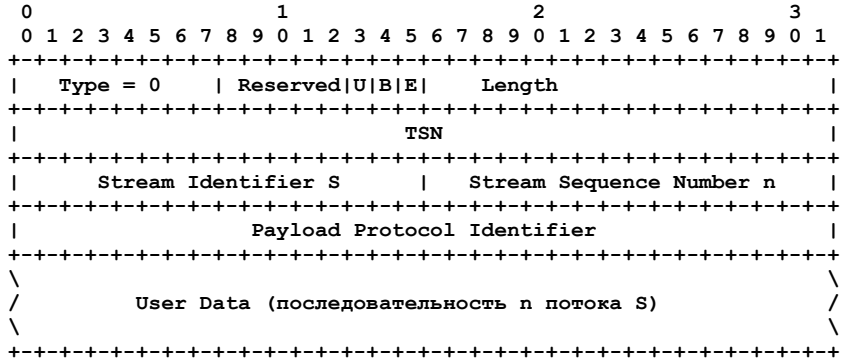
Параметры SCTP рассматриваются при описании конкретных блоков SCTP. Определяемые IETF параметры расширения описаны в параграфе 13.2.

## 3.3 Определения блоков SCTP

В этом разделе описываются форматы блоков SCTP различных типов.

### *3.3.1 Пользовательские данные (DATA) (0)*

Для блоков DATA **должен** использоваться показанный на рисунке формат.




**Reserved**: 5 битов

Отправитель должен заполнить это поле нулями, а получатель - игнорировать его.

**U bit**: 1 бит

Флаг разупорядочивания, устанавливаемый для блоков DATA, которые могут передаваться без сохранения порядка. Для таких блоков значение поля Stream Sequence Number не устанавливается, следовательно, получатель **должен** его игнорировать.

После сборки (если она требуется) неупорядоченные блоки DATA **должны** диспетчеризоваться получателем на вышележащий уровень без попыток восстановления порядка, если флаг U имеет значение 1.

[1] Type-Length-Value - Тип-Размер-Значение.

**B bit**: 1 бит

Флаг первого фрагмента пользовательского сообщения.

**E bit**: 1 бит

Флаг последнего фрагмента пользовательского сообщения.

Для нефрагментированных пользовательских сообщений устанавливаются оба флага B и E. Нулевые значения обоих флагов указывают на один из промежуточных (не первый и не последний) фрагментов. Варианты состояния флагов показаны в таблице Ошибка: источник перекрёстной ссылки не найден.

*Таблица 1. Состояния флагов B и E.*

| *B* | *E* | *Описание* |
|---|---|---|
| 1 | 0 | Первый фрагмент |
| 0 | 0 | Один из средних фрагментов |
| 0 | 1 | Последний фрагмент |
| 1 | 1 | Нефрагментированное сообщение |

При фрагментировании пользовательского сообщения на множество блоков получатель использует для сборки номера TSN. Это означает, что фрагменты **должны** передаваться в строгой последовательности.

**Length**: 16 битов (целое без знака)

Это поле показывает размер блока DATA от начала поля типа и до конца пользовательских данных (без учёта байтов заполнения). Для блоков DATA, не содержащих пользовательских данных, Length = 16.

**TSN**: 32 бита (целое без знака)

Это поле содержит порядковый номер TSN для блока DATA. Значения номеров TSN могут находиться в диапазоне от 0 до 4294967295 ($2^{32}$ - 1). После достижения TSN максимального значения 4294967295 нумерация продолжается с 0.

**Stream Identifier S**: 16 битов (целое без знака)

Идентификатор потока, к которому относится блок данных.

**Stream Sequence Number n**: 16 битов (целое без знака)

Это значение представляет порядковый номер в потоке S. Допустимые значения лежат в диапазоне от 0 до 65535.

При фрагментировании пользовательского сообщения протоколом SCTP в каждом фрагмент **должен** указываться одинаковый порядковый номер в потоке.

**Payload Protocol Identifier**: 32 бита (целое без знака)

Это поле содержит заданный приложением идентификатор протокола. SCTP получает идентификатор от вышележащего уровня и передаёт его удалённому партнёру. Значение идентификатора не используется протоколом SCTP, но может быть использовано некоторыми сетевыми объектами и приложениями на удалённой стороне для идентификации типа информации, передаваемой в блоке DATA. Это поле должно передаваться даже для фрагментированных блоков DATA (чтобы обеспечить доступность информации для агентов в сети).

Нулевое значение показывает что протокол вышележащего уровня не указал идентификатор протокола для этого блока.

**User Data**: переменный размер

Это поле переменной длины содержит пользовательскую информацию. Реализация протокола **должна** обеспечивать выравнивание размеров поля по 4-байтовой границе путём добавления от 1 до 3 байтов с нулевым значением. **Недопустимо** учитывать байты заполнения в поле размера блока. Для отправителя **недопустимо** использование более 3 байтов заполнения.

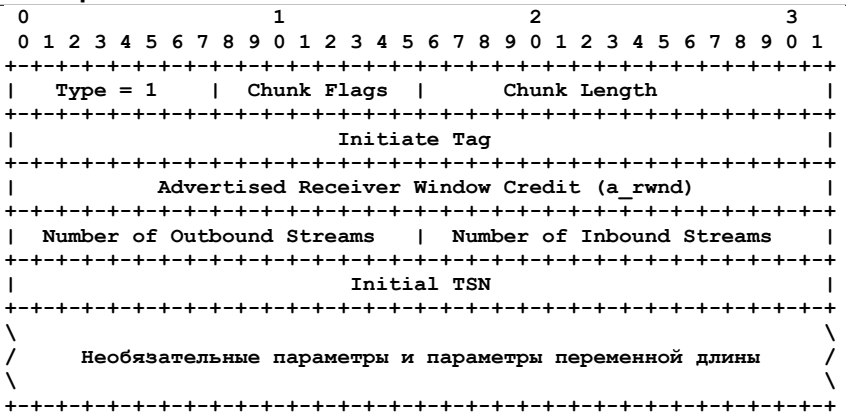
## *3.3.2 Инициализация (INIT) (1)*

Этот блок используется для создания ассоциации SCTP между парой конечных точек. Формат блока INIT показан на рисунке.

Параметры блока INIT перечислены ниже. Если в описании явно не указано иное, каждый параметр **должен** включаться в блок INIT не более 1 раза.

Параметры Initiate Tag, Advertised Receiver Window Credit, Number of Outbound Streams, Number of Inbound Streams и Initial TSN являются обязательными и имеют фиксированные размеры. Кроме того, существует ряд необязательных параметров переменной или фиксированной длины - IPv4 Address[1] (5), IPv6 Address[1] (6), Cookie Preservative (9), ECN Capable[2] (32768 - 0x8000), Host Name Address[3] (11), Supported Address Types[4] (12).

[1]Блоки INIT могут содержать множество адресов IPv4 и/или IPv6 в произвольных комбинациях.
[2]Поле ECN capable зарезервировано для будущего использования с ECN.
[3]**Недопустимо** включение в блок INIT более 1 параметра Host Name Address. Более того, для отправителя блока INIT **недопустимо** комбинировать в блоке INIT любые другие типы адресов с Host Name Address. Получатель блока INIT **должен** игнорировать адреса всех остальных типов при наличии в блоке INIT параметра Host Name Address.
[4]Этот параметр используется для указания всех типов адресов, которые поддерживает передающая сторона. Отсутствие данного параметра говорит о поддержке передающей стороной адресов всех типов.




Поле Chunk Flags в блоках INIT является резервным и отправитель должен установить нулевые значения во всех битах, а получатель - игнорировать это поле. Последующие параметры блока INIT могут обрабатываться в произвольном порядке.

**Initiate Tag**: 32 бита (целое без знака)

Получатель INIT (откликающаяся сторона) записывает значение параметра Initiate Tag. Это значение **должно** включаться в поле Verification Tag каждого пакета SCTP, который получатель данного блока INIT будет передавать через эту ассоциацию.

Поле Initiate Tag может содержать любые значения кроме 0. Рекомендации по выбору значений этого тега приводятся в параграфе 5.3.1.

Если в принятом блоке INIT значение Initiate Tag = 0, получатель **должен** трактовать это как ошибку и закрывать ассоциацию, передавая ABORT.

**Advertised Receiver Window Credit (a_rwnd)**: 32 бита (целое без знака)

Это значение представляет размер (в байтах) выделенного буферного пространства, которое отправитель INIT зарезервировал для данной ассоциации. В течение срока существования ассоциации **не следует** снижать размер этого буфера (т. е., буфер может быть удалён только вместе с ассоциацией), однако конечная точка **может** изменить значение с помощью параметра a_rwnd, переданного в SACK.

**Number of Outbound Streams (OS)**: 16 битов (целое без знака)

Указывает число исходящих потоков, которые отправитель блока INIT желает открыть для данной ассоциации. Для этого параметра **недопустимо** использование значения 0. При получении блока INIT с OS = 0 получателю **следует** прервать ассоциацию.

**Number of Inbound Streams (MIS)**: 16 битов (целое без знака)

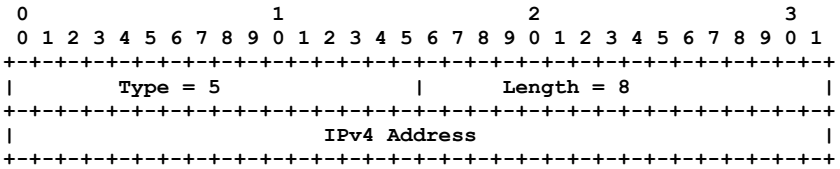
Указывает максимальное число потоков, которые отправитель данного блока INIT готов принять от удалённого партнёра для этой ассоциации[1]. Использование значений 0 в данном поле **недопустимо**. При получении блока INIT с MIS = 0 получателю **следует** разорвать ассоциацию.

**Initial TSN (I-TSN)**: 32 бита (целое без знака)

Определяет начальное значение TSN, которое будет использовать отправитель (диапазон допустимых значений - от 0 до 42949672950). В это поле **можно** помещать значение поля Initiate Tag.

### 3.3.2.1 Необязательные параметры и параметры переменного размера в блоке INIT

Перечисленные ниже параметры используют формат TLV, как описано в параграфе 3.2.1. Любые комбинации параметров TLV **должны** размещаться в блоке после параметров фиксированного размера, описанных в предыдущем параграфе.




Параметр IPv4 Address (5)

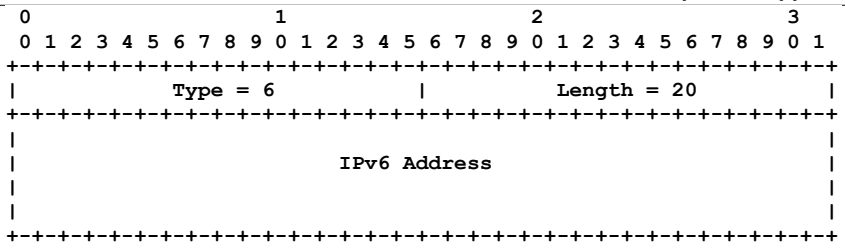
**IPv4 Address**: 32 бита (целое без знака)

Содержит адрес IPv4 передающей стороны в двоичном формате.

Параметр IPv6 Address (6)

**IPv6 Address**: 128 битов (целое без знака)

[1]В протоколе не используется согласование числа потоков - каждая сторона просто выбирает меньшее из двух значений - предложенного и запрашиваемого. Более подробное описание приводится в параграфе 5.1.1.



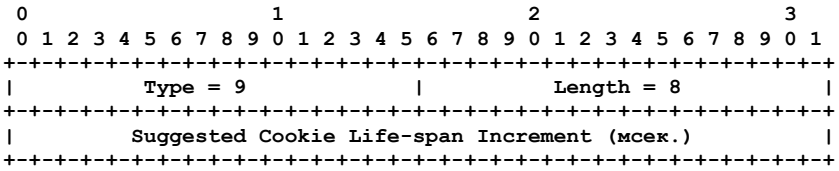

Содержит адрес IPv6 передающей стороны в двоичном представлении[1].

Вместе с параметром Source Port Number в общем заголовке SCTP адреса IPv4 и IPv6 определяют адрес транспортного уровня, который отправитель блока INIT будет поддерживать для создаваемой ассоциации. Т. е., этот IP-адрес в течении срока существования данной ассоциации может появляться в поле отправителя дейтаграмм IP, передаваемых отправителем данного блока INIT, и может использоваться как IP-адрес получателя в дейтаграммах, передаваемых получателем данного блока INIT.

В блоке INIT можно указывать более одного адреса IP, если отправитель INIT является многодомным хостом. Более того, многодомные хосты могут быть подключены к разнотипным сетям и в блоках INIT могут указываться одновременно адреса IPv4 и IPv6.

Если INIT содержит хотя бы один параметр IP Address, адрес отправителя в дейтаграмме, содержащей блок INIT, и любые адреса, указанные в INIT, могут использоваться другой стороной при ответе в качестве адреса получателя. Если INIT не содержит ни одного параметра IP Address, получившая блок INIT конечная точка **должна** отправлять ответ по адресу отправителя в заголовке дейтаграммы IP, содержащей блок INIT.

Отметим, что отсутствие параметров IP address в блоках INIT и INIT-ACK упрощает организацию ассоциаций при работе через устройства NAT.




**Cookie Preservative (9)**

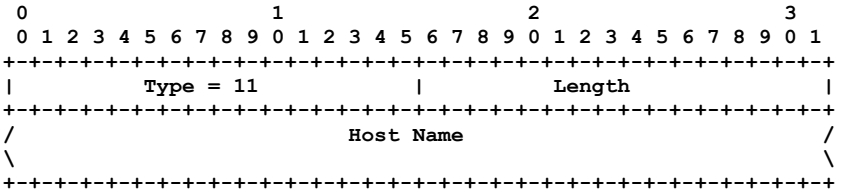
Отправителю блока INIT следует использовать этот параметр для того, чтобы предложить получателю INIT увеличение срока жизни State Cookie.

**Suggested Cookie Life-span Increment**: 32 бита (целое без знака)

Этот параметр показывает получателю размер увеличения срока жизни cookie в миллисекундах.

Этот необязательный параметр отправителю следует добавлять в блок INIT при попытке создания ассоциации после того, как предыдущая попытка сорвалась по причине ошибки в работе stale cookie. Получатель **может** проигнорировать предложенное увеличение срока жизни cookie в соответствии со своей политикой безопасности.

**Параметр Host Name (11)**

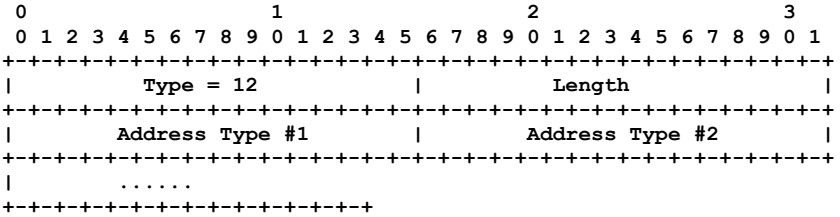



Отправитель блока INIT использует этот параметр для передачи партнёру своего имени (Host Name) взамен адреса IP. Преобразование имени осуществляется на приёмной стороне. Использование этого параметра может упростить создание ассоциаций через NAT.

**Host Name**: переменный размер

Это поле содержит имя хоста с использованием синтаксиса, определённого в параграфе 2.1 RFC1123 [RFC1123]. Метод преобразования имени в адрес выходит за пределы спецификации протокола SCTP.

Имя хоста должно содержать по крайней мере один завершающий нуль-символ, который учитывается в поле размера.




**Supported Address Types (12)**

Отправитель блока INIT использует этот параметр для перечисления всех типов поддерживаемых им адресов.

**Address Type**: 16 битов (целое без знака)

[1]Для отправителя **недопустимо** использование отображения адресов IPv4 на адреса IPv6, описанное в [RFC2373]. Адреса IPv4 следует указывать с помощью параметра IPv4 Address Parameter.

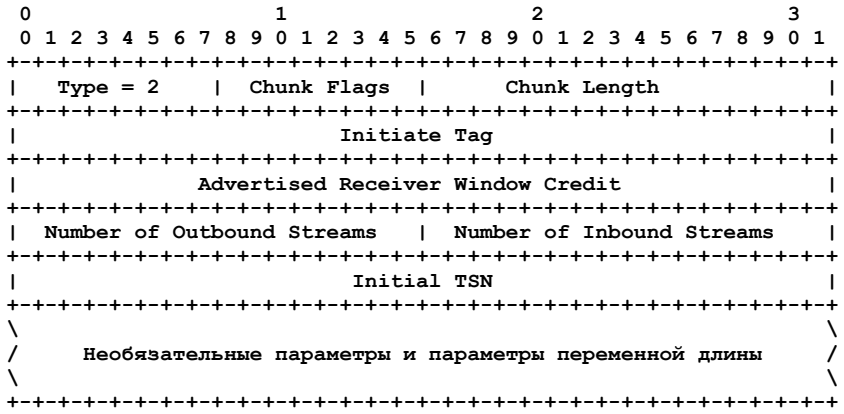
В этом поле указывается значение типа адреса из соответствующего TLV (например, IPv4 = 5, IPv6 = 6, Hostname = 11).

### *3.3.3 Подтверждение инициализации (INIT ACK) (2)*

Блок INIT ACK используется для подтверждения инициирования ассоциации SCTP.

Параметры INIT ACK форматируются подобно параметрам блока INIT. Кроме того, блоки данного типа содержат два дополнительных параметра: State Cookie и Unrecognized Parameter.

Формат блока INIT ACK показан на рисунке.




**Initiate Tag**: 32 бита (целое без знака)

Получатель блока INIT ACK записывает значение параметра Initiate Tag. Это значение **должно** помещаться в поле Verification Tag каждого пакета SCTP, который получатель INIT ACK будет передавать в данной ассоциации.

Для поля Initiate Tag **недопустимо** использование значения 0. Выбор значений параметра Initiate Tag рассматривается в параграфе 5.3.1.

Если в полученном блоке INIT ACK параметр Initiate Tag = 0, приёмная сторона **должна** трактовать это как ошибку и прерывать ассоциацию с помощью ABORT.

**Advertised Receiver Window Credit (a_rwnd)**: 32 бита (целое без знака)

Это значение указывает размер буфера (в байтах), выделенного отправителем INIT ACK для данной ассоциации. Во время существования ассоциации **не следует** уменьшать размер буфера (т. е., выделенный буфер должен умереть вместе с ассоциацией).

**Number of Outbound Streams (OS)**: 16 битов (целое без знака)

Определяет число исходящих потоков, которые отправитель INIT ACK желает создать для данной ассоциации. **Недопустимо** устанавливать для этого параметра значение 0.

Получателю блока INIT ACK с параметром OS = 0 **следует** закрыть ассоциацию, отбросив TCB.

**Number of Inbound Streams (MIS)**: 16 битов (целое без знака)

Определяет максимальное число потоков, которые отправитель INIT ACK позволяет создать удалённому партнёру в данной ассоциации[1]. **Недопустимо** использование нулевого значения для этого параметра.

Получателю блока INIT ACK с параметром MIS = 0 **следует** разорвать ассоциацию, отбрасывая TCB.

**Initial TSN (I-TSN)**: 32 бита (целое без знака)

Определяет начальный номер TSN, который отправитель INIT-ACK будет использовать. Корректные значения лежат в диапазоне от 0 до 4294967295. Это поле **может** содержать значение параметра Initiate Tag.

Параметры Initiate Tag, Advertised Receiver Window Credit, Number of Outbound Streams, Number of Inbound Streams, Initial TSN имеют фиксированный размер и являются обязательными. Обязательный параметр State Cookie (7) не имеет фиксированного размера, а параметры IPv4 Address[2] (5), IPv6 Address[14] (6), Unrecognized Parameters (8), ECN Capable[3] (32768 - 0x8000) и Host Name[4] (11) являются необязательными.

**Замечание для разработчиков**: реализация протокола **должна** быть готова к приёму достаточно больших блоков INIT ACK (более 1500 байтов) по причине наличия параметров переменной длины для state cookie и списка адресов. Например, если отвечающий на блок INIT хост имеет 1000 адресов IPv4, которые он желает сообщить партнёру, ему потребуется не менее 8000 байтов для включения этого списка в INIT ACK.

В комбинации с параметром Source Port из общего заголовка SCTP каждый параметр IP Address в блоке INIT ACK показывает получателю INIT ACK транспортный адрес отправителя INIT ACK, корректный в течение срока жизни создаваемой ассоциации.

Если INIT ACK содержит хотя бы один параметр IP Address, адрес отправителя в дейтаграмме, содержащей блок INIT, и любые адреса, указанные в INIT ACK, могут использоваться другой стороной при ответе в качестве адреса

[1]В протоколе не используется согласование числа потоков - каждая сторона просто выбирает меньшее из двух значений - предложенного и запрашиваемого. Более подробное описание приводится в параграфе 5.1.1.
[2]Блоки INIT ACK могут содержать множество адресов IPv4 и/или IPv6 в произвольных комбинациях.
[3]Поле ECN capable зарезервировано для будущего использования с ECN.
[4]**Недопустимо** включение в блок INIT ACK более 1 параметра Host Name. Более того, для получателя **недопустимо** комбинировать в блоке INIT ACK любые другие типы адресов с Host Name. Получатель блока INIT ACK **должен** игнорировать адреса всех остальных типов при наличии в блоке INIT ACK параметра Host Name.

получателя. Если INIT ACK не содержит ни одного параметра IP Address, получившая блок INIT ACK конечная точка **должна** отправлять ответ по адресу отправителя в заголовке дейтаграммы IP, содержащей блок INIT ACK.

Параметры State Cookie и Unrecognized Parameters используют формат TLV в соответствии с определением параграфа 3.2.1 и описаны ниже.

Остальные поля соответствуют одноимённым полям блока INIT.

#### 3.3.3.1 Необязательные параметры и параметры переменной длины

Параметр State Cookie (7)

**Type**: 7

**Length**: переменный размер, зависящий от длины Cookie

**Value**:

Этот параметр **должен** содержать всю информацию о параметрах и состоянии, требуемую отправителю данного блока INIT ACK для создания ассоциации вместе с кодом аутентификации MAC. Описание State Cookie приводится в параграфе 5.1.3.

**Параметр Unrecognized Parameters (8)**

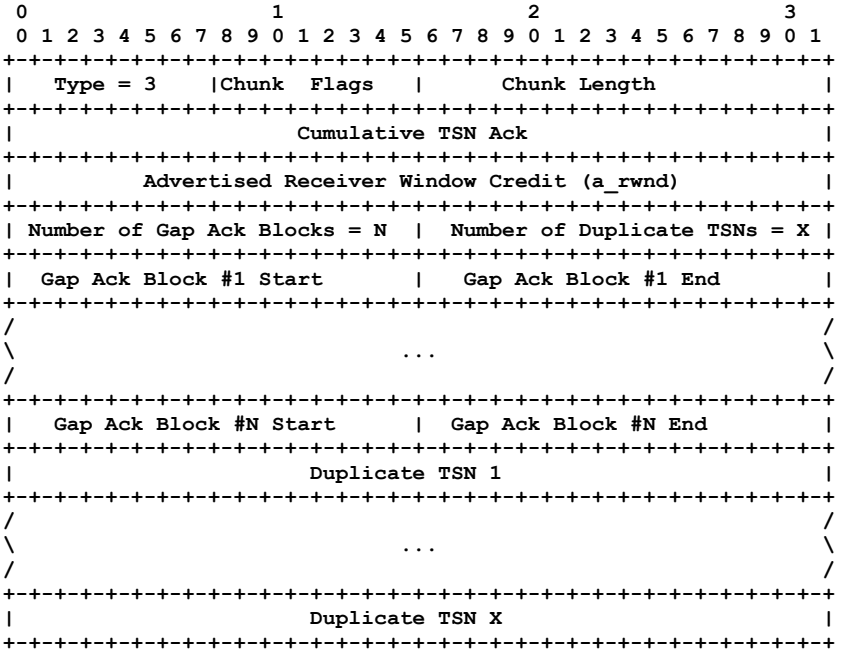
**Type**: 8

**Length**: переменный размер.

**Value**:

Этот параметр возвращается отправителю блока INIT, если этот блок содержит значения, для которых отправителю должен передаваться отчёт. Значение данного параметра будет содержать поле нераспознанного параметра, скопированное из блока INIT (все 3 поля Type, Length и Value).

### *3.3.4 Селективное подтверждение (SACK) (3)*




Этот блок передаётся партнёру для подтверждения приёма. блоков DATA и информирования партнёра о пропусках в порядковых номерах блоков DATA, представленных в TSN.

Блок SACK **должен** содержать параметры Cumulative TSN Ack и Advertised Receiver Window Credit (a_rwnd).

По определению параметр Cumulative TSN Ack содержит значение последнего номера TSN, полученного до того, как была нарушена последовательность принятых TSN; следующее за этим значение TSN (на 1 большее) ещё не получено стороной, передающей SACK. Этот параметр, следовательно, подтверждает доставку всех TSN, номера которых не превышают значение параметра.

Обработка a_rwnd на принявшей SACK стороне рассматривается в параграфе 6.2.1.

SACK может также содержать параметры Gap Ack Block, каждый из которых подтверждает последовательность TSN, полученных после прерывания основной последовательности номеров TSN. По определению все TSN, подтверждённые с помощью Gap Ack Block, имеют значения, превышающие Cumulative TSN Ack.

**Chunk Flags**: 8 битов

Заполняется нулями на передающей стороне и игнорируется на приёмной.

**Cumulative TSN Ack**: 32 бита (целое без знака)

Этот параметр содержит значение TSN последнего блока DATA, который был принят до прерывания последовательности номеров.

**Advertised Receiver Window Credit (a_rwnd)**: 32 бита (целое без знака)

Это поле показывает обновлённое значение размера (в байтах) приёмного буфера на стороне отправителя данного блока SACK (см. параграф 6.2.1).

**Number of Gap Ack Blocks**: 16 битов (целое без знака)

Показывает число параметров Gap Ack Block, включённых в SACK.

**Number of Duplicate TSNs**: 16 битов

Это поле показывает число дубликатов TSN, полученных конечной точкой. Каждый дубликат TSN указывается в последующем списке Gap Ack Block.

**Gap Ack Block:**

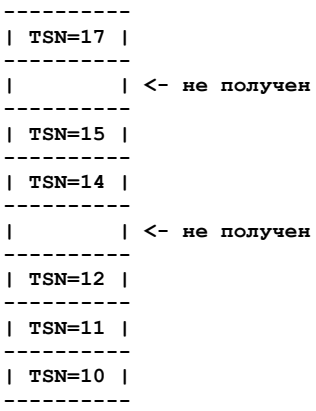
Эти поля содержат параметры Gap Ack Block, число которых задано значением поля Number of Gap Ack Blocks. Все блоки DATA с номерами TSN, большими или равными Cumulative TSN Ack + Gap Ack Block Start и меньшими или равными Cumulative TSN Ack + Gap Ack Block End из каждого предполагаются принятыми без ошибок.

**Gap Ack Block Start**: 16 битов (целое без знака)

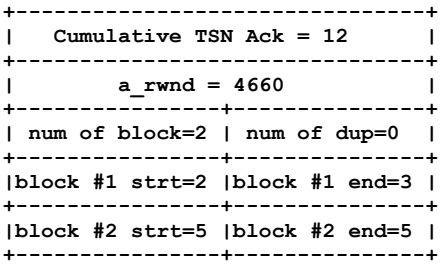
Показывает стартовое смещение TSN для данного Gap Ack Block. Для расчёта реального номера TSN это смещение добавляется к значению параметра Cumulative TSN Ack. Рассчитанное таким способом значение TSN указывает первый номер TSN в данном Gap Ack Block, который был принят.

**Gap Ack Block End**: 16 битов (целое без знака)

Показывает финишное смещение TSN для данного Gap Ack Block. Для расчёта реального номера TSN это смещение добавляется к значению параметра Cumulative TSN Ack. Рассчитанное таким способом значение TSN указывает последний номер TSN в данном Gap Ack Block для принятого блока DATA.




Предположим для примера, что недавно принятые блоки DATA в момент принятия решения о передаче SACK образуют последовательность, показанную на рисунке справа. Тогда часть блока SACK **должна** включать поля, показанные на рисунке слева (предполагается, что новое значение a_rwnd = 4660).

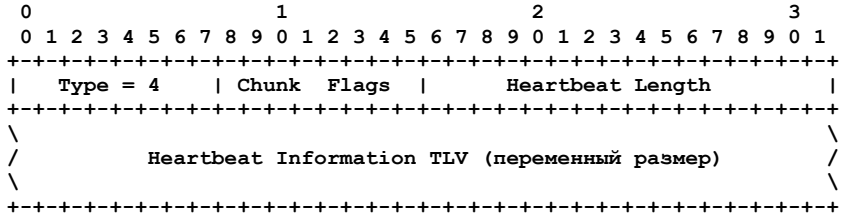



**Duplicate TSN**: 32 бита (целое без знака)

Показывает количество случаев, когда номер TSN был принят как дубликат, с момента передачи последнего блока SACK. При получении каждого дубликата TSN (до момента передачи SACK) этот дубликат добавляется в список. Счётчик дубликатов сбрасывается в 0 после отправки каждого блока SACK.

Например, если получатель принял TSN 19 трижды, номер 19 будет указан два раза в блоке SACK. После отправки SACK, если TSN 19 будет принят снова, он будет указан в списке нового блока SACK.

## *3.3.5 Запрос Heartbeat (HEARTBEAT) (4)*




Конечной точке следует передавать такой запрос своему партнёру для проверки его доступности через тот или иной транспортный адрес, указанный для данной ассоциации.

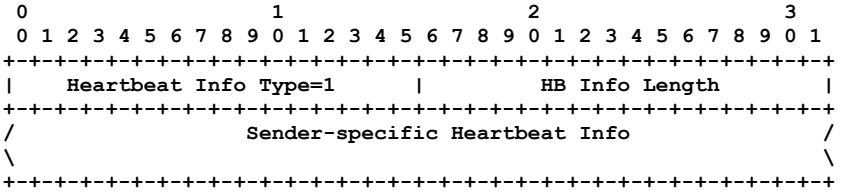
Поле Heartbeat Information имеет переменный размер и содержит структуру данных, понятную только отправителю.

**Chunk Flags**: 8 битов

Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

**Heartbeat Length**: 16 битов (целое без знака)

Задаёт размер блока в байтах с учётом заголовка и поля Heartbeat Information.

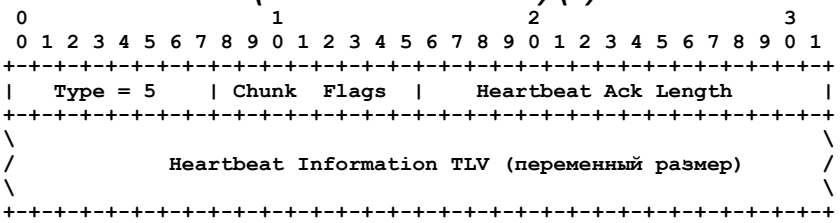



**Heartbeat Information**: переменный размер

Содержит обязательный параметр Heartbeat Info (1) переменного размера, описанный в параграфе 3.2.1.

Устанавливаемое отправителем значение поля Heartbeat Info обычно включает текущее время отправителя в момент отправки блока HEARTBEAT и описание транспортного адреса, по которому передаётся HEARTBEAT (см. параграф 8.3).

## *3.3.6 Подтверждение Heartbeat (HEARTBEAT ACK) (5)*




Конечной точке следует передавать такой блок в ответ на запрос HEARTBEAT (см. параграф 8.3). Блок HEARTBEAT ACK всегда передаётся по тому адресу IP, который был указан в заголовке дейтаграммы, содержавшей HEARTBEAT.

Параметр Heartbeat Info (1) является обязательным и имеет переменный размер.

**Chunk Flags**: 8 битов

Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

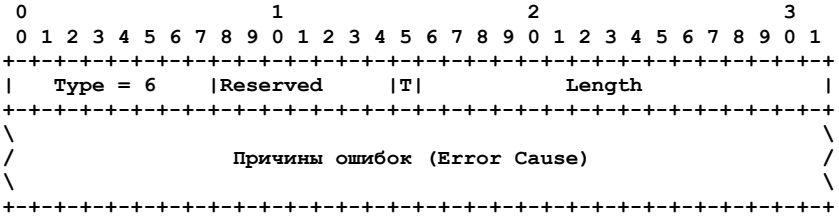
**Heartbeat Ack Length**: 16 битов (целое без знака)

Задаёт размер блока в байтах с учётом заголовка и поля Heartbeat Information.

**Heartbeat Information**: переменный размер

Это поле **должно** содержать параметр Heartbeat Information из запроса Heartbeat, на который передаётся отклик.

## *3.3.7 Разрыв ассоциации (ABORT) (6)*




Блок ABORT передаётся партнёру для незамедлительного разрыва ассоциации. Блок ABORT может содержать параметры Error Cause, информирующие партнёра о причинах разрыва ассоциации. **Недопустимо** группировать блоки DATA с блоком ABORT. Блоки управления (кроме INIT, INIT ACK и SHUTDOWN COMPLETE) **могут** группироваться с ABORT, но эти блоки **должны** размещаться перед блоком ABORT в пакете SCTP, поскольку в противном случае получатель проигнорирует их.

Если конечная точка получает блок ABORT с некорректным форматом или для несуществующей ассоциации, она должна отбросить такой блок. Более того, в некоторых случаях для получившей такой блок ABORT конечной точки недопустимо передавать в ответ свой блок ABORT.

**Chunk Flags**: 8 битов

**Reserved**: 7 битов

Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

**T bit**: 1 бит

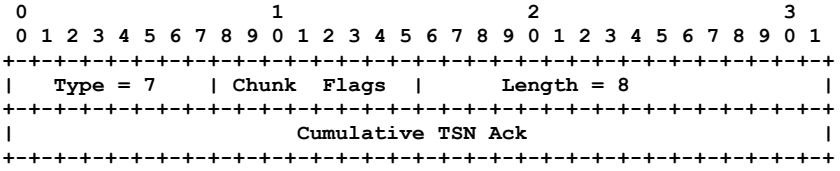
Бит T сбрасывается в 0, если отправитель имеет TCB, который будет разрушен. При отсутствии разрушаемых TCB этот бит должен иметь значение 1.

Примечание: для верификации этого типа блоков служат специальные правила, описанные в параграфе 8.5.1.

**Length**: 16 битов (целое без знака)

Указывает размер блока в байтах с учётом заголовка и всех полей Error Cause. Определения причин ошибки (Error Cause) приведены в параграфе 3.3.10.

### 3.3.8 Завершение ассоциации (SHUTDOWN) (7)




Конечная точка ассоциации **должна** использовать этот блок для аккуратного завершения работы ассоциации. Формат блока описан ниже.

**Chunk Flags**: 8 битов

Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

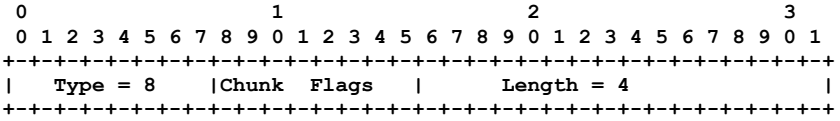
**Length**: 16 битов (целое без знака)

Показывает размер параметра и имеет значение 8.

**Cumulative TSN Ack**: 32 бита (целое без знака)

Этот параметр содержит номер последнего TSN, принятого без пропусков в нумерации[1].

### 3.3.9 Подтверждение закрытия ассоциации (SHUTDOWN ACK) (8)



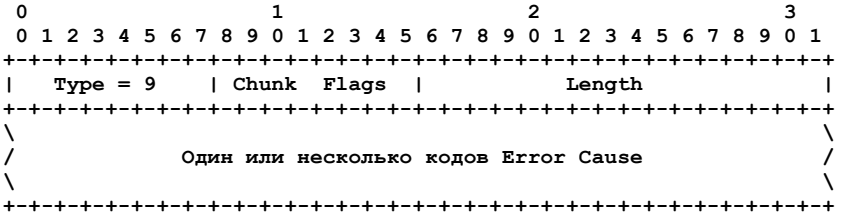

Этот блок **должен** использоваться для подтверждения приёма блока SHUTDOWN при завершении процесса закрытия, описанного в параграфе 9.2.

Блок SHUTDOWN ACK не содержит параметров.

**Chunk Flags**: 8 битов

Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

### 3.3.10 Ошибка при работе (ERROR) (9)




Конечные точки передают блоки этого типа для уведомления партнёра о некоторых типах ошибок. Блок содержит один или несколько кодов ошибок. Приём сообщения об ошибке не обязывает партнёра разрывать ассоциацию, однако такие блоки могут передаваться вместе с блоком ABORT в качестве отчёта о причине разрыва. Параметры блока описаны ниже.

**Chunk Flags**: 8 битов

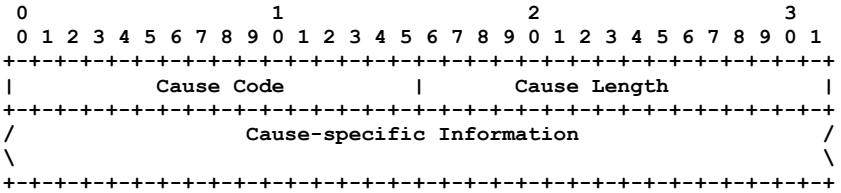
Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

**Length**: 16 битов (целое без знака)

Указывает размер блока в байтах с учётом заголовка и всех полей Error Cause.

Причины ошибок указываются как параметры переменного размера в соответствии с определением в параграфе 3.2.1.

**Cause Code**: 16 битов (целое без знака)




Указывает тип причины, которая вызвала передачу сообщения.

| Код | Описание | Код | Описание |
|---|---|---|---|
| 1 | Некорректный идентификатор потока | 6 | Нераспознанный тип блока |
| 2 | Отсутствие обязательного параметра | 7 | Некорректный обязательный параметр |
| 3 | Ошибка Stale Cookie | 8 | Нераспознанные параметры |
| 4 | Нехватка ресурсов | 9 | Отсутствие пользовательских данных |
| 5 | Не удалось преобразовать адрес | 10 | Получение Cookie в процессе закрытия |

**Cause Length**: 16 битов (целое без знака)

[1] Поскольку блок SHUTDOWN не содержит параметров Gap Ack Block, он не может служить подтверждением TSN, принятых с нарушением порядка. В блоках SACK отсутствие Gap Ack Block, которые были указаны в предыдущих сообщениях, показывает, что получатель данных отказался от соответствующих блоков DATA. Поскольку SHUTDOWN не содержит Gap Ack Block, получателю блока SHUTDOWN не следует интерпретировать отсутствие Gap Ack Block как отказ (см. параграф 6.2).

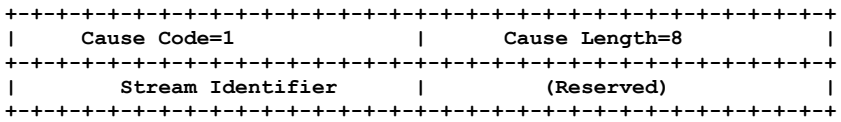
Указывает размер параметра в байтах с учётом полей Cause Code, Cause Length и Cause-Specific Information

**Cause-specific Information**: переменный размер

В этом поле указываются сведения об ошибке. Причины ошибок SCTP рассматриваются в параграфах 3.3.10.1 - 3.3.10.10. Рекомендации по определению новых типов ошибок для IETF даны в параграфе 13.3.

### 3.3.10.1 Некорректный идентификатор потока (1)

**Причина ошибки**




**Invalid Stream Identifier**: показывает, что конечная точка получила блок DATA, переданный в несуществующий поток.

**Stream Identifier**: 16 битов (целое без знака)

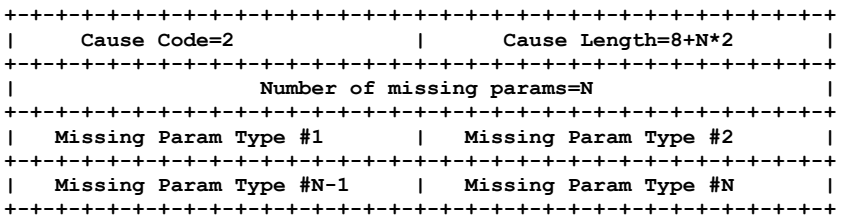
Этот параметр содержит значение Stream Identifier из блока DATA, вызвавшего ошибку.

**Reserved**: 16 битов

Зарезервированное поле. Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

### 3.3.10.2 Отсутствует обязательный параметр (2)

**Причина ошибки**




**Missing Mandatory Parameter**: говорит об отсутствии одного или нескольких обязательных параметров TLV в принятом блоке INIT или INIT ACK.

**Number of Missing params**: 32 бита (целое без знака)

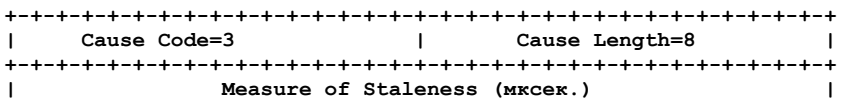
Это значение указывает число отсутствующих параметров, перечисленных в поле Cause-specific Information.

**Missing Param Type**: 16 битов (целое без знака)

Каждое поле содержит пропущенный обязательный параметр.

### 3.3.10.3 Ошибка Stale Cookie (3)

**Причина ошибки**




**Stale Cookie Error**: говорит о получении корректного значения State Cookie с истекшим сроком.

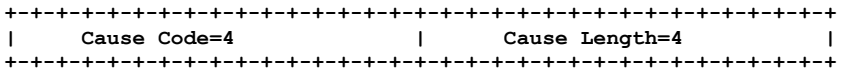
**Measure of Staleness**: 32 бита (целое без знака)

В этом поле указывается разница между текущим временем и временем окончания срока действия State Cookie (в микросекундах).

Отправитель этого сообщения **может** указать время, прошедшее после завершения срока действия State Cookie, отличным от нуля значением поля Measure of Staleness. Если отправитель не хочет передавать эти сведения, ему следует установить нулевое значение в поле Measure of Staleness.

### 3.3.10.4 Нехватка ресурсов (4)

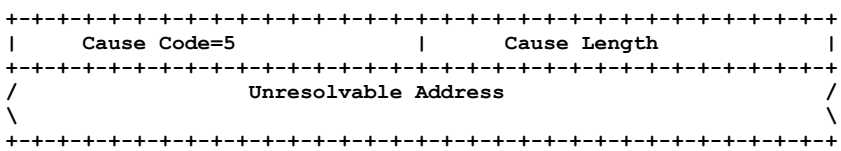
**Причина ошибки**




**Out of Resource**: говорит о нехватке ресурсов. Обычно эта информация передаётся вместе с блоком ABORT.

### 3.3.10.5 Не удалось преобразовать адрес (5)

**Причина ошибки**



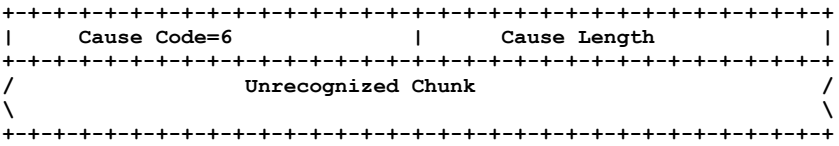

**Unresolvable Address**: говорит о том, что конечной точке не удалось преобразовать полученный адрес (например, данный тип адресации не поддерживается). Обычно такое сообщение передаётся вместе с блоком ABORT.

**Unresolvable Address**: переменный размер

Поле Unresolvable Address содержит полное значение параметра TLV, задающего адрес (или параметра Host Name), который не удалось преобразовать.

### 3.3.10.6 Не распознан тип блока (6)

**Причина ошибки**



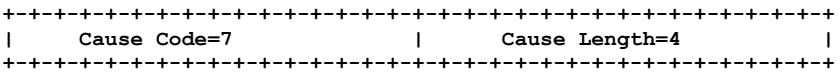

**Unrecognized Chunk Type**: эта информация передаётся отправителю блока, если получатель не смог определить тип блока и два старших бита поля Chunk Type имеют значение 01 или 11.

**Unrecognized Chunk**: переменный размер

Поле Unrecognized Chunk содержит нераспознанный блок из пакета SCTP, включая поля Chunk Type, Chunk Flags и Chunk Length.

### 3.3.10.7 Некорректный обязательный параметр (7)

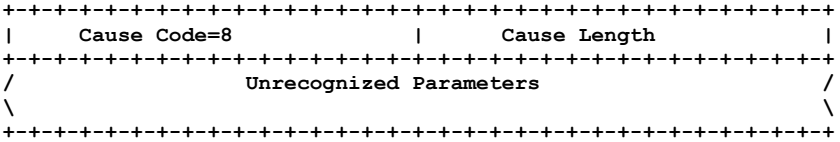
**Причина ошибки**




**Invalid Mandatory Parameter**: это сообщение передаётся отправителю блока INIT или INIT ACK, когда один или несколько обязательных параметров имеют некорректные значения.

### 3.3.10.8 Нераспознанные параметры (8)

**Причина ошибки**



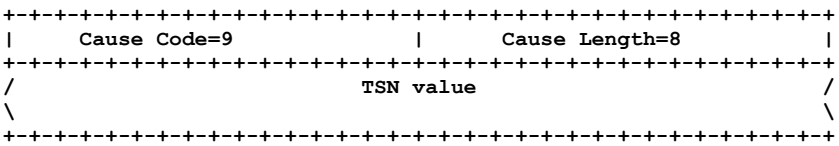

**Unrecognized Parameters**: это сообщение возвращается отправителю блока INIT ACK, если получателю не удаётся распознать один или несколько необязательных параметров TLV в блоке INIT ACK.

**Unrecognized Parameters**: переменный размер

Поле Unrecognized Parameters содержит нераспознанные параметры, скопированные из блока INIT ACK полностью в форме TLV. Это сообщение обычно передаётся в блоках ERROR, объединённых с блоком COOKIE ECHO при отклике на INIT ACK, когда отправитель блока COOKIE ECHO желает сообщить о нераспознанных параметрах.

### 3.3.10.9 Нет пользовательских данных (9)

**Причина ошибки**




**No User Data**: это сообщение передаётся отправителю блока DATA, если принятый блок не содержит пользовательских данных.

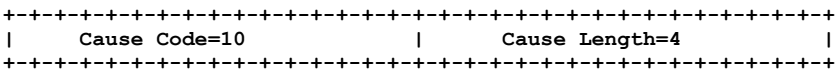
**TSN value**: 32 бита (целое без знака)

Значение поля TSN из блока DATA, с которым связана ошибка.

Это сообщение обычно передаётся в блоке ABORT (см. параграф 6.2).

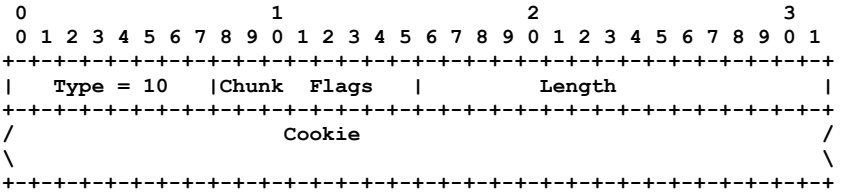
### 3.3.10.10 Получение Cookie во время процедуры закрытия (10)

**Причина ошибки**




**Cookie Received While Shutting Down**: это сообщение говорит о приёме COOKIE ECHO в то время, когда конечная точка находилась в состоянии SHUTDOWN-ACK-SENT. Обычно этот код возвращается в блоке ERROR, сгруппированном с повторным блоком SHUTDOWN ACK.

### 3.3.11 Cookie Echo (COOKIE ECHO) (10)




Этот блок используется только в процессе создания ассоциации. Он передаётся инициатором ассоциации удалённому партнёру для завершения процесса инициирования ассоциации. Такой блок **должен** предшествовать любому блоку

DATA, передаваемому через ассоциацию, но **может** быть сгруппирован с одним или несколькими блоками DATA в один пакет.

**Chunk Flags**: 8 bit

Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

**Length**: 16 битов (целое без знака)

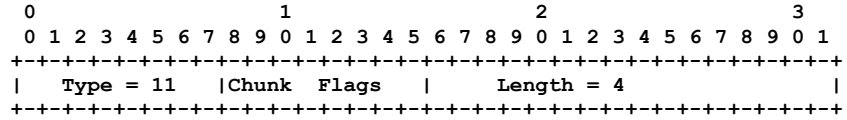
Указывает размер блока в байтах, включая 4 байта заголовка блока и размер Cookie.

**Cookie**: переменный размер

Это поле должно содержать точную копию параметра State Cookie, полученного в предыдущем блоке INIT ACK.

Реализациям протокола следует стремиться к уменьшению размера cookie в целях обеспечения взаимодействия.

### *3.3.12 Подтверждение Cookie (COOKIE ACK) (11)*




Этот тип блоков используется только при создании ассоциации и служит подтверждением приёма блока COOKIE ECHO. Данный блок **должен** предшествовать любому блоку DATA или SACK в данной ассоциации, но **может** группироваться с одним или несколькими блоками DATA или блоком SACK в одном пакете SCTP.

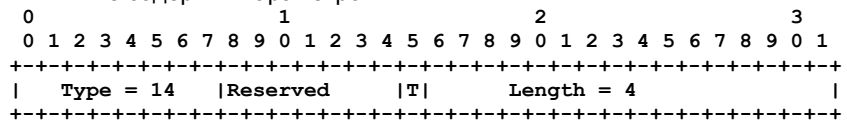
**Chunk Flags**: 8 битов

Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

### *3.3.13 Закрытие ассоциации завершено (SHUTDOWN COMPLETE) (14)*

Этот тип блоков **должен** использоваться для подтверждения приёма блока SHUTDOWN ACK при завершении ассоциации (см. параграф 9.2).

Блок SHUTDOWN COMPLETE не содержит параметров.




**Chunk Flags**: 8 битов

**Reserved**: 7 битов

Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

**T bit**: 1 бит

Бит T устанавливается в 0, если отправитель имеет TCB, который будет разрушен. Если таких TCB, бит должен иметь значение 1.

Примечание: Для верификации этого типа блоков используются специальные правила, описанные в параграфе 8.5.1.

## *4. Диаграмма состояний ассоциации SCTP*

В течение срока существования ассоциации SCTP участвующие в ней конечные точки могут переходить из одного состояния в другое в результате тех или иных событий. События, способные изменить состояние ассоциации, включают:

- вызовы пользовательских примитивов SCTP (например, [ASSOCIATE], [SHUTDOWN], [ABORT]);
- приём управляющих блоков INIT, COOKIE ECHO, ABORT, SHUTDOWN и т. п.;
- тайм-ауты.

Приведённые рисунки показывают возможные состояния ассоциации и переходы между ними вместе с вызывающими эти переходы событиями и выполняемыми при смене состояния действиями. Некоторые ошибки не показаны на схеме состояний. Полное описание всех состояний и переходов приводится в тексте документа.

На схеме состояний имена блоков (Chunk) указаны заглавными буквами, а имена параметров начинаются с заглавной буквы (например, блок COOKIE ECHO, параметр State Cookie). Если смену состояния могут вызывать различные события, эти события выделены метками (A), (B) и т. д.

Примечания:

1) Если параметр State Cookie в принятом блоке COOKIE ECHO некорректен (например, не прошла проверка целостности), получатель **должен** отбросить пакет без уведомления. Если получен State Cookie с истекшим сроком (см. параграф 5.1.5), получатель **должен** оправить в ответ блок ERROR. В обоих случаях получатель остаётся в состоянии CLOSED.
2) Если истекло время по таймеру T1-init, конечная точка **должна** повторить передачу INIT и заново запустить таймер T1-init без смены своего состояния. Эти действия **должны** повторяться до Max.Init.Retransmits раз после чего конечная точка **должна** прервать процесс инициирования ассоциации и возвратить сообщение об ошибке пользователю SCTP.

3) Если истекло время по таймеру T1-cookie, конечная точка **должна** повторить передачу COOKIE ECHO и заново запустить таймер T1-cookie без смены состояния. Эта процедура **должна** повторяться до Max.Init.Retransmits раз, после чего конечная точка должна прервать процесс инициирования ассоциации и возвратить сообщение об ошибке пользователю SCTP.

4) В состоянии SHUTDOWN-SENT конечная точка **должна** подтверждать все принятые блоки DATA без задержек.

5) В состоянии SHUTDOWN-RECEIVED для конечной точки **недопустимо** принимать любые новые запросы на передачу от пользователя SCTP.

6) В состоянии SHUTDOWN-RECEIVED конечная точка **должна** передать (возможно повторно) данные и выйти из этого состояния после того, как будут переданы все данные из очереди.

7) В состоянии SHUTDOWN-ACK-SENT для конечной точки **недопустимо** принимать от пользователя SCTP любые новые запросы на передачу.

Состояние CLOSED на схеме используется для того, чтобы показать, что ассоциация ещё не создана (т .е., не существует).

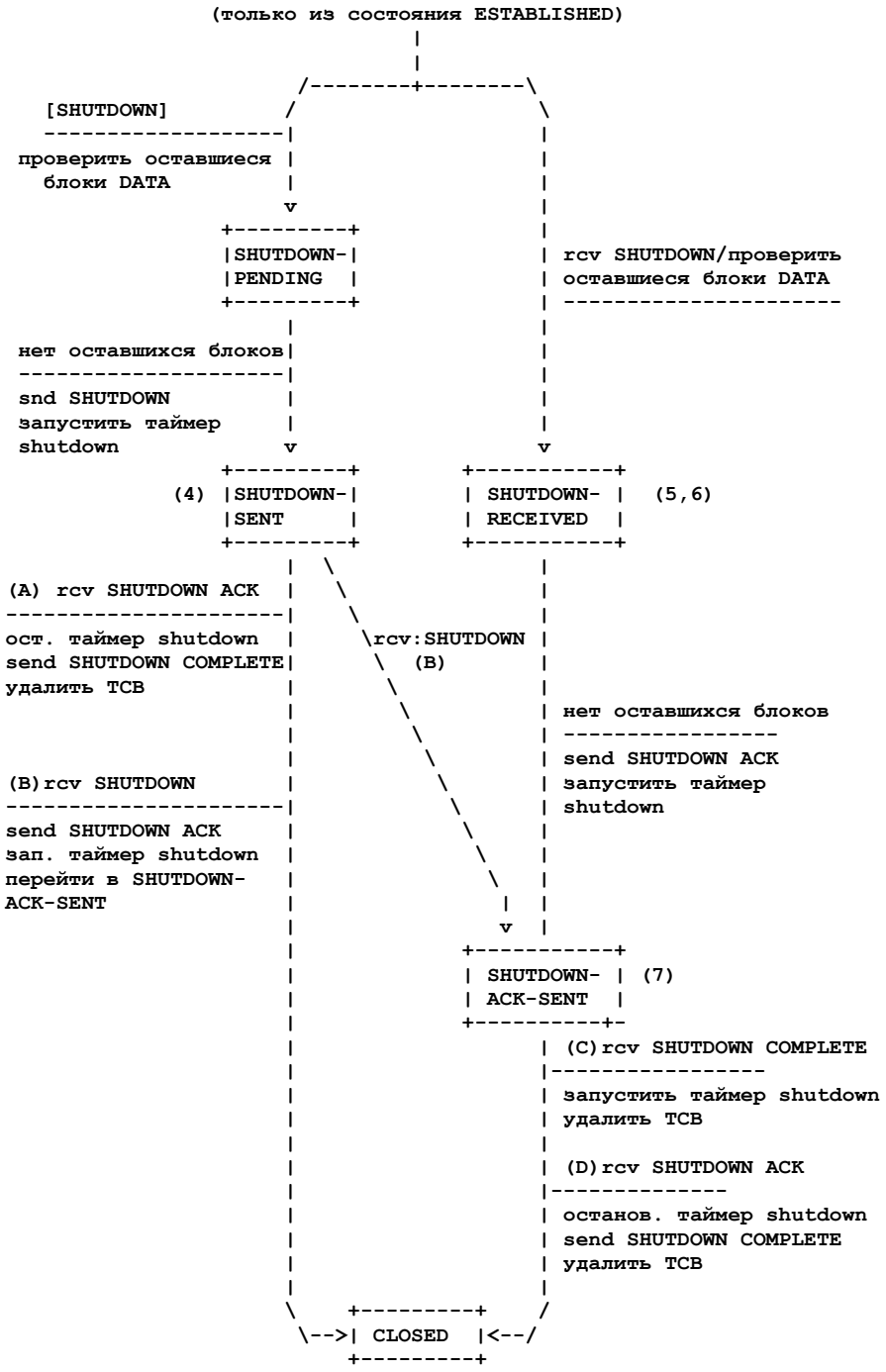



*Рисунок 3. Диаграмма состояний протокола SCTP (продолжение).*

# *5. Создание ассоциации*

До того, как сможет произойти передача первых данных от одной конечной точки SCTP ("A") к другой ("Z"), эти две точки должны выполнить полностью процесс создания ассоциации.

Пользователю SCTP в конечной точке следует применять примитив ASSOCIATE для создания ассоциации с другой конечной точкой SCTP.

Примечания для разработчиков: С точки зрения пользователя SCTP ассоциация может быть создана неявно без использования примитива ASSOCIATE (см. 10.1 B) просто путём передачи первых пользовательских данных удалённому адресату. Инициирующий ассоциацию узел SCTP будет задавать принятые по умолчанию значения для всех обязательных и дополнительных параметров INIT/INIT ACK.

После создания ассоциации организуются двунаправленные потоки данных для передачи информации в обе стороны (см. параграф 5.1.1).

## 5.1 Нормальное создание ассоциации

Процесс инициализации описан ниже в предположении, что конечная точка "A" пытается создать ассоциацию, а точка "Z" принимает вызов.

A) Точка "A" сначала передаёт блок INIT точке "Z". В блока INIT точка "A" должна указать свой Verification Tag (Tag_A) в поле Initiate Tag. Для Tag_A **следует** использовать случайное число в диапазоне от 1 до 4294967295 (см. рекомендации по выбору значения тега в параграфе 5.3.1). После передачи блока INIT точка "A" запускает таймер T1-init и переходит в состояние COOKIE-WAIT.

B) Точке "Z" следует незамедлительно ответить на запрос блоком INIT ACK. IP-адрес получателя в блоке INIT ACK **должен** совпадать с адресом отправителя блока INIT, для которого передаётся INIT ACK. Кроме заполнения других полей отклика точка "Z" должна скопировать в поле Verification Tag значение Tag_A, а также указать своё значение Verification Tag (Tag_Z) в поле Initiate Tag.
Кроме того, точка "Z" **должна** сгенерировать и передать с INIT ACK значение State Cookie[1] (см. параграф 5.1.3).

C) При получении блока INIT ACK от "Z" точке "A" следует остановить таймер T1-init и выйти из состояния COOKIE-WAIT. Далее точке "A" следует передать значение State Cookie из принятого блока INIT ACK в ответном блоке COOKIE ECHO[2], запустить таймер T1-cookie и перейти в состояние COOKIE-ECHOED.

D) При получении блока COOKIE ECHO точка "Z" будет передавать блок COOKIE ACK после создания TCB и перехода в состояние ESTABLISHED. Блок COOKIE ACK может объединяться с любыми ожидающими передачи блоками DATA и/или SACK, но блок COOKIE ACK **должен** быть первым в пакете.
Примечание для разработчиков: реализация протокола может передавать уведомление Communication Up пользователю SCTP при получении корректного блока COOKIE ECHO.

E) Приняв блок COOKIE ACK, точка "A" будет переходить из состояния COOKIE-ECHOED в состояние ESTABLISHED, останавливая таймер T1-cookie. Она может также уведомить ULP об успешном создании ассоциации с помощью Communication Up (см. раздел 10).

Блоки INIT и INIT ACK **недопустимо** группировать с другими блоками. Такой блок **должен** быть единственным в содержащем его пакете SCTP.

Конечная точка **должна** передавать блок INIT ACK по адресу IP, с которого был передан блок INIT.

Примечание: Для таймеров T1-init и T1-cookie должны выполняться правила, описанные в параграфе 6.3.

Если конечная точка, получившая блок INIT, INIT ACK или COOKIE ECHO, решает не создавать новую ассоциацию по причине отсутствия обязательных параметров в блоке INIT или INIT ACK, некорректных значений параметров или нехватки локальных ресурсов, эта точка **должна** передать в ответ блок ABORT. Этой точке также **следует** указать причину отказа (тип отсутствующих обязательных параметров и т. п.), включив соответствующий параметр в блок ABORT. Поле Verification Tag в общем заголовке исходящего пакета SCTP, содержащего блок ABORT, **должно** содержать значение Initiate Tag, полученное от партнёра.

После получения первого блока DATA в ассоциации конечная точка **должна** без промедления передать SACK для подтверждения приёма блока DATA. Последующие подтверждения передаются в соответствии с рекомендациями параграфа 6.2.

При создании TCB каждая точка **должна** установить для внутреннего параметра Cumulative TSN Ack Point значение переданного Initial TSN - 1.

Примечание для разработчиков: В качестве ключей поиска TCB для данного экземпляра SCTP обычно используются IP-адреса и номер порта SCTP.

### *5.1.1 Обработка параметров потока*

В блоках INIT и INIT ACK отправителю следует указывать число исходящих потоков (OS), которые он желает поддерживать для данной ассоциации, а также максимальное число входящих потоков (MIS), которые он будет принимать от удалённого партнёра.

После получения сведений о конфигурации потоков от другой стороны каждой конечной точке следует выполнить ряд проверок. Если значение полученное от партнёра. значение MIS < OS, это означает, что удалённый партнёр не сможет поддерживать все исходящие потоки и данная точка **должна** установить для числа исходящих потоков значение MIS или разорвать ассоциацию и сообщить вышележащему уровню о нехватке ресурсов на удалённой стороне.

После того, как ассоциация инициализирована, приемлемые значения идентификаторов исходящих потоков для неё могут находиться в диапазоне [0 - min(local OS, remote MIS)-1].

### *5.1.2 Обработка адресов*

В процессе создания ассоциации конечным точкам следует использовать перечисленные ниже правила для определения и сохранения транспортных адресов своего партнёра.

[1]Сразу после передачи INIT ACK с параметром State Cookie для точки "Z" **недопустимо** выделять какие-либо ресурсы или сохранять какие-либо состояния для новой ассоциации, поскольку это сделает точку "Z" уязвимой для атак на ресурсы системы.
[2]Блок COOKIE ECHO может группироваться с любыми ожидающими передачи блоками DATA, но эти блоки **должны** размещаться в пакете после блока COOKIE ECHO. До получения ответного блока COOKIE ACK **недопустима** передача каких-бы то ни было пакетов удаленному партнеру.

A) Если в принятом блоке INIT или INIT ACK нет адресных параметров, следует взять адрес отправителя из заголовка пакета IP, в котором был доставлен блок, и сохранить этот адрес вместе с номером порта SCTP, использованным отправителем пакета, как единственный транспортный адрес партнёра.

B) Если в полученном блоке INIT или INIT ACK присутствует параметр Host Name, имя следует преобразовать в адрес (список адресов) IP и сохранить транспортные адреса партнёра., как комбинации полученных при преобразовании адресов IP и номера порта отправителя SCTP. Конечная точка **должна** игнорировать все дополнительные параметры с IP-адресами, если они присутствуют в блоке INIT или INIT ACK.

Пока получатель блока INIT выполняет преобразование имени хоста в адрес, он подвержен потенциальному риску атак на SCTP. Если получатель INIT преобразует имя хоста в адрес при получении блока и используемый механизм преобразования может вносить достаточно большую задержку (например, запросы DNS), получатель может быть открыт для атаки на ресурсы в течение периода ожидания результатов преобразования имени до создания State Cookie и освобождения локальных ресурсов.

Следовательно, в тех случаях, когда преобразование имён может происходить достаточно долго, получатель INIT **должен** отложить процедуру преобразования до того, как будет получен блок COOKIE ECHO от партнёра. В таких случаях получателю INIT **следует** создать State Cookie с использованием полученного значения Host Name (вместо транспортного адреса получателя) и передать блок INIT ACK по адресу отправителя из заголовка IP в пакете, содержащем принятый блок INIT.

Получателю INIT ACK всегда следует незамедлительно предпринять попытку преобразования имени после получения блока.

Для получателя INIT или INIT ACK **недопустима** передача пользовательских данных до преобразования имени хоста в адрес.

Если не удаётся преобразовать имя, конечная точка **должна** незамедлительно передать своему партнёру блок ABORT с причиной ошибки Unresolvable Address. Блок ABORT следует отправлять по адресу IP из заголовка пакета, в котором был получен последний пакет от партнёра.

C) Если в принятом блоке INIT или INIT ACK присутствуют только адреса IPv4/IPv6, получателю следует выделить и записать все транспортные адреса из полученного блока и адреса отправителя в заголовке пакета IP, содержащего блок INIT или INIT ACK. Транспортные адреса представляют собой комбинацию номера порта SCTP (из общего заголовка) и адресов IP, переданных в блоке INIT или INIT ACK и заголовке пакета IP, доставившего блок. Получателю следует использовать только эти транспортные адреса для передачи последующих пакетов удалённому партнёру.

Примечания для разработчиков: В некоторых случаях (например, когда реализация не контролирует IP-адреса отправителя, используемые при передаче) конечной точке может потребоваться включение в блок INIT или INIT ACK всех адресов IP, которые могут использоваться для передачи.

После выделения транспортного адреса из блока INIT или INIT ACK с использованием описанных выше правил конечной точке следует выбрать один из таких адресов для организации первичного пути.

Примечание: Блок INIT-ACK **должен** передаваться по адресу отправителя блока INIT.

Отправитель блока INIT может включить в этот блок параметр Supported Address Types, показывающий поддерживаемые типы адресов. При наличии такого параметра получатель блока INIT **должен** использовать один из указанных этим параметром типов адресов при передаче отклика на INIT или разорвать ассоциацию с помощью блока ABORT с причиной ошибки Unresolvable Address, если конечная точка не желает или не может использовать ни один из типов адресов, указанных партнёром.

Примечания для разработчиков: В тех случаях, когда получателю блока INIT ACK не удаётся выполнить преобразование адреса вследствие отсутствия поддержки указанного типа, попытка создания ассоциации может быть прервана, после чего предпринимается попытка повторной организации с использованием параметра Supported Address Types в новом блоке INIT для индикации предпочтительных типов адресов.

### *5.1.3 Генерация State Cookie*

При передаче блока INIT ACK в ответ на INIT отправитель INIT ACK создаёт значение State Cookie и передаёт его в одноимённом параметре блока INIT ACK. В параметр State Cookie отправителю следует включить MAC (см., например, [RFC2104]), временную метку генерации State Cookie и время жизни параметра State Cookie, а также другую информацию, требуемую для создания ассоциации.

При генерации State Cookie следует выполнить перечисленные ниже операции:

1) Создать для ассоциации TCB с использованием информации из полученного блока INIT и передаваемого блока INIT ACK.

2) В TCB установить время создания по текущему времени суток, а для времени жизни использовать протокольный параметр 'Valid.Cookie.Life'.

3) Из TCB выделить и сохранить минимальный набор информации, требуемой для повторного создания TCB, и создать MAC с использованием этого набора и секретного ключа (см. пример генерации MAC в [RFC2104]).

4) Создать State Cookie, объединив минимальный набор информации и полученное значение MAC.

После передачи блока INIT ACK с параметром State Cookie отправителю **следует** удалить TCB и все прочие локальные ресурсы, связанные с новой ассоциацией в целях предотвращения атак на ресурсы.

Метод хеширования, используемый для генерации MAC является приватным для получателя блока INIT. Использование MAC является обязательным с целью предотвращения атак на службы. В качестве секретного ключа **следует** использовать случайное значение (см. рекомендации [RFC1750]), которое **следует** менять достаточно часто.

Для идентификации ключа, который будет использоваться для проверки MAC, **может** служить временная метка момента создания State Cookie.

Для обеспечения взаимодействия реализациям протокола следует минимизировать размер cookie.

### *5.1.4 Обработка State Cookie*

Когда конечная точка (в состоянии COOKIE WAIT) получает блок INIT ACK с параметром State Cookie, она **должна** незамедлительно передать своему партнёру блок COOKIE ECHO с полученным значением State Cookie. Отправитель **может** также добавить в пакет ожидающие обработки блоки DATA после блока COOKIE ECHO.

Конечной точке следует также запустить таймер T1-cookie после передачи блока COOKIE ECHO. По истечении заданного для таймера периода конечной точке следует повторить передачу блока COOKIE ECHO и заново включить таймер T1-cookie. Эту процедуру следует повторять до получения блока COOKIE ACK или исчерпания числа попыток 'Max.Init.Retransmits'. Если заданное число попыток не привело к успеху, партнёр помечается как недоступный и ассоциация переводится в положение CLOSED.

### *5.1.5 Аутентификация State Cookie*

Когда конечная точка получает блок COOKIE ECHO от другой конечной точки, с которой нет действующей ассоциации, следует выполнить перечисленные ниже операции:

1) Рассчитать MAC с использованием данных TCB, полученных в State Cookie, и секретного ключа (отметим, что для выбора секретного ключа может использоваться временная метка из State Cookie). Рекомендации по расчёту MAC приведены в [RFC2104].

2) Проверить State Cookie, сравнивая рассчитанное значение MAC с полученным в State Cookie. При наличии расхождений пакет SCTP, включающий COOKIE ECHO и любые блоки DATA, следует отбросить без уведомления отправителя.

3) Сравнить временную метку в State Cookie с текущим временем локальной системы. Если разница превышает заданный срок существования State Cookie, пакет, содержащий COOKIE ECHO и любые блоки DATA, **следует** отбросить. Конечная точка в таком случае **должна** передать партнёру блок ERROR с причиной ошибки Stale Cookie.

4) Если значение State Cookie корректно, создаётся ассоциация с отправителем COOKIE ECHO, создаётся TCB с использованием данных из блока COOKIE ECHO и конечная точка переходит в состояние ESTABLISHED.

5) Передать блок COOKIE ACK удалённому партнёру для подтверждения приёма COOKIE ECHO. Блок COOKIE ACK **может** группироваться с пользовательскими блоками DATA или блоком SACK, однако блок COOKIE ACK **должен** размещаться в пакете SCTP первым.

6) Незамедлительно подтвердить получение любых блоков DATA, сгруппированных с COOKIE ECHO, путём передачи блока SACK (подтверждения последовательных блоков DATA передаются с использованием правил, рассмотренных в параграфе 6.2). Как было отмечено в п. 5), блок SACK, группируемый с COOKIE ACK, **должен** размещаться в пакете SCTP после блока COOKIE ACK.

При получении блока COOKIE ECHO от конечной точки, с которой получатель имеет работающую ассоциацию, выполняются операции, рассмотренные в параграфе 5.2.

### *5.1.6 Пример нормального создания ассоциации*

В показанном на рисунке примере точка "A" инициирует создание ассоциации и передаёт пользовательское сообщение точке "Z", а "Z", в свою очередь, передаёт точке "A" два пользовательских сообщения (предполагается отсутствие группировки и фрагментирования).

Если закончилось время T1-init в точке "A" после передачи блока INIT или COOKIE ECHO, повторяется передача блока INIT или COOKIE ECHO с тем же значением Initiate Tag (т. е., Tag_A) или State Cookie и таймер запускается снова. Эта процедура повторяется до Max.Init.Retransmits раз после чего точка "A" принимает решение о недоступности "Z" и сообщает вышележащему протоколу об ошибке (ассоциация переводится в состояние CLOSED).

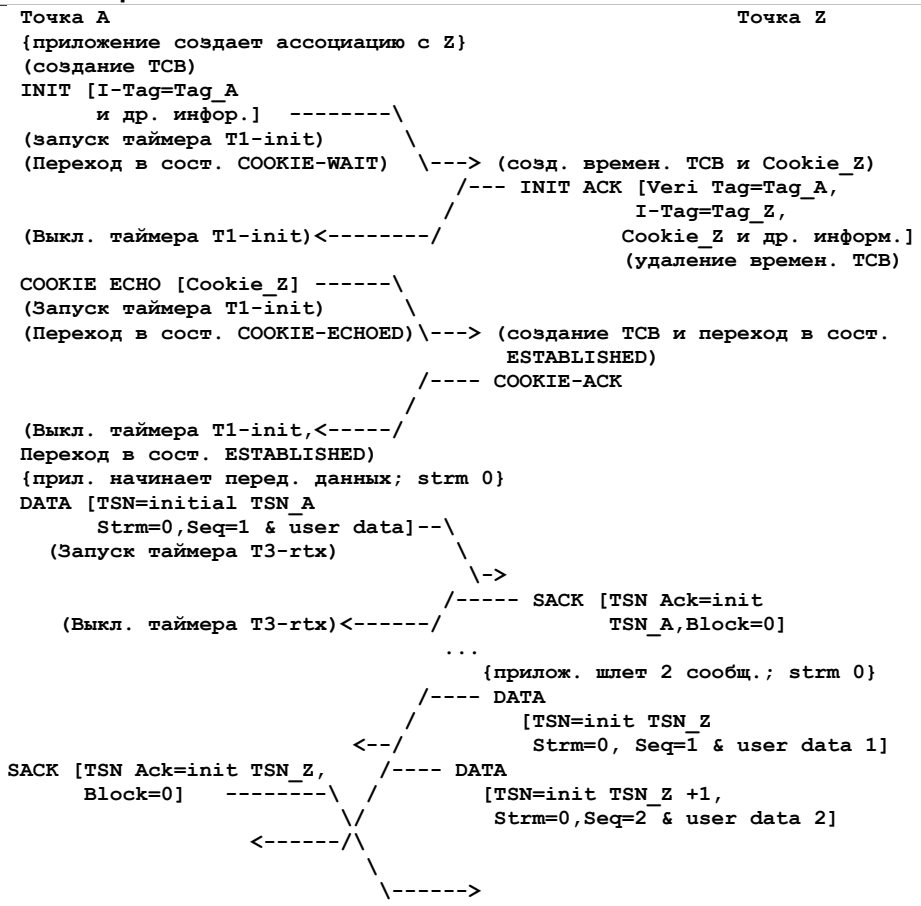



*Рисунок 4. Пример создания ассоциации.*

При повторной передаче INIT конечная точка **должна** следовать правилам, описанным в параграфе 6.3, для определения подходящего значения таймера.

## 5.2 Обработка дубликатов и неожиданных блоков INIT, INIT ACK, COOKIE ECHO, COOKIE ACK

В течение срока жизни ассоциации (в одном из возможных состояний) конечная точка может получить от своего партнёра. один из установочных блоков (INIT, INIT ACK, COOKIE ECHO, COOKIE ACK)[1]. Получателю следует трактовать такие установочные блоки как дубликаты и обрабатывать их в соответствии с приведёнными в этом параграфе рекомендациями.

Появление дубликатов и неожиданных установочных блоков может быть вызвано любой из перечисленных ниже причин.

A) Критическая ошибка на удалённой стороне с перезапуском партнёра. и передачей нового блока INIT для восстановления ассоциации.

B) Обе стороны предприняли одновременные попытки создания ассоциации.

C) Блок был получен в старом пакете, который использовался для предыдущей ассоциации или ассоциации, которой уже нет.

D) Блок является фальшивым (атака).

E) Партнёр не получил блок COOKIE ACK и повторно передаёт COOKIE ECHO.

При получении таких блоков следует применять описанные ниже правила, позволяющие идентифицировать и корректно обрабатывать подобные ситуации.

### *5.2.1 Блок INIT получен в состоянии COOKIE-WAIT или COOKIE-ECHOED (п. B)*

Такие ситуации обычно возникают в результате конфликта при создании ассоциации, когда обе стороны начинают процесс создания одновременно.

При получении блока INIT в состоянии COOKIE-WAIT или COOKIE-ECHOED конечная точка **должна** ответить блоком INIT ACK, используя те же параметры, которые были переданы в исходном блоке INIT (включая параметр Initiate Tag). Сохранённые параметры при передаче отклика комбинируются с параметрами из недавно полученного блока INIT. Конечной точке следует также генерировать параметр State Cookie для блока INIT ACK, используя параметры, переданные ею в блоке INIT.

После этого для конечной точки **недопустимо** изменение своего состояния и удаление соответствующего TCB, таймеру T1-init следует продолжать отсчёт. Обычная процедура обработки State Cookies при наличии TCB позволит избавиться от дубликатов INIT в одной ассоциации.

[1]Конечная точка не получит блок, пока тот не будет передан по транспортному адресу SCTP и далее локально от транспортного адреса SCTP, связанного с данной точкой. Следовательно, обрабатывающая такие блоки конечная точка является элементом текущей ассоциации.

Конечная точка, находящаяся в состоянии COOKIE-ECHOED при получении блока INIT **должна** заполнить поля Tie-Tags информацией из параметра Tag (своей или партнёра., как описано в параграфе 5.2.2).

### *5.2.2 Неожиданный блок INIT в состоянии, отличном от CLOSED, COOKIE-ECHOED, COOKIE-WAIT и SHUTDOWN-ACK-SENT*

Если явно не указано иное, при получении блока INIT в таких случаях конечная точка должна генерировать блок INIT ACK с параметром State Cookie. В передаваемый блок INIT ACK **должны** быть скопированы текущее значение Verification Tag данной точки и Verification Tag с использованием зарезервированного пространства в State Cookie. Положение этих параметров следует указать в параметрах Peer-Tie-Tag и Local-Tie-Tag. В исходящем пакете SCTP с блоком INIT ACK **должно** содержаться значение Verification Tag, совпадающее с параметром Initiation Tag в неожиданно полученном блоке INIT. Блок INIT ACK **должен** содержать новое значение Initiation Tag (случайное число, созданное в соответствии с 5.3.1). Остальные параметры для конечной точки (например, число исходящих потоков) **следует** скопировать из существующих параметров ассоциации в блок INIT ACK и cookie.

После передачи INIT ACK конечной точке не следует предпринимать каких-либо действий, т. е., существующую ассоциацию, включая текущее состояние и соответствующее значение TCB изменять **недопустимо**.

Примечание: Значения Tie-Tags заполняются только в том случае, когда существует TCB и ассоциация не находится в состоянии COOKIE-WAIT. При нормальном создании ассоциации (конечная точка находится в состоянии COOKIE-WAIT) параметр Tie-Tags **должен** быть установлен в 0 (это показывает отсутствие предыдущего TCB). INIT ACK и State Cookie заполняются в соответствии с рекомендациями параграфа 5.2.1.

### *5.2.3 Неожиданный блок INIT ACK*

При получении блока INIT ACK конечной точкой, находящейся в отличном от COOKIE-WAIT состоянии, этой точке следует отбросить блок INIT ACK. Неожиданные блоки INIT ACK обычно связаны с обработкой старых или дублированных блоков INIT.

### *5.2.4 Обработка COOKIE ECHO при наличии TCB*

При получении блока COOKIE ECHO конечной точкой, находящейся в любом состоянии для существующей ассоциации (состояние отлично от CLOSED), нужно следовать приведённым ниже правилам.

1) Рассчитать MAC в соответствии с рекомендациями п. 1 в параграфе 5.1.5,
2) Проверить State Cookie как описано в п. 2 параграфа 5.1.5 (случай C или D в начале параграфа 5.2).
3) Сравнить временную метку State Cookie с текущим временем. Если срок жизни State Cookie истёк и значение Verification Tag, содержащееся в State Cookie, не соответствует Verification Tag для текущей ассоциации, пакет вместе с входящими в него блоками COOKIE ECHO и DATA следует отбросить. Конечная точка **должна** также передать блок ERROR с причиной ошибки "Stale Cookie" своему партнёру (случай C или D в параграфе 5.2).
   Если параметры Verification Tag в State Cookie и текущей ассоциации совпадают, следует считать параметр State Cookie корректным (случай E в параграфе 5.2) даже по истечении срока жизни.
4) При корректном значении State Cookie распаковать TCB во временный TCB.
5) Выполнить подходящее действие из таблицы 2.

*Таблица 2. Обработка COOKIE ECHO при наличии TCB.*

| *Local Tag* | *Peer's Tag* | *Local-Tie-Tag* | *Peer's-Tie-Tag* | *Действие* |
|---|---|---|---|---|
| X | X | M | M | (A) |
| M | X | A | A | (B) |
| M | 0 | A | A | (B) |
| X | M | 0 | 0 | (C) |
| M | M | A | A | (D) |

X - тег не соответствует существующему TCB

M - тег соответствует существующему TCB

0 - нет Tie-Tag в Cookie (неизвестно).

A - Все случаи (M, X, 0).

Для всех ситуаций, не рассмотренных в таблице 2, cookie следует отбрасывать без уведомления.

Действия

A) Этот случай может быть связан с рестартом на удалённой стороне. Когда конечная точка распознает возможный “рестарт”, существующая сессия трактуется, как случай получения блока ABORT, за которым сразу же следовал новый блок COOKIE ECHO, с перечисленными ниже исключениями:

   - любые блоки DATA **могут** быть сохранены (в зависимости от реализации протокола);
   - протоколу вышележащего уровня **следует** передать уведомление RESTART взамен “COMMUNICATION LOST”.

   Все параметры контроля насыщения (например, cwnd, ssthresh), связанные с этим партнёром., **должны** быть сброшены в исходное состояние (см. параграф 6.2.1).

   После этого конечной точке следует перейти в состояние ESTABLISHED.

   Если конечная точка находится в состоянии SHUTDOWN-ACK-SENT и определяет рестарт партнёра. (п. A), для этой точки **недопустимо** создание новой ассоциации и следует передать своему партнёру блок SHUTDOWN ACK и ERROR с причиной ошибки "Cookie Received while Shutting Down".

B) В этой ситуации обе стороны могут пытаться организовать ассоциацию одновременно, но удалённая точка передаст свой блок INIT уже после отклика на INIT от локальной точки. В результате новое значение Verification Tag не будет включать информацию из тега, переданного ранее той же конечной точкой. Конечной точке следует сохранить состояние ESTABLISHED или перейти в него, но она **должна** обновить значение Verification Tag из параметра State Cookie, остановить запущенные таймеры init и cookie и передать блок COOKIE ACK.

C) В этом случае информация cookie локальной точки поступает с опозданием. До этого локальная точка передала блок INIT и приняла INIT-ACK, а также передала блок COOKIE ECHO с тегом партнёра., а новый тег принадлежит локальной точке. Данные cookie следует отбросить без уведомления. Конечной точке **не следует** менять своё состояние и отключать запущенные таймеры.

D) Когда теги локальной и удалённой точки совпадают, конечной точке следует переходить в состояние ESTABLISHED, если она ещё не находится в нем. Следует также остановить таймеры init и cookie и передать блок COOKIE ACK.

Примечание: Verification Tag партнёра. - это тег, полученный в поле Initiate Tag блока INIT или INIT ACK.

#### 5.2.4.1 Пример перезапуска ассоциации

В приведённом на рисунке 5 примере точка "A" инициирует ассоциацию после рестарта. Точка "Z" пока не знает о рестарте (т. е., Heartbeat ещё не удалось обнаружить недоступность точки "A"). Предполагается отсутствие группировки и фрагментирования.

### *5.2.5 Обработка дубликатов COOKIE-ACK*

Во всех состояниях, кроме COOKIE-ECHOED, конечной точке следует отбрасывать без уведомления блоки COOKIE ACK.

### *5.2.6 Обработка ошибок Stale COOKIE*

Приём блока ERROR с причиной Stale Cookie может быть обусловлен несколькими причинами.

A) Создание ассоциации завершилось неудачей до того, как была выполнена обработка State Cookie.

B) После создания ассоциации обработана старая переменная State Cookie.

C) Получена старая переменная State Cookie от кого-то, с кем получатель не намерен создавать ассоциацию, но блок ABORT был утерян.

При обработке блока ERROR с причиной ошибки Stale Cookie конечной точке следует сначала проверить завершился ли процесс создания ассоциации (состояние отличается от COOKIE-ECHOED). Если ассоциация не находится в состоянии COOKIE-ECHOED, блок ERROR следует отбросить без уведомления.

Если ассоциация находится в состоянии COOKIE-ECHOED, конечная точка может выбрать один из перечисленных

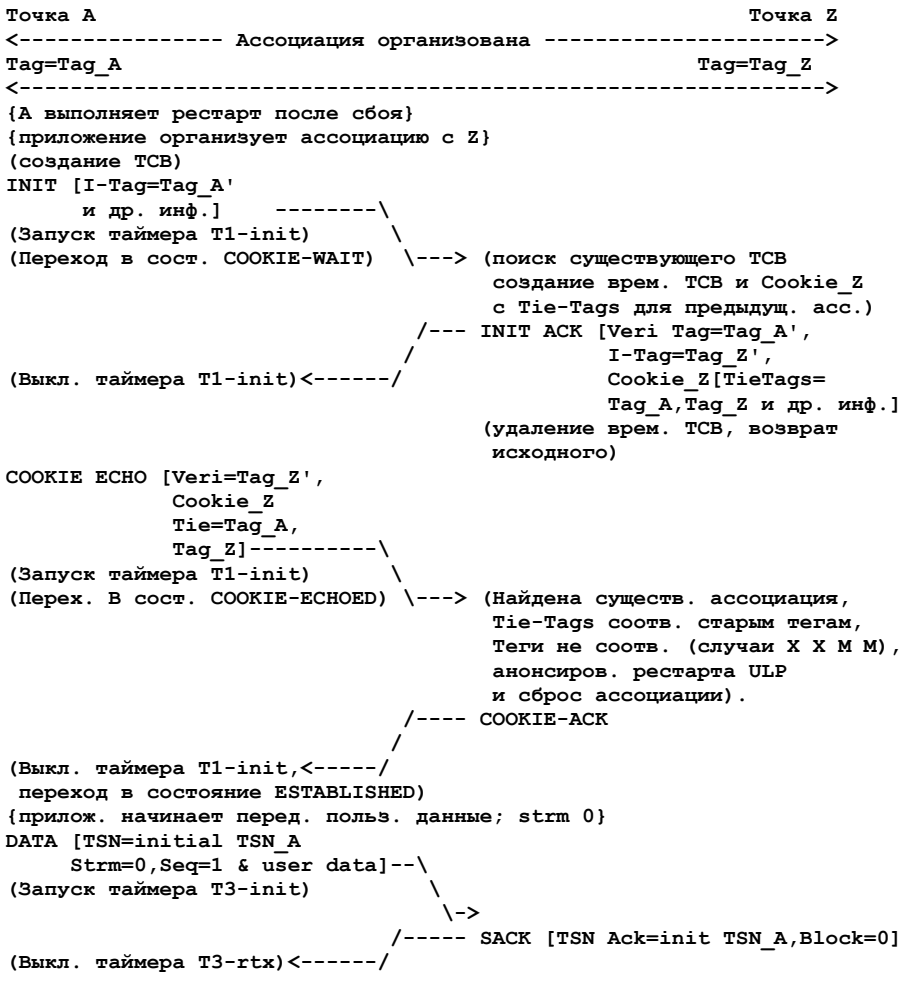



*Рисунок 5. Пример перезапуска.*

вариантов.

1) Передать новый блок INIT удалённой конечной точке для генерации нового значения State Cookie и повторить процедуру создания ассоциации.

2) Отбросить TCB и сообщить вышележащему уровню о невозможности создания ассоциации.

3) Передать новый блок INIT удалённой конечной точке, добавив в него параметр Cookie Preservative, запрашивающий продление срока жизни State Cookie. При расчёте дополнительного времени реализации протокола следует использовать данные RTT, полученные во время предыдущего обмена блоками COOKIE ECHO/ERROR. К полученному значению RTT следует добавить не более 1 секунды, поскольку увеличение срока жизни State Cookie подвергает конечную точку риску replay-атак.

## 5.3 Другие вопросы инициализации

### *5.3.1 Выбор значений тегов*

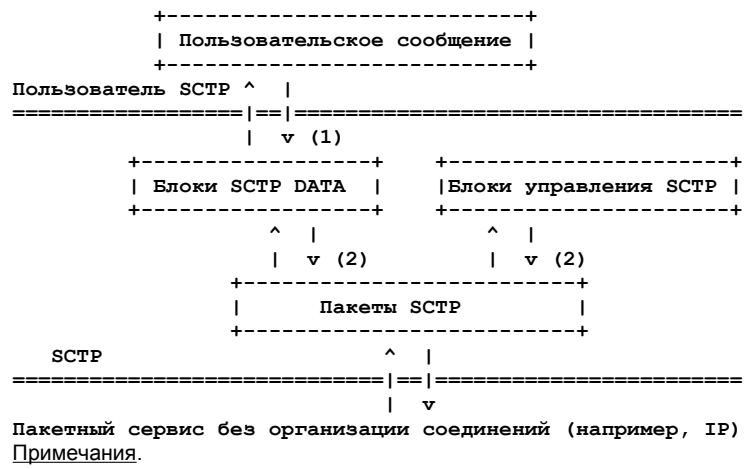
Значения Initiate Tag следует выбирать из диапазона 1 - $2^{32}$-1. Важно, чтобы значение Initiate Tag было случайным - это поможет в защите от атак man in the middle[1] и sequence number[2]. Для создания случайных значений Initiate Tag можно использовать методы, описанные в [RFC1750]. Аккуратный подбор Initiate Tag позволяет также избавиться от появления дубликатов из предыдущих ассоциаций, которые могут быть ошибочно направлены в текущую ассоциацию.

Важно помнить, что значение Verification Tag, используемое любой из конечных точек данной ассоциации, **недопустимо** изменять в течение срока существования ассоциации. Каждый раз, когда конечная точка перезапускается и заново создаёт ассоциацию с тем же партнёром., **должно** использоваться новое значение Verification Tag.

# *6. Передача пользовательских данных*

Данные **должны** передаваться только в состояниях ESTABLISHED, SHUTDOWN-PENDING и SHUTDOWN-RECEIVED. Единственным исключением из этого правила является возможность включения блоков DATA в пакеты, содержащие блок COOKIE ECHO, в состоянии COOKIE-WAIT.

Блоки DATA **должны** приниматься только в соответствии с приведёнными ниже правилами в состояниях ESTABLISHED, SHUTDOWN-PENDING, SHUTDOWN-SENT. Блок DATA, полученный в состоянии CLOSED, **следует** обрабатывать в соответствии со специальными правилами, описанными в параграфе 8.4. Блоки DATA, полученные во всех прочих состояниях, **следует** отбрасывать.




Примечания.

1) При преобразовании пользовательских сообщений в блоки DATA конечная точка будет фрагментировать сообщения, размер которых превышает значение path MTU, в несколько блоков DATA. Получатель будет собирать принятые фрагменты из блоков DATA до передачи сообщения пользователю (см. параграф 6.9).

2) Множество блоков DATA и блоков управления может группироваться отправителем в один пакет SCTP, пока размер результирующего пакета не будет превышать значение path MTU. Получатель будет разбирать групповой пакет, выделяя из него отдельные блоки. Блоки управления **должны** размещаться в пакете перед блоками DATA.

*Рисунок 6. Пример передачи пользовательских данных.*

Подтверждения SACK **должны** обрабатываться в состояниях ESTABLISHED, SHUTDOWN-PENDING и SHUTDOWN-RECEIVED. Входящий блок SACK **может** быть обработан ассоциацией в состоянии COOKIE-ECHOED. Подтверждения SACK в состоянии CLOSED **следует** обрабатывать в соответствии со специальными правилами, рассмотренными в параграфе 8.4. Во всех прочих состояниях блоки SACK **следует** отбрасывать.

Получатель SCTP **должен** быть способен принимать пакеты SCTP размером как минимум 1500 байтов. Это означает, что для конечной точки SCTP **недопустимо** указывать значение меньше 1500 в стартовом параметре a_rwnd, передаваемом в INIT или INIT ACK.

Для эффективной передачи трафика протокол SCTP поддерживает механизм группировки мелких пользовательских сообщений и фрагментации больших сообщений. На рисунке 6 показана схема обмена пользовательскими сообщениями в SCTP.

В этом параграфе термин «отправитель» (data sender) будет указывать конечную точку, которая передаёт блок DATA, а термин «получатель» (data receiver) - конечную точку, принимающую блок DATA. Получатель подтверждает приём данных блоком SACK.

[1] "Человек в центре атаки" - тип атак, в которых используются интеллектуальные средства перехвата и подмены пакетов. *Прим. перев.*
[2] Атаки с подменой порядковых номеров.

Механизмы фрагментации и группировки, описанные в параграфах 6.9 и 6.10, являются **необязательными** для отправителя, но они **должны** быть реализованы на приёмной. стороне (т. е., конечная точка **должна** принимать и обрабатывать фрагментированные и сгруппированные данные).

## 6.1 Передача блоков DATA

В этом документе предполагается использование одного таймера повторной передачи на транспортный адрес получателя, но реализации протокола **могут** поддерживать отдельный таймер повтора для каждого блока DATA.

Ниже приведены правила общего назначения, используемые отправителем для передачи или повтора исходящего блока DATA.

A) В любой момент для отправителя **недопустимо** передавать новые данные по любому транспортному адресу получателя, если значение rwnd, полученное от партнёра., говорит об отсутствии свободного пространства в буфере (т. е., rwnd = 0, см. параграф 6.2.1). Однако, независимо от значения rwnd (включая 0), отправитель может всегда держать один блок DATA пути к получателю, если позволяет cwnd (см. правило B). Это позволяет отправителю проверять изменение rwnd, о котором отправитель не знает по причине утери подтверждения SACK в процессе доставки от получателя.

B) В любой момент времени для отправителя **недопустимо** передавать информацию по данному транспортному адресу, если имеется cwnd или более неподтвержденных байтов данных для этого адреса.

C) Когда приходит время отправителю передавать данные, перед отправкой новых блоков DATA он **должен** сначала передать неподтвержденные блоки DATA, которые помечены для повтора (не более cwnd).

D) Отправитель может передать столько новых блоков DATA, сколько позволяют правила A и B.

Множество блоков DATA, подготовленных для передачи, **можно** сгруппировать в один пакет. Более того, передаваемые повторно блоки DATA **можно** группировать с новыми блоками DATA, пока размер результирующего пакета не превышает path MTU. Протокол вышележащего уровня (ULP) может запросить передачу сообщений без группировки, но это означает лишь отключение задержек, которые реализация SCTP может использовать для повышения эффективности группировки. Группировка может происходить и в этом случае (например, при перегрузках или повторе передачи).

До того, как конечная точка передаст блок DATA, отправителю следует создать блок SACK для всех неподтвержденных данных и сгруппировать его с передаваемыми блоками DATA, пока размер результирующего пакета не превышает значение MTU (см. параграф 6.2).

Примечание для разработчиков. При заполнении окна (передача запрещена правилом A и/или B), отправитель **может** по-прежнему принимать запросы на передачу от вышележащего протокола, но он **должен** остановить передачу блоков DATA, пока некоторые или все остающиеся блоки DATA не будут подтверждены и правила A и B не будут выполнены.

Когда передача или повтор передачи происходит по любому из адресов и таймер T3-rtx для этого адреса не включён, отправитель **должен** запустить этот таймер. Если таймер для данного адреса уже включён, отправитель **должен** сбросить и запустить его заново, если по этому адресу происходит повтор передачи передачи остающихся в очереди (т. е., с меньшим значением TSN) блоков данных. В остальных случаях перезапуск таймера **недопустим**.

При старте или перезапуске таймера T3-rtx его значение должно быть установлено в соответствии с правилами, приведёнными в параграфах 6.3.2 и 6.3.3.

Примечание: Отправителю **не следует** использовать значения TSN, которые превышают стартовое значение TSN для текущего окна более, чем на $2^{31} - 1$.

## 6.2 Подтверждение приёма блоков DATA

Конечная точка SCTP всегда **должна** подтверждать получение каждого корректного блока DATA.

При передаче подтверждений **следует** придерживаться рекомендаций по использованию алгоритма отложенных подтверждений, описанного в параграфе 4.2 [RFC2581]. В частности, подтверждения **следует** генерировать по крайней мере по факту доставки каждого второго пакета (не обязательно с блоком DATA), а также **следует** в течение 200 мсек генерировать подтверждение для любого ещё не подтверждённого блока DATA. В некоторых случаях для отправителя SCTP разумно быть более консервативным, нежели предлагает описанный в данном документе алгоритм подтверждения. Однако для отправителей SCTP **недопустимо** быть более агрессивным, нежели предлагает описанный ниже алгоритм.

Для отправителя SCTP **недопустимо** генерировать более 1 блока SACK для каждого входящего пакета, который не является обновлением предложенного размера окна при запросе приёмной. стороной новых данных.

Примечание для разработчиков. Максимальная задержка при генерации подтверждений может задаваться администратором SCTP статически или динамически в соответствии с реальными требованиями протоколов вышележащих уровней.

Для реализации протокола **недопустимо** позволять установку задержки сверх 500 мсек. Иными словами, реализация **может** устанавливать задержки менее 500 мсек, но задержка ни в коем случае **не должна** превышать 500 мсек.

Подтверждения **должны** передаваться в блоках SACK до тех пор, пока со стороны ULP не поступит запроса на завершение ассоциации (в последнем случае подтверждение **может** быть передано в блоке SHUTDOWN). Блок SACK может использоваться для подтверждения доставки нескольких блоков DATA. Формат блоков SACK описан в параграфе 3.3.4. В частности, конечная точка SCTP **должна** заполнить поле Cumulative TSN Ack, чтобы показать последний номер TSN для полученных без пропусков корректных блоков DATA. Все принятые блоки DATA с номерами TSN, превышающими значение Cumulative TSN Ack, **следует** указывать в полях Gap Ack Block.

Примечание. Блок SHUTDOWN не включает поля Gap Ack Block. Поэтому конечной точке **следует** использовать SACK, а не SHUTDOWN для подтверждения доставки блоков DATA, принятых с нарушением порядка.

При получении пакета с дубликатом блока(ов) DATA, в котором нет нового блока(ов) DATA, конечная точка **должна** незамедлительно передать SACK. Если пакет с дубликатом(ами) DATA содержит также новый блоки(и) DATA, конечная точка **может** передать SACK незамедлительно. Обычно получение дублирующих блоков DATA связано с потерей исходного блока SACK и тайм-аутом RTO у партнёра. Номера TSN для дубликатов **следует** указывать в блоке SACK как дубликаты.

При получении блока SACK конечная точка **может** использовать информацию Duplicate TSN для обнаружения потери блоков SACK. Другие варианты использования этой информации требуют дальнейшего изучения.

Получатель данных отвечает за поддержку буферов приёма. Получателю **следует** своевременно уведомлять отправителя об изменении своих возможностей приёма данных. Способы управления приёмными буферами в реализации протокола зависят от множества факторов (например, от операционной системы, системы управления памятью, размера ОЗУ и т. п.). Однако описанная в параграфе 6.2.1 стратегия передачи основана на предположении, что получатель использует описанный ниже алгоритм.

A) При создании ассоциации конечная точка сообщает партнёру размер своего приёмного буфера для данной ассоциации в блоке INIT или INIT ACK. Переданное значение размера буфера устанавливается для переменной a_rwnd.

B) При получении и буферизации блоков DATA значение a_rwnd уменьшается на размер принятых и помещённых в буфер данных. Это по сути закрывает окно rwnd у отравителя и ограничивает количество данных, которые тот может передать.

C) Когда блоки DATA передаются ULP и буфер освобождается, значение a_rwnd увеличивается на размер данных, которые отправлены протоколу вышележащего уровня. Таким образом окно rwnd открывается снова, позволяя отправителю передавать новые данные. Получателю **не следует** увеличивать значение a_rwnd, пока данные не будут освобождены из буфера. Например, если получатель удерживает фрагментированные блоки DATA в очереди на сборку, ему не следует увеличивать значение a_rwnd.

D) При передаче блока SACK получателю данных **следует** поместить текущее значение переменной a_rwnd в поле a_rwnd передаваемого блока. Получателю данных **следует** принимать во внимание, что отправитель не будет повторять передачу блоков DATA, указанных в Cumulative TSN Ack (т. е., удалит их из очереди на повтор).

Возможны ситуации, когда получателю потребуется отбросить блоки DATA, которые были приняты, но ещё не освобождены из приёмного буфера (т. е., не переданы ULP). Такие блоки DATA могут быть подтверждены в Gap Ack Block. Например, получатель может удерживать принятые данные в буфере для сборки фрагментов пользовательского сообщения, когда обнаружится нехватка буферного пространства. Получатель в этом случае может отбросить блоки DATA из буфера, несмотря на то, что они уже подтверждены в Gap Ack Block. Если получатель отбрасывает блоки DATA, их **недопустимо** включать в Gap Ack Block последующих блоков SACK, пока отброшенные блоки не будут получены снова в повторных пакетах. В дополнение к сказанному конечной точке следует принимать во внимание отброшенные блоки при расчёте значения a_rwnd.

Конечной точке **не следует** отзывать SACK и отбрасывать данные. Описанной процедурой следует пользоваться только в экстремальных ситуациях (например, при нехватке памяти). Получателю данных следует принимать во внимание, что отбрасывание данных, которые были подтверждены в Gap Ack Block, может привести к неоптимальной работе отправителя данных и снижению производительности.

Приведённый на рисунке 7 пример иллюстрирует использование отложенных подтверждений.

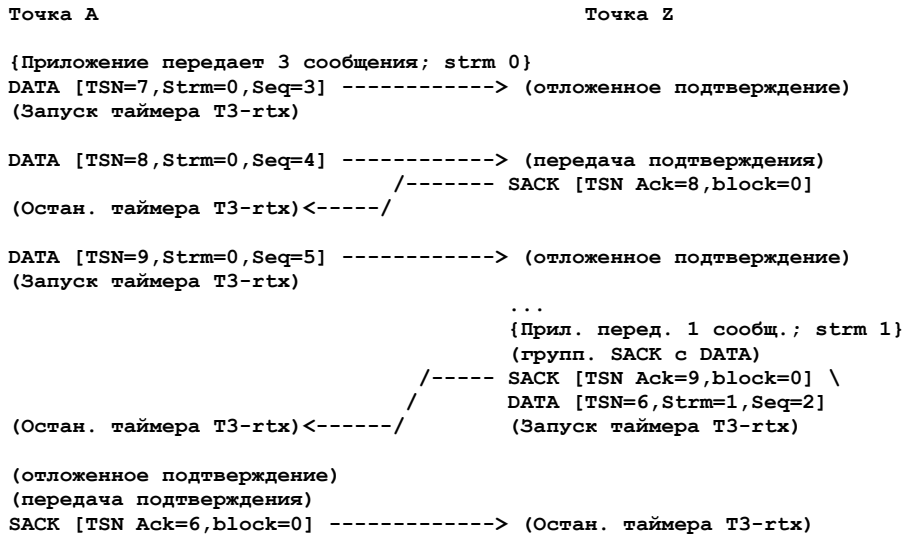



*Рисунок 7. Пример подтверждения с задержкой.*

Если конечная точка получает блок DATA без пользовательских данных (Length = 16), она **должна** передать блок ABORT с причиной ошибки No User Data.

Конечной точке **не следует** передавать блоков DATA без пользовательских данных.

## *6.2.1 Обработка подтверждений SACK*

Каждый блок SACK, полученный конечной точкой, содержит значение a_rwnd. Это значение представляет объем свободного буферного пространства на приёмной. стороне в момент передачи блока SACK, которое осталось от приёмного буфера, указанного в блоке INIT/INIT ACK. Используя значения a_rwnd, Cumulative TSN Ack и Gap Ack Block, отправитель может создать своё представление о буферном пространстве партнёра.

Одна из проблем, возникающих при обработке отправителем данных блока SACK, связана с тем, что подтверждения SACK могут доставляться с нарушением порядка (т. е., отправленный раньше блок SACK может быть доставлен позднее своих последователей). При нарушении порядка доставки блоков SACK у отправителя данных может сложиться превратное представление о буферном пространстве его партнёра.

Поскольку явных идентификаторов, позволяющих детектировать нарушение порядка доставки SACK, не предусмотрено, отправитель данных должен использовать эвристические методы проверки полученных подтверждений SACK.

Конечной точке **следует** использовать приведённые ниже правила расчёта rwnd на основе значений a_rwnd, Cumulative TSN Ack и Gap Ack Block, полученных в блоке SACK.

A) При создании ассоциации конечная точка устанавливает rwnd = Advertised Receiver Window Credit (a_rwnd), указанное партнёром. в блоке INIT или INIT ACK.

B) Всякий раз при передаче (или повторе) блока DATA партнёру конечная точка вычитает размер переданной информации из значения rwnd для этого партнёра.

C) Всякий раз, когда блок DATA помечается для повтора (по таймеру T3-rtx (параграф 6.3.3) или fast retransmit (параграф 7.2.4))[1], размер этого блока добавляется к значению rwnd.

D) Всякий раз при получении SACK конечная точка выполняет следующие операции.

   i. Если Cumulative TSN Ack < Cumulative TSN Ack Point, блок SACK отбрасывается. Поскольку Cumulative TSN Ack возрастает монотонно, блок SACK, в котором Cumulative TSN Ack < Cumulative TSN Ack Point говорит о нарушении порядка доставки SACK.

   ii. Для переменной rwnd устанавливается значение a_rwnd за вычетом числа байтов, остающихся не подтверждёнными, после обработки Cumulative TSN Ack и Gap Ack Block.

   iii. Если SACK является отсутствующим TSN, который был ранее подтверждён с использованием Gap Ack Block (например, получатель отказался от данных), соответствующий блок DATA помечается как доступный для повтора. Блок помечается как отсутствующий для быстрого повтора (параграф 7.2.4) и при отсутствии работающего таймера для адреса получателя, по которому блок DATA был передан изначально, для этого адреса запускается таймер T3-rtx.

## 6.3 Управление таймером повтора передачи

Конечная точка SCTP использует таймер повтора передачи T3-rtx для обеспечения доставки данных при отсутствии обратной связи с партнёром. Время этого таймера называют тайм-аутом повтора или RTO[2].

Когда партнёр является многодомным хостом, конечная точка будет рассчитывать значение RTO для каждого транспортного адреса удалённого партнёра.

Расчёт и обслуживание RTO для протокола SCTP осуществляются так же, как это делается для таймера повтора передачи в TCP. Для расчёта текущего значения RTO конечная точка поддерживает две переменных состояния для каждого адреса получателя - SRTT[3] и RTTVAR[4].

### *6.3.1 Расчёт RTO*

Ниже приводятся правила расчёта значений SRTT, RTTVAR и RTO.

C1) До того, как будет измерено значение RTT для пакета, переданного по данному транспортному адресу, для RTO следует использовать параметр протокола RTO.Initial.

C2) После того, как будет определено значение RTT (обозначим его R), следует установить SRTT <- R, RTTVAR <- R/2, RTO <- SRTT + 4 * RTTVAR.

C3) Когда будет получено для RTT новое значение R', следует установить[5] RTTVAR <- (1 - RTO.Beta) * RTTVAR + RTO.Beta * |SRTT - R'| и SRTT <- (1 - RTO.Alpha) * SRTT + RTO.Alpha * R'

C4) Когда данные находятся в процессе доставки и выполняется правило C5, для каждого кругового обхода **должно** быть выполнено новое измерение RTT. Новое измерение RTT **следует** проводить не более одного раза на круговой обход для данного транспортного адреса. Такое ограничение обусловлено 2 причинами. Во-первых, более частые измерения не имеют смысла, поскольку не дают заметных преимуществ [ALLMAN99]; во-вторых, при частых измерениях значения RTO.Alpha и RTO.Beta в правиле C3 следует подбирать так, чтобы значения SRTT и RTTVAR рассчитывались примерно с такой же частотой (в терминах количества круговых обходов, при котором новые значения вступают в силу) как при одном измерении на круговой обход и с использованием RTO.Alpha и RTO.Beta в правиле C3. Однако точная процедура расчётов требует дополнительных исследований.

C5) Алгоритм Karn: измерение RTT **недопустимо** выполнять с использованием передаваемых повторно пакетов, поскольку нет возможности различить, к какой из переданных копий относится полученный отклик.

C6) Если после расчёта RTO получается значение меньше RTO.Min, устанавливается RTO = RTO.Min. Причина этого заключается в том, что использование слишком малых значений RTO будет приводить к возникновению неоправданных тайм-аутов [ALLMAN99].

C7) Максимальное значение RTO составляет по крайней мере RTO.max секунд.

---

[1]Если реализация поддерживает таймеры для каждого блока DATA, для повторной передачи помечаются только блоки DATA, для которых истекло заданное таймером время.
[2]Retransmission timeout - тайм-аут для повтора.
[3]Smoothed round-trip time - усредненное время кругового обхода.
[4]Round-trip time variation - вариации времени кругового обхода.
[5]Значение SRTT используемое для обновления RTTVAR представляет собой SRTT до обновления. После расчета следует обновить RTO <- SRTT + 4 * RTTVAR.

К дискретности временных параметров (G) при измерении RTT и различных переменных состояния применяется единственное правило.

G1) Если при расчёте RTTVAR получено нулевое значение, следует установить RTTVAR <- G.

Опыт показывает [ALLMAN99], что дискретность не более 100 мсек является предпочтительной.

### *6.3.2 Правила для таймера повторной передачи*

Ниже приведены правила управления таймером повтора передачи.

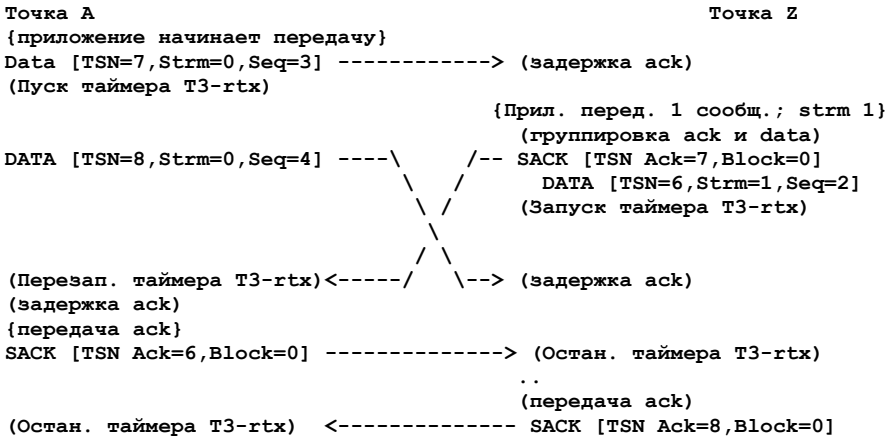



*Рисунок 8. Пример правил для таймера.*

R1) Каждый раз при передаче (включая повторы) блока DATA по любому из адресов следует запустить для этого адреса таймер T3-rtx (если он не работает) на время RTO. Используемое для таймера значение RTO удваивается после каждого случая тайм-аута для предыдущего таймера T3-rtx, связанного с этим адресом, как указано ниже в правиле E2.

R2) После подтверждения всех остающихся для этого адреса данных таймер T3-rtx для данного адреса сбрасывается.

R3) Всякий раз при получении SACK, подтверждающего блок DATA с остающимся TSN для данного адреса, таймер T3-rtx для этого адреса запускается заново с текущим значением RTO (если для этого адреса ещё остаются данные).

R4) При получении SACK для отсутствующего TSN, который ранее был подтверждён с помощью Gap Ack Block, включается таймер T3-rtx для адреса получателя, по которому блок DATA был передан изначально (если этот таймер ещё не запущен).

На рисунке 8 показан пример использования правил для таймера (предполагается, что получатель использует отложенные подтверждения).

### *6.3.3 Обработка завершения отсчёта T3-rtx*

При завершении отсчёта T3-rtx для адреса получателя выполняются перечисленные ниже действия.

E1) Для этого адреса изменяется значение ssthresh в соответствии с правилами, приведёнными в параграфе 7.2.3, и устанавливается cwnd <- MTU.

E2) Для этого адреса устанавливается RTO <- RTO * 2. Максимальное значение для таймера рассматривается в приведённом выше правиле C7 (RTO.max). Это значение может служить верхней границей для операций удваивания.

E3) Определяется количество более ранних (т. е., с меньшими номерами TSN) неподтвержденных блоков DATA для этого адреса, которые можно поместить в один пакет с учётом ограничений MTU на пути доставки по этому транспортному адресу (для разных путей могут быть разные значения MTU - см. параграф 6.4). Обозначим найденное число блоков K. Эти K блоков DATA группируются в один пакет и передаются удалённому партнёру по его транспортному адресу.

E4) Запускается таймер повтора T3-rtx для адреса, по которому был передан повторный пакет, если приведённое выше правило R1 указывает это. Используемое для таймера значение RTO следует брать для одного из адресов, по которым передаётся повтор - в случае многодомного получателя значения могут различаться для разных адресов получателя (см. параграф 6.4 ).

После повтора передачи, когда будет проведено новое измерение RTT (это может случиться только в том случае, когда новые данные были переданы и подтверждены в соответствии с правилом C5, или измерение сделано на основе HEARTBEAT (см. параграф 8.3)), выполняется расчёт по правилу C3 (включая вычисление RTO), который может привести к “коллапсу” RTO (со снижением значения до начального уровня) после того, как это значение было удвоено (правило E2).

Примечание. Любые блоки DATA, переданные по адресу, для которого истекло время T3-rtx, но не был заполнен MTU (правило E3), следует пометить для повторной передачи и передать, как только позволит cwnd (обычно при получении SACK).

Заключительное правило управления таймером повтора передачи связано с переключением на другой адрес получателя (см. параграф 6.4.1).

F1) Всякий раз, когда конечная точка переключается с текущего транспортного адреса получателя на другой транспортный адрес, текущий таймер повтора передачи продолжает работать. Как только конечная точка передаст пакет, содержащий блок(и) DATA, по новому транспортному адресу, запускается таймер для этого адреса с

использованием значения RTO для адреса получателя, по которому были посланы данные, если правило R1 указывает, что это нужно сделать.

## 6.4 Многодомные точки SCTP

Конечная точка SCTP рассматривается как многодомная, если для доставки данных в эту точку может использоваться более одного транспортного адреса.

Протоколу вышележащего уровня (ULP) в конечной точке следует выбрать один из множества адресов многодомного партнёра. в качестве первичного пути (см. параграфы 5.1.2 и 10.1).

По умолчанию конечной точке **следует** всегда передавать данные по первичному пути, если пользователь SCTP явно не указал транспортный адрес получателя (возможно и транспортный адрес отправителя).

Конечной точке **следует** передавать блоки откликов (например, SACK, HEARTBEAT ACK и т. п.) по тому же транспортному адресу, с которого был получен блок DATA или управляющий блок, вызвавший передачу отклика. Этому правилу нужно следовать также в тех случаях, когда конечная точка объединяет с откликом блоки DATA.

Однако при подтверждении множества блоков DATA, полученных в пакетах с разных адресов, в одном SACK этот блок SACK может передаваться по одному из транспортных адресов, с которых были получены подтверждаемые блоки DATA или управляющие блоки.

Когда получатель дубликата блока DATA передаёт блок SACK многодомной конечной точке, он **может** воспользоваться выбором адреса получателя и не использовать в качестве такового адрес отправителя блока DATA. Причина этого заключается в том, что получение дубликата от многодомной конечной точки может указывать на то, что путь возврата, указанный в поле отправителя блока DATA, может быть недоступен (разорван) для передачи SACK.

Конечной точке, имеющей многодомного партнёра., **следует** пытаться повторять передачу блока по активному транспортному адресу, который отличается от последнего адреса получателя, по которому был передан блок DATA.

Повтор передачи не оказывает влияния на общий счётчик неподтвержденных данных. Однако, если блок DATA передаётся повторно с использованием другого адреса получателя, счётчики неподтвержденных данных для старого и нового адресов получателя должны быть согласованно изменены.

### *6.4.1 Переключение с неактивного адреса получателя*

Некоторые транспортные адреса многодомной конечной точки SCTP могут утратить активность в результате ошибок (см. параграф 8.2) или по требованию пользователя SCTP.

Когда имеются данные для передачи и основной путь становится неактивным (например, по причине отказа) или пользователь SCTP явно запрашивает передачу данных по неактивному транспортному адресу получателя, до передачи сообщения об ошибке протоколу вышележащего уровня конечной точке SCTP следует предпринять попытку передачи данных по другому активному адресу получателя, если таковой имеется.

При повторе передачи данных, если конечная точка является многодомной, ей следует рассматривать каждую пару адресов “отправитель-получатель” в политике выбора при повторе. Передающей конечной точке следует пытаться выбрать для повтора пару адресов, наиболее сильно отличающуюся от использованной при первой попытке пары “отправитель-получатель”.

Примечание: Правила выбора максимально отличающейся пары зависят от реализации и не определяются данной спецификацией.

## 6.5 Идентификаторы и порядковые номера в потоке

Каждый блок DATA должен содержать корректный идентификатор потока. Если конечная точка получает блок DATA с некорректным идентификатором, ей следует подтвердить приём данных в соответствии с обычной процедурой, незамедлительно передать блок ERROR с причиной ошибки Invalid Stream Identifier (некорректный идентификатор потока, см. параграф 3.3.10) и отбросить блок DATA. Конечная точка может группировать блок ERROR в один пакет с блоком SACK, если ERROR следует после SACK.

Порядковые номера во всех потоках должны начинаться с нуля при создании ассоциации. При достижении порядковым номером в потоке значения 65535 следующим номером снова будет 0.

## 6.6 Упорядоченная и неупорядоченная доставка

Внутри потока конечная точка **должна** доставлять блоки DATA, полученные с флагом U = 0, на вышележащий уровень в соответствии с порядковыми номерами блоков в потоке. Если блоки DATA поступают с нарушением порядка, конечная точка **должна** удерживать полученные блоки DATA от передачи ULP, пока не будет восстановлен порядок.

Однако конечная точка SCTP может запросить неупорядоченную доставку для определённого блока DATA, передаваемого в потоке, установив для этого блока флаг U = 1.

При получении конечной точкой блока DATA с флагом U = 1, этот блок должен обрабатываться в обход механизма упорядочивания и незамедлительно доставляться на вышележащий уровень (после сборки фрагментов, если отправитель фрагментировал блок).

Это обеспечивает эффективный способ передачи “дополнительных” (out-of-band) данных в потоке. Кроме того, весь поток можно сделать “неупорядоченным”, устанавливая флаг U = 1 для каждого блока DATA, передаваемого в этом потоке.

Примечание для разработчиков. При передаче неупорядоченного блока DATA реализация протокола может помещать такой блок DATA в начало очереди на передачу, если такая возможность имеется.

Поле 'Stream Sequence Number' в блоке DATA с флагом U = 1 не имеет смысла. Отправитель может указывать в этом поле произвольное значение, а получатель **должен** игнорировать это поле.

Примечание. При передаче упорядоченных и неупорядоченных данных, конечная точка не увеличивает своё значение поля Stream Sequence Number, передавая блок DATA с флагом U = 1.

## 6.7 Информация о пропусках в порядковых номерах TSN блоков DATA

При получении нового блока DATA конечной точке следует проверить непрерывность порядковых номеров TSN. При обнаружении пропуска в порядковых номерах принятых блоков DATA конечной точке **следует** незамедлительно передать блок SACK с Gap Ack Block. Получатель продолжает передачу SACK после получения каждого пакета SCTP, который не закрывает пропуск в порядковых номерах.

На основе Gap Ack Block из полученного блока SACK конечная точка может определить пропущенные блоки DATA и принять решение о необходимости повторной передачи таких блоков (см. параграф 6.2.1).

В одном блоке SACK может передаваться информация о нескольких обнаруженных пропусках (см. параграф 3.3.4).

Если партнёр является многодомным, конечной точке SCTP всегда **следует** пытаться передать SACK по тому адресу, с которого был получен последний блок DATA.

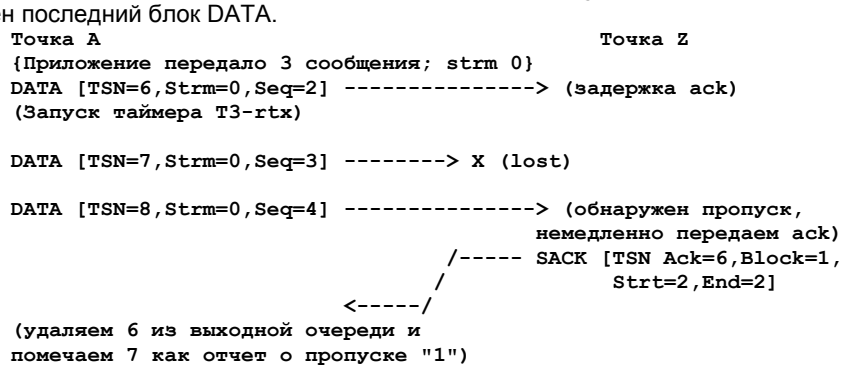



*Рисунок 9. Отчет о пропуске с использованием SACK.*

При получении блока SACK конечная точка **должна** удалить из выходной очереди все блоки DATA, которые были подтверждены в Cumulative TSN Ack блока SACK. Конечная точка **должна** также трактовать все блоки DATA с номерами TSN, не включёнными в Gap Ack Block из блока SACK, как “потерянные”. Число отчётов о “потерях” для каждого неподтвержденного блока DATA должно быть записано отправителем, чтобы принять решение о повторной передаче (см. параграф 7.2.4).

На рисунке 9 приведён пример использования SACK для передачи информации о пропуске.

Максимальное число Gap Ack Block, которые могут быть включены в один блок SACK, ограничивается текущим значением MTU для пути. Когда в один блок SACK невозможно включить все Gap Ack Block, которые нужно передать, по причине ограничения MTU, конечная точка **должна** передать только один блок SACK, включающий все Gap Ack Block от младших номеров TSN к старшим, которые могут поместиться в блок, ограниченный значением MTU, и оставить старшие номера TSN неподтвержденными.

## 6.8 Расчёт контрольной суммы Adler-32

При передаче пакета SCTP конечная точка **должна** для обеспечения возможности контроля целостности данных включить в пакет значение контрольной суммы Adler-32 в соответствии с приведённым ниже описанием.

После создания пакета (содержащего общий заголовок SCTP и один или несколько управляющих блоков или блоков DATA), конечная точка должна выполнить следующие операции:

1) Заполнить поле Verification Tag общего заголовка SCTP и установить нулевое значение в поле контрольной суммы.
2) Рассчитать контрольную сумму по алгоритму Adler-32 с включением общего заголовка SCTP и всех блоков из пакета. Описание алгоритма Adler-32 приводится в Приложении B.
3) Поместить полученное значение в поле контрольной суммы общего заголовка, не изменяя остальных битов.

При получении пакета SCTP принимающая конечная точка **должна** проверить значение контрольной суммы Adler-32 в заголовке:

1) Сохранить полученное в пакете значение контрольной суммы Adler-32.
2) Заменить 32-битовое поле контрольной суммы принятого пакета SCTP значением 0 и рассчитать контрольную сумму по алгоритму Adler-32 для изменённого пакета.
3) Сравнить сохранённое значение (из пакета) контрольной суммы Adler-32 с контрольной суммой, полученной в результате расчёта. Если значения не совпадут, получатель **должен** трактовать принятый пакет как некорректный.

По умолчанию некорректные пакеты SCTP отбрасываются без уведомления.

## 6.9 Фрагментация и сборка

Конечная точка **может** поддерживать фрагментацию при передаче блоков DATA и **должна** поддерживать сборку фрагментов для принимаемых блоков DATA. Если конечная точка поддерживает фрагментацию, она **должна** фрагментировать пользовательские сообщения, когда их размер приводит к созданию пакетов SCTP, превышающих по своему размеру значение MTU. Если реализация не поддерживает фрагментацию, конечная точка должна возвращать вышележащему уровню код ошибки, не пытаясь передать сообщение избыточного размера.

Примечание для разработчиков. В случае ошибки для возврата кода ошибки вышележащему уровню нужно использовать примитив Send, рассматриваемый в параграфе 10.1.

Если партнёр является многодомным, конечной точке следует выбирать размер пакета, не превышающий значение Path MTU для ассоциации (наименьшее из значений Path MTU для всех адресов, участвующих в ассоциации).

Примечание. После фрагментации сообщения оно не может быть фрагментировано ещё раз. Если же значение Path MTU уменьшилось, в таких случаях должна использоваться фрагментация IP. Определение значений Path MTU рассматривается в параграфе 7.3.

При решении вопроса о необходимости фрагментирования реализация SCTP **должна** принимать во внимание размер заголовка пакета SCTP и заголовков блоков DATA. **Должен** также приниматься во внимание размер блоков SACK, если они группируются с блоком DATA.

Фрагментация выполняется следующим образом.

1) Отправитель **должен** разбить пользовательское сообщение не несколько блоков DATA, чтобы размер каждого блока в сумме со служебной информацией SCTP не превышал размер поля данных дейтаграммы IP, которая по своему размеру равна или меньше значения Path MTU для ассоциации.
2) Отправитель **должен** выделить по порядку номера TSN для каждого из блоков DATA, содержащих фрагменты сообщения. Всем блокам DATA, содержащим фрагменты одного сообщения, присваиваются одинаковые номера SSN. Если пользователь указал, что сообщение доставляется без соблюдения порядка, для каждого блока DATA **должен** быть установлен флаг U = 1.
3) Отправитель **должен** также установить биты B/E блока DATA для всех фрагментов (10 для первого, 01 для последнего и 00 для всех остальных).

Конечная точка **должна** идентифицировать фрагментированные блоки DATA путём проверки битов B/E в каждом принятом блоке DATA и помещать фрагменты в очередь для сборки. После сборки пользовательского сообщения из фрагментов, протокол SCTP будет передавать собранное сообщение в конкретный поток для возможного упорядочивания и окончательной диспетчеризации.

Примечание. Если на передающей стороне недостаточно памяти для буферизации фрагментов в процессе ожидания их доставки, следует направить полученную часть входящего сообщения через API частичной доставки (см. главу 10) для освобождения буферного пространства.

## 6.10 Группировка блоков

Конечная точка группирует блоки путём простого включения множества блоков в один исходящий пакет SCTP. Суммарный размер получающейся в результате дейтаграммы IP (включая пакет SCTP и заголовок IP) **должен** быть не более текущего значения Path MTU.

Если партнёр является многодомным, передающей стороне следует выбирать размер пакета так, чтобы он не превышал последнее значение MTU для первичного пути.

При группировке управляющих блоков с блоками DATA конечная точка **должна** помещать управляющие блоки перед блоками данных. Отправитель **должен** передавать блоки DATA внутри пакета SCTP в соответствии с ростом номеров TSN.

Примечание. Поскольку управляющие блоки должны размещаться в пакете перед блоками DATA, а блоки DATA должны передаваться до управляющих блоков SHUTDOWN и SHUTDOWN ACK, блоки DATA не могут группироваться с блоками SHUTDOWN или SHUTDOWN ACK.

**Недопустимо** включение в пакет SCTP неполных блоков (Partial chunk).

Принимающая сторона **должна** обрабатывать полученные блоки в порядке их следования в пакете. Получатель использует поле длины блока для определения конца данного блока и начала следующего. При определении начала блока не следует забывать о выравнивании блоков по 4-байтовой границе. Если получатель обнаруживает в пакете неполный блок, такой блок **должен** быть отброшен.

**Недопустимо** группировать блоки INIT, INIT ACK, SHUTDOWN COMPLETE с любыми другими блоками.

# *7. Контроль насыщения*

Контроль насыщения является одной из базовых функций SCTP. Ясно, что для некоторых приложений может быть выделено достаточное количество ресурсов, которые позволят обеспечить незамедлительную доставку критичного к задержкам трафика SCTP и при нормальной работе будет казаться не очевидной возможность возникновения перегрузок в сети. Однако протокол SCTP должен работать и в неблагоприятных условиях, когда возможны отказы в сети или неожиданные всплески трафика. В таких ситуациях SCTP должен использовать корректные механизмы контроля насыщения для обеспечения доставки данных за приемлемое время. При отсутствии перегрузок алгоритмы контроля насыщения не должны оказывать влияния на работу протокола.

Примечание для разработчиков. После удовлетворения заданных требований по производительности разработчики могут выбрать более консервативный алгоритм контроля насыщения из числа описанных ниже.

Алгоритмы контроля насыщения протокола SCTP основаны на [RFC2581]. В этой главе описано, как алгоритмы, определённые в RFC2581, адаптированы для использования в SCTP. Сначала рассматриваются различия между протоколами TCP и SCTP, а после этого описывается схема контроля насыщения в SCTP. В описании используется та же терминология, которая принята для протокола TCP. Контроль насыщения в SCTP всегда применяется ко всей ассоциации (соединению), а не к отдельным потокам.

## 7.1 Различия в контроле насыщения для SCTP и TCP

Gap Ack Block в SCTP SACK имеют такой же смысл, как TCP SACK. Протокол TCP рассматривает информацию в SACK только как консультативную. SCTP также рассматривает информацию, передаваемую в Gap Ack Blocks блоков SACK как консультативную. В SCTP любой блок DATA, подтверждённый с помощью SACK (включая блоки DATA, полученные на приёмной. стороне с нарушением порядка), не рассматривается как доставленный окончательно, пока Cumulative

TSN Ack Point не перейдёт значение TSN блока DATA (т. е., пока блок DATA не будет подтверждён полем Cumulative TSN Ack в SACK). Следовательно, значение параметра cwnd контролирует количество неподтвержденных данных, а не (как в случае TCP без SACK) верхнюю границу между максимальным подтверждённым порядковым номером и последним блоком DATA, который может быть передан в окне насыщения. SCTP SACK обусловливает отличие реализации механизмов ускоренного повтора (fast-retransmit) и быстрого восстановления (fast-recovery) от случая TCP без SACK. Пример этого приведён в [FALL96].

Наиболее серьёзным различием между SCTP и TCP является поддержка в SCTP многодомных конечных точек. Протокол SCTP разработан для организации устойчивых соединений (ассоциаций) между конечными точками, каждая из которых может быть доступна с использованием множества транспортных адресов. Использование различных адресов может вести к организации разных путей между парой конечных точек и в идеальном случае для каждого пути могут использоваться свои параметры контроля насыщения. Приведённая здесь трактовка контроля насыщения для многодомных получателей является новинкой протокола SCTP и может быть уточнена в будущем. Текущий алгоритм основан на следующих допущениях:

- Отправитель обычно не меняет адрес получателя, пока не получит запрос на такую замену от протокола вышележащего уровня; однако SCTP может переключиться на другой адрес получателя, если прежний адрес был помечен как неактивный (см. параграф 8.2). Кроме того, SCTP может повторять передачу пакетов по адресам, отличающимся от тех, которые использовались для передачи исходного пакета.
- Отправитель сохраняет отдельный набор параметров контроля насыщения для каждого адреса получателя, по которому он может передавать данные (не для каждой пары адресов “отправитель-получатель”, а для каждого адреса получателя). Параметры следует отбрасывать, если адрес не используется достаточно долго.
- Для каждого адреса получателя конечная точка выполняет процедуру замедленного старта (slow-start) в начале передачи данных по этому адресу.

Примечание. TCP гарантирует упорядоченную доставку данных протоколу вышележащего уровня в рамках одной сессии TCP. Это означает, что при получении протоколом TCP информации о пропуске в порядковых номерах он будет ждать заполнения пропуска и только после этого передаст данные вышележащему уровню. SCTP может доставлять данные на вышележащий уровень даже при наличии пропусков в порядковых номерах TSN, если порядковые номера в потоке (Stream Sequence Number) упорядочены в рамках данного потока (т. е., пропущенный блок DATA относится к другому потоку) или при запросе неупорядоченной доставки. Это различие не оказывает влияния на cwnd, но может влиять на расчёт rwnd.

## 7.2 Процедуры Slow-Start[1] и Congestion Avoidance[2]

Алгоритмы замедленного старта и предотвращения перегрузки **должны** использоваться конечной точкой для контроля количества данных, которые будут передаваться в сеть. Контроль насыщения работает в SCTP на уровне ассоциаций, а не отдельных потоков в ассоциации. В некоторых ситуациях отправителю SCTP может давать некоторые преимущества более консервативное поведение, нежели предлагают эти алгоритмы, однако для отправителя **недопустимо** быть более агрессивным, нежели позволяют алгоритмы.

Подобно TCP, конечная точка SCTP использует три перечисленных ниже переменных для управления скоростью передачи.

- Анонсируемый получателем размер окна приёма (rwnd[3], в байтах) устанавливается получателем данных с учётом возможностей выделения буферов для принимаемых пакетов.

  Примечание. эта переменная имеет одно значение для всей ассоциации.

- Окно контроля насыщения (cwnd[4], в байтах) устанавливается отправителем с учётом условий в сети.

  Примечание. Эта переменная поддерживается для каждого адреса получателя.

- Порог замедленного старта (ssthresh[5], в байтах) используется отправителем для принятия решения о выборе используемого алгоритма контроля насыщения (slow start или congestion) на данной фазе.

  Примечание. Эта переменная поддерживается для каждого адреса получателя.

Протокол SCTP использует также одну дополнительную переменную - partial_bytes_acked, которая применяется в фазе предотвращения перегрузки для упрощения расчёта значения cwnd.

В отличие от TCP, отправитель SCTP **должен** хранить набор из трёх переменных (cwnd, ssthresh и partial_bytes_acked) для **каждого** адреса получателя на удалённой стороне (если партнёр является многодомным). Только переменная rwnd имеет значение для всей ассоциации (независимо от того, является ли партнёр многодомным).

### *7.2.1 Замедленный старт*

Начиная передачу данных в сеть с неизвестными условиями или возобновляя передачу после достаточно долгого простоя, протокол SCTP должен проверить сеть на предмет пропускной способности. Для этих целей используется алгоритм slow start на начальной стадии передачи или при восстановлении потерь, обнаруженных по таймеру повтора передачи.

- Стартовое значение cwnd до передачи блоков DATA или после достаточно долгого бездействия **должно** быть не более 2*MTU.
- Начальное значение cwnd после тайм-аута для повтора передачи **должно** быть не более 1*MTU.

---

[1] Замедленный старт.
[2] Предотвращение перегрузки.
[3] Receiver advertised window size.
[4] Congestion control window. Для краткости будем называть его просто окном насыщения. *Прим. перев.*
[5] Slow-start threshold.

- Начальное значение ssthresh **может** быть произвольно большим (например, реализация **может** просто установить для этого параметра значение размера окна приёма, анонсируемого получателем).

- При положительном значении cwnd конечная точка может иметь cwnd ожидающих подтверждения байтов данных для этого транспортного адреса.

- Если cwnd меньше или равно ssthresh, конечная точка SCTP **должна** использовать алгоритм slow start для увеличения размера cwnd (предполагается, что текущее окно насыщения полностью используется). Если входящее подтверждение SACK указывает за пределы Cumulative TSN Ack Point, значение cwnd **должно** быть увеличено не более чем на меньшее из двух значений - 1) общий размер подтверждённых блоков DATA или 2) значение path MTU для данного адресата. Такой подход обеспечивает защиту от атаки ACK-Splitting, описанной в [SAVAGE99].

В случаях, когда партнёр является многодомным и конечная точка получает подтверждение SACK, указывающее за пределы Cumulative TSN Ack Point, ей следует обновить значение cwnd (или несколько значений cwnd) для адреса получателя, по которому были переданы подтверждённые данные. Однако, если принятое подтверждение SACK не указывает вперёд Cumulative TSN Ack Point, для конечной точки **недопустимо** менять значение cwnd для любого из адресов получателя.

Поскольку значение cwnd для конечной точки не связано с Cumulative TSN Ack Point для неё, при получении дубликата SACK (даже если он не указывает за пределы Cumulative TSN Ack Point), конечная точка может продолжать использование этого подтверждения для синхронизации отправки новых данных. Т. е., данные, подтверждённые последним SACK уменьшают количество данных, находящихся в пути до значения меньше cwnd и даже неизменное значение cwnd позволяет передать новую порцию данных в сеть. С другой стороны, увеличение cwnd должно быть связано с подтверждением сверх Cumulative TSN Ack Point, как было сказано выше. В противном случае дубликат SACK будет не только разрешать передачу в сеть новых данных, но и увеличивать количество данных, которые можно передать, не дожидаясь подтверждения, хотя в это время сеть может оказаться перегруженной.

- Когда конечная точка не передаёт данных по какому-либо транспортному адресу, в качестве значения cwnd для этого адреса следует установить max(cwnd/2, 2*MTU) в расчёте на RTO.

### *7.2.2 Предотвращение перегрузки*

При cwnd > ssthresh, значение cwnd следует увеличивать на 1*MTU за каждый период RTT, если у отправителя имеется не менее cwnd байтов неподтвержденных данных для соответствующего транспортного адреса.

На практике эта процедура может быть реализована следующим способом:

- Устанавливается partial_bytes_acked = 0.

- Если cwnd > ssthresh, всякий раз при получении SACK, указывающего за пределы Cumulative TSN Ack Point, значение partial_bytes_acked увеличивается на число байтов во всех блоках, подтверждённых этим SACK, включая блоки, подтверждённые новым значением Cumulative TSN Ack и Gap Ack Block.

- Когда значение partial_bytes_acked становится равным cwnd или превышает его и до прибытия SACK у отправителя имеется не менее cwnd остающихся данных (т. е., до прибытия SACK, объем переданных, но ещё не подтверждённых данных больше или равен размеру окна cwnd), значение cwnd увеличивается на MTU, а значение partial_bytes_acked уменьшается на cwnd.

- Как и при замедленном старте в качестве значения cwnd для адреса получателя следует установить max(cwnd/2, 2*MTU) в расчёте на RTO, если отправитель не передаёт блоков DATA по данному адресу.

- Когда все переданные отправителем данные подтверждены получателем, устанавливается partial_bytes_acked = 0.

### *7.2.3 Контроль насыщения*

При обнаружении потери пакета из подтверждения SACK (см. параграф 7.2.4), конечной точке следует установить значения:

```
ssthresh = max(cwnd/2, 2*MTU)
cwnd = ssthresh
```

Обычно потеря пакета приводит к уменьшению cwnd вдвое.

Когда завершается отсчёт. по таймеру T3-rtx для этого адреса, SCTP следует выполнить процедуру замедленного старта, установив:

```
ssthresh = max(cwnd/2, 2*MTU)
cwnd = 1*MTU
```

и обеспечивать наличие не более одного пакета SCTP на пути от отправителя к получателю, пока конечная точка не получит подтверждения доставки данных по этому адресу.

### *7.2.4 Ускоренный повтор при пропуске*

При отсутствии потерь данных конечная точка может использовать подтверждение с задержкой. Однако при обнаружении пропуска в порядковых номерах TSN **следует** начать передачу подтверждений SACK незамедлительно после доставки каждого пакета пока пропуск в порядковых номерах не будет заполнен.

Всякий раз при получении SACK, указывающего на пропуск некоторых номеров TSN, конечной точке **следует** дождаться 3 последующих индикаций потери (в следующих подтверждениях SACK) для тех же номеров, прежде, чем начать реализацию механизма ускоренного повтора (Fast Retransmit).

Когда отсутствующие номера TSN указаны в 4 последовательных подтверждениях SACK, отправителю следует:

1) Пометить указанные блоки DATA для повторной передачи.

2) Изменить значения ssthresh и cwnd для адреса (или нескольких адресов) получателя, по которому были переданы утерянные данные в последний раз, согласно выражениям, приведённым в параграфе 7.2.3.

3) Определить, сколько наиболее ранних (т. е., с меньшими номерами TSN) блоков DATA может быть включено в один пакет с учётом значения path MTU для адреса получателя, по которому будут передаваться данные (обозначим это количество буквой K). Повторно передать эти K блоков DATA в одном пакете.

4) Заново запустить таймер T3-rtx, если последнее подтверждение SACK относится к меньшему из остающихся в сети номеров TSN для данного адреса получателя или конечная точка повторяет передачу первого из остающихся в сети блоков DATA, переданных по этому адресу.

Примечание. Если полученный блок SACK также подтверждает новые блоки DATA за пределами Cumulative TSN AckPoint, перед выполнением перечисленных в пп. 1 - 4 операций нужно сначала изменить значение cwnd в соответствии с правилами, приведёнными в параграфах 7.2.1 и 7.2.2.

Корректная реализация приведённых выше правил будет подсчитывать все пропуски TSN, о которых сообщалось в SACK. Значение счётчика увеличивается для каждого последующего подтверждения SACK, указывающего на пропуск TSN. После того, как значение счётчика достигнет 4 и будет начата процедура ускоренного повтора передачи, значение счётчика сбрасывается в 0.

Поскольку значение cwnd в SCTP опосредованно ограничивает количество остающихся в сети TSN, механизм быстрого восстановления TCP (fast-recovery) реализуется автоматически без изменения размера окна контроля насыщения.

## 7.3 Определение MTU для пути

В [RFC1191] описан метод "Path MTU Discovery", позволяющий конечной точке оценить значение максимального размера пакета (MTU) на данном пути через Internet и воздерживаться от передачи по этому пути пакетов, размер которых превышает MTU, без использования методов зондирования с целью определения изменений Path MTU (PMTU). RFC 1191 предлагает механизм и стратегию определения “сквозного” значения MTU для пути и изменений этого параметра. В [RFC1981] определяется аналогичный метод для протокола IPv6. Отправитель SCTP, использующий IPv6, **должен** применять метод Path MTU Discovery, если размер пакетов может превышать минимальное значение MTU для IPv6 [RFC2460].

Конечной точке **следует** применять этот метод определения MTU и **следует** выполнять такое определение независимо для каждого адреса получателя.

При определении значения MTU для протокола SCTP существуют несколько отличий от описания, приведённого в RFC 1191 для TCP:

1) Ассоциация SCTP может включать множество адресов. Конечная точка **должна** поддерживать отдельную оценку значения MTU для каждого из адресов своего партнёра.

2) Термин "MTU" в данном документе всегда относится к значению MTU для адреса получателя, к которому относится обсуждение.

3) В отличие от TCP, протокол SCTP не использует понятия “максимальный размер сегмента” (MSS[1]). Соответственно, MTU для каждого адреса получателя **следует** инициализировать значением, не превышающее MTU канала для локального интерфейса, через который пакет будет маршрутизироваться к получателю.

4) Поскольку передача данных в SCTP структурируется естественным способом в терминах порядковых номеров TSN, а не в байтах (как в TCP), обсуждение в параграфе 6.5 RFC 1191 применимо и к этому протоколу: “При повторе передачи дейтаграммы IP по удалённому адресу, для которого исходная дейтаграмма оказалась слишком большой по сравнению с MTU для пути к данному адресу, дейтаграмму IP **следует** передавать повторно без флага DF, что позволит фрагментировать её. При передаче новых дейтаграмм IP флаг DF должен устанавливаться.

5) Отправителю следует контролировать значение PMTU для ассоциации в целом, выбирая для этого минимальное из значений PMTU для путей ко всем адресам партнёра. При фрагментации сообщений значение PMTU для ассоциации в целом следует использовать для определении размера фрагментов. Такой подход позволит передавать фрагменты по любому из возможных путей без дополнительной фрагментации на уровне IP.

За исключением перечисленных выше различий методы определения MTU для пути, описанные в RFC 1191 и 1981 для протокола TCP, применимы к SCTP независимо для каждого адреса получателя.

Примечание: Для адресов IPv6 флаг DF не используется и дейтаграммы должны фрагментироваться в соответствии с [RFC2460].

# *8. Контроль отказов*

## 8.1 Обнаружение отказов конечных точек

Конечной точке следует подсчитывать общее количество последовательных повторов передачи своему партнёру (включая повторы по всем адресам для многодомных партнёров). Если значение этого счётчика превысит порог, указанный параметром протокола Association.Max.Retrans[2], конечной точке следует рассмотреть вопрос о доступности партнёра. и прекращении передачи ему каких-либо данных (т. е., перевода ассоциации в состояние CLOSED). Кроме того, конечной точке следует передать информацию об отказе (и, возможно, об оставшихся в выходной очереди данных) протоколу вышележащего уровня. Ассоциация автоматически закрывается, когда удалённый партнёр становится недоступным.

Счётчик повторов следует сбрасывать всякий раз, когда переданный партнёру блок DATA подтверждается с помощью SACK или от удалённого партнёра. принимается HEARTBEAT-ACK.

[1]Maximum Segment Size.
[2]Максимальное число повторов передачи для ассоциации.

## 8.2 Обнаружение сбоев в пути

Если партнёр является многодомным, конечной точке следует поддерживать счётчики ошибок для каждого транспортного адреса этого партнёра.

Каждый раз, когда завершается отсчёт. таймера T3-rtx для любого из адресов или передача HEARTBEAT по бездействующему адресу не подтверждается в течение RTO, значение счётчика ошибок для соответствующего адреса будет увеличиваться на 1. Когда значение счётчика превысит значение параметра протокола Path.Max.Retrans[1] для данного адреса, конечной точке следует пометить этот адрес как неактивный, а также **следует** передать уведомление об этом протоколу вышележащего уровня.

Когда остающиеся в сети номера TSN подтверждаются или блок HEARTBEAT, переданный по бездействующему адресу, подтверждается блоком HEARTBEAT ACK, конечная точка должна сбрасывать счётчик ошибок для транспортного адреса получателя, по которому был отправлен последний блок DATA (или блок HEARTBEAT). Если партнёр является многодомным и последний блок был передан ему в качестве повтора с изменением адреса получателя, возникает неоднозначность в выборе адреса, для которого следует учитывать полученное подтверждение. Однако эта неоднозначность не оказывает существенного влияния на дальнейшее поведение SCTP. Если такие неоднозначности нежелательны, отправитель может не сбрасывать счётчик ошибок, когда последний блок передавался повторно.

Примечание. При настройке конфигурации конечной точки SCTP следует избегать установки для параметра Association.Max.Retrans значения, превышающего любое из значений Path.Max.Retrans для адресов удалённой конечной точки. В противном случае все адреса удалённого партнёра. могут стать неактивными, а данная точка будет по-прежнему считать этого партнёра. доступным. При возникновении такой ситуации поведение протокола SCTP будет зависеть от конкретной реализации.

Когда основной путь помечен как неактивный (например, в результате множества повторов), отправитель **может** автоматически начать передачу всех новых пакетов по другому адресу, если он имеется и активен. При наличии в момент потери активности на основном пути нескольких активных дополнительных адресов **следует** выбирать только **один** транспортный адрес партнёра. и использовать его для передачи новых пакетов.

## 8.3 Проверка жизнеспособности пути

По умолчанию конечная точка SCTP должна вести мониторинг доступности бездействующих транспортных адресов своего партнёра. путём периодической отправки по таким адресам блоков HEARTBEAT.

Транспортный адрес рассматривается как бездействующий (idle), если нет новых блоков, которые могут использоваться для обновления периода RTT для пути (обычно, к таким блокам относятся DATA, INIT, COOKIE ECHO, HEARTBEAT и т. п.), и в течение текущего периода “проверки пульса”[2] по этому адресу не было передано блоков HEARTBEAT. Эта трактовка относится как к активным адресам, так и к неактивным.

Вышележащий уровень может дополнительно инициировать следующие функции:

A) запрет HEARTBEAT для определённого транспортного адреса в рамках данной ассоциации;

B) изменение интервала HB.interval;

C) восстановление передачи HEARTBEAT для указанного адреса в данной ассоциации;

D) запрос на передачу HEARTBEAT по указанному адресу в данной ассоциации.

Конечной точке следует увеличивать значение соответствующего счётчика ошибок для транспортного адреса партнёра. всякий раз, когда переданный по этому адресу блок HEARTBEAT не был подтверждён в течение периода RTO.

Когда значение счётчика достигает значения протокольного параметра Path.Max.Retrans, конечной точке следует пометить соответствующий адрес получателя как неактивный (если он ещё не помечен). Кроме того, конечная точка может сообщить вышележащему уровню об изменении состояния доступности для данного адреса получателя. После этого конечной точке следует продолжать передачу блоков HEARTBEAT по этому адресу, но без дальнейшего увеличения значения счётчика ошибок.

Отправителю блока HEARTBEAT следует включать в поле Heartbeat Information этого блока текущее время в момент передачи пакета.

Примечание для разработчиков. Для увеличения значений счётчиков может использоваться иной механизм, при котором значение счётчика увеличивается при передаче каждого блока HEARTBEAT. В таких случаях по прибытию подтверждения HEARTBEAT ACK **следует** сбрасывать счётчик для того адреса, по которому был передан подтверждённый блок HEARTBEAT. Такое уменьшение будет компенсировать рост значения счётчика при передаче каждого блока, а также сбрасывать результаты увеличения счётчика вследствие других ошибок.

Получателю блока HEARTBEAT следует незамедлительно передать в ответ подтверждение HEARTBEAT ACK, содержащее Heartbeat Information из полученного блока HEARTBEAT.

При получении блока HEARTBEAT ACK отправителю блока HEARTBEAT следует сбросить счётчик ошибок для транспортного адреса получателя, по которому был отправлен подтверждённый блок HEARTBEAT, и пометить адрес как активный (если он не был помечен ранее). Конечная точка может также сообщить вышележащему уровню об активизации транспортного адреса в результате получения последнего блока HEARTBEAT ACK. Получатель блока HEARTBEAT ACK должен также сбросить значение общего счётчика ошибок для ассоциации (см. параграф 8.1).

Получателю HEARTBEAT ACK следует также выполнить определение RTT для транспортного адреса партнёра., используя значение времени из блока HEARTBEAT ACK.

[1] Максимальное число повторов для пути.
[2] Heartbeat period.

Для бездействующего транспортного адреса, которому разрешено передавать блоки HEARTBEAT, рекомендуется передавать один блок в каждый период RTO для данного адреса получателя + HB.interval с вариациями ± 50% от RTO и экспоненциальным снижением RTO, если доставка предыдущего блока HEARTBEAT не подтверждена.

Для пользователей SCTP обеспечивается примитив, позволяющий изменять значение HB.interval и включать/выключать передачу блоков HEARTBEAT для данного адреса. Интервал передачи HEARTBEAT, устанавливаемый пользователем SCTP, добавляется к RTO для данного адресата (с учётом экспоненциального снижения RTO). Следует передавать только один блок HEARTBEAT в течение каждого периода отсчёта heartbeat-таймера (если бездействует множество адресов). Разработчики вольны определить способ выбора кандидата для передачи блока при наличии множества бездействующих адресов.

Примечание. При подстройке интервала heartbeat может возникать побочный эффект, который следует принимать во внимание. Когда этот интервал возрастает (блоки HEARTBEAT передаются реже), детектирование потери сообщений ABORT также замедляется. Если партнёр прервёт (ABORT) ассоциацию по какой-либо причине и блок ABORT будет потерян, локальная точка обнаружит прерывание ассоциации только при передаче блока DATA или HEARTBEAT (это вынудит партнёра. передать блок ABORT повторно). Такой эффект следует учитывать при установке значения для таймера HEARTBEAT. Если передача HEARTBEAT запрещена, потеря блока ABORT будет обнаружена только после передачи блока DATA.

## 8.4 Обработка пакетов OOTB

Пакет SCTP называется пакетом OOTB[1], если он правильно сформирован (т. е., включает корректное значение контрольной суммы Adler-32, описанной в параграфе 6.8), но получатель не может идентифицировать ассоциацию, к которой этот пакет относится.

Получатель пакета OOTB **должен** выполнить следующие операции с соблюдением их порядка:

1) Если в пакете OOTB указан отличный от индивидуального (non-unicast) адрес отправителя или получателя, такой пакет отбрасывается без уведомления.
2) Если пакет OOTB содержит блок ABORT, получатель **должен** отбросить такой пакет без уведомления и выполнения каких-либо дополнительных действий.
3) Если пакет содержит блок INIT с полем Verification Tag = 0, пакет обрабатывается как описано в параграфе 5.1.
4) Если пакет содержит COOKIE ECHO в первом блоке, обработка пакета выполняется в соответствии с параграфом 5.1.
5) Если пакет содержит блок SHUTDOWN ACK, получателю следует передать отправителю пакета OOTB блок SHUTDOWN COMPLETE. При передаче блока SHUTDOWN COMPLETE получатель пакета OOTB должен заполнить поле Verification Tag, скопировав в него значение Verification Tag из блока SHUTDOWN ACK и установить бит T в поле флагов блока для индикации отсутствия порядкового номера TCB.
6) Если пакет содержит блок SHUTDOWN COMPLETE, получателю следует отбросить пакет без уведомления и выполнения каких-либо дополнительных действий.
7) Если пакет содержит Stale cookie ERROR или блок COOKIE ACK, пакет следует отбросить без уведомления.
8) В остальных случаях получателю следует ответить отправителю пакета OOTB пакетом с блоком ABORT. При передаче блока ABORT получатель пакета OOTB **должен** указать в поле Verification Tag значение Verification Tag из пакета OOTB и установить бит T в поле флагов для индикации отсутствия порядкового номера TCB. После передачи блока ABORT получатель пакета OOTB должен отбросить этот пакет, не выполняя каких-либо дополнительных действий.

## 8.5 Правила для тегов верификации

Правила Verification Tag, определённые в этом параграфе, применяются для передачи и приёма пакетов SCTP, не содержащих блоков INIT, SHUTDOWN COMPLETE, COOKIE ECHO (см. параграф 5.1), ABORT или SHUTDOWN ACK. Для перечисленных блоков правила рассмотрены отдельно в параграфе 8.5.1.

При передаче пакета SCTP конечная точка **должна** заполнить поле Verification Tag, указав в нем значение параметра Initiate Tag из полученного от партнёра. блока INIT или INIT ACK.

При получении пакета SCTP конечная точка **должна** проверить соответствие полученного значения Verification Tag своему значению тега. Если полученное значение Verification Tag не соответствует тегу локальной точки, получателю следует отбросить пакет без уведомления и не выполнять каких-либо дополнительных операций за исключением случаев, описанных в параграфе 8.5.1.

### *8.5.1 Исключения из правил для Verification Tag*

A) Правила для пакетов с блоками INIT:

- Отправитель **должен** установить Verification Tag = 0.
- Когда конечная точка получает пакет SCTP с Verification Tag = 0, ей следует убедиться, что пакет содержит только блок INIT. В противном случае пакет **должен** быть отброшен без уведомления.

B) Правила для пакетов с блоками ABORT:

- Конечная точка всегда должна указывать в поле Verification Tag передаваемых пакетов значение тега адресата, если этот тег известен.
- Если блок ABORT передаётся в ответ на пакет OOTB, конечная точка **должна** выполнить процедуру, описанную в параграфе 8.4.

[1] out of the blue

- Получатель блока ABORT **должен** воспринять пакет, если значение Verification Tag соответствует его собственному тегу **или** тегу партнёра. В противном случае пакет **должен** отбрасываться без уведомления и выполнения каких-либо других действий.

C) Правила для пакетов с блоками SHUTDOWN COMPLETE:

- При передаче блока SHUTDOWN COMPLETE, если получатель блока SHUTDOWN ACK имеет значение TCB, в поле верификации **должен** использоваться тег адресата. При отсутствии TCB отправителю следует использовать значение Verification Tag из блока SHUTDOWN ACK.
- Получатель SHUTDOWN COMPLETE должен воспринять пакет, если поле Verification Tag соответствует его собственному тегу **или** тегу его партнёра. и в поле флагов установлен бит T. В противном случае получатель **должен** отбрасывать пакет без уведомления и выполнения каких-либо иных действий. Конечная точка **должна** игнорировать SHUTDOWN COMPLETE, если она не находится в состоянии SHUTDOWN-ACK-SENT.

D) Правила для пакетов с блоками COOKIE ECHO

- При передаче COOKIE ECHO конечная точка **должна** использовать значение Initial Tag из принятого блока INIT ACK.
- Получателю COOKIE ECHO следует выполнить процедуры, описанные в разделе 5.

E) Правила для пакетов с блоками SHUTDOWN ACK

- Если получатель находится в состоянии COOKIE-ECHOED или COOKIE-WAIT, **следует** выполнить процедуры, описанные в параграфе 8.4 (т. е., пакет должен трактоваться как OOTB).

# *9. Прекращение работы ассоциации*

Конечной точке следует прерывать ассоциацию, когда сервис более не требуется. Ассоциация может быть прервана путём разрыва (abort) или завершения (shutdown). Разрыв ассоциации представляет собой прекращение всякой передачи данных с отбрасыванием всех оставшихся не доставленными данных. Завершение ассоциации представляет собой процесс контролируемого прекращения обмена данными с доставкой партнёру всех данных, остающихся в очередях. Однако в случае завершения (shutdown) протокол SCTP не поддерживает полуоткрытых состояний (как в TCP), когда одна сторона может продолжать передачу данных в то время, как другая уже закрыла ассоциацию. Когда конечная точка выполняет процедуру завершения, обе стороны ассоциации будут прекращать приём новых данных от пользователя и доставлять только данные, которые находились в очередях на момент приёма или передачи блока SHUTDOWN.

## 9.1 Разрыв ассоциации (Abort)

Когда конечная точка принимает решение о разрыве существующей ассоциации, она должна передать своему партнёру блок ABORT. Отправитель **должен** включить значение Verification Tag своего партнёра. в исходящий пакет. **Недопустимо** группировать блок ABORT с блоками DATA.

Для конечной точки **недопустима** передача откликов на любой принятый пакет, содержащий блок ABORT (см. также параграф 8.4).

Конечная точка, получившая блок ABORT, должна выполнить специальную проверку Verification Tag в соответствии с правилами параграфа 8.5.1.

После проверки Verification Tag принявшая блок конечная точка должна удалить ассоциацию из своих записей и сообщить о разрыве ассоциации на вышележащий уровень.

## 9.2 Завершение ассоциации (Shutdown)

Используя примитив SHUTDOWN (параграф 10.1), вышележащий уровень конечной точки ассоциации может выполнить аккуратное завершение работы ассоциации[1]. В этом случае все остающиеся блоки DATA от партнёра., инициировавшего завершение ассоциации, будут доставлены до завершения работы.

При получении от вышележащего уровня примитива SHUTDOWN конечная точка переходит в состояние SHUTDOWN-PENDING и продолжает находиться в этом состоянии, пока все остающиеся данные не будут подтверждены партнёром. Конечная точка не принимает новых данных от вышележащего уровня, но будет повторять передачу данных удалённому партнёру, если в этом возникает необходимость (заполнение пропуска в порядковых номерах).

После того, как все остающиеся в сети данные будут подтверждены партнёром., конечная точка должна передать своему партнёру блок SHUTDOWN, содержащий в поле Cumulative TSN Ack последний порядковый номер TSN, полученный от партнёра. После этого конечная точка должна запустить таймер T2-shutdown и перейти в состояние SHUTDOWN-SENT. По завершении отсчёта таймера конечная точка должна повторно передать блок SHUTDOWN с обновлённым значением последнего порядкового номера TSN, полученного от партнёра.

**Должны** быть выполнены правила параграфа 6.3 для определения корректного значения таймера T2-shutdown. Для индикации пропусков в порядковых номерах TSN конечная точка может группировать SACK и блок SHUTDOWN в одном пакете SCTP.

Конечной точке следует ограничивать число повторов передачи блока SHUTDOWN с помощью протокольного параметра Association.Max.Retrans. После превышения этого порога конечной точке следует уничтожить TCB и **обязательно** передать информацию о недоступности партнёра. на вышележащий уровень (тем самым ассоциация переводится в состояние CLOSED). При получении любых пакетов от партнёра. (блоки DATA из очереди) следует сбрасывать счётчик повтора передачи и заново запускать таймер T2-Shutdown, давая партнёру дополнительную возможность передачи всех остающихся блоков DATA из его очередей.

При получении блока SHUTDOWN конечная точка должна:

[1] Разрыв соединений. *Прим. перев.*

- перейти в состояние SHUTDOWN-RECEIVED;
- прекратить приём новых данных от своего пользователя SCTP;
- убедиться путём проверки поля Cumulative TSN Ack, что все блоки DATA приняты отправителем блока SHUTDOWN.

После перехода конечной точки в состояние SHUTDOWN-RECEIVED для неё **недопустимо** передавать SHUTDOWN в ответ на запрос ULP и следует отбрасывать последующие блоки SHUTDOWN.

При наличии остающихся блоков DATA получатель SHUTDOWN должен продолжать нормальные процедуры передачи, описанные в главе 6, пока все остающиеся блоки DATA не будут подтверждены; однако для получателя блока SHUTDOWN **недопустимо** принимать новые данные от своего пользователя SCTP.

Находясь в состоянии SHUTDOWN-SENT, отправитель блока SHUTDOWN **должен** незамедлительно передавать в ответ на каждый принятый пакет, содержащий хотя бы один блок DATA, подтверждение SACK и блок SHUTDOWN, а также заново запускать таймер T2-shutdown. При наличии остающихся блоков DATA получатель блока SHUTDOWN должен передать блок SHUTDOWN ACK, запустить таймер T2-shutdown и перейти в состояние SHUTDOWN-ACK-SENT. По завершении отсчёта таймера конечная точка должна повторить передачу блока SHUTDOWN ACK.

Отправителю SHUTDOWN ACK следует ограничивать число повторов передачи SHUTDOWN ACK с помощью протокольного параметра Association.Max.Retrans. После превышения заданного порога конечной точке следует уничтожить TCB и можно сообщить вышележащему уровню о недоступности партнёра. (тем самым ассоциация переводится в состояние CLOSED).

При получении блока SHUTDOWN ACK отправитель SHUTDOWN должен остановить таймер T2-shutdown, передать своему партнёру блок SHUTDOWN COMPLETE и удалить все записи для данной ассоциации.

При получении блока SHUTDOWN COMPLETE конечная точка будет проверять, что она находится в состоянии SHUTDOWN-ACK-SENT и отбрасывать полученный блок, если состояние отличается от указанного. Если же конечная точка находится в состоянии SHUTDOWN-ACK-SENT, ей следует остановить таймер T2-shutdown и удалить все связанные с ассоциацией записи (тем самым ассоциация переводится в состояние CLOSED).

Конечной точке перед началом процедуры завершения ассоциации **следует** удостовериться, что все остающиеся блоки DATA были подтверждены.

Конечной точке, находящейся в состоянии SHUTDOWN-PENDING, SHUTDOWN-SENT, SHUTDOWN-RECEIVED или SHUTDOWN-ACK-SENT, следует отвергать все новые запросы данных от вышележащего уровня.

Если конечная точка, находящаяся в состоянии SHUTDOWN-ACK-SENT, получает блок INIT (например, при утере блока SHUTDOWN COMPLETE) с транспортными адресами отправителя и получателя (в заголовке IP или блоке INIT), относящимися к данной ассоциации, ей следует отбросить блок INIT и повторить передачу блока SHUTDOWN ACK.

Примечание. Получение блока INIT с совпадающими адресами IP, но другим номером порта говорит о попытке создания новой ассоциации.

Отправителю блока INIT или COOKIE ECHO следует отвечать на получение блока SHUTDOWN-ACK пакетом SCTP, содержащим только блок SHUTDOWN COMPLETE, а в поле Verification Tag общего заголовка следует включать значение тега из полученного пакета SHUTDOWN ACK. Такой пакет рассматривается как OOTB (см. параграф 8.4). Отправитель INIT оставляет работать свой таймер T1-init и сохраняет состояние COOKIE-WAIT или COOKIE-ECHOED. Завершение отсчёта таймера T1-init будет приводить к повтору передачи блока INIT или COOKIE и последующему созданию новой ассоциации.

Если получатель блока SHUTDOWN находится в состоянии COOKIE WAIT или COOKIE ECHOED, блок SHUTDOWN **следует** отбрасывать без уведомления.

Если конечная точка находится в состоянии SHUTDOWN-SENT и получает от своего партнёра. блок SHUTDOWN, ей следует незамедлительно ответить блоком SHUTDOWN ACK и перейти в состояние SHUTDOWN-ACK-SENT с перезапуском своего таймера T2-shutdown.

Если конечная точка находится в состоянии SHUTDOWN-ACK-SENT и получает от партнёра. блок SHUTDOWN ACK, она должна остановить таймер T2-shutdown, передать партнёру блок SHUTDOWN COMPLETE и удалить все связанные с ассоциацией записи.

# *10. Интерфейс с вышележащим уровнем*

Протоколы вышележащих уровней (ULP) должны запрашивать сервис путём передачи примитивов протоколу SCTP и получать уведомления о различных событиях от SCTP.

Примитивы и уведомления, описанные в этом разделе, следует рассматривать, как рекомендации для разработчиков SCTP. Приведённое ниже функциональное описание примитивов интерфейса с ULP служит для иллюстрации. Различные реализации протокола SCTP могут использовать разные интерфейсы с ULP. Однако все реализации SCTP должны обеспечивать некий минимальный сервис, гарантирующий что все реализации SCTP могут поддерживать одинаковую иерархию протоколов.

## 10.1 ULP -> SCTP

В последующих параграфах приведены функциональные характеристики интерфейса ULP-SCTP. Используемая при описании нотация похожа на описания вызовов процедур или функций в большинстве языков высокого уровня.

Примитивы ULP, описанные ниже, задают базовые функции, которые протокол SCTP должен выполнять для поддержки обмена данными между процессами. Та или иная реализация протокола может определять свой формат и поддерживать комбинации или подмножества базовых функций в одном вызове.

**A) Initialize - инициализация**

*Формат:* INITIALIZE ([local port], [local eligible address list]) -> имя локального экземпляра SCTP

Этот примитив позволяет протоколу SCTP инициализировать внутренние структуры данных и выделить ресурсы, требуемые для создания рабочей среды. После инициализации SCTP протокол ULP может напрямую обмениваться информацией с удалёнными конечными точками без повторного использования данного примитива.

SCTP будет возвращать ULP имя локального экземпляра SCTP.

*Обязательные атрибуты:*

Нет.

*Необязательные атрибуты:*

С примитивом могут передаваться следующие типы атрибутов:

- local port - номер порта SCTP, если ULP хочет задать порт;
- local eligible address list - список адресов, с которыми следует связать локальную точку SCTP. По умолчанию при отсутствии списка локальная точка связывается со всеми адресами IP, присвоенными данному хосту.

Примечание для разработчиков. Если реализация поддерживает этот атрибут, она принимает на себя ответственность за то, что в любом исходящем от данной точки пакете SCTP будет указан в поле отправителя адрес IP из заданного параметром списка.

**B) Associate - создать ассоциацию**

*Формат:* ASSOCIATE(local SCTP instance name, destination transport addr, outbound stream count)

-> association id [,destination transport addr list] [,outbound stream count]

Этот примитив позволяет инициировать создание ассоциации с указанным партнёром.

Конечная точка партнёра. по создаваемой ассоциации должна указываться одним из транспортных адресов данной точки (см. параграф 1.4). Если локальный экземпляр SCTP ещё не был инициализирована, вызов ASSOCIATE рассматривается как ошибка.

При успешном создании ассоциации будет возвращаться идентификатор созданной ассоциации SCTP. Если SCTP не может создать ассоциацию с удаленным партнёром., возвращается код ошибки.

При создании ассоциации могут возвращаться другие параметры ассоциации, включая полные транспортные адреса партнёра., а также счётчик исходящих потоков локальной точки. Один из транспортных адресов удалённого партнёра. выбирается локальной точкой в качестве первичного (используемого по умолчанию) транспортного адреса для передачи пакетов этому партнёру. Возвращаемый список транспортных адресов партнёра. (destination transport addr list) может использоваться ULP для смены первичного пути или задания передачи по определённому транспортному адресу партнёра.

Примечание для разработчиков. Если примитив ASSOCIATE реализован, как блокируемый вызов функции, он может возвращать кроме идентификатора ассоциации дополнительные параметры. Если примитив ASSOCIATE реализован как неблокируемый вызов, возвращается только идентификатор ассоциации, а параметры созданной ассоциации передаются с использованием уведомления COMMUNICATION UP.

*Обязательные атрибуты:*

- local SCTP instance name - возвращается операцией INITIALIZE.
- destination transport addr - содержит один из транспортных адресов партнёра., с которым создаётся ассоциация.
- outbound stream count - число исходящих потоков, которые ULP может открывать для передачи данных удалённому партнёру.

*Необязательные атрибуты:*

Нет.

**C) Shutdown - завершение ассоциации**

*Формат:* SHUTDOWN(association id) -> результат

Завершает работу ассоциации с доставкой партнёру всех данных из локальных очередей. Ассоциация будет разрываться лишь после того, как будут подтверждены все переданные пакеты SCTP. При успешном завершении ассоциации будет возвращаться код завершения, а при отказе - код ошибки.

*Обязательные атрибуты:*

- association id - локальный идентификатор ассоциации SCTP.

*Необязательные атрибуты:*

Нет.

**D) Abort - разрыв ассоциации**

*Формат:* ABORT(association id [, cause code]) -> результат

Прерывает работу ассоциации без завершения процесса передачи данных из очередей. Все пользовательские данные из локальных очередей отбрасываются, а партнёру передаётся блок ABORT. При успешном разрыве ассоциации возвращается код разрыва, а при отказе - код ошибки.

*Обязательные атрибуты:*

- association id - локальный идентификатор ассоциации SCTP.

*Необязательные атрибуты:*

- cause code - причина разрыва ассоциации, передаваемая партнёру.

**E) Send - передача данных**

*Формат:* SEND(association id, buffer address, byte count [,context] [,stream id] [,life time] [,destination transport address] [,unorder flag] [,no-bundle flag] [,payload protocol-id] ) -> результат

Этот примитив обеспечивает основной метод передачи пользовательских данных по протоколу SCTP.

*Обязательные атрибуты:*

- association id - локальный идентификатор ассоциации SCTP;
- buffer address - адрес, по которому сохраняется передаваемое пользовательское сообщение;
- byte count - размер пользовательских данных в байтах.

*Необязательные атрибуты:*

- context - необязательное 32-битовое целое число, которое будет передаваться в уведомлении об отказе протоколу ULP, если при передаче пользовательского сообщения произойдёт ошибка;
- stream id - идентификатор потока, используемого для передачи данных (по умолчанию используется поток 0);
- life time - задаёт время жизни пользовательских данных. По истечении заданного времени SCTP уже не будет пытаться передавать эти данные. Параметр может использоваться для предотвращения передачи устаревших пользовательских сообщений. SCTP уведомляет ULP, если данные не удаётся передать с помощью SCTP в течение заданного этой переменной времени. Однако пользовательские данные будут передаваться, если протокол SCTP начал попытки передачи блока данных до истечения заданного времени.

  Примечание для разработчиков. Для более эффективной поддержки опции времени жизни данных передающая сторона может присваивать передаваемому блоку номер TSN в последний момент. Для упрощения реализации после присвоения номера TSN отправителю следует рассматривать передачу блока DATA как начавшееся событие, не принимая уже во внимание ограничение времени жизни, заданное для блока DATA.
- destination transport address - задаёт один из транспортных адресов получателя, по которому пакет должен передаваться. По возможности протоколу SCTP следует использовать заданный адрес получателя вместо адреса первичного пути.
- unorder flag - этот флаг указывает что пользовательские данные доставляются партнёру без соблюдения порядка (т. е., устанавливается флаг U = 1 во всех блоках DATA, содержащих это сообщение).
- no-bundle flag - указывает протоколу SCTP, что эти данные не должны группироваться с другими передаваемыми блоками DATA. SCTP **может** выполнять в целях предотвращения перегрузки группировку блоков, игнорируя этот флаг.
- payload protocol-id - 32-битовое целое число без знака, которое будет передаваться партнёру для индикации типа передаваемых данных. Это значение передаётся не обрабатывается протоколом SCTP.

**F) Set Primary - выбор основного пути**

*Формат:* SETPRIMARY(association id, destination transport address, [source transport address] ) -> результат

Падает для локальной точки SCTP использование указанного транспортного адреса в качестве первичного пути передачи пакетов.

Примитив должен возвращать результат попытки задания первичного пути. Если заданного адреса нет в списке destination transport address list, возвращённом ранее командой создания ассоциации или уведомлением COMMUNICATION UP, должен возвращаться код ошибки.

*Обязательные атрибуты:*

- association id - локальный идентификатор ассоциации SCTP;
- destination transport address - задаёт один из транспортных адресов партнёра. для использования в качестве основного пути передачи пакетов. Это значение заменяет собой адрес первичного пути, поддерживавшегося до этого локальной точкой SCTP.

*Необязательные атрибуты:*

- source transport address - некоторые реализации могут поддерживать задание адреса отправителя, который будет по умолчанию включаться во все исходящие дейтаграммы IP.

**G) Receive - приём данных**

*Формат:* RECEIVE(association id, buffer address, buffer size [,stream id])

-> byte count [,transport address] [,stream id] [,stream sequence number] [,partial flag] [,delivery number] [,payload protocol-id]

Этот примитив должен считывать первое пользовательское сообщение из входной очереди SCTP в буфер, указанный ULP, если такой буфер имеется. После выполнения команды возвращается размер прочитанных данных в байтах. В зависимости от реализации может также возвращаться такая информация, как адрес отправителя, идентификатор потока, из которого получены данные, сведения о наличии дополнительных данных для прочтения и т. п. Для упорядоченных сообщений может также возвращаться порядковый номер в потоке.

В зависимости от реализации вызов данного примитива в момент отсутствия данных для чтения будет возвращать уведомление об отсутствии данных и блокировать вызванный процесс пока данные не поступят.

*Обязательные атрибуты:*

- association id - локальный идентификатор ассоциации SCTP;
- buffer address - адрес в памяти, указанный ULP для считывания сообщения.
- buffer size - максимальный размер считываемых данных в байтах.

*Необязательные атрибуты:*

- stream id - для индикации потока, из которого были получены данные.
- stream sequence number - порядковый номер в потоке, присвоенный передающим партнёром. SCTP.
- partial flag - наличие этого флага говорит о том, что данный вызов Receive возвращает часть данных из сообщения. Установка данного флага **должна** сопровождаться возвратом идентификатора потока и порядкового номера в потоке. Нулевое значение флага показывает, что для данного порядкового номера в потоке больше не будет доставлено данных[1].
- payload protocol-id - 32-битовое целое число без знака, полученное от партнёра. и указывающее тип данных в принятом сообщении. Значение этого параметра передаётся не обрабатывается протоколом SCTP.

**H) Status - статус**

*Формат:* STATUS(association id) -> данные о состоянии

Этот примитив должен возвращать блок данных, содержащий следующую информацию:

- состояние соединения для ассоциации;
- список адресов транспортного уровня для получателя;
- состояния доступности транспортных адресов получателя;
- текущий размер окна приёма;
- текущий размер окна насыщения;
- число неподтвержденных блоков DATA;
- число блоков DATA ожидающих приёма;
- основной путь;
- самое новое значение SRTT на основном пути;
- значение RTO для основного пути;
- значения SRTT и RTO для других путей и т. п.

*Обязательные атрибуты:*

- association id - локальный идентификатор ассоциации SCTP.

*Необязательные атрибуты:*

Нет.

**I) Change Heartbeat - смена режима HeartBeat**

*Формат:* CHANGEHEARTBEAT(association id, destination transport address, new state [,interval]) -> результат

Говорит локальной точке о необходимости включить или отключить функцию heartbeat для указанного адреса получателя, возвращая результат операции.

Примечание. Даже при включённой функции heartbeat реальных проверок может не выполняться, если указанный адрес не относится к бездействующим.

*Обязательные атрибуты:*

- association id - локальный идентификатор ассоциации SCTP;
- destination transport address - один из транспортных адресов партнёра.;
- new state - новое состояние heartbeat для данного транспортного адреса (enabled или disabled).

*Необязательные атрибуты:*

- interval - этот параметр определяет частоту передачи heartbeat, если эта функция включена для транспортного адреса партнёра. Значение этого параметра добавляется к RTO транспортного адреса. Данный параметр имеет значение для всех транспортных адресов получателя.

**J) Request HeartBeat - запрос на выполнение HeartBeat**

*Формат:* REQUESTHEARTBEAT(association id, destination transport address) -> результат

[1]Сообщение прочитано целиком. *Прим. перев.*

Говорит локальной точке о необходимости выполнения HeartBeat для указанного транспортного адреса в данной ассоциации. Возвращаемый результат должен показывать, была ли передача блока HEARTBEAT по заданному транспортному адресу успешной.

*Обязательные атрибуты:*

- association id - локальный идентификатор ассоциации SCTP;
- destination transport address - транспортный адрес, для которого запрашивается передача блока HEARTBEAT.

**K) Get SRTT Report - запрос значения SRTT**

*Формат:* GETSRTTREPORT(association id, destination transport address) -> результат srtt

Запрашивает у локального SCTP результаты текущего измерения SRTT для указанного транспортного адреса в данной ассоциации. Возвращаемое значение может быть целым числом, указывающим последнее измеренное значение SRTT в миллисекундах.

*Обязательные атрибуты:*

- association id - локальный идентификатор ассоциации SCTP;
- destination transport address - транспортный адрес партнёра. в данной ассоциации, для которого определяется значение SRTT.

**L) Set Failure Threshold - задать порог детектирования отказа**

*Формат:* SETFAILURETHRESHOLD(association id, destination transport address, failure threshold) -> результат

Этот примитив позволяет локальному модулю SCTP задать значение порога детектирования отказа Path.Max.Retrans для заданного транспортного адреса.

*Обязательные атрибуты:*

- association id - локальный идентификатор ассоциации SCTP;
- destination transport address - транспортный адрес партнёра. в данной ассоциации, для которого задаётся порог;
- failure threshold - новое значение параметра Path.Max.Retrans для заданного транспортного адреса.

**M) Set Protocol Parameters - установить параметры протокола**

*Формат:* SETPROTOCOLPARAMETERS(association id, [,destination transport address,] protocol parameter list) -> результат

Этот примитив позволяет локальному модулю установить параметры протокола.

*Обязательные атрибуты:*

- association id - локальный идентификатор ассоциации SCTP;
- protocol parameter list - список имён и значений протокольных параметров (например, Association.Max.Retrans), которые будут установлены для протокола SCTP (см. главу 14).

*Необязательные атрибуты:*

- destination transport address - некоторые параметры протокола могут независимо устанавливаться для каждого транспортного адреса партнёра.

**N) Receive unsent message - получить неотправленное сообщение**

*Формат:* RECEIVE_UNSENT(data retrieval id, buffer address, buffer size

[,stream id] [, stream sequence number] [,partial flag] [,payload protocol-id])

- data retrieval id - идентификатор, передаваемый ULP в уведомлении об отказе.
- buffer address - адрес буфера, указанный ULP для записи полученного сообщения.
- buffer size - максимальный размер принимаемых данных в байтах.

*Необязательные атрибуты:*

- stream id - это возвращаемое значение указывает идентификатор потока, в который были переданы данные.
- stream sequence number - это возвращаемое значение указывает порядковый номер в потоке, связанный с сообщением.
- partial flag - этот возвращаемый флаг указывает на частичную доставку сообщения. При установке этого флага **должны** также возвращаться идентификатор потока и порядковый номер в потоке. Нулевое значение флага показывает, что для данного порядкового номера в потоке больше не будет доставлено данных.
- payload protocol-id - это 32-битовое целое число без знака, которое было передано для идентификации типа полученных данных.

**O) Receive unacknowledged message - получить неподтвержденное сообщение**

*Формат:* RECEIVE_UNACKED(data retrieval id, buffer address, buffer size,

[,stream id] [, stream sequence number] [,partial flag] [,payload protocol-id])

- data retrieval id - идентификатор, передаваемый ULP в уведомлении об отказе.
- buffer address - адрес буфера, указанный ULP для записи полученного сообщения.
- buffer size - максимальный размер принимаемых данных в байтах.

*Необязательные атрибуты:*

- stream id - это возвращаемое значение указывает идентификатор потока, в который были переданы данные.
- stream sequence number - это возвращаемое значение указывает порядковый номер в потоке, связанный с сообщением.
- partial flag - этот возвращаемый флаг указывает на частичную доставку сообщения. При установке этого флага **должны** также возвращаться идентификатор потока и порядковый номер в потоке. Нулевое значение флага показывает, что для данного порядкового номера в потоке больше не будет доставлено данных.
- payload protocol-id - это 32-битовое целое число без знака, которое было передано для идентификации типа полученных данных.

**P) Destroy SCTP instance - уничтожить экземпляр SCTP**

*Формат:* DESTROY(local SCTP instance name)

- local SCTP instance name - значение, переданное приложению из примитива инициализации; указывает уничтожаемый экземпляр SCTP.

## 10.2 SCTP -> ULP

Предполагается, что операционная система или прикладная среда обеспечивает механизм асинхронной передачи сигналов SCTP процессу ULP. Когда SCTP подаёт сигнал процессу ULP на вышележащий уровень передаётся та или иная информация.

Примечание для разработчиков. В некоторых случаях передача на верхний уровень может выполняться через отдельный сокет или канал вывода ошибок.

**A) уведомление DATA ARRIVE**

Протокол SCTP должен передавать такое уведомление ULP в тех случаях, когда пользовательское сообщение получено и готово для считывания.

Уведомление может включать следующие необязательные параметры:

- association id - локальный идентификатор ассоциации SCTP;
- stream id - идентификатор потока, в котором были получены данные.

**B) уведомление SEND FAILURE**

Если сообщение не может быть доставлено, протокол SCTP должен передать это уведомление ULP.

Уведомление может включать следующие необязательные параметры:

- association id - локальный идентификатор ассоциации SCTP;
- data retrieval id - идентификатор, используемый для считывание неотправленных или неподтвержденных данных.
- cause code - указывает причину отказа (например, слишком большой размер сообщения, завершение срока жизни сообщения и т. п.).
- context - дополнительная информация, связанная с сообщением (см. D в параграфе 10.1).

**C) уведомление NETWORK STATUS CHANGE**

Когда транспортный адрес получателя помечается как неактивный (например, при детектировании отказа) или, наоборот, становится активным (например, при детектировании восстановления), протоколу SCTP следует передать такое уведомление ULP.

В уведомлении должна содержаться следующая информация:

- association id - локальный идентификатор ассоциации SCTP;
- destination transport address - указывает транспортный адрес партнёра. для которого зафиксировано изменение состояния;
- new-status - указывает новое состояние.

**D) уведомление COMMUNICATION UP**

Этот тип уведомлений используется для индикации готовности протокола SCTP к приёму или передаче пользовательских сообщений, а также после восстановления разорванной связи с удалённой точкой.

Примечание для разработчиков. Если примитив ASSOCIATE реализован, как блокируемый вызов функции, параметры ассоциации возвращаются самим примитивом ASSOCIATE. В таких случаях на стороне инициатора ассоциации уведомления COMMUNICATION UP становятся необязательными.

В уведомлении должна содержаться следующая информация:

- association id - локальный идентификатор ассоциации SCTP;

- status - указывает тип события, с которым связано уведомление;
- destination transport address list - полный набор транспортных адресов партнёра.;
- outbound stream count - максимальное число исходящих потоков, которые ULP может использовать в данной ассоциации;
- inbound stream count - число потоков, запрошенных партнёром. (это значение может отличаться от outbound stream count).

**E) уведомление COMMUNICATION LOST**

Это уведомление должно передаваться протоколом SCTP в случае полной утраты связи с удалённой точкой (например, обнаруженной с помощью Heartbeat) или выполнения удалённой точкой операции прерывания ассоциации (abort).

В уведомлении должна содержаться следующая информация:

- association id - локальный идентификатор ассоциации SCTP;
- status - указывает тип события, с которым связано уведомление; этот параметр может говорить об отказе **или** обычном прекращении работы ассоциации с помощью запроса shutdown или abort.

Уведомление может включать следующие необязательные параметры:

- data retrieval id - идентификатор, используемый для считывание неотправленных или неподтвержденных данных;
- Last-acked - последний номер TSN, подтверждённый партнёром.;
- last-sent - последний номер TSN, переданный партнёру.

**F) уведомление COMMUNICATION ERROR**

Когда SCTP получает блок ERROR от своего партнёра. и решает уведомить об ошибке ULP, этот тип служит для передачи уведомления.

Уведомление может включать следующие параметры:

- association id - локальный идентификатор ассоциации SCTP;
- error info - указывает тип ошибки и может содержать дополнительную информацию, полученную из блока ERROR.

**G) уведомление RESTART**

При обнаружении рестарта на стороне партнёра. SCTP может использовать этот тип для передачи уведомления ULP.

Уведомление может включать следующие параметры:

- association id - локальный идентификатор ассоциации SCTP;

**H) уведомление SHUTDOWN COMPLETE**

Это уведомление SCTP передаёт на вышележащий уровень при завершении процедуры (параграф 9.2).

Уведомление может включать следующие параметры:

- association id - локальный идентификатор ассоциации SCTP;

# *11. Вопросы безопасности*

## 11.1 Цели защиты

Для протокола транспортного уровня общего назначения, разработанных для гарантированной доставки чувствительной к задержкам информации (например, сигнальных сообщений телефонных систем или данных билинга) между парами точек сети важны следующие аспекты, связанные с безопасностью:

- доступность сервиса с гарантией своевременной доставки;
- целостность пользовательской информации, передаваемой через SCTP.

## 11.2 Реакция SCTP на потенциальные угрозы

Протокол SCTP может использоваться в различных системах, связанных с риском. Для операторов систем, использующих SCTP, важен анализ конкретной среды для принятия соответствующих мер безопасности.

Операторам систем, использующих SCTP следует использовать [RFC2196] в качестве руководства по обеспечению безопасности своего сайта.

### *11.2.1 Чет возможности атак изнутри*

Следует использовать принципы, изложенные в [RFC2196] для минимизации риска утечки информации или саботажа со стороны сотрудников. Такие процедуры включают публикацию политики безопасности, контроль доступа к оборудованию, программам и сети, а также разделение служб.

### *11.2.2 Защита от повреждения данных в сети*

Если риск возникновения недетектируемых ошибок в дейтаграммах, доставляемых с помощью транспорта нижележащих уровней, слишком велик, требуются дополнительные меры по обеспечению целостности данных. Если эта дополнительная защита обеспечивается на прикладном уровне, заголовок SCTP продолжает оставаться уязвимым

для преднамеренных атак с целью повреждения данных. Хотя существующих механизмов детектирования подмены пакетов в SCTP вполне достаточно для работы в нормальных условиях, требуется более сильная защита SCTP в тех случаях, когда рабочая среда характеризуется высоким уровнем риска преднамеренных атак со стороны изощрённых противников.

Для обеспечения повторного использования кода, а также для предотвращения попыток изобрести велосипед и избыточной сложности SCTP **следует** использовать заголовки аутентификации IP AH[1] [RFC2402] SHOULD, если опасная среда требует более сильной защиты целостности, но не требует конфиденциальности.

Существует широко распространённое расширение BSD Sockets API для приложений, которым требуется сервис IP типа AH или ESP на уровне ядра операционной системы. Приложения могут использовать этот API для использования AH в тех случаях, когда это возможно.

## *11.2.3 Сохранение конфиденциальности*

В большинстве случаев вопросы конфиденциальности относятся к данным, передающим сигнальную информацию, а не к заголовкам SCTP или протоколов нижележащих уровней. В этом случае можно ограничиться шифрованием пользовательских данных SCTP. Как и дополнительные контрольные суммы, шифрование данных **может** выполняться пользовательским приложением SCTP.

В дополнение к этому пользовательское приложение **может** использовать специфические для реализации API для запроса сервиса IP ESP[2] [RFC2406], обеспечивающего конфиденциальность и целостность.

Практические требования конфиденциальности для мобильных пользователей могут включать маскирование адресов IP и номеров портов. В таких случаях **следует** использовать ESP вместо обеспечения конфиденциальности на уровне приложений. При использовании ESP для обеспечения конфиденциальности трафика SCTP **должно** применяться криптографическое преобразование ESP, которое включает криптографическую защиту целостности, поскольку в таких случаях кроме угрозы конфиденциальности данных имеется достаточно серьёзная угроза их целостности.

При использовании ESP шифрование данных на прикладном уровне в общем случае не требуется.

Независимо от способа обеспечения конфиденциальности **следует** использовать механизмы ISAKMP [RFC2408] и IKE[3] [RFC2409] для управления ключами.

Операторам следует обратиться к [RFC2401] для получения дополнительной информации по средствам обеспечения безопасности непосредственно поверх уровня IP.

## *11.2.4 Защита от атак на службы вслепую (Blind DoS)*

Атака вслепую представляет собой случай, когда атакующий не может перехватывать и просматривать содержимое потока данных, передаваемых узлу SCTP или от него. Атаки вслепую на службы могут иметь форму лавинной рассылки (flooding), маскирования (masquerade) или недозволенного монопольного захвата сервиса.

### 11.2.4.1 Лавинная атака (Flooding)

Задачей лавинной атаки является выведение сервиса из строя и некорректное поведение системы путём истощения ресурсов, интерференции с другими легитимными транзакциями или использования связанных с буферами программных ошибок. Лавинная атака может быть направлена на узел SCTP или каналы доступа. Лавинная атака может приводить к недоступности сервиса даже в случаях использования межсетевых экранов для защиты.

В общем случае защита от лавинной атаки начинается при разработке оборудования и включает такие меры, как:

- предотвращение выделения ограниченных ресурсов до проверки легитимности сервисного запроса;
- предоставление более высокого приоритета уже выполняющимся процессам по сравнению с новыми;
- идентификация и удаление дубликатов и просроченных запросов из очередей;
- игнорирование неожиданных пакетов, направленных по адресам, не являющимся индивидуальными (non-unicast).

Сетевое оборудование должно обеспечивать возможность генерации сигналов тревоги и записи в журнальные файлы сведений о подозрительных всплесках трафика. В журнальный файл следует включать информацию, которая позволит идентифицировать входящий канал и адреса отправителей, которые могут помочь администратору сети или SCTP принять адекватные меры по защите. Следует разработать для операторов процедуры, описывающие действия по сигналу тревоги в тех случаях, когда картина атаки достаточно ясна.

Протокол SCTP устойчив к лавинным атакам в частности, благодаря использованию четырехэтапного согласования при старте ассоциации, механизма cookie для блокирования ресурсов отвечающего узла до тех пор, пока не будет завершено согласование, и тегов верификации (Verification Tag) для защиты от вставки дополнительных пакетов в поток данных действующей ассоциации.

Механизмы IP AH и ESP могут также оказаться полезными для снижения риска при некоторых типах атак на службы.

Параметр Host Name в блоках INIT можно использовать для лавинной атаки на сервер DNS. Большое количество backlog-запросов к DNS для преобразования Host Name из блоков INIT в IP-адреса может быть вызвано передачей множества запросов INIT в один домен. Кроме того, атакующий может воспользоваться Host Name для непрямой атаки на сторонний сервер, рассылая большое число блоков INIT с именем хоста-жертвы по случайно выбранным адресам. Помимо увеличения нагрузки на DNS, такая атака может привести к генерации и передаче большого числа блоков INIT ACK по адресу атакуемого хоста. Одним из способов защиты от этого типа атак является проверка наличия в числе адресов IP, полученных от DNS, IP-адреса отправителя исходного блока INIT. Если список адресов IP, полученных от сервера DNS, не включает IP-адрес отправителя INIT, конечная точка **может** отбрасывать блок INIT без уведомления. Отметим, что эта мера не защищает от атак на DNS.

[1]Authentication Header.
[2]Encapsulating Security Payload.
[3]Internet Key Exchange.

#### 11.2.4.2 Слепое маскирование

Маскирование (Masquerade) может использоваться для атак на службы несколькими способами:

- путём связывания ресурсов целевого узла SCTP, к которому ролевой (impersonated) узел имеет ограниченный доступ. Например, политика целевого узла может разрешать не более одной ассоциации SCTP с ролевого узла SCTP. Замаскированный атакующий может попытаться организовать ассоциацию и прикинуться ролевым узлом, чтобы впоследствии настоящий ролевой узел не мог подключиться к сервису.
- посредством преднамеренного обнаружения подмены с целью провоцирования ролевого узла на блокирование целевого узла SCTP.
- путём взаимодействия с существующей ассоциацией посредством вставки в поток добавочной информации (например, запроса SHUTDOWN).

SCTP снижает риск атак с помощью слепого маскирования с подменой адресов (IP spoofing) за счёт использования четырехэтапного согласования при старте ассоциации. Маскирование с человеком в центре (Man-in-the-middle masquerade attack) рассматривается ниже в параграфе 11.3. Поскольку начальный обмен не запоминается, при атаках с маскированием вслепую нет возможности использования механизмов захвата. Кроме того, подтверждение INIT ACK, содержащее State Cookie, передаётся по адресу IP, с которого был получен блок INIT. Таким образом атакующий не получит блок INIT ACK, содержащий State Cookie. SCTP обеспечивает защиту от вставки дополнительных пакетов в поток данных существующей ассоциации за счёт использования тегов верификации (Verification Tag).

Протоколирование полученных запросов INIT и аномалий типа неожиданных блоков INIT ACK может быть полезно в целях обнаружения враждебных действий. Однако пользу от такого протоколирования нужно сопоставить с усложнением обработки при старте ассоциации SCTP, которое к тому же делает узел SCTP более уязвимым к лавинным атакам. Протоколирование не имеет смысла без создания рабочих процедур повседневного просмотра и анализа журнальных файлов.

#### 11.2.4.3 Неправомерная монополизация

Атаки этого типа выполняются злоумышленником открыто с использованием легитимного доступа. Они направлены против других пользователей узла SCTP или ресурсов, разделяемых между атакующим и целевым узлом. Возможные атаки включают создание большого числа ассоциаций между атакующим узлом и объектом атаки или перенос больших объёмов данных в рамках легитимных ассоциаций.

Следует вводить административные ограничения на число ассоциаций, создаваемых узлами. Пользовательским приложениям SCTP следует обеспечивать возможность детектирования передачи больших объёмов информации или сообщений no-op в данной ассоциации, а также другие механизмы протоколирования и разрыва ассоциаций в соответствии с принятой локальной политикой.

## 11.3 Защита от мошенничества и отказов от оплаты

Целью мошенничества является получение доступа к сервису без проверки полномочий и, в частности, без оплаты услуг. Для достижения своей цели атакующий должен склонить пользовательское приложение SCTP[1] на атакуемом узле SCTP обеспечить желаемый сервис, несмотря на подставные билинговые данные или отказ при сборе учётной информации. Отказ от оплаты является близкой проблемой, поскольку он может быть частью мошеннических действий или просто сохранением отказывающейся стороной неверной информации о предоставленных ей услугах.

Потенциальные мошеннические атаки включают перехват и последующее использование сведений, передающихся при авторизации (например, номеров кредитных карт), маскирование и повтор вслепую, а также атаки с человеком в центре, в которых пакеты, проходящие через ассоциацию SCTP подменяются в реальном масштабе времени.

Атаки, использующие перехват, можно предотвратить за счёт средств обеспечения конфиденциальности, описанных в параграфе 11.2.3.

Параграф 11.2.4.2 описывает устойчивость SCTP к атакам с маскированием вслепую, обеспечиваемую 4-этапным согласованием при старте и тегами верификации. Verification Tag вместе с номерами TSN обеспечивают защиту от атак с повтором вслепую (blind replay), где данные помещаются в существующую ассоциацию.

Однако SCTP не обеспечивает защиты от атак с человеком в центре (man-in-the-middle), где атакующий может перехватывать и изменять пакеты, передаваемые через ассоциацию. Например, блок INIT ACK включает информацию, которую злоумышленник на пути передачи может перехватить из существующей ассоциации SCTP. Там, где существует возможность организации таких атак или связанный с этим отказ от оплаты, рекомендуется использовать IPSEC AH для обеспечения целостности и аутентичности передаваемых пакетов SCTP.

SCTP также не обеспечивает защиты от атак, начинающихся на узле SCTP или за ним и происходящих в контексте существующей ассоциации. Предотвращение таких атак является одной из задач политики безопасности для хоста, упомянутой в параграфе 11.2.1.

# *12. Рекомендуемые параметры TCB*

В этом разделе описываются рекомендуемые параметры, которые должны содержаться в TCB. Раздел имеет иллюстративное значение и его содержимое не следует трактовать, как требования к реализациям или исчерпывающий список всех параметров, включаемых в SCTP TCB. Конкретной реализации для обеспечения оптимальной работы может потребоваться свой набор параметров.

## 12.1 Параметры, требуемые для экземпляра SCTP

**Associations - ассоциации**: список существующих ассоциаций и отображений потребителей данных для каждой ассоциации. Информация может храниться в форме хэш-таблицы или ином формате, определяемом реализацией. В качестве потребителей данных могут указываться информационные идентификаторы процессов (например, файловые

[1]Сервер, предоставляющий в сеть те или иные услуги. *Прим. перев.*

дескрипторы, указатели на именованные каналы или таблицы указателей) в зависимости от конкретной реализации SCTP.

**Secret Key - закрытый ключ**: закрытый ключ используется данной конечной точкой для расчёта MAC. В качестве ключа **следует** использовать случайное число криптографического качества и достаточной длины. Для выбора ключей могут быть полезны рекомендации [RFC1750].

**Address List - список адресов**: список адресов IP, с которыми связан данный экземпляр. Эта информация передаётся партнёру в блоках INIT и INIT ACK.

**SCTP Port - номер порта**: локальный номер порта, с которым связана конечная точка SCTP.

## 12.2 Параметры, требуемые для ассоциации в целом (например, TCB)

**Peer Verification Tag**: значение тега партнёра., передаваемое в каждом пакете и полученное из блока INIT или INIT ACK.

**My Verification Tag**: значение тега, ожидаемое в каждом входящем пакете и передаваемое партнёру в блоке INIT или INIT ACK.

**State**: переменная, показывающая в каком состоянии находится ассоциация (COOKIE-WAIT, COOKIE-ECHOED, ESTABLISHED, SHUTDOWN-PENDING, SHUTDOWN-SENT, SHUTDOWN-RECEIVED, SHUTDOWN-ACK-SENT).

Примечание. состояние CLOSED не указано в списке, поскольку для ассоциации в этом состоянии порядковый номер TCB **следует** удалять.

**Peer Transport Address List**: список транспортных адресов SCTP, с которыми связан партнёр. Эта информация извлекается из блоков INIT или INIT ACK и используется для связывания входящих пакетов с данной ассоциацией. Обычно эта информация хэшируется или индексируется (keyed) для быстрого поиска и доступа к TCB.

**Primary Path**: текущее значение первичного транспортного адреса партнёра. Это значение может также указывать адрес отправителя пакетов, получаемых этой точкой.

**Overall Error Count**: общий счётчик ошибок для всей ассоциации.

**Overall Error Threshold**: значение количества ошибок для ассоциации, при достижении которого данная ассоциация будет разорвана.

**Peer Rwnd**: текущее значение окна приёма (rwnd), рассчитанное для партнёра.

**Next TSN**: значение следующего номера TSN, которое будет присвоено новому блоку DATA. Начальный номер передаётся партнёру в блоке INIT или INIT ACK и далее номер увеличивается каждый раз, когда блоку DATA присваивается значение TSN (обычно непосредственно перед передачей или в процессе фрагментации).

**Last Rcvd TSN**: последний номер TSN, полученный с сохранением порядка. Начальное значение устанавливается на основе параметра Initial TSN, полученного от партнёра. в блоке INIT или INIT ACK, путём вычитания 1 из полученного значения.

**Mapping Array**: массив битов или байтов, показывающий какие упорядоченных номеров TSN были получены (относительно Last Rcvd TSN). При отсутствии пропусков (т. е., все пакеты получены с сохранением порядка доставки), этот массив может содержать только нулевые значения. Эта структура может использовать кольцевой буфер или битовый массив.

**Ack State**: это поле показывает, будет ли передано подтверждение SACK при получении следующего пакета. Начальное значение поля равно 0. При получении пакета значение увеличивается на 1. При достижении значения 2 или более в ответ на получение пакета передаётся подтверждение SACK и значение снова сбрасывается в 0. Отметим, что описанный механизм используется лишь при отсутствии нарушений в порядке доставки блоков DATA. Если имеется нарушение порядка доставки DATA, подтверждения SACK не задерживаются (см. раздел 6).

**Inbound Streams**: массив структур для отслеживания входящих потоков. Обычно данные о потоках включают следующий (ожидаемый) порядковый номер и могут включать номер потока.

**Outbound Streams**: массив структур для отслеживания исходящих потоков. Обычно данные о потоках включают следующий порядковый номер для передачи в поток.

**Reasm Queue**: очередь сборки фрагментов.

**Local Transport Address List**: список локальных адресов IP, связанных с данной ассоциацией.

**Association PMTU**: наименьшее значение PMTU среди всех путей к партнёру.

## 12.3 Данные для каждого транспортного адреса

Для каждого транспортного адреса партнёра. из списка, полученного в блоке INIT или INIT ACK, поддерживается группа параметров, включающая:

**Error count**: текущее значение счётчика ошибок для данного транспортного адреса.

**Error Threshold**: текущее значение порога для числа ошибок, связанных с данным транспортным адресом. При достижении этого порога транспортный адрес помечается как недоступный.

**cwnd**: текущий размер окна насыщения.

**ssthresh**: текущее значение порога Slow Start (ssthresh).

**RTO**: текущее значение тайм-аута для повтора передачи.

**SRTT**: текущее значение среднего времени кругового обхода.

**RTTVAR**: изменение текущего значения RTT.

**partial bytes acked**: метод слежения, используемый для увеличения размера cwnd в режиме предотвращения перегрузки (congestion avoidance, см. параграф 6.2.2).

**state**: текущее состояние адресата (DOWN, UP, ALLOW-HB, NO-HEARTBEAT и т. п.).

**PMTU**: текущее известное значение MTU для пути.

**Per Destination Timer**: таймер, используемый для каждого адреса получателя.

**RTO-Pending**: флаг, указывающий на то, что один из блоков DATA, переданных по этому адресу, в данный момент используется для расчёта RTT. Если флаг сброшен (0), ближайший блок DATA, отправляемый по этому адресу, следует использовать для расчёта RTT и установить данный флаг. При завершении расчёта RTT (получение подтверждения SACK для блока DATA) флаг сбрасывается.

**last-time used**: время передачи адресату последнего пакета. Это значение может использоваться для определения необходимости передачи HEARTBEAT.

## 12.4 Требуемые параметры общего назначения

**Out Queue**: очередь исходящих блоков DATA.

**In Queue**: очередь входящих блоков DATA.

# *13. Взаимодействие с IANA*

Данный протокол требует резервирования портов (как в TCP) для использования на широко используемых (well known) серверах в Internet. Все порты, зарезервированные для протокола TCP, автоматически резервируются и в пространстве портов SCTP. Запросы на выделение новых номеров должны следовать текущим механизмам IANA для выделения портов TCP.

Данный протокол может также расширяться по согласованию с IANA в трёх направлениях:

- определение новых типов блоков (chunk type),
- определение новых типов параметров,
- определение дополнительных кодов причин ошибок для блоков ERROR.

В тех случаях, когда для того или иного протокола ULP требуется использование определённых портов SCTP ответственность за регистрацию номеров этих портов в IANA ложится на ULP.

## *13.1 Расширения для блоков, определяемые IETF*

Определение и использование новых типов блоков (chunk type) является частью SCTP. Таким образом, новые типы блоков выделяются IANA по согласованию с IETF, как описано в [RFC2434].

Документация для новых типов блоков должна включать следующую информацию:

a) полное и сокращённое имя нового типа блоков;

b) детальное описание структуры блока, которое **должно** соответствовать базовой структуре, определённой в параграфе 3.2;

c) определение и детальное описание каждого поля блока, включая флаги, если они используются;

d) детальное описание процедур использования нового типа блоков в процессе работы протокола.

Максимальный номер типа (255) зарезервирован для будущих расширений.

## 13.2 Определяемые IETF расширения для параметров блоков

Выделение новых кодов для параметров блоков осуществляется через IETF Consensus, как описано в [RFC2434]. Документация для новых параметров блока **должна** включать следующую информацию:

a) Название типа параметра.

b) Детальное описание структуры поля параметра. Структура **должна** соответствовать общему формату TLV[1], описанному в параграфе 3.2.1.

c) детальное описание каждой компоненты значения параметра.

d) Детальное описание использования этого типа параметра и индикацию возможности и условий, при которых в одном блоке может содержаться более одного экземпляра параметров данного типа.

## 13.3 Определяемые IETF дополнительные коды причин ошибок

Дополнительные коды причин ошибок могут выделяться из диапазона [11: 65535] с использованием процедуры Specification Required, описанной в [RFC2434].

Документация для кода причины ошибки должна включать:

a) имя ошибочной ситуации;

b) детальное описание условий, при которых конечной точке SCTP следует генерировать блок ERROR (или ABORT) с данным кодом причины ошибки;

[1]Type-length-value - тип-размер-значение.

c) предполагаемое действие конечной точки SCTP, получившей блок ERROR (или ABORT) с данным кодом;

d) детальное описание структуры и содержимого полей данных, сопровождающих данный код;

Первое слово (32 бита) кода причины ошибки **должно** соответствовать формату, описанному в параграфе 3.3.10:

- первые два байта содержат значение идентификатора кода
- последние два байта содержат размер Cause Parameter.

## 13.4 Идентификаторы передаваемых данных (Payload)

За исключением значения 0, зарезервированного SCTP для индикации неуказанного протокола в блоках DATA, SCTP не принимает на себя ответственности за стандартизацию или проверку идентификаторов передаваемых протоколов. SCTP просто получает идентификатор от вышележащего уровня и передаёт его вместе с соответствующими данными.

Вышележащему уровню (например, пользовательскому приложению SCTP) **следует** стандартизовать нужные значение идентификаторов протоколов в IANA, если это нужно. Использование тех или иных значений идентификаторов передаваемых протоколов выходит за пределы спецификации SCTP.

# *14. Предлагаемые параметры протокола SCTP*

**Рекомендуется** использовать следующие значения параметров протокола:

RTO.Initial - 3 секунды

RTO.Min - 1 секунда

RTO.Max - 60 секунд

RTO.Alpha - 1/8

RTO.Beta - 1/4

Valid.Cookie.Life - 60 секунд

Association.Max.Retrans - 10 попыток

Path.Max.Retrans - 5 попыток (для каждого адреса получателя)

Max.Init.Retransmits - 8 попыток

HB.interval - 30 секунд

Примечание для разработчиков. Реализация SCTP может разрешать ULP изменение некоторых параметров протокола (см. раздел 10).

Примечание. Для RTO.Min **следует** использовать приведённое выше значение.

# *15. Благодарности*

Авторы выражают свою признательность Mark Allman, R.J. Atkinson, Richard Band, Scott Bradner, Steve Bellovin, Peter Butler, Ram Dantu, R. Ezhirpavai, Mike Fisk, Sally Floyd, Atsushi Fukumoto, Matt Holdrege, Henry Houh, Christian Huitema, Gary Lehecka, Jonathan Lee, David Lehmann, John Loughney, Daniel Luan, Barry Nagelberg, Thomas Narten, Erik Nordmark, Lyndon Ong, Shyamal Prasad, Kelvin Porter, Heinz Prantner, Jarno Rajahalme, Raymond E. Reeves, Renee Revis, Ivan Arias Rodriguez, A. Sankar, Greg Sidebottom, Brian Wyld, La Monte Yarroll и другим людям, внёсшим свой вклад в обсуждение протокола.

# *16. Адреса авторов*

**Randall R. Stewart**
24 Burning Bush Trail.
Crystal Lake, IL 60012
USA
Phone: +1-815-477-2127
EMail: rrs@cisco.com

**Qiaobing Xie**
Motorola, Inc.
1501 W. Shure Drive, #2309
Arlington Heights, IL 60004
USA
Phone: +1-847-632-3028
EMail: qxie1@email.mot.com

**Ken Morneault**
Cisco Systems Inc.
13615 Dulles Technology Drive
Herndon, VA. 20171
USA
Phone: +1-703-484-3323
EMail: kmorneau@cisco.com

**Chip Sharp**
Cisco Systems Inc.
7025 Kit Creek Road
Research Triangle Park, NC 27709
USA
Phone: +1-919-392-3121
EMail: chsharp@cisco.com

**Hanns Juergen Schwarzbauer**
SIEMENS AG
Hofmannstr. 51
81359 Munich
Germany
Phone: +49-89-722-24236
EMail: HannsJuergen.Schwarzbauer@icn.siemens.de

Tom Taylor
Nortel Networks
1852 Lorraine Ave.
Ottawa, Ontario
Canada K1H 6Z8
Phone: +1-613-736-0961
EMail: taylor@nortelnetworks.com

**Ian Rytina**
Ericsson Australia
37/360 Elizabeth Street

Melbourne, Victoria 3000
Australia
Phone: +61-3-9301-6164
EMail: ian.rytina@ericsson.com

**Malleswar Kalla**
Telcordia Technologies
3 Corporate Place
PYA-2J-341
Piscataway, NJ 08854
USA
Phone: +1-732-699-3728
EMail: mkalla@telcordia.com

**Lixia Zhang**
UCLA Computer Science Department
4531G Boelter Hall
Los Angeles, CA 90095-1596
USA
Phone: +1-310-825-2695
EMail: lixia@cs.ucla.edu

**Vern Paxson**
ACIRI
1947 Center St., Suite 600,
Berkeley, CA 94704-1198
USA
Phone: +1-510-666-2882
EMail: vern@aciri.org

**Перевод на русский язык**

Николай Малых

nmalykh@protokols.ru

## 17. Документы

[RFC768] Postel, J. (ed.), "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC793] Postel, J. (ed.), "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1123] Braden, R., "Requirements for Internet hosts - application and support", STD 3, RFC 1123, October 1989.

[RFC1191] Mogul, J. and S. Deering, "Path MTU Discovery", RFC 1191, November 1990.

[RFC1700] Reynolds, J. and J. Postel, "Assigned Numbers", STD 2, RFC 1700[1], October 1994.

[RFC1981] McCann, J., Deering, S. and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.

[RFC2026] Bradner, S., "The Internet Standards Process - Revision 3", BCP 9, RFC 2026, October 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[RFC2402] Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[RFC2406] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[RFC2408] Maughan, D., Schertler, M., Schneider, M. and J. Turner, "Internet Security Association and Key Management Protocol", RFC 2408, November 1998.

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC2581] Allman, M., Paxson, V. and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

## 18. Библиография

[ALLMAN99] Allman, M. and Paxson, V., "On Estimating End-to-End Network Path Properties", Proc. SIGCOMM'99, 1999.

[FALL96] Fall, K. and Floyd, S., Simulation-based Comparisons of Tahoe, Reno, and SACK TCP, Computer Communications Review, V. 26 N. 3, July 1996, pp. 5-21.

[RFC1750] Eastlake, D. (ed.), "Randomness Recommendations for Security", RFC 1750, December 1994.

[RFC1950] Deutsch P. and J. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

[RFC2104] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, March 1997.

[RFC2196] Fraser, B., "Site Security Handbook", FYI 8, RFC 2196, September 1997.

[RFC2522] Karn, P. and W. Simpson, "Photuris: Session-Key Management Protocol", RFC 2522, March 1999.

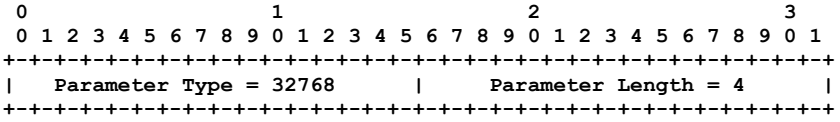
[SAVAGE99] Savage, S., Cardwell, N., Wetherall, D., and Anderson, T., "TCP Congestion Control with a Misbehaving Receiver"[2], ACM Computer Communication Review, 29(5), October 1999.

[1] В соответствии с RFC 3232 документ Assigned Numbers утратил прежнее значение, а информация о выделенных номерах хранится в реестре на сайте http://www.iana.org/numbers.html. *Прим. перев.*

[2] Копию этой статьи можно загрузить с сайта http://www.cs.ucsd.edu/~savage/papers/CCR99.pdf. *Прим. перев.*

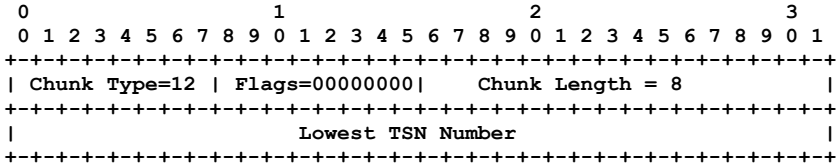
# Приложение A. Явное уведомление о насыщении (ECN)[3]

ECN (Ramakrishnan, K., Floyd, S., "Explicit Congestion Notification", RFC 2481, January 1999) описывает предложенное расширение протокола IP, включающее метод получения информации о насыщении в сети, отличный от детектирования потери дейтаграмм. Эта дополнительная функция **может** быть реализована и в протоколе SCTP. В данном приложении рассматриваются незначительные отличия, которые следует принимать во внимание при реализации этого механизма в SCTP. В целом нужно следовать рекомендациям RFC 2481 с учётом перечисленных ниже отличий.



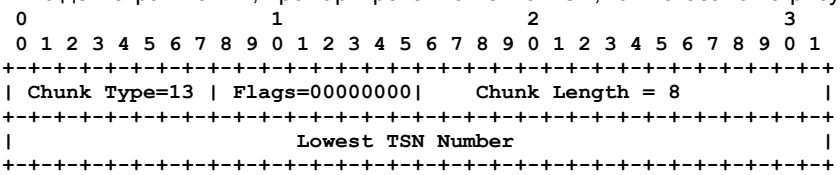

**Negotiation**

RFC 2481 описывает согласование ECN на этапах SYN и SYN-ACK организации соединения TCP. Отправитель SYN устанавливает два бита в поле флагов TCP, а отправитель SYN-ACK - только 1 бит. Это сделано для того, чтобы убедиться в поддержке ECN на обеих сторонах соединения. Для протокола SCTP этого не требуется. Для индикации поддержки механизма ECN конечной точке следует добавить в блок INIT или INIT ACK структуру TLV, зарезервированную для ECN. TLV не включает параметров и имеет формат, показанный на рисунке.




**ECN-Echo**

В соответствии с RFC 2481 получатель пакета устанавливает в передаваемом подтверждении специальный бит, который уведомляет отправителя CE[4] о получении информации. Для SCTP подобная индикация осуществляется с путём включения специального блока ECNE. Этот блок содержит единственный элемент данных - минимальный номер TSN, связанный с дейтаграммой IP, промаркированной битом CE, как показано на рисунке.




Примечание. ECNE рассматривается как управляющий блок (Control chunk).

**CWR**

В соответствии с RFC 2481 отправитель устанавливает специальный бит CWR[2] в заголовке следующего исходящего сегмента TCP для индикации снижения размера окна насыщения. Для SCTP такая индикация может обеспечиваться включением блока CWR. Этот блок содержит единственный элемент данных - номер TSN, который был передан в блоке ECNE. Этот элемент представляет наименьший номер TSN в дейтаграмме, которая была изначально помечена битом CE.

Примечание. Блок CWR относится к числу управляющих блоков.

# Приложение B Алгоритм Alder 32 для расчёта контрольных сумм

Приведённый в этом приложении алгоритм расчёта контрольной суммы Adler-32 скопирован из [RFC1950].

Контрольная сумма Adler-32 состоит из двух сумм, собранных для каждого байта: s1 представляет собой сумму значений всех байтов, а s2 - сумму всех значений s1. Обе суммы вычисляются по модулю 65521. Сумма s1 инициализируется значением 1, s2 - значением 0. Контрольная сумма Adler-32 сохраняется как s2*65536 + s1 с использованием сетевого порядка байтов.

Приведённый ниже код на языке C обеспечивает расчёт контрольной суммы Adler-32 для буфера данных. Код оптимизирован с точки зрения понимания, а не скорости выполнения. В примере используется диалект ANSI C. Для читателей, не знакомых с языком C ниже приведены некоторые пояснения:

```
&       битовый оператор AND (И).
>>       оператор сдвига битов вправо. При использовании с числами без знака (как в приведённом примере)
освобождающиеся слева биты заполняются нулями.
<<      оператор сдвига битов влево. Освобождающиеся справа биты заполняются нулями.
++      "n++" увеличивает значение переменной n на 1.
%       остаток от деления: a % b даёт остаток от деления a на b.

#define BASE 65521 /* наибольшее значение суммы, не превышающее 65536 */
/*
Обновляет контрольную сумму Adler-32, учитывая в ней байты buf[0..len-1], и возвращает обновлённое
значение. Контрольную сумму Adler-32 следует инициализировать значением 1.
Пример использования:
        unsigned long adler = 1L;

        while (read_buffer(buffer, length) != EOF) {
        adler = update_adler32(adler, buffer, length);
        }
```

[3]Explicit Congestion Notification.
[4]Congestion Experienced.
[2]Congestion Window Reduced - окно насыщения уменьшено.

```
        if (adler != original_adler) error();
*/
unsigned long update_adler32(unsigned long adler, unsigned char *buf, int len)
{
        unsigned long s1 = adler & 0xffff;
        unsigned long s2 = (adler >> 16) & 0xffff;
        int n;

        for (n = 0; n < len; n++) {
         s1 = (s1 + buf[n]) % BASE;
                s2 = (s2 + s1)     % BASE;
        }
        return (s2 << 16) + s1;
}

/* Возвращает значение суммы Adler-32 для байтов buf[0..len-1] */
unsigned long adler32(unsigned char *buf, int len)
{
        return update_adler32(1L, buf, len);
}
```

**Полное заявление авторских прав**



**Подтверждение**

Финансирование функций RFC Editor обеспечено Internet Society.