Internet Engineering Task Force (IETF)
Request for Comments: 9260
Obsoletes: 4460, 4960, 6096, 7053, 8540
Category: Standards Track
ISSN: 2070-1721

R. Stewart
Netflix, Inc.
M. Tüxen
Münster Univ. of Appl. Sciences
K. Nielsen
Kamstrup A/S

June 2022

Протокол SCTP

Stream Control Transmission Protocol

Аннотация

Этот документ описывает протокол управления потоковой передачей (Stream Control Transmission Protocol или SCTP) и отменяет RFC 4960. Он включает спецификацию реестра флагов блоков (chunk) из RFC 6096 и спецификацию бита I в блоках DATA из RFC 7053, поэтому отменяет также RFC 6096 и RFC 7053. Кроме того, документ отменяет RFC 4460 и RFC 8540, где указаны ошибки для SCTP.

Протокол SCTP изначально был предназначен для передачи сигнальных сообщений PSTN[1] через сети IP, но может использоваться и для других приложений, например, WebRTC.

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[2];
- 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 gthtuheprb и устойчивости к атакам с использованием лавины пакетов (flooding) или маскированием адресов (masquerade).

Статус документа

Документ относится к категории Internet Standards Track.

Документ является результатом работы IETF[3] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[4]. Дополнительную информацию о стандартах Internet можно найти в разделе 2 в RFC 7841.

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке https://www.rfc-editor.org/info/rfc9260.

[1] Public Switched Telephone Network - коммутируемая телефонная сеть общего пользования. *Прим. перев.*
[2] Максимальный размер передаваемых пакетов. *Прим. перев.*
[3] Internet Engineering Task Force - комиссия по решению инженерных задач Internet.
[4] Internet Engineering Steering Group - комиссия по инженерным разработкам Internet.

Оглавление

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.

Этот документ заменяет собой [RFC4960]. Кроме того, он включает спецификацию реестра флагов блоков (chunk) из RFC 6096 и спецификацию бита I в блоках DATA из RFC 7053, поэтому отменяет также [RFC 6096] и [RFC 7053].

1.1. Мотивация

Протокол TCP [RFC0793] 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 [RFC0768]. Ниже перечислены основные ограничения TCP, которые пользователи стремятся обойти.

- 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 хостов.
- Протокол TCP достаточно слабо защищён от атак на службы, 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. Одним из примеров является использование каналов данных в инфраструктуре WebRTC.

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 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.

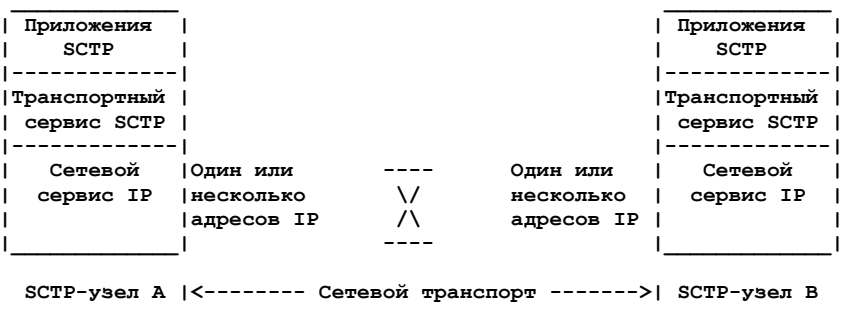


Рисунок 1. SCTP-ассоциация.

В дополнение к инкапсуляции пакетов SCTP в IPv4 и IPv6 возможна инкапсуляция в UDP, как указано в [RFC6951], или в DTLS, как указано в [RFC8261].

1.3. Основные термины

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.

Association Maximum DATA Chunk Size (AMDCS) - максимальный размер блока DATA для ассоциации
Меньшее из значение PMDCS (Path Maximum DATA Chunk Size) среди всех адресов получателя.

Bundling of Chunks - группировка блоков
Дополнительная операция мультиплексирования, позволяющая передать в одном пакете несколько блоков SCTP.

[1]Connection-oriented.

Bundling of User messages - группировка пользовательских сообщений

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 (перегрузки)

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.

Flightsize - размер данных «в пути»

Число остающихся данных для конкретного транспортного адреса получателя в данный момент.

Idle destination address - неиспользуемый адрес получателя

Адрес, который не используется для передачи пользовательских сообщений в течение некоторого периода (обычно ≥ HB.interval).

Inactive destination transport address - неактивный транспортный адрес получателя

Адрес, который считается неактивным в результате обнаружения ошибки, и не доступен для доставки пользовательских сообщений.

Message (user message) - (пользовательское) сообщение

Данные, полученные протоколом SCTP от протокола вышележащего уровня (Upper Layer Protocol или ULP).

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 Data (Data Outstanding или Data In Flight) - остающиеся данные (данные «в пути»)

Общий размер блоков DATA связанных с остающимися TSN. Повторно передаваемый блок DATA учитывается в остающихся данных один раз. Блок DATA, сочтённый потерянным, но ещё не переданный повторно не относится к остающимся данным.

Outstanding TSN (в конечной точке SCTP)

Номер TSN (и связанный с ним блок DATA), который был передан конечной точкой, но его получение ещё не подтверждено.

"Out of the Blue" (OOTB) Packet

Пакет с корректным форматом, для которого получатель не может определить ассоциацию (см. параграф 8.4).

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. В этой спецификации путь указывается транспортным адресом получателя, поскольку маршрутизация считается стабильной. Это включает, в частности, выбор адреса отправителя при отправке пакета получателю.

Path Maximum DATA Chunk Size (PMDCS) - максимальный размер блока DATA для пути

Максимальный размер блока DATA (включая заголовок блока), который помещается в пакет SCTP, не превышающий по размеру PMTU для конкретного адреса получателя.

Path Maximum Transmission Unit (PMTU)

Максимальный размер пакета SCTP (включая общий заголовок SCTP и все блоки вместе с их заполнением), который можно передать по конкретному адресу получателя без использования фрагментации на уровне IP.

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) - окно получателя

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 (т. е. транспортный адрес каждой конечной точки SCTP уникален).

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, 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.

State Cookie - куки состояния

Контейнер со всей информацией, требуемой для создания ассоциации.

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[1].

Stream Sequence Number - порядковый номер в потоке

16-битовый порядковый номер, используемый SCTP для упорядоченной доставки пользовательских сообщений в данном потоке. Каждому пользовательскому сообщению в потоке присваивается один порядковый номер.

Tie-Tags

Два 32-битовых случайных значений, которые вместе образуют 64-битовое одноразовое значение nonce. Эти теги используются в State Cookie и TCB для связывания недавно перезапущенной ассоциации с её предшественником на конечной станции, которая не была перезагружена, но, тем не менее, не может найти тегов Verification существующей ассоциации.

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).

Unordered Message - неупорядоченное сообщение

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.

1.4. Сокращения

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

RTO - Retransmission Timeout (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).

1.5. Функциональность SCTP

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

[1]Соотношения между номерами потоков в противоположных направлениях сильно зависят от того, как приложения используют эти потоки. Ответственность за поддержку корреляции между порядковыми номерами (если она нужна) ложится на пользовательское приложение SCTP.

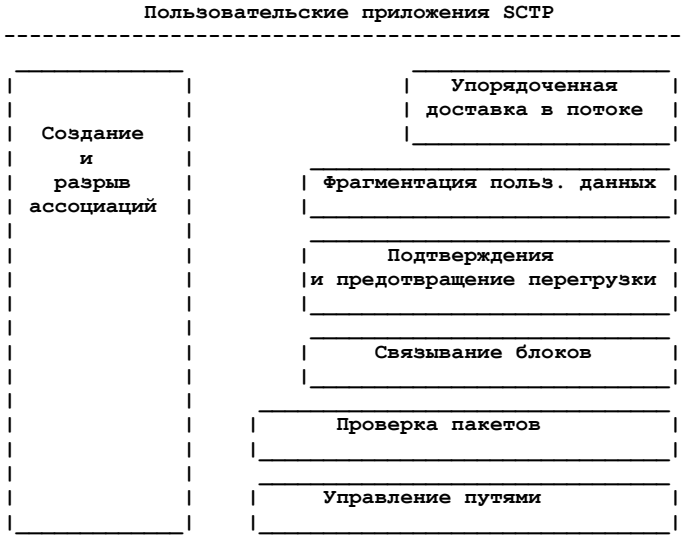


Рисунок 2. Функциональное представление сервиса SCTP.

1.5.1. Создание и разрыв ассоциаций

Ассоциации создаются по запросам пользователей SCTP (см. описание примитива ASSOCIATE на стр. 49 или SEND на стр. 50).

В процессе создания ассоциаций используется механизм cookie, подобный предложенному Кэрном (Karn) и Симпсоном (Simpson) в [RFC2522], для обеспечения защиты от атак на синхронизацию. 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 разделе 5. Создание ассоциации.

Протокол SCTP обеспечивает аккуратное завершение работы активных ассоциаций (shutdown) по запросу пользователя SCTP (см. описание примитива SHUTDOWN на стр. 50). Протокол SCTP позволяет также разрывать ассоциации (abort) по запросу пользователя (примитив ABORT) или в результате обнаружения ошибок на уровне SCTP. В разделе 9 описаны оба варианта завершения работы ассоциаций.

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

1.5.2. 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 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 этой ассоциацией. Это значение согласуется с удалённой стороной (см. 5.1.1. Обработка параметров потока). Пользовательские сообщения связываются с номерами потоков (см. примитивы SEND на стр. 50 и RECEIVE на стр. 51). 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.

1.5.3. Фрагментация пользовательских данных

При необходимости 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.

1.5.4. Подтверждения и предотвращение перегрузки

SCTP присваивает номер TSN каждому фрагменту и нефрагментированному пользовательскому сообщению. Нумерация TSN не зависит от Stream Sequence Number, присваиваемых на уровне потока. Принимающая сторона подтверждает все полученные TSN даже при обнаружении пропуска в номерах. Если нужно повторно передать фрагмент данных пользователя или нефрагментированное сообщение, используется выделенный номер TSN. Такой подход обеспечивает независимую функциональность для гарантии доставки и сохранения порядка сообщений.

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

1.5.5. Группировка блоков

Как описано в разделе 3, пакет SCTP, передаваемый на нижележащий уровень, содержит общий заголовок, за которым следует один или несколько информационных блоков (chunk), каждый из них содержит пользовательские данные или управляющую информацию SCTP. Реализация SCTP, поддерживающая группировку, может задержать на стороне отправителя отправку пользовательских сообщений для группировки соответствующих блоков DATA.

Пользователь SCTP может запросить у реализации передачу своих сообщений без задержки, связанной с группировкой. Однако даже при выборе пользователем этой опции реализация SCTP может задерживать передачу по другим причинам (например, связанным с контролем перегрузки или управлением потоком данных) и группировать блоки DATA, если это возможно.

1.5.6. Проверка пакетов

Обязательное поле Verification Tag и 32-битовая контрольная сумма (см. Приложение A. Расчет контрольной суммы CRC32c), 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 CRC32c помещается отправителем в каждый пакет SCTP для обеспечения дополнительной защиты от повреждения данных в сети. Получатель пакета SCTP с некорректной контрольной суммой CRC32 с отбрасывает такие пакеты без уведомления.

1.5.7. Управление путями

Передающий пакеты SCTP пользователь может манипулировать набором транспортных адресов, используемых для указания получателя пакетов SCTP с помощью примитивов, описанных в разделе 11. Функция управления путями отслеживает доступность 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 SCTP выбирает транспортный адрес получателя для каждого исходящего пакета SCTP на основе пользовательских инструкций и текущей информации о доступности желанного получателя. Функция управления путями отвечает также за передачу информации о доступных локальных адресах транспортного уровня партнёру в процессе создания ассоциации и за передачу пользователю SCTP сведений, полученных от партнёра.

При создании ассоциации определяется основной путь (primary path) для каждой конечной точки SCTP, который используется для нормальной передачи пакетов SCTP.

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

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

1.6. Порядковые номера

Важно помнить, что пространство порядковых номеров TSN ограничено, хотя и достаточно велико. Значения порядковых номеров могут находиться в диапазоне от 0 до 232 - 1. Ввиду конечности пространства TSN все арифметические операции с ними **должны** осуществляться по модулю 232 - 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. Символы =< применительно к TSN означают «меньше или равно» (модуль 232).

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 231 - 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 232 - 1. Т. е., следующий блок DATA после блока с TSN = 232 - 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.

1.7. Отличия от RFC 4960

Протокол SCTP был изначально определён в [RFC4960], который данный документ отменяет. Читателям, интересующимся подробностями отличий, рекомендуется прочесть [RFC8540].

В дополнение к этим и последующим редакторским правкам в документ внесены перечисленные ниже изменения.

- Обновлены ссылки.
- Улучшен язык описания уровней требований.
- Примитиву ASSOCIATE разрешено принимать несколько удалённых адресов (см. спецификацию API сокетов).
- Ссылка на спецификацию Packetization Layer Path MTU Discovery (PLPMTUD) для определения MTU на пути.
- Описание обработки ICMP перенесено из приложения в основной текст.
- Удалено приложение с описанием обработки явных уведомлений о перегрузке (ECN).
- Более точно описана обработка размера пакетов, путём введения PMTU, PMDCS и AMDCS.

- Добавлено определение управляющего блока (control chunk).
- Улучшено описание обработки блоков INIT и INIT ACK с непригодными обязательными параметрами.
- Разрешено применение L > 1 для подсчёта байтов (Appropriate Byte Counting или ABC) при замедленном старте.
- Явно описана повторная инициализация контроллера перегрузок при смене маршрутов.
- Улучшена терминология для более ясного представления, что данная спецификация не описывает полносвязную (full mesh) архитектуру.
- Улучшено описание генерации номеров (Transmission Sequence Number и Stream Sequence Number).
- Улучшено описание «отречения» (reneging).
- Больше не требуется изменение Cumulative TSN Ack для увеличения окна перегрузки. Это улучшает согласованность с действиями по предотвращению перегрузки.
- Улучшено описание State Cookie.
- Исправлен API для извлечения сообщений при отказах ассоциации.

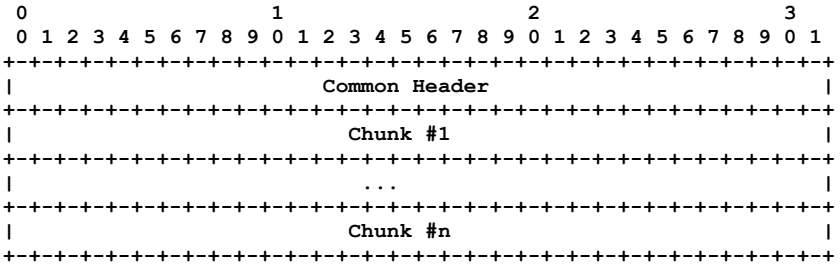
2. Соглашения о терминах

Ключевые слова **должно** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не следует** (SHALL NOT), **следует** (SHOULD), **не нужно** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **не рекомендуется** (NOT RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе интерпретируются в соответствии с BCP 14 [RFC2119] [RFC8174] тогда и только тогда, когда они выделены шрифтом, как показано здесь.

3. Формат пакетов SCTP

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

Формат пакетов SCTP показан на рисунке.

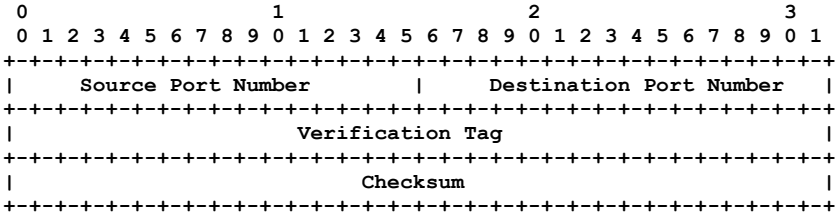


Недопустимо группировать INIT, INIT ACK, и SHUTDOWN COMPLETE с другими блоками в один пакет SCTP. Все прочие блоки **можно** группировать в одном пакете SCTP, пока его размер не превышает значение PMTU. Более подробное описание группировки блоков приводится в параграфе 6.10.

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



Source Port Number - 16 битов (целое число 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. Значение 0 для номера порта **недопустимо**.

Destination Port Number - 16 битов (целое число 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. Значение 0 для номера порт **недопустимо**.

Verification Tag - 32 бита (целое число 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 инициализации ассоциации, за исключением перечисленных ниже случаев.

- В пакетах, содержащих блок INIT, тег Verification **должен** быть установлен в 0.
- В пакетах, содержащих блок SHUTDOWN-COMPLETE с установленным флагом T, значение тега Verification **должно** копироваться из пакета с блоком SHUTDOWN-ACK.

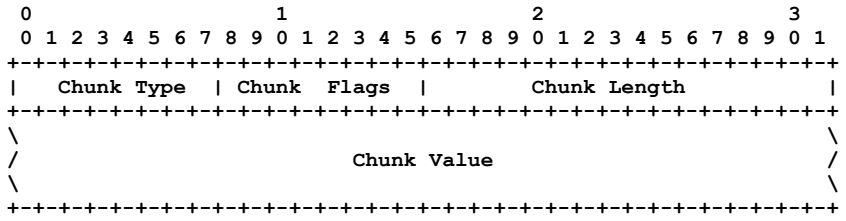
- В пакетах, содержащих блок ABORT, тег верификации **может** быть копией Verification Tag из пакета, вызвавшего передачу блока ABORT. Более подробное описание приведено в параграфах 8.4 и 8.5.

Checksum - 32 бита (целое число без знака)

Это поле содержит контрольную сумму для данного пакета SCTP. Расчёт контрольных сумм описан в параграфе 6.8. Протокол SCTP использует алгоритм CRC32c, описанный в Приложении A.

3.2. Описание поля Chunk

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



Chunk Type - 8 битов (целое число 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 будущего использования в качестве расширения.

Значения идентификаторов типа приведены в таблице.

Таблица 1. Типы блоков.

ID	*Тип блока*	*ID*	*Тип блока*
0	Пользовательские данные (DATA)	12	Резерв для Explicit Congestion Notification Echo (ECNE)[1]
1	Инициализация (INIT)	13	Резерв для Congestion Window Reduced (CWR)
2	Подтверждение инициирования (INIT ACK)	14	Окончание работы завершен0 (SHUTDOWN COMPLETE)
3	Выборочное подтверждение (SACK)	15 - 62	Не выделены
4	Запрос Heartbeat (HEARTBEAT)	63	Резерв для определённых IETF расширений
5	Подтверждение Heartbeat (HEARTBEAT ACK)	64 - 126	Не выделены
6	Прерывание (ABORT)	127	Резерв для определённых IETF расширений
7	Завершение работы (SHUTDOWN)	128 - 190	Не выделены
8	Подтверждение завершения (SHUTDOWN ACK)	191	Резерв для определённых IETF расширений
9	Ошибка при операции (ERROR)	192 - 254	Не выделены
10	State Cookie (COOKIE ECHO)	255	Резерв для определённых IETF расширений
11	Подтверждение Cookie (COOKIE ACK)		

Коды Chunk Type выбраны таким образом, чтобы 2 старших бита определяли действие, которое следует предпринять конечной точке на приёмной стороне, если Chunk Type не удаётся распознать.

Таблица 2. Обработка неизвестных блоков.

00 Прекратить обработку данного пакета SCTP и отбросить нераспознанный блок и следующие за ним блоки.
01 Прекратить обработку данного пакета SCTP и отбросить нераспознанный блок и следующие за ним блоки, а также сообщить о неопознанном блоке в блоке ERROR с причиной ошибки Unrecognized Parameter Type.
10 Пропустить данный блок и продолжить обработку пакета.
11 Пропустить данный блок и продолжить обработку пакета, а также сообщить о неопознанном блоке в блоке ERROR с причиной ошибки Unrecognized Parameter Type.

Chunk Flags - 8 битов

Использование этих битов зависит от поля 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 число без знака)

Размер блока в байтах с учётом полей Chunk Type, Chunk Flags, Chunk Length и Chunk Value. Для блоков с пустым Chunk Value поле размера имеет значение 4. Байты заполнения блока в поле Chunk Length не включаются, однако считаются байты заполнения имеющихся параметров переменного размера, за исключением последнего[2].

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 15.1.

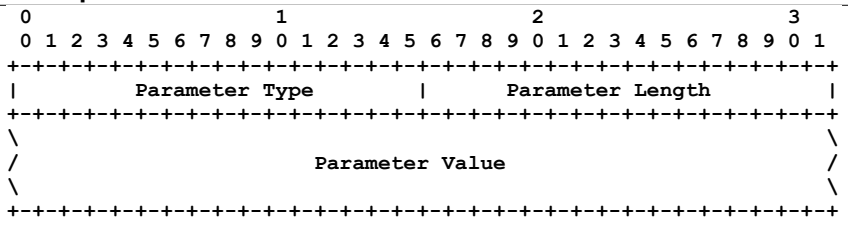
3.2.1. Необязательные параметры и поля переменного размера

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[3] как описано ниже.

Parameter Type - 16 битов (целое число без знака)

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

[1]Типы типы ECNE и CWR зарезервированы для использования в будущем явных уведомлений о перегрузке (ECN).
[2]Предполагается, что отказоустойчивые реализации будут воспринимать блоки даже в тех случаях, когда поле Chunk Length учитывает заполнение в конце блока.
[3]Type-Length-Value - Тип-Размер-Значение.



Parameter Length - 16 битов (целое число без знака)

Размер параметра в байтах с учётом полей Parameter Type, Parameter Length и Parameter Value. Для параметров с пустым полем Parameter Value поле размера имеет значение 4. Размер 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 (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 обрабатывающей конечной точкой при обнаружении неизвестного типа параметра.

Таблица 3. Обработка неизвестных параметров.

00	Прекратить обработку данного параметра и следующих за ним параметров в этом блоке.
01	Прекратить обработку данного параметра и следующих за ним параметров в этом блоке, а также сообщить о неопознанном параметре, как указано в параграфе 3.2.2.
10	Пропустить данный параметр и продолжить обработку.
11	Пропустить данный параметр и продолжить обработку пакета, а также сообщить о неопознанном параметре, как указано в параграфе 3.2.2.

Следует отметить, что при получении блока INIT или INIT ACK во всех 4 случаях в ответ передаётся блок INIT ACK или COOKIE ECHO. Для случаев 00 и 01 обработка параметров после неизвестного параметра не производится, но уже обработанные параметры не отбрасываются.

Реальные параметры SCTP определяются в параграфах, посвящённых конкретным блокам SCTP. Правила для определённых IETF расширений параметров определены в параграфе 15.3. Отметим, что тип параметра **должен** быть уникальным для всех блоков. Например, тип параметра 5 используется для представления адресов IPv4 (см. параграф 3.3.2.1.1). Значение 5 в этом случае резервируется во всех блоках для представления адресов IPv4 и **недопустимо** его использование в ином смысле в любом другом блоке.

3.2.2. Уведомления о неизвестных параметрах

Если получатель блока INIT обнаруживает неизвестный параметр и сообщает о нем в соответствии с параграфом 3.2.1, он **должен** указать параметр(ы) в поле Unrecognized Parameter блока INIT ACK, передаваемого в ответ на INIT. Отметим, что если получатель блока INIT не организует ассоциацию (например, по причине нехватки ресурсов), поле Unrecognized Parameter не включается в блок ABORT, передаваемый в ответ на INIT.

Если получатель блока любого другого блока (например, INIT ACK) обнаруживает неизвестные параметры и сообщает о них в соответствии с параграфом 3.2.1, ему **следует** сгруппировать блок ERROR, содержащий поля Unrecognized Parameter с указанием причины ошибки, с блоком, передаваемым в ответ (например, COOKIE ECHO). Если получатель INIT ACK не группирует блок COOKIE ECHO с блоком ERROR, последний **можно** передать отдельно, но не раньше получения COOKIE ACK.

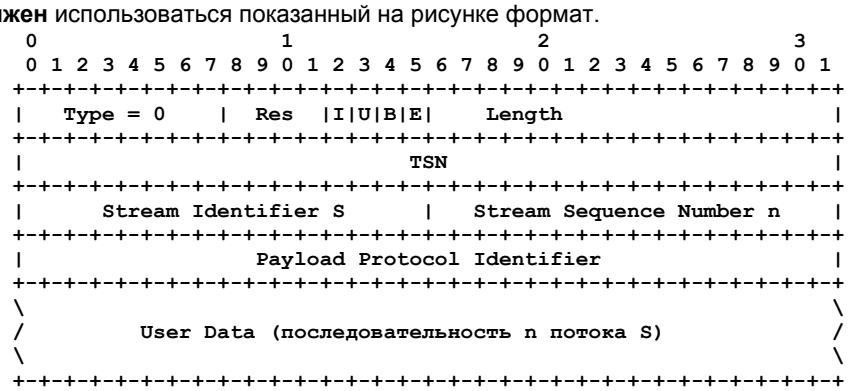
В любом случае передачи в пакете блока COOKIE ECHO он **должен** быть первым блоком.

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.



Res - 4 бита

Отправителю следует заполнить это поле нулями, а получателю - игнорировать его.

I - 1 бит

Бит I (immediate) отправитель может устанавливать всякий раз, когда может при отправке блока DATA получить преимущество в результате отправки соответствующего блока SACK без задержки (см. раздел 4 в [RFC7053].

U - 1 бит

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

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

Если неупорядоченное пользовательское сообщение фрагментируется, для каждого фрагмента **должен** устанавливаться флаг U = 1.

B - 1 бит

Флаг первого фрагмента пользовательского сообщения.

E - 1 бит

Флаг последнего фрагмента пользовательского сообщения.

Length - 16 битов (целое число без знака)

Размер блока DATA в байтах от начала поля типа и до конца поля User Data (без учёта байтов заполнения). Для блока DATA с одним байтом пользовательских данных Length = 17 (17 байтов). Блок DATA с полем User Data размера L будет иметь поле Length со значением (16 + L), где L **должно** быть больше 0.

TSN - 32 бита (целое число без знака)

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 число без знака)

Идентификатор потока, к которому относится блок данных.

Stream Sequence Number n - 16 битов (целое число без знака)

Порядковый номер пользовательских данных 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 число без знака)

Заданный приложением или вышележащим протоколом идентификатор протокола данных. SCTP получает идентификатор от вышележащего уровня и передаёт его партнёру. Значение идентификатора не используется протоколом SCTP, но может быть использовано некоторыми сетевыми объектами и приложениями у партнёра 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). Отметим, что реализации SCTP не работают с этим полем и за преобразование между сетевым и хостовым порядком байтов в этом поле отвечает вышележащий уровень.

Значение 0 говорит, что протокол вышележащего уровня не указал идентификатор протокола для этого блока.

User Data - переменный размер

Поле переменной длины, содержащее пользовательскую информацию. Реализация протокола **должна** дополнять поле до 4-байтовой границе путём добавления байтов с нулевым значением. **Недопустимо** учитывать байты заполнения в поле размера блока. Для отправителя **недопустимо** использование более 3 байтов заполнения.

В нефрагментированных пользовательских сообщениях **должны** устанавливаться (1) оба флага B и E. Нулевые значения обоих флагов устанавливаются в средних (не первом и не последнем) фрагментах пакета, как показано в таблице 4.

Таблица 4. 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.

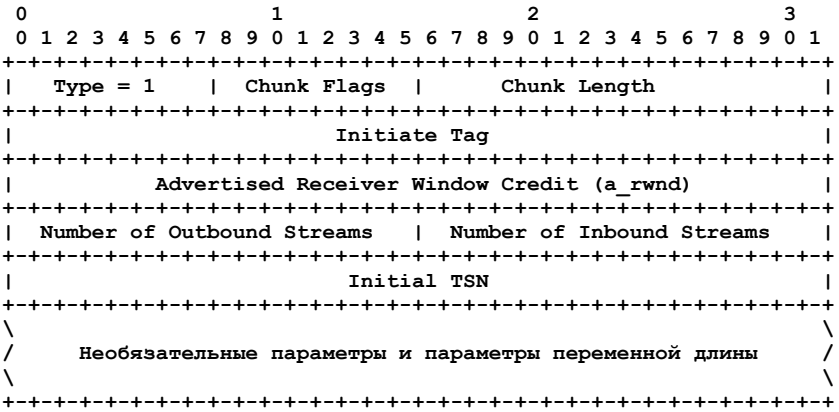
При фрагментировании пользовательского сообщения на множество блоков получатель использует TSN для сборки сообщения. Это означает, что TSN фрагментов **должны** устанавливаться в строгом порядке.

TSN в блоках DATA **следует** строго упорядочивать.

Примечание. Можно применять расширение, описанное в [RFC8260] для снижения блокировки head-of-line при передаче больших пользовательских сообщений.

3.3.2. Инициализация (INIT) (1)

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



Блок INIT содержит перечисленные ниже параметры. Если явно не сказано иное, каждый параметр **должен** включаться в блок INIT в одном экземпляре.

Таблица 5. Параметры фиксированного размера в блоках INIT.

Фиксированные параметры	***Статус***

Initiate Tag	Обязательный
Advertised Receiver Window Credit	Обязательный
Number of Outbound Streams	Обязательный
Number of Inbound Streams	Обязательный
Initial TSN	Обязательный

Таблица 6. Параметры переменного размера в блоках INIT.

Переменные параметры	***Статус***	***Тип***
IPv4 Address[1]	Необязательный	5
IPv6 Address[1]	Необязательный	6
Cookie Preservative	Необязательный	9
Reserved for ECN Capable[2]	Необязательный	32768 (0x8000)
Host Name Address[3]	Отменён	11
Supported Address Types[4]	Необязательный	12

Если принят блок INIT со всеми обязательными параметрами, заданными для блока INIT, получателю **следует** обработать блок INIT и возвратить отправителю INIT ACK. Получатель блока INIT **может** позднее сгруппировать блок ERROR с блоком COOKIE ACK. Однако ограниченная реализация **может** возвратить блок ABORT в ответ на блок INIT.

Поле Chunk Flags в INIT является резервным и отправителю **следует** устанавливать все его биты в 0, а получателю - игнорировать их.

Initiate Tag - 32 бита (целое число без знака)

Получатель INIT (отвечающая 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 отбрасывать пакет без иных действий.

Advertised Receiver Window Credit (a_rwnd) - 32 бита (целое число без знака)

Размер (в байтах) выделенного буферного пространства, которое отправитель INIT зарезервировал для данной ассоциации.

Для Advertised Receiver Window Credit **недопустимы** значения меньше 1500.

Получатель блока INIT с a_rwnd < 1500 **должен** отбросить пакет, ему **следует** передать в ответ пакет с блоком ABORT и использовать Initiate Tag как Verification Tag, а также **недопустимо** менять состояние оюбой имеющейся ассоциации.

В течение срока существования ассоциации **не следует** снижать размер этого буфера (т. е., буфер может быть удалён из ассоциации), однако конечная точка **может** изменить размер a_rwnd, передаваемого в SACK.

Number of Outbound Streams (OS) - 16 битов (целое число без знака)

Число исходящих потоков, которые отправитель блока INIT желает открыть для данной ассоциации. Для этого параметра недопустимо использование значения 0.

Получатель блока INIT с OS = 0 **должен** отбросить пакет, ему **следует** передать в ответ пакет с блоком ABORT и использовать Initiate Tag как Verification Tag, а также **недопустимо** менять состояние оюбой имеющейся ассоциации.

Number of Inbound Streams (MIS) - 16 битов (целое число без знака)

Максимальное число потоков, которые отправитель данного блока INIT готов принять от партнёра для этой ассоциации[5]. Использование значений 0 в данном поле **недопустимо**.

Получатель блока INIT с с MIS = 0 **должен** отбросить пакет, ему **следует** передать в ответ пакет с блоком ABORT и использовать Initiate Tag как Verification Tag, а также **недопустимо** менять состояние оюбой имеющейся ассоциации.

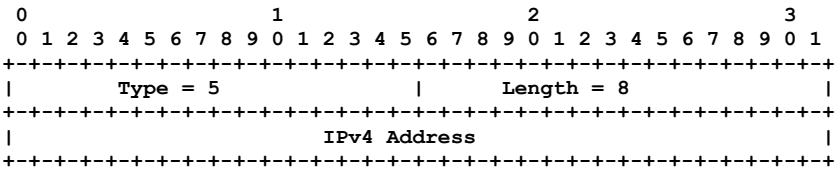
Initial TSN (I-TSN) - 32 бита (целое число без знака)

Определяет начальное значение TSN, которое будет использовать отправитель. Диапазон допустимых значений - от 0 до 4294967295 и **следует** устанавливать случайное значение из этого диапазона. Для выбора случайного значения можно использовать методы, описанные в [RFC4086].

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.

3.3.2.1.1. Параметр IPv4 Address (5)



IPv4 Address - 32 бита (целое число без знака)

Адрес IPv4 передающей стороны в двоичном формате.

3.3.2.1.2. Параметр IPv6 Address (6)

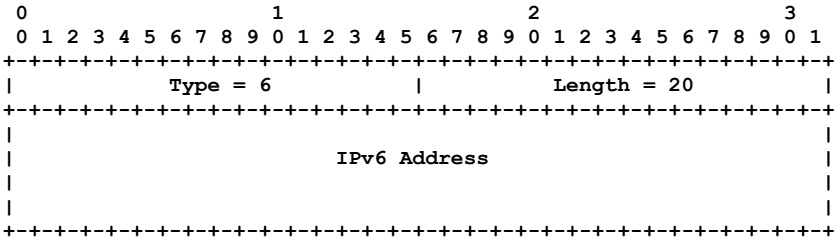
[1]Блоки INIT могут содержать множество адресов IPv4 и/или IPv6 в произвольных комбинациях.

[2]Поле ECN capable зарезервировано для будущего использования с ECN.

[3]**Недопустимо** включение в блок INIT параметра Host Name Address. Получатель блока INIT с Host Name Address **должен** передать блок ABORT и **может** включить в него причину ошибки Unresolvable Address.

[4]Этот параметр (при наличии) задаёт все типы адресов, которые может поддерживать передающая сторона. Отсутствие данного параметра говорит о поддержке передающей стороной адресов всех типов.

[5]Здесь 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.



IPv6 Address - 128 битов (целое число без знака)

Адрес IPv6 [RFC2460] передающей стороны в двоичном формате.

Отправителю **недопустимо** использовать адреса отображённые на IPv4 адреса IPv6 [RFC4291], вместо этого **следует** применять параметр IPv4 Address для адреса IPv4.

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

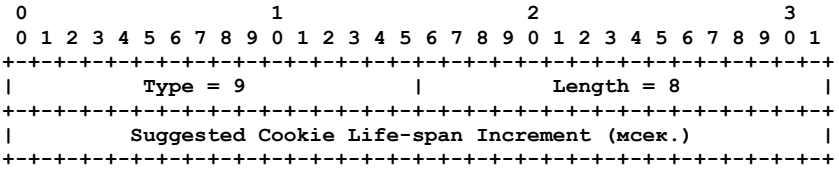
В блоке INIT можно указывать более одного адреса IP, если отправитель INIT является многодомным хостом. Кроме того, многодомные хосты могут быть подключены к разнотипным сетям и в блоках INIT могут указываться одновременно адреса IPv4 и IPv6.

Если INIT содержит хотя бы один параметр IP Address, адрес отправителя в дейтаграмме, содержащей блок INIT, и любые адреса, указанные в INIT, могут использоваться партнёром при ответе в качестве адреса получателя. Если INIT не содержит параметров IP Address, получившая блок INIT конечная точка **должна** использовать адрес отправителя в заголовке дейтаграммы IP, содержащей блок INIT, как единственный адрес партнёра в данной ассоциации.

Отметим, что отсутствие параметров IP Address в блоках INIT и INIT ACK упрощает организацию ассоциаций при работе через системы трансляции адресов (Network Address Translation или NAT).

3.3.2.1.3. Параметр Cookie Preservative (9)

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



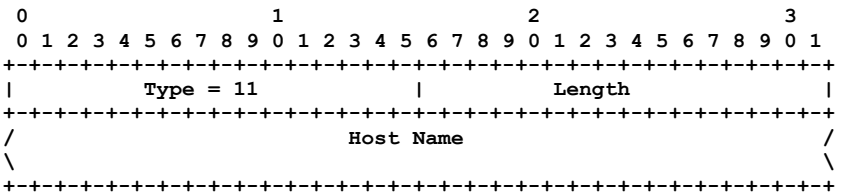
Suggested Cookie Life-Span Increment - 32 бита (целое число без знака)

Показывает получателю размер увеличения срока действия cookie в миллисекундах.

Этот необязательный параметр отправителю следует добавлять в блок INIT при попытке создания ассоциации после отказа в предыдущей попытке по причине ошибки в работе stale cookie. Получатель **может** проигнорировать предложенное увеличение срока действия cookie в соответствии со своей политикой безопасности.

3.3.2.1.4. Параметр Host Name Address (11)

Отправителю блока INIT или INIT ACK **недопустимо** включать этот параметр, поскольку его использование отменено. Получатель блока INIT или INIT ACK с параметром Host Name Address должен передать блок ABORT и **может** включить в него причину Unresolvable Address.



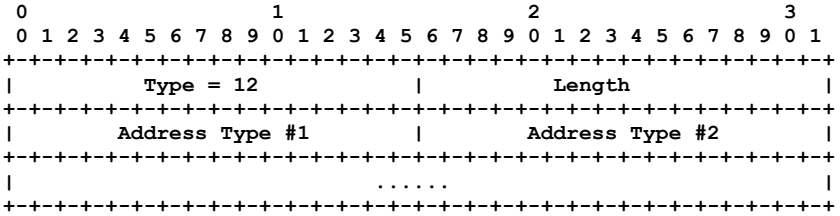
Host Name - переменный размер

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

Имя хоста **должно** содержать хотя бы один завершающий null-символ, который учитывается в поле размера.

3.3.2.1.5. Параметр Supported Address Types (12)

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

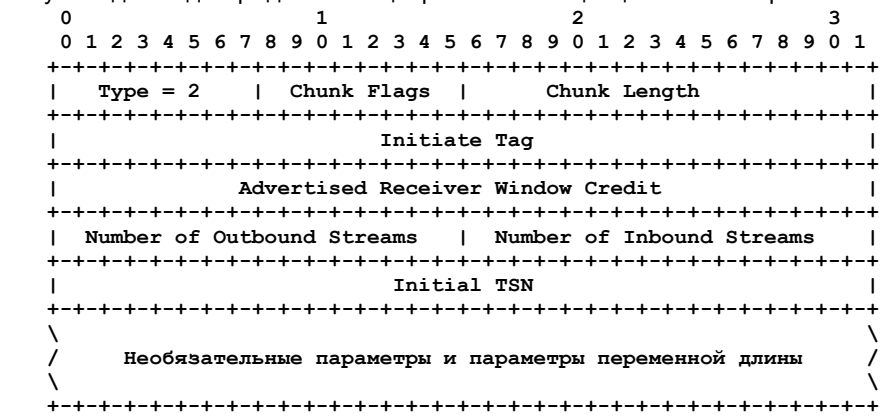


Address Type - 16 битов (целое число без знака)

В этом поле указываются типы адреса из соответствующих TLV (например, IPv4 = 5, IPv6 = 6). Значение, указывающее параметр Host Name Address, **недопустимо** использовать при передачи, а при получении оно **должно** игнорироваться.

3.3.3. Подтверждение инициализации (INIT ACK) (2)

Блок INIT ACK используется для подтверждения инициирования ассоциации SCTP. Формат блока показан на рисунке.



Параметры INIT ACK форматируются подобно параметрам блока INIT и показаны в таблицах ниже.

Таблица 7. Параметры фиксированного размера в блоках INIT ACK.

Фиксированные параметры	***Статус***
Initiate Tag	Обязательный
Advertised Receiver Window Credit	Обязательный
Number of Outbound Streams	Обязательный
Number of Inbound Streams	Обязательный
Initial TSN	Обязательный

Блоки INIT ACK содержат два дополнительных параметра - State Cookie и Unrecognized Parameter.

Таблица 8. Параметры переменного размера в блоках INIT ACK.

Переменные параметры	***Статус***	***Тип***
State Cookie	Обязательный	7
IPv4 Address[1]	Необязательный	5
IPv6 Address[1]	Необязательный	6
Unrecognized Parameter	Необязательный	8
Reserved for ECN Capable[2]	Необязательный	32768 (0x8000)
Host Name Address[3]	Отменён	11

Поле Chunk Flags в блоках INIT ACK является резервным, в нем **следует** устанавливать 0 при передаче и игнорировать при получении.

Initiate Tag - 32 бита (целое число 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 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** отбросить TCB и ей **следует** передать блок ABORT с установленным флагом T. Если такой блок INIT ACK получен в состоянии, отличном от CLOSED и COOKIE-WAIT, его **следует** просто отбросить (см. параграф 5.2.3).

Advertised Receiver Window Credit (a_rwnd) - 32 бита (целое число без знака)

Это значение указывает размер буфера (в байтах), выделенного отправителем INIT ACK для данной ассоциации.

Недопустимо устанавливать для этого параметра значение < 1500.

Получатель блока INIT ACK с a_rwnd < 1500 **должен** отбросить пакет, **следует** передать блок ABORT и использовать Initiate Tag в качестве Verification Tag. Менять состояние ассоциации **недопустимо**.

В течение срока действия ассоциации **не следует** уменьшать размер буфера (т. е., выделенный буфер не удаляется из ассоциациеи), однако конечная точка может изменить значение a_rwnd в блоках SACK.

Number of Outbound Streams (OS) - 16 битов (целое число без знака)

Число исходящих потоков, которые отправитель INIT ACK желает создать для данной ассоциации. **Недопустимо** устанавливать для этого параметра значение 0, **недопустимо** также устанавливать значение, превышающее значение MIS, переданное в блоке INIT.

Если конечная точка в состоянии COOKIE-WAIT получает INIT ACK с параметром OS = 0, она **должна** отбросить TCB и **следует** передать блок ABORT. Если такой блок INIT ACK получен в состоянии, отличном от CLOSED и COOKIE-WAIT, его **следует** просто отбросить (см. параграф 5.2.3).

Number of Inbound Streams (MIS) - 16 битов (целое число без знака)

Максимальное число потоков, которые отправитель INIT ACK позволяет создать удалённому партнёру в данной ассоциации[4]. **Недопустимо** использование нулевого значения для этого параметра.

Если конечная точка в состоянии COOKIE-WAIT получает INIT ACK с параметром MIS = 0, она **должна** отбросить TCB и **следует** передать блок ABORT. Если такой блок INIT ACK получен в состоянии, отличном от CLOSED и COOKIE-WAIT, его **следует** просто отбросить (см. параграф 5.2.3).

Initial TSN (I-TSN) - 32 бита (целое число без знака)

Начальный номер TSN, который отправитель INIT ACK будет использовать. Корректные значения лежат в диапазоне от 0 до 4294967295 и **следует** устанавливать случайное значение из этого диапазона. Для выбора случайного значения можно использовать методы, описанные в [RFC4086].

[1]Блоки INIT ACK 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 ACK параметра Host Name Address. Получатель блока INIT с Host Name Address **должен** передать блок ABORT и **может** включить в него причину ошибки Unresolvable Address.
[4]Здесь 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.

Примечание для разработчиков. Реализации **должны** быть готовы к получению INIT ACK достаточно большого (более 1500 байтов) размера по причине переменного размера State Cookie и списка адресов. Например, если отвечающая на INIT сторона имеет 1000 адресов IPv4, которые она желает сообщить, для них потребуется не менее 8000 байтов в блоке INIT ACK.

Если принят блок INIT ACK со всеми обязательными для него параметрами, получателю **следует** обработать блок INIT ACK и передать в ответ COOKIE ECHO. Получатель INIT ACK **может** группировать блок ERROR с блоком COOKIE ECHO. Однако ограниченные реализации **могут** передавать блок ABORT в ответ на INIT ACK.

Вместе с параметром Source Port Number в общем заголовке SCTP параметр IP Address в INIT ACK указывает получателю INIT ACK адрес транспортного уровня, который отправитель блока INIT ACK будет поддерживать в течение срока действия создаваемой ассоциации.

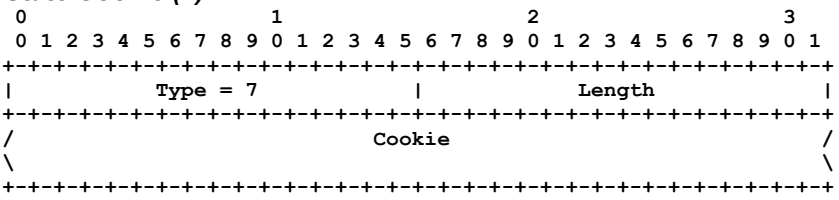
Если INIT ACK содержит хотя бы один параметр IP Address, указанные в нем адреса вместе с адресом отправителя в заголовке дейтаграммы IP, содержащей INIT ACK, **могут** использоваться получателем INIT ACK блока в качестве адресов получателя для этой ассоциации. Если в INIT ACK нет параметра IP Address, получатель блока INIT ACK **должен** использовать адрес отправителя содержащей этот блок дейтаграммы IP в качестве единственного адреса получателя для этой ассоциации.

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 в INIT ACK

Для State Cookie и Unrecognized Parameters применяется формат TLV, как определено в параграфе 3.2.1 и описано ниже. Параметр IPv4 Address описан в параграфе 3.3.2.1.1, а IPv6 Address - в 3.3.2.1.2. Параметр Host Name Address, описанный в параграфе 3.3.2.1.4, **недопустимо** включать в блок INIT ACK. Все поля TLV **должны** размещаться после полей фиксированного размера (они определены в предыдущем параграфе).

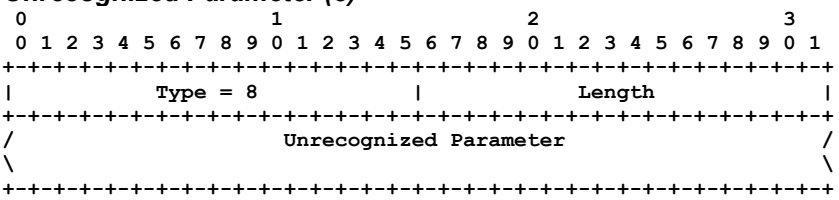
3.3.3.1.1. Параметр State Cookie (7)



Cookie - переменный размер

Значение этого параметра должно включать всю необходимую информацию о состоянии и параметрах, требуемую отправителю данного блока INIT ACK для создания ассоциации, а также код аутентификации сообщения (Message Authentication Code или MAC). Определение State Cookie дано в параграфе 5.1.3.

3.3.3.1.2. Параметр Unrecognized Parameter (8)



Unrecognized Parameter - переменный размер

Это поле содержит нераспознанный параметр (полный TLV), скопированный из блока INIT.

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** включать поля Cumulative TSN Ack, Advertised Receiver Window Credit (a_rwnd), Number of Gap Ack Blocks и Number of Duplicate TSNs.

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

Обработка a_rwnd получателем SACK рассмотрена 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. Блокам Gap Ack Block **следует** быть изолированными, т. е. TSN непосредственно перед и сразу после Gap Ack Block не получены. По определению все TSN, подтверждённые Gap Ack Block, имеют значения, превышающие Cumulative TSN Ack.

Chunk Flags - 8 битов

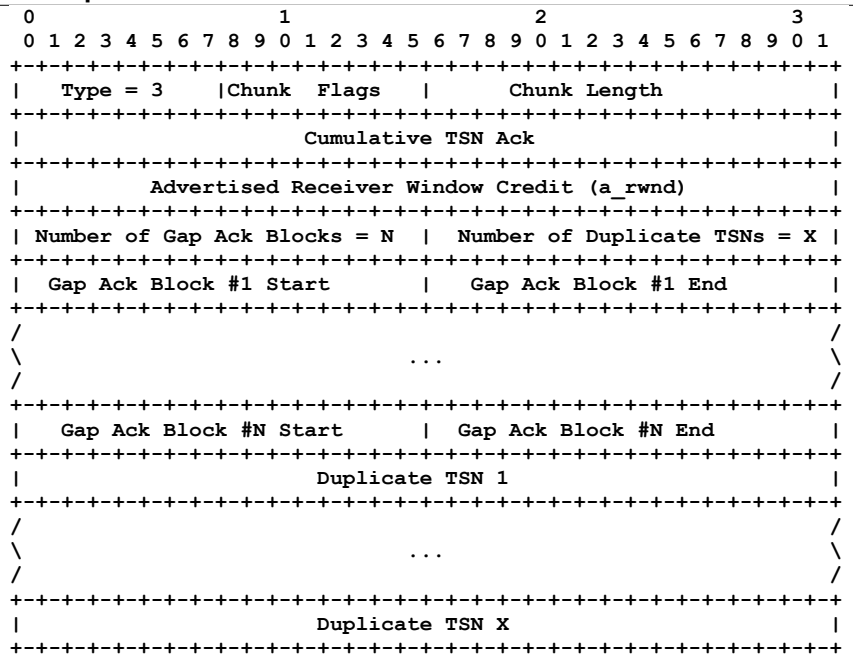
Заполняется нулями на передающей стороне и игнорируется на приёмной.

Cumulative TSN Ack - 32 бита (целое число без знака)

Наибольшее значение TSN, такое что все значения, не превышающие его, уже получены, а следующее не получено. Если блоков DATA ещё не получено, это поле содержит партнерское значение Initial TSN - 1.

Advertised Receiver Window Credit (a_rwnd) - 32 бита (целое число без знака)

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 число без знака)

Число параметров Gap Ack Block, включённых в SACK.

Number of Duplicate TSNs - 16 битов

Число дубликатов TSN, полученных конечной точкой. Каждый дубликат TSN указывается в последующем списке Gap Ack Block.

Блоки Gap Ack

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

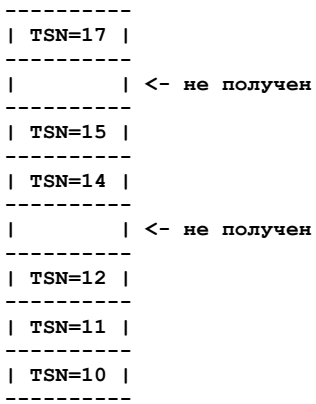
Gap Ack Block Start - 16 битов (целое число без знака)

Показывает стартовое смещение TSN 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 получен.

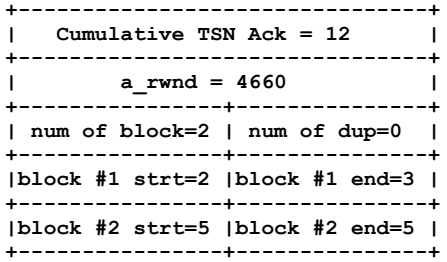
Gap Ack Block End - 16 битов (целое число без знака)

Показывает финишное смещение TSN 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 полученного блока DATA.

Предположим для примера, что недавно полученные блоки DATA в момент принятия решения о передаче SACK образуют последовательность, показанную на рисунке.



Тогда часть блока SACK должна включать поля, показанные на следующем рисунке (предполагается, что новое значение a_rwnd = 4660).



Duplicate TSN - 32 бита (целое число без знака)

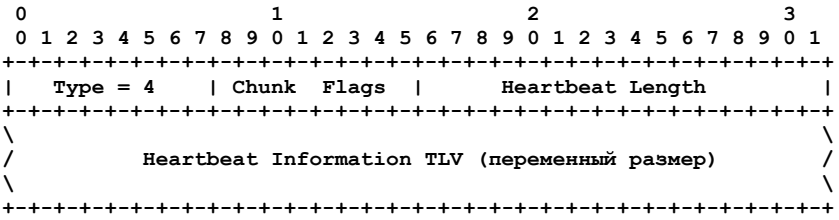
Показывает количество случаев, когда номер TSN был принят как дубликат, с момента передачи последнего блока SACK. При получении каждого дубликата TSN (до момента передачи SACK) этот дубликат добавляется в список. Счётчик дубликатов сбрасывается в 0 после отправки каждого блока SACK.

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 следующего блока SACK.

3.3.5. Запрос Heartbeat (HEARTBEAT) (4)

Конечной точке следует передавать блок HEARTBEAT (HB) своему партнёру для проверки его доступности через конкретный транспортный адрес, заданный для данной ассоциации.

Поле параметров содержит Heartbeat Information - не анализируемую (opaque) структуру данных переменного размера, понятную только отправителю.



Chunk Flags - 8 битов

Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

Heartbeat Length - 16 битов (целое число без знака)

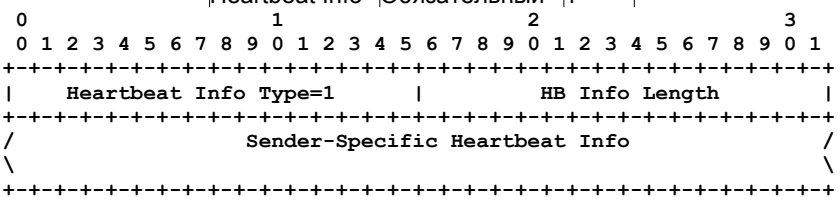
Размер блока в байтах с учётом заголовка и поля Heartbeat Information.

Heartbeat Information - переменный размер

Параметр переменного размера, который использует формат, описанный в параграфе 3.2.1.

Таблица 9. Параметры переменного размера в блоке HEARTBEAT.

Параметр	Статус	Тип
Heartbeat Info	Обязательный	1

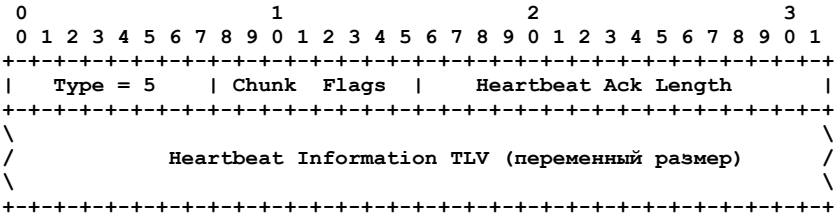


В поле Sender-Specific Heartbeat Info обычно **следует** включать информацию о текущем времени отправителя на момент передачи данного блока HEARTBEAT и транспортный адрес получателя этого блока (см. параграф 8.3). Эта информация просто «отражается» получателем в его блоке HEARTBEAT ACK (см. параграф 3.3.6). Отметим также, что сообщения HEARTBEAT служат для проверки доступности и верификации пути (см. параграф 5.4). При использовании блока HEARTBEAT для проверки пути он **должен** включать случайное значение не менее 64 битов (см. рекомендации по созданию случайных чисел в [RFC4086]).

3.3.6. Подтверждение Heartbeat (HEARTBEAT ACK) (5)

Конечная точка **должна** передавать такой блок в ответ на запрос HEARTBEAT (см. параграф 8.3). Пакет с блоком HEARTBEAT ACK всегда передаётся по тому адресу IP, который был указан в заголовке дейтаграммы, содержавшей запрос HEARTBEAT, на который передаётся отклик.

Поле параметра содержит «непрозрачную» структуру данных с переменным размером.



Chunk Flags - 8 битов

Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

Heartbeat Length - 16 битов (целое число 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 отвечает данный блок Heartbeat Acknowledgement.

Таблица 10. Параметры переменного размера в блоке HEARTBEAT ACK.

Параметр	Статус	Тип
Heartbeat Info	Обязательный	1

3.3.7. Разрыв ассоциации (ABORT) (6)

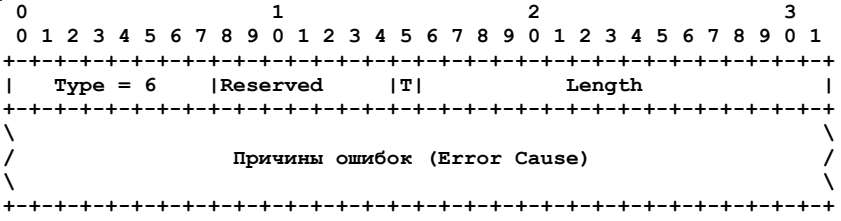
Блок ABORT передаётся партнёру для разрыва ассоциации. Блок ABORT **может** содержать параметры Error Cause, информирующие партнёра о причинах разрыва ассоциации. **Недопустимо** группировать блоки DATA с блоком ABORT. Блоки управления (кроме INIT, INIT ACK и SHUTDOWN COMPLETE) **могут** группироваться 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 TCB не найден, такой блок **должен** быть просто отброшен. 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 - 1 бит

Бит T сбрасывается в 0, если отправитель заполнил поле Verification Tag, ожидаемое партнёром. Если в Verification Tag используется «отражённое» значение, **должно** устанавливаться T = 1. Отражение означает, что переданное значение Verification Tag совпадает с принятым.

Length - 16 битов (целое число без знака)

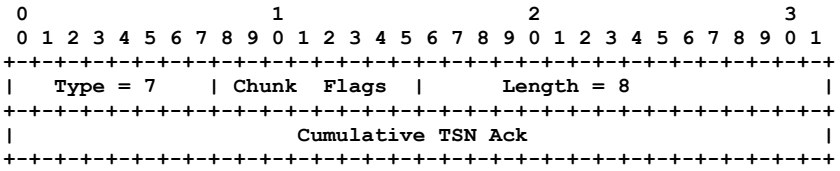
Указывает размер блока в байтах с учётом заголовка и всех полей Error Cause.

Определения причин ошибки (Error Cause) приведены в параграфе 3.3.10.

Примечание. Для проверки этого типа блоков применяются специальные правила, описанные в параграфе 8.5.1.

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 число без знака)

Показывает размер параметра и имеет значение 8.

Cumulative TSN Ack - 32 бита (целое число без знака)

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

Примечание. 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.

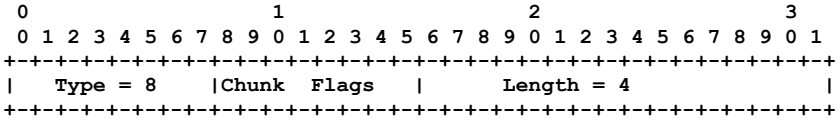
Поскольку SHUTDOWN не содержит Gap Ack Block, получателю блока SHUTDOWN **недопустимо** интерпретировать отсутствие Gap Ack Block как отказ (см. параграф 6.2).

Отправитель блока SHUTDOWN может сгруппировать его с блоком SACK для указания пропусков в полученных TSN.

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.

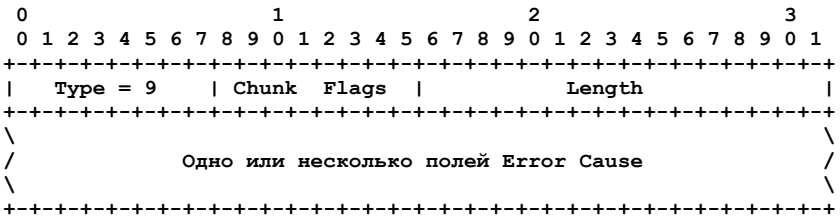


Chunk Flags - 8 битов

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 Operational Error не обязывает партнёра разрывать ассоциацию, однако такие блоки **могут** передаваться вместе с блоком ABORT как отчёт 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 число 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 параграфом 3.2.1.

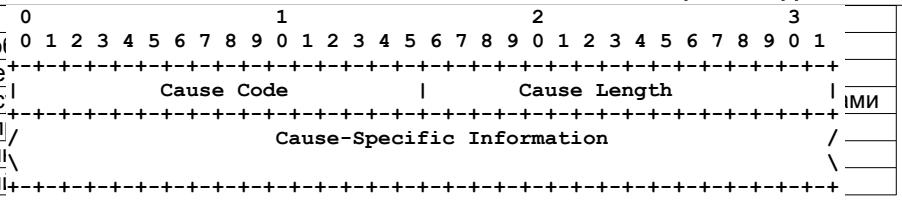
Cause Code - 16 битов (целое число без знака)

Указывает код причины, которая вызвала передачу блока.

Таблица 11. Коды причин.

Код	*Описание*	*Код*	*Описание*
1	Некорректный идентификатор потока	8	Нераспознанные параметры

Код	
2	Отсутствие об
3	Ошибка Stale
4	Нехватка рес
5	Не удалось п
6	Нераспознан
7	Некорректны



ими

Cause Length - 16 битов (целое число без знака)

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

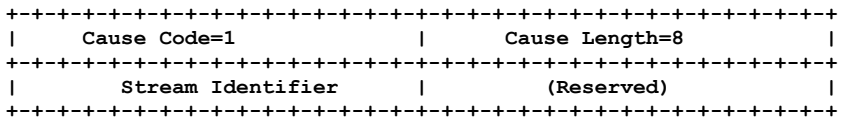
Cause-Specific Information - переменный размер

В этом поле указываются сведения об ошибке.

Причины ошибок SCTP рассматриваются в параграфах 3.3.10.1 - 3.3.10.13. Рекомендации по определению новых типов ошибок для IETF даны в параграфе 15.4.

3.3.10.1. Неприемлемый идентификатор потока (1)

Показывает, что конечная точка получила блок DATA, переданный в несуществующий поток.



Stream Identifier - 16 битов (целое число без знака)

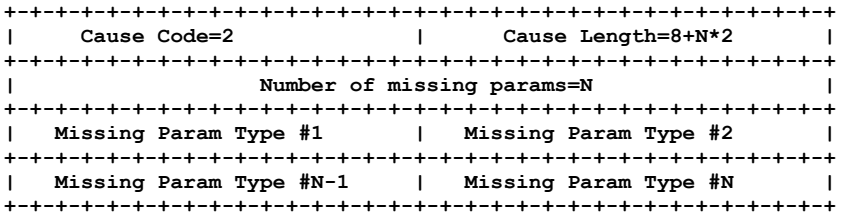
Значение Stream Identifier из блока DATA, вызвавшего ошибку.

Reserved - 16 битов

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

3.3.10.2. Отсутствует обязательный параметр (2)

Говорит об отсутствии одного или нескольких обязательных параметров TLV в принятом блоке INIT или INIT ACK.



Number of Missing params - 32 бита (целое число без знака)

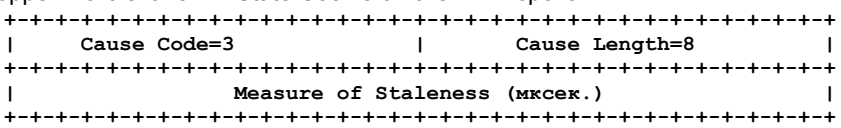
Число отсутствующих параметров, перечисленных в Cause-Specific Information (поля Missing Param Type).

Missing Param Type - 16 битов (целое число без знака)

Каждое поле содержит пропущенный обязательный параметр.

3.3.10.3. Ошибка Stale Cookie (3)

Говорит о получении корректного значения State Cookie с истекшим сроком.



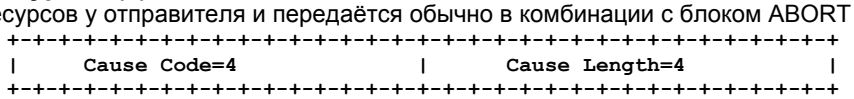
Measure of Staleness - 32 бита (целое число без знака)

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

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.

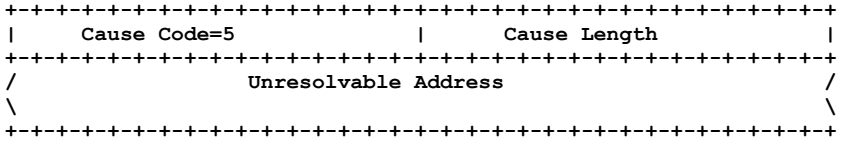
3.3.10.4. Нехватка ресурсов (4)

Указывает нехватку ресурсов у отправителя и передаётся обычно в комбинации с блоком ABORT или в нем.



3.3.10.5. Нераспознаваемый адрес (5)

Говорит о том, что отправителю не удалось распознать указанный адресный параметр (например, отправитель не поддерживает этот тип адресов) и передаётся обычно в комбинации с блоком ABORT или в нем.

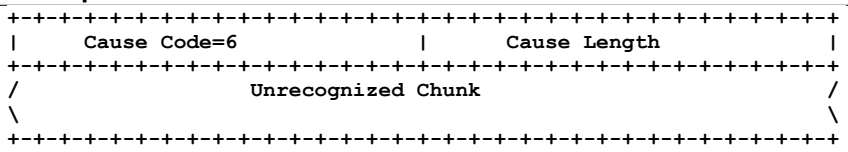


Unresolvable Address - переменный размер

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

3.3.10.6. Не распознан тип блока (6)

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

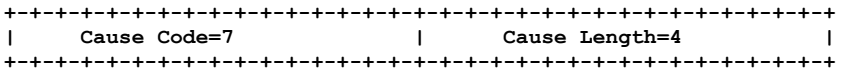


Unrecognized Chunk - переменный размер

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

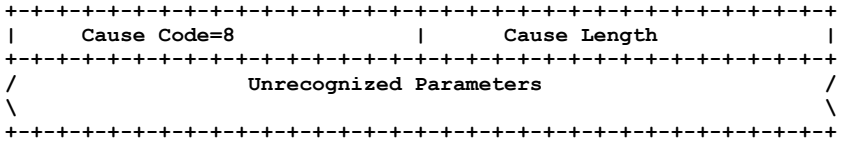
3.3.10.7. Недействительный обязательный параметр (7)

Это сообщение передаётся отправителю блока INIT или INIT ACK, когда один или несколько обязательных параметров имеют недействительные значения.



3.3.10.8. Нераспознанные параметры (8)

Это сообщение возвращается отправителю блока INIT ACK, если получателю не удаётся распознать один или несколько необязательных параметров TLV в блоке INIT ACK.

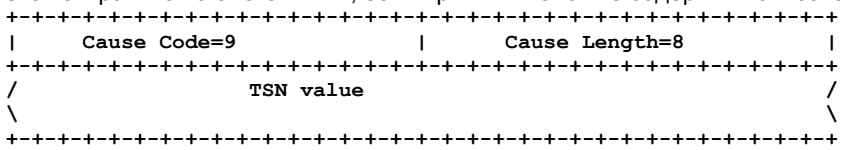


Unrecognized Parameters - переменный размер

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)

Это сообщение передаётся отправителю блока DATA, если принятый блок не содержит пользовательских данных.



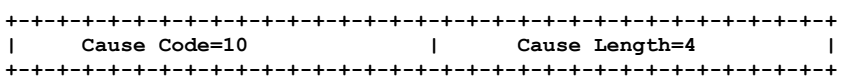
TSN - 32 бита (целое число без знака)

Значение поля TSN из блока DATA, принятого без пользовательских данных.

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

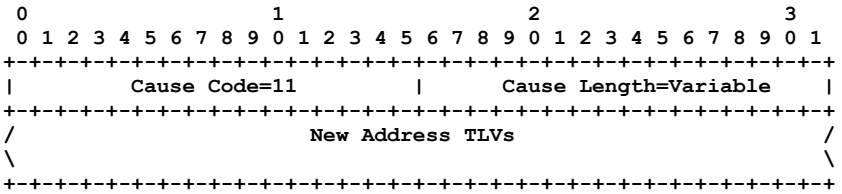
3.3.10.10. Получение Cookie во время процедуры закрытия (10)

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.



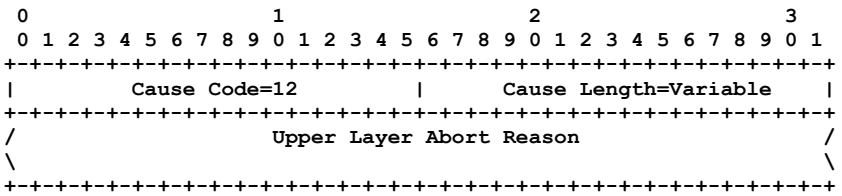
3.3.10.11. Перезапуск ассоциации с новыми адресами (11)

Был получен блок INIT в существующей ассоциации и этот блок добавляет адреса, которые раньше не были частью данной ассоциации. Новые адреса указываются в коде ошибки. Этот сообщение обычно передаётся как часть блока ABORT, отвергающего INIT (см. параграф 5.2).



Примечание. Каждый элемент New Address TLV является точной копией связанного с новым адресом TLV из блока INIT, включая Parameter Type и Parameter Length.

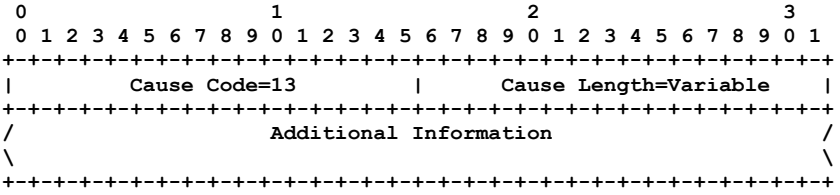
3.3.10.12. Разрыв по инициативе пользователя (12)



Эта причина ошибки **может** включаться в блоки ABORT, передаваемые по запросам вышележащего уровня. Этот уровень может указать значение Upper Layer Abort Reason, которое без изменения передаётся протоколом SCTP и **может** быть доставлено протоколу прикладного уровня у партнёра.

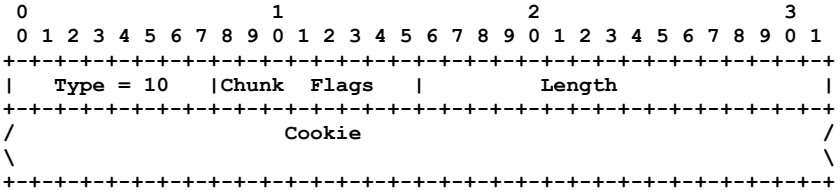
3.3.10.13. Протокольное нарушение (13)

Эта причина ошибки может включаться в блоки ABORT, передаваемые в результате обнаружения конечной точкой SCTP нарушения партнёром протокола, которое не может быть описано причинами, указанными в параграфах 3.3.10.1 - 3.3.10.12. Реализация **может** предоставить дополнительную информацию о нарушении протокола.



3.3.11. Cookie Echo (COOKIE ECHO) (10)

Этот блок используется только в процессе создания ассоциации. Он передаётся инициатором удалённому партнёру для завершения процесса создания 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 битов

Устанавливается в 0 на передающей стороне и игнорируется на приёмной.

Length - 16 битов (целое число без знака)

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

Cookie - переменный размер

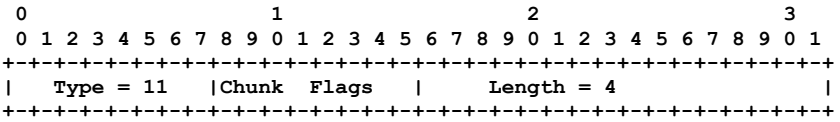
Это поле **должно** содержать точную копию параметра State Cookie, полученного в предыдущем блоке INIT ACK.

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

Примечание. Блок Cookie Echo не включает параметр State Cookie, вместо этого данные из поля Parameter Value в State Cookie становятся Chunk Value в Cookie Echo. Это позволяет реализациям поменять лишь два первых байта параметра State Cookie, чтобы сделать из него блок COOKIE ECHO.

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.



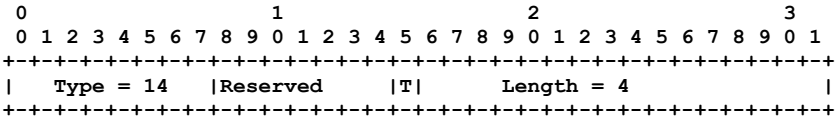
Chunk Flags - 8 битов

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 - 1 бит

Бит T сбрасывается в 0, если отправитель заполнил поле Verification Tag, ожидаемое партнёром. Если в Verification Tag используется «отражённое» значение, **должно** устанавливаться T = 1. Отражение означает, что переданное значение Verification Tag совпадает с принятым.

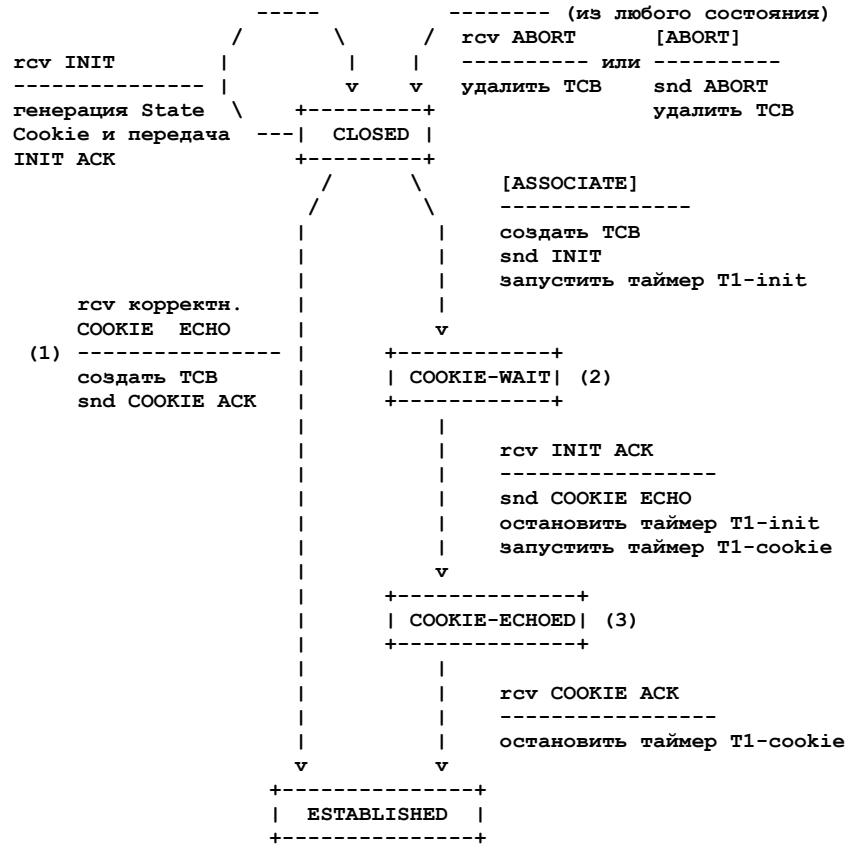
Примечание. Для проверки этого типа применяются специальные правила, описанные в параграфе 8.5.1.

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 разных 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 ниже 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[1].



(продолжение рисунка на следующей странице)

[1]Имена блоков указаны заглавными буквами, а в именах параметров только первая буква является заглавной (например, блок COOKIE ECHO и параметр State Cookie). Если смену состояния могут вызвать несколько событий или сообщений, они помечаются (A), (B) и т. д.

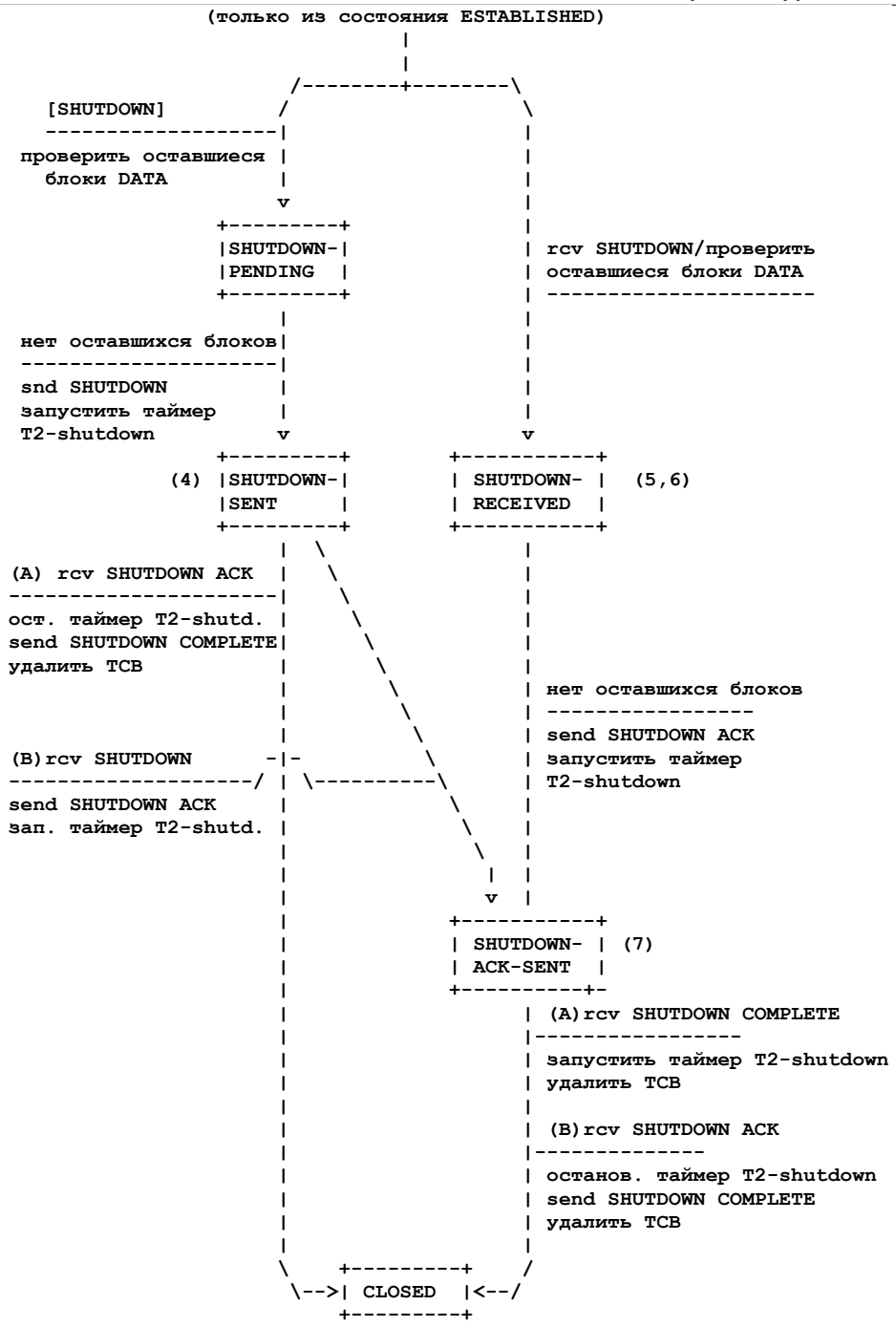


Рисунок 3. Диаграмма состояний протокола SCTP (продолжение).

1) Если параметр State Cookie в принятом блоке COOKIE ECHO недействителен (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, сохраняя состояние COOKIE-WAIT. 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, сохраняя состояние COOKIE-ECHOE. 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** воспринимать любые новые запросы на передачу от пользователя SCTP.

6) В состоянии SHUTDOWN-RECEIVED конечная точка **должна** передать или повторить 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).

5. Создание ассоциации

До того, как сможет начаться передача данных от одной конечной точки SCTP (A) к другой (Z), эти две точки должны выполнить весь процесс создания ассоциации между собой.

Пользователь SCTP в конечной точке может применять примитив ASSOCIATE для создания ассоциации с другой конечной точкой SCTP.

Примечание для разработчиков. С точки зрения пользователя SCTP ассоциация может быть создана неявно без обращения к примитиву ASSOCIATE (см. параграф 11.1.2) 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 двухсторонние потоки данных для передачи информации в обоих направлениях (см. параграф 5.1.1).

5.1. Обычное создание ассоциации

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

A) A создаёт TCB и передаёт блок INIT точке Z. В блоке INIT точка A **должна** указать свой Verification Tag (Tag_A) в поле Initiate Tag. Для Tag_A **следует** использовать случайное число из диапазона 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) Z сразу же отвечает 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 в INIT ACK значение State Cookie (см. параграф 5.1.3).

   После передачи INIT ACK с параметром State Cookie точке Z **недопустимо** выделять какие-либо ресурсы или сохранять какие-либо состояния для новой ассоциации, поскольку это сделает её уязвимой для атак на ресурсы.

C) При получении блока INIT ACK от Z точка A останавливает таймер T1-init и выходит из состояния COOKIE-WAIT. Далее A передаёт значение State Cookie из принятого блока INIT ACK в ответном блоке COOKIE ECHO, запускает таймер T1-cookie и переходит в состояние COOKIE-ECHOED.

   Блок COOKIE ECHO **может** группироваться с любыми ожидающими передачи блоками DATA, но эти блоки **должны** размещаться в пакете после COOKIE ECHO. До получения ответного блока COOKIE ACK **недопустима** передача каких-либо пакетов удалённому партнёру.

D) При получении блока COOKIE ECHO точка Z передаёт блок COOKIE ACK после создания TCB и перехода в состояние ESTABLISHED. Блок COOKIE ACK **может** объединяться с 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 переходит 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 11).

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 получен блок INIT.

Для таймеров T1-init и T1-cookie **следует** применять правила, описанные в параграфе 6.3. Если приложение сообщает партнёру несколько адресов IP, **следует** устанавливать таймеры T1-init и T1-cookie для каждого из этих адресов. При повторе блоков INIT и COOKIE ECHO **следует** использовать все адреса партнёра как при повторе блоков DATA.

Если конечная точка, получившая блок INIT, INIT ACK или COOKIE ECHO, решает не создавать ассоциацию по причине отсутствия обязательных параметров в блоке INIT или INIT ACK, недействительных значений параметров или нехватки локальных ресурсов, ей **следует** 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 в блоке INIT или INIT ACK, вызвавшего отправку блока ABORT.

Отметим, что блок COOKIE ECHO, не прошедший проверки целостности, не считается недействительным обязательным параметром и требует специальной обработки, описанной в параграфе 5.1.5.

После получения первого блока DATA в ассоциации конечная точка **должна** без промедления передать блок 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 отправитель должен 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 каждая конечная точка **должна** выполнить ряд проверок. Если значение полученное от партнёра значение MIS меньше локального OS, это означает, что удалённый партнёр не сможет поддерживать все исходящие потоки и данная точка **должна** установить для числа исходящих потоков полученное значение MIS или **может** сообщить вышележащему уровню о нехватке ресурсов на удалённой стороне. На основании этого вышележащий уровень может принять решение о разрыве ассоциации, если работа при нехватке ресурсов не приемлема для него.

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.

A) Если в принятом блоке INIT или INIT ACK нет адресных параметров, конечная точка **должна** 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, конечная точка **должна** сразу же передать блок ABORT и может указать в нем причину Unresolvable Address. Блок ABORT **следует** передавать по адресу IP, а которого получен последний пакет от партнёра.

C) Если в принятом блоке INIT или INIT ACK присутствуют только адреса IPv4/IPv6, получатель **должен** 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 партнёру.

D) Блок INIT или INIT ACK **должен** трактоваться, как относящийся к организованной ранее (или организуемой сейчас) ассоциации, если использование любого из содержащихся в нем корректных адресных параметров уже зафиксировано в существующих TCB.

Примечание 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 партнёру.

После выделения транспортного адреса из блока INIT или INIT ACK с использованием описанных выше правил конечная точка выбирает один из таких адресов как начальный основной путь.

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.

Примечание 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.

Если конечная точка SCTP, поддерживающая только один из типов адресов (IPv4 или IPv6), получает адреса IPv4 и IPv6 в блоке INIT или INIT ACK от своего партнёра, она **должна** использовать все указанные партнёром адреса, которые относятся к поддерживаемому типу. Остальные адреса **можно** игнорировать. **Не следует** в таких случаях передавать какие-либо сообщения об ошибке.

Если конечная точка SCTP указывает в параметре Supported Address Types только один из типов IPv4 и IPv6, но использует для передачи пакета с блоком INIT другой тип или перечисляет адрес другого типа в блоке INIT, адреса не указанного в параметре Supported Address Types типа получателю блока INIT также **следует** считать поддерживаемыми. **Не следует** в таких случаях передавать какие-либо сообщения об ошибке.

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 отправитель **должен** включить MAC (см., например, [RFC2104]) для защиты целостности. **Следует** также включать временную метку генерации State Cookie и время действия параметра State Cookie, а также другую информацию, требуемую для создания ассоциации, включая номера портов и Verification Tag.

Метод, используемый для генерации MAC является строго приватным для получателя блока INIT. Использование MAC обязательно для предотвращения атак на службы. Алгоритмы MAC могут различаться по производительности в зависимости от платформы. Выбор скоростного алгоритма MAC повышает устойчивость к лавинным атакам на Cookie. **Следует** применять MAC с подходящими свойствами защиты. В качестве секретного ключа **следует** использовать случайное значение (см. рекомендации [RFC4086]) подходящего размера, которое **следует** менять достаточно часто (например, каждый час). Для идентификации ключа, который будет использоваться для проверки MAC, **может** служить временная метка момента создания State Cookie.

Если State Cookie не шифруется, **недопустимо** включать в него сведения, не рассчитанные на общее применение.

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.

Конечная точка **должна** также запустить таймер T1-cookie после передачи блока COOKIE ECHO. По истечении заданного для таймера интервала конечная точка **должна** 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 (см. раздел 16). Если заданное число попыток не привело к успеху, партнёр помечается как недоступный (и ассоциация переходит в статус CLOSED).

5.1.5. Аутентификация State Cookie

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

1) Рассчитать MAC с использованием данных TCB, полученных в State Cookie, и секретного ключа. Для выбора секретного ключа **может** использоваться временная метка из State Cookie. Если секрет хранится лишь ограниченное время и секретный ключ больше не доступен, пакет с COOKIE ECHO **должен** отбрасываться без уведомления. Расчёт MAC можно выполнять в соответствии с рекомендациями [RFC2104].

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

3) Сравнить номера портов и Verification Tag в блоке COOKIE ECHO с реальными номерами портов и полем Verification Tag в общем заголовке SCTP принятого пакета. При наличии расхождений пакет **должен** быть отброшен без уведомления отправителя.

4) 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.

5) Если значение State Cookie действительно, создать ассоциацию с отправителем COOKIE ECHO, создать TCB с использованием данных из блока COOKIE ECHO и перевести конечную точку в состояние ESTABLISHED.

6) 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.

7) 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. 6), при группировке COOKIE ACK с блоком SACK, блок COOKIE ACK **должен** размещаться в пакете SCTP первым.

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

5.1.6. Пример нормального создания ассоциации

В показанном на рисунке 4 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).

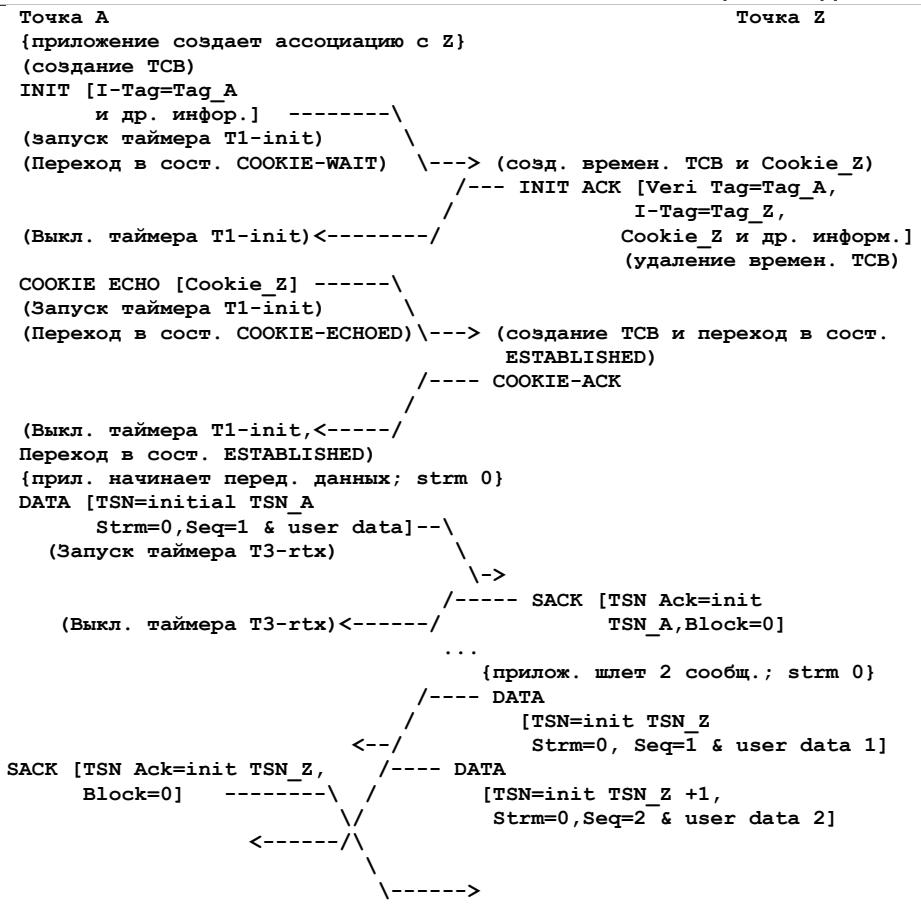


Рисунок 4. Пример создания ассоциации.

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 установочных блоков

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

Примечание. Конечная точка не получит блок, если тот не передан по транспортному адресу SCTP с транспортного адреса SCTP, связанного с данной точкой. Следовательно, обрабатывающая такие блоки конечная точка является элементом текущей ассоциации.

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 конечная точка **должна** ответить блоком INIT ACK, используя те же параметры, которые она передавала в исходном блоке INIT (включая параметр Initiate Tag). При ответе **должны** применяться приведённые ниже правила.

1) Пакет с блоком INIT ACK **должен** отправляться только по тому адресу, который был передан вышележащим уровнем в запросе на создание ассоциации.

2) Пакет с блоком INIT ACK **должен** отправляться только по адресу, указанному в принятом блоке INIT.

3) Пакет с блоком INIT ACK **следует** передавать по адресу отправителя принятого пакета с блоком INIT.

При получении блока INIT в состоянии COOKIE-ECHOED конечная точка **должна** ответить блоком INIT ACK, используя те же параметры, которые она передавала в исходном блоке INIT (включая параметр Initiate Tag), чтобы в создаваемую ассоциацию не добавлялись новые адреса. Если блок INIT показывает добавление к ассоциации нового адреса, весь блок INIT **должен** быть отброшен и в существующую ассоциацию **не следует** вносить изменений. В ответ **следует** передать блок ABORT, который **может** включать информацию об ошибке Restart of an association with new addresses (перезапуск ассоциации с новыми адресами). В информацию об ошибке **следует** включать список адресов, которые были добавлены при перезапуске ассоциации.

При отклике INIT ACK из состояния COOKIE-WAIT или COOKIE-ECHOED исходные параметры комбинируются с параметрами из полученного недавно блока INIT. Конечная точка **должна** 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, таймер T1-init **должен** продолжать отсчёт. Обычная процедура обработки State Cookie при наличии TCB позволит избавиться от дубликатов INIT в одной ассоциации.

Конечная точка, находящаяся в состоянии COOKIE-ECHOED при получении блока INIT **должна** заполнить поля Tie-Tag в TCB ассоциации и State Cookie (см. параграф 5.2.2).

5.2.2. INIT в состоянии, отличном от CLOSED, COOKIE-ECHOED, COOKIE-WAIT, 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. Перед отправкой отклика конечная точка **должна** проверить, добавляет ли неожиданный блок INIT новые адреса для ассоциации. При наличии новых адресов конечная точка **должна** отвечать блоком ABORT, копируя Initiate Tag из неожиданного блока INIT в поле Verification Tag исходящего пакета с блоком ABORT. В блоке ABORT **можно** указать причину ошибки Restart of an association with new addresses (перезапуск ассоциации с новыми адресами). В информацию об ошибке **следует** включать список адресов, добавленных к ассоциации. Если новых адресов нет, в ответ на неожиданный блок INIT блоком INIT ACK конечная точка **должна** копировать свои текущие значения Tie-Tag в резервное пространство State Cookie и TCB этой ассоциации. Эти места внутри cookie по-прежнему обозначаются Peer's-Tie-Tag и Local-Tie-Tag. Копии внутри TCB ассоциации будем обозначать Local Tag и Peer's Tag. Исходящий пакет SCTP с блоком INIT ACK **должен** включать значение Verification Tag, совпадающее с Initiate Tag в неожиданном блоке INIT. Блок INIT ACK **должен** включать новое значение Initiate Tag (случайное, см. параграф 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 или ABORT конечная точка **должна** отказаться от каких-либо действий, т. е., существующую ассоциацию, включая текущее состояние и соответствующее значение TCB, изменять **недопустимо**.

Поля Tie-Tag заполняются отличными от 0 случайными значениями только в том случае, когда существует TCB и ассоциация не находится в состоянии COOKIE-WAIT или SHUTDOWN-ACK-SENT. При обычном создании ассоциации (конечная точка находится в состоянии CLOSED) для Tie-Tag **должно** устанавливаться значение 0 (это показывает отсутствие предыдущего TCB).

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 подлинность 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 действия 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 партнёру блок ERROR с причиной ошибки Stale Cookie (случай C или D в параграфе 5.2).

   Если параметры Verification Tag в State Cookie и текущей ассоциации совпадают, следует считать параметр State Cookie действительным (случай E в параграфе 5.2) даже по истечении срока действия.

4) При действительном значении State Cookie распаковать TCB во временный TCB.

5) Выполнить подходящее действие из таблицы 12.

Таблица 12. 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 12, 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 перегрузки (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 T1-init и T1-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, но новый тег принадлежит локальной точке. Данные cookie **следует** отбросить без уведомления. Конечной точке **не следует** менять своё состояние, а запущенные таймеры **следует** сохранить.

D) Когда теги локальной и удалённой точки совпадают, конечной точке **следует** перейти в состояние ESTABLISHED, если она находится в состоянии COOKIE-ECHOED. Ей **следует** остановить таймер T1-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 перезапуске (т. е., Heartbeat ещё не удалось обнаружить недоступность точки A). Предполагается отсутствие группировки и фрагментирования.

Рисунок 5. Пример перезапуска.

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 одной из перечисленных ниже ситуаций.

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 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 действия 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** добавлять не более 1 секунды, поскольку увеличение срока действия 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 - 232-1. Важно, чтобы значение Initiate Tag было случайным - это поможет в защите от атак извне пути[1]. Для создания случайных значений Initiate Tag можно использовать методы, описанные в [RFC4086]. 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 снова создаёт ассоциацию с тем же партнёром, **должно** выбираться новое значение Verification Tag.

5.4. Проверка пути

В процессе создания ассоциации партнёры обмениваются списками адресов. Чаще всего эти списки содержат адреса, принадлежащие каждому из партнёров. Однако возможны ситуации, когда некорректно ведущий себя партнёр предложит адреса, которые ему не принадлежат. Для предотвращения нежелательных последствий этого ко всем адресам в новой ассоциации применяются приведённые ниже правила.

1) Любой адрес, переданный отправителю блока INIT его вышележащим уровнем в запросе на создание ассоциации, автоматически считается **подтверждённым**.

2) Для получателя COOKIE ECHO единственным **подтверждённым** адресом является тот, по которому был передан блок INIT-ACK.

3) Остальные адреса (кроме перечисленных в п. 1 и 2) считаются **неподтверждёнными** и должны проверяться.

Для проверки адреса конечная точка передаёт блоки HEARTBEAT, включающее 64-битовое случайное значение nonce и индикатор пути (для идентификации адреса, по которому передаётся HEARTBEAT) в параметре HEARTBEAT.

При получении HEARTBEAT ACK проверяется значение nonce в параметре Heartbeat Info на предмет совпадения с переданным по адресу, указанному в Heartbeat Info. При совпадении адрес, по которому был передан исходный блок HEARTBEAT, считается **подтверждённым** и может применяться для обычной передачи данных.

Эти процедуры проверки запускаются при переходе ассоциации в состояние ESTABLISHED и завершаются после проверки всех путей.

В каждый период RTO **может** быть отправлен пробный пакет для активного **неподтверждённого** пути в попытке перевести этот путь в состояние **подтверждённого**. Если в процессе проверки путь становится неактивным, скорость отправки проб снижается до обычной скорости передачи HEARTBEAT. По завершении отсчёта таймера RTO значения счётчика ошибок для протестированного, но не **подтверждённого** пути увеличивается на 1 и рассматривается вопрос отказа на этом пути (см. параграф 8.2). При проверке **неподтвержденных** адресов значение общего счётчика ошибок в ассоциации не инкрементируется.

Число HEARTBEAT, передаваемых в течение RTO, **следует** ограничивать значением параметра HB.Max.Burst. Распределение HEARTBEAT по адресам партнёра при проверке путей реализация определяет самостоятельно.

При подтверждении пути вышележащему уровню **может** передаваться уведомление.

Конечной точке **недопустимо** передавать какие-либо блоки по **неподтверждённому** адресу за исключением перечисленных ниже случаев.

- Блоки HEARTBEAT, включающие nonce, **можно** передавать по **неподтвержденным** адресам.
- Блоки HEARTBEAT ACK **можно** передавать по **неподтвержденным** адресам.

[1]off-path.

- Блоки COOKIE ACK **можно** передавать по **неподтвержденным** адресам, но они **должны** группироваться с блоками HEARTBEAT, включающими nonce. Реализациям, не поддерживающим группировку блоков, **недопустимо** передавать COOKIE ACK по **неподтвержденным** адресам.

- Блоки COOKIE ECHO **можно** передавать по **неподтвержденным** адресам, но они **должны** группироваться с блоками HEARTBEAT, включающими nonce, и **недопустимо** использование пакетов с размером больше MTU на этом пути. Если реализация не поддерживает группировку блоков или после группировки COOKIE ECHO с HEARTBEAT (включающим nonce) размер пакета превысит MTU для пути, **недопустимо** передавать COOKIE ECHO по **неподтвержденным** адресам.

6. Передача пользовательских данных

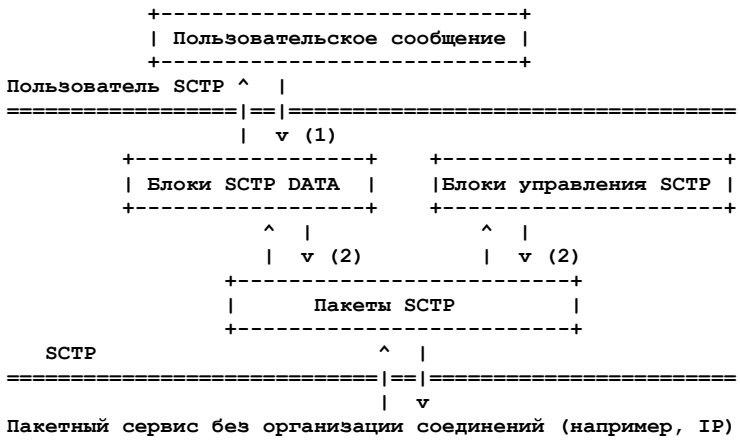
Данные **должны** передаваться только в состояниях ESTABLISHED, SHUTDOWN-PENDING и SHUTDOWN-RECEIVED. Единственным исключением из этого правила является возможность включения блоков DATA в пакеты, содержащие блок COOKIE ECHO, в состоянии COOKIE-WAIT.

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

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

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) - точку, принимающую блок DATA. Получатель подтверждает приём данных блоком SACK.



Примечания.

1) При преобразовании пользовательских сообщений в блоки DATA конечная точка **должна** фрагментировать большие сообщения. Размер каждого блока DATA **следует** делать не больше максимального размера блока данных в ассоциации (Association Maximum DATA Chunk Size или AMDCS). Получатель обычно будет собирать принятые фрагменты из блоков DATA до передачи сообщения пользователю (см. параграф 6.9).

2) Несколько блоков DATA и блоков управления **может** группироваться отправителем в один пакет SCTP, пока размер результирующего пакета не будет превышать значение PMTU. 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.

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 параграфе приведены правила для отправки блоков DATA. В частности, определена проба нулевого окна, требуемая для предотвращения неопределённого ожидания ассоциации в случае потери пакетов с блоками SACK для обновления окна.

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 исходящих блоков DATA.

A) В любой момент для отправителя **недопустимо** передавать новые данные по какому-либо транспортному адресу партнёра, если значение rwnd, полученное от того, говорит об отсутствии свободного пространства в буфере (т. е., rwnd меньше размера следующего блока данных, см. параграф 6.2.1), за исключением пробы нулевого окна.

Пробой нулевого окна (zero window probe) является, передаваемый, когда анонсированное получателем окно имеет нулевой размер. Это позволяет отправителю проверить изменение rwnd, которое он пропустил из-за потери блоков SACK, направленных от получателя к отправителю. Зонд нулевого окна **должен** передаваться лишь в том случае, когда это разрешает cwnd (см. п. B). Такие пробы **следует** передавать только в тех случаях, когда все остающиеся блоки DATA имеют кумулятивное подтверждение и нет находящихся в пути блоков DATA. Реализации **должны** поддерживать проверку нулевого окна.

Если отправитель продолжает принимать новые пакеты от партнёра во время проверки нулевого окна, отсутствие подтверждения проб **не следует** использовать для инкрементирования счётчика ошибок ассоциации или каких-либо транспортных адресов партнёра. Это связано с тем, что получатель **может** держать своё окно закрытым в течение неопределённого времени. Поведение получателя, анонсирующего нулевое окно, описано в параграфе 6.2. Отправителю **следует** передавать первую пробу нулевого окна по истечении 1 RTO с момента обнаружения закрытия окна получателем, а также **следует** экспоненциально увеличивать интервал между проверками. Отметим также, что значение cwnd **следует** подстраивать в соответствии с параграфом 7.2.1. Проверки нулевого окна не оказывают влияния на расчёт cwnd.

Отправитель **должен** также поддерживать алгоритм отправки новых блоков DATA, позволяющий предотвратить синдром SWS[1], как описано в [RFC1122]. Алгоритм может быть похож на описанный в параграфе 4.2.3.4 [RFC1122].

B) В любой момент времени для отправителя **недопустимо** передавать информацию по данному транспортному адресу, если у него имеется не менее cwnd + (PMDCS - 1) неподтвержденных байтов данных для этого адреса. Если данные доступны, отправителю **следует** превышать cwnd на величину до PMDCS - 1 при новой передаче данных, если размер находящихся в пути данных не достигает значения cwnd. Нарушение cwnd **должно** составлять лишь 1 пакет.

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

D) Когда приходит время отправителю передавать новые блоки DATA, **следует** использовать протокольный параметр Max.Burst для ограничения числа передаваемых пакетов. Ограничение **может** быть реализовано путём изменения cwnd

```
if((flightsize + Max.Burst*PMDCS) < cwnd)
   cwnd = flightsize + Max.Burst*PMDCS
```

или просто **может** быть строго ограничено число пакетов, выдаваемых модулем отправки. При расчёте числа пакетов для передачи, в частности, по приведённой выше формуле, **не следует** менять cwnd на постоянной основе.

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

Несколько блоков DATA, подготовленных для передачи, **можно** сгруппировать в один пакет. Кроме того, передаваемые повторно блоки DATA **можно** группировать с новыми блоками DATA, пока размер результирующего пакета не превышает PMTU. 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, если имеются неподтвержденные блоки DATA, отправителю **следует** создать блок SACK для всех неподтвержденных данных и сгруппировать его с передаваемыми блоками DATA, пока размер результирующего пакета не превышает значение PMTU (см. параграф 6.2).

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.

При старте или перезапуске таймера T3-rtx его значение **следует** устанавливать в соответствии с правилами, приведёнными в параграфах 6.3.2 и 6.3.3.

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

Для каждого потока отправителю **недопустимо** иметь более $2^{16} - 1$ упорядоченных пользовательских сообщений в текущем окне передачи.

Когда отправитель блока DATA может получить преимущества от незамедлительной передачи ему соответствующего блока SACK, отправитель **может** установить флаг I в заголовке блока DATA. Следует отметить, что причина установки отправителем бита I не имеет значения для получателя.

Причины установки бита I включают (но не ограничиваются) указанные ниже (см. раздел 4 в [RFC7053]):

- приложение запрашивает установку бита I в последнем блоке DATA пользовательского сообщения для представления этого сообщения реализации SCTP (параграф 11.1);
- отправитель находится в состоянии SHUTDOWN-PENDING;
- отправка блока DATA заполняет окно перегрузки или окно получателя.

6.2. Подтверждение приёма блоков DATA

Конечная точка SCTP всегда **должна** подтверждать получение каждого корректного блока DATA, когда этот блок приходит в окне приёма.

[1]Silly window syndrome - синдром неразумного окна.

Когда получатель анонсирует размер окна 0, он **должен** отбрасывать все новые входящие блоки DATA со значением TSN, превышающим максимальное полученное до этого значение TSN. Если в новом входящем блоке DATA значение TSN меньше максимального из принятых до этого блоков, получателю **следует** отбросить наибольшее значение TSN, хранящееся для упорядочения, и воспринять новый блок DATA. В любом случае при отбрасывании блока DATA получатель **должен** незамедлительно вернуть блок SACK с текущим окном приёма, показывающим лишь блоки DATA, полученные и воспринятые к этому моменту. Отброшенные блоки DATA **недопустимо** включать в SACK, поскольку они не были восприняты. Получатель **должен** также иметь алгоритм анонсирования своего приёмного окна для предотвращения синдрома SWS, как описано в [RFC0813]. Это алгоритм может быть похож на описанный в параграфе 4.2.3.3 [RFC1122].

Следует придерживаться рекомендаций по использованию алгоритма отложенных подтверждений, описанного в параграфе 4.2 [RFC5681]. В частности, подтверждения **следует** генерировать по меньшей мере для каждого второго принятого 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 агрессивными, 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 по мере потребления данных принимающим приложением. При расширении окна (opens up) получателю SCTP **следует** передавать дополнительные блоки SACK для обновления окна, даже если новых данных не получено. Получатель **должен** избегать передачи многочисленных обновлений окна и, в частности, больших пиков (large burst) таких обновлений. Одним из способов достижения этого является передача обновления окна лишь при его увеличении по меньшей мере на четверть размера приёмного буфера ассоциации.

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** разрешать установку максимальной задержки (параметр SACK.Delay) более 500 мсек. Иными словами, реализация **может** устанавливать SACK.Delay менее 500 мсек, но в любом случае **недопустимо** устанавливать значение более 500 мсек.

Подтверждения **должны** передаваться в блоках SACK до тех пор, пока от 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). 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, указываются в полях Gap Ack Block. Конечная точка SCTP **должна** включать столько Gap Ack Block, сколько можно поместить в один блок SACK с учётом текущего значения PMTU.

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 SCTP, содержащего блок DATA с установленным битом I, получателю **не следует** задерживать отправку соответствующего блока SACK, т. е. ему **следует** передать этот блок SACK незамедлительно.

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 DATA содержит также новый блоки DATA, конечная точка **может** передать SACK незамедлительно. Обычно получение дубликатов 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 сведения о дубликатах 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. Это 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, сокращает 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).

В некоторых случаях получатель может 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, хотя 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** учитывать отброшенные блоки при расчёте a_rwnd.

Конечной точке **не следует** отзывать SACK и отбрасывать данные. Этой процедурой следует пользоваться только в экстремальных ситуациях (например, при нехватке памяти). Получателю данных **следует** учитывать, что отбрасывание данных, подтверждённых в Gap Ack Block, может привести к неоптимальной работе отправителя и потере производительности.

Приведённый на рисунке 7 пример иллюстрирует использование отложенных подтверждений.

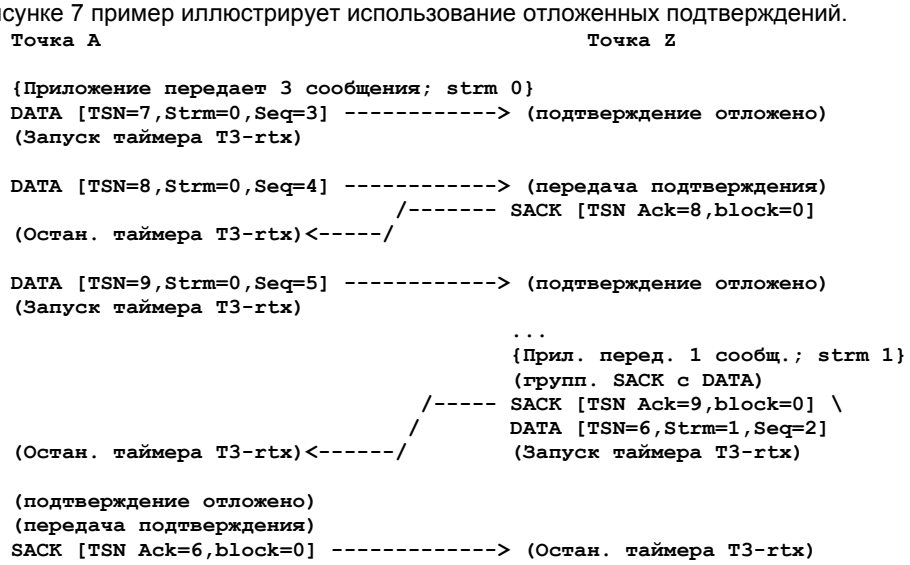


Рисунок 7. Пример подтверждения с задержкой.

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

Конечной точке **не следует** передавать блоков DATA без пользовательских данных. Это избавляет от необходимости возврата пустых пользовательских сообщений в API сокета, как отмечено в [RFC6458].

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 обнаружить 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 = a_rwnd[1] из блока INIT или INIT ACKот партнёра.

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), размер этого блока добавляется к значению rwnd.

D) Всякий раз при получении SACK конечная точка выполняет указанные ниже 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 в SACK отсутствует TSN, который был ранее подтверждён в Gap Ack Block (например, получатель отказался от данных), рассматривается соответствующий блок DATA, который может оказаться пропущенным. Блок считается отсутствующим для Fast Retransmit (параграф 7.2.4) и если нет запущенного таймера для адреса получателя, по которому блок DATA был передан изначально, для этого адреса запускается таймер T3-rtx.

   iv. Если Cumulative TSN Ack совпадает со значением для точки выхода из процедуры Fast Recovery (параграф 7.2.4) или превышает его, процедура Fast Recovery завершается.

[1]Advertised Receiver Window Credit - анонсированное окно приёма.

6.3. Управление таймером повтора передачи

Конечная точка SCTP использует таймер повтора передачи T3-rtx для обеспечения доставки данных при отсутствии обратной связи от партнёра. Время этого таймера называют тайм-аутом повтора или RTO.

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 и RTTVAR.

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

```
RTTVAR = (1 - RTO.Beta) * RTTVAR + RTO.Beta * |SRTT - R'|
SRTT = (1 - RTO.Alpha) * SRTT + RTO.Alpha * R'
```

Примечание. Значение SRTT, используемое для обновления RTTVAR, представляет собой SRTT до обновления с использованием второго назначения.

После расчёта следует обновить

```
RTO = SRTT + 4 * RTTVAR.
```

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 корректировались примерно с одной же частотой (в терминах числа 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 блоков, поскольку нет возможности различить, к какой из переданных копий относится полученный отклик.

Измерение RTT с использованием блока с TSN r **следует** выполнять лишь в тех случаях, когда нет блоков с номерами TSN не больше r, повторно переданными с момента первой передачи TSN r.

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 **можно** поместить в RTO при условии, что оно не меньше RTO.max секунд.

К дискретности часов (G) при измерении RTT и различных переменных состояния требований не предъявляется.

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

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

Предложенные значения параметров приведены в разделе 16.

6.3.2. Правила для таймера повторной передачи

R1) Каждый раз при передаче блока DATA (включая повторы) 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 тайм-аута предыдущего таймера T3-rtx, связанного с этим адресом, как указано ниже в правиле E2.

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

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

R4) При получении SACK для пропущенного 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).

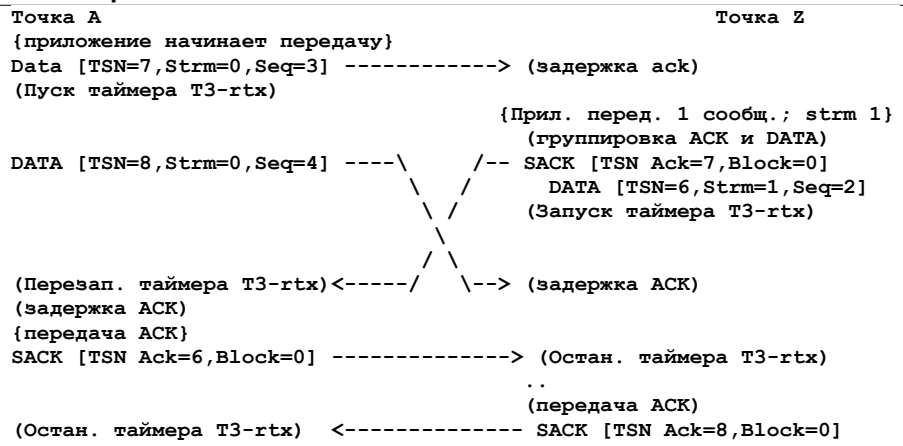


Рисунок 8. Пример правил для таймера.

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 = PMDCS.

E2) Для этого адреса устанавливается RTO = RTO * 2. Максимальное значение, рассмотренное в правиле C7 (RTO.max), **может** служить верхней границей для операций удваивания.

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

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).

Любые блоки DATA, переданные по адресу, для которого истекло время T3-rtx, но не был заполнен пакет SCTP размером не более PMTU (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 восстановлением при отказах (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 переданы 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.

ULP в конечной точке выбирает один из адресов многодомного партнёра для основного пути (параграфы 5.1.2 и 10.1).

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

Конечной точке **следует** передавать блоки откликов (например, INIT ACK, COKIE ACK, HEARTBEAT ACK) по тому же транспортному адресу, с которого был получен управляющий блок, вызвавший передачу отклика.

Выбор транспортного адреса получателя пакетов с блоками SACK зависит от реализации. Однако конечной точке **не следует** менять транспортный адрес получателя SACK в случае приёма блоков DATA с тем же адресом отправителя.

При подтверждении нескольких 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 применять 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, отличающемуся от последнего адреса получателя, по которому был передан блок.

Когда партнёр конечной точки является многодомным, ей **следует** передавать ускоренные повторы по тому же транспортному адресу получателя, куда данные направлялись исходно. Если основной путь сменился, а исходные данные переданы по старому основному пути до Fast Retransmit, реализация **может** отправить повтор по новому пути.

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 желанию 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 ULP, конечной точке SCTP следует предпринять попытку передачи данных по другому активному адресу получателя, если таковой имеется.

При повторе передачи данных в результате тайм-аута, 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 действительный идентификатор потока. Если конечная точка принимает блок DATA с недействительным идентификатором, ей **следует** подтвердить получение данных в соответствии с обычной процедурой, незамедлительно передать блок ERROR с причиной ошибки Invalid Stream Identifier (недейсвительный идентификатор потока, см. параграф 3.3.10) и отбросить блок DATA. Конечная точка **может** группировать блок ERROR в один пакет с блоком SACK.

Порядковые номера во всех потоках **должны** начинаться с нуля при создании ассоциации. Значение Stream Sequence Number **должно** увеличиваться на 1 для каждого упорядоченного пользовательского сообщения, переданного в выходной поток. При достижении порядковым номером в потоке значения 65535 следующим номером снова **должен** быть 0. Для неупорядоченных пользовательских сообщение менять Stream Sequence Number **недопустимо**.

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 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 TSN блоков DATA

При получении нового блока DATA конечная точка проверяет непрерывность порядковых номеров TSN. Если обнаружен пропуск в порядковых номерах принятых блоков DATA, конечной точке **следует** незамедлительно передать блок SACK с Gap Ack Block. Получатель продолжает передачу SACK после приёма 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.

При получении блока SACK конечная точка **должна** удалить из выходной очереди все блоки DATA, которые были подтверждены в Cumulative TSN Ack блока SACK. Передающая конечная точка **должна** считать «отсутствующими» все блоки DATA с номерами TSN, не включёнными в Gap Ack Block, которые меньше максимального TSN из блока SACK. 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 порядковых номеров.

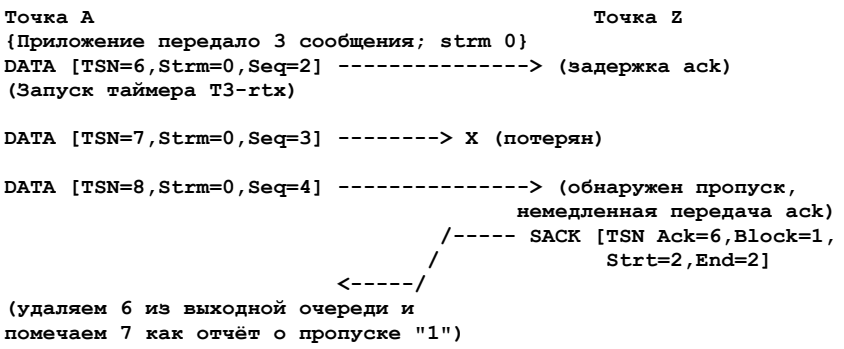


Рисунок 9. Отчёт о пропуске с использованием SACK.

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

6.8. Расчёт контрольных сумм CRC32c

При передаче пакета SCTP конечная точка **должна** для обеспечения возможности контроля целостности данных включить в пакет значение контрольной суммы CRC32c в соответствии с приведённым ниже описанием.

После создания пакета (общий заголовок SCTP и один или несколько управляющих блоков или блоков DATA) отправитель **должен** выполнить перечисленные ниже операции.

1) Заполнить поле Verification Tag общего заголовка SCTP и установить нулевое значение в поле контрольной суммы.
2) Рассчитать контрольную сумму CRC32c с включением общего заголовка SCTP и всех блоков из пакета. Описание алгоритма CRC32c приводится в Приложении A.
3) Поместить полученное значение в поле контрольной суммы общего заголовка, не изменяя остальных битов.

При получении пакета SCTP принимающая конечная точка **должна** сначала проверить значение контрольной суммы CRC32c в заголовке, как описано ниже.

1) Сохранить полученное в пакете значение контрольной суммы CRC32c.
2) Заменить 32-битовое поле контрольной суммы принятого пакета SCTP значением 0 и рассчитать контрольную сумму CRC32c для полученного пакета.
3) Сравнить сохранённое значение (из пакета) CRC32c с контрольной суммой, полученной в результате расчёта. Если значения не совпадут, получатель **должен** считать принятый пакет недействительным.

По умолчанию недействительные пакеты SCTP отбрасываются без уведомления.

Аппаратным реализациям **следует** обеспечивать возможность программной проверки контрольных сумм.

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 PMTU. Если реализация не поддерживает фрагментацию, конечная точка **должна** возвращать вышележащему уровню код ошибки, когда размер пакета SCTP превышает текущее значение PMTU, передать сообщения избыточного размера **недопустимо**.

Реализация SCTP **может** обеспечивать вышележащему уровню механизм запрета фрагментирования при передаче блоков DATA. При отключённой фрагментации блоков DATA реализация SCTP **должна** вести себя так, будто фрагментация не поддерживается, т. е. отвергать вызовы, приводящие к созданию пактов SCTP, размер которых превышает PMTU.

Примечание для разработчиков. В случае ошибки для возврата кода вышележащему уровню нужно использовать примитив SEND, рассматриваемый в параграфе 11.1.5.

Если партнёр является многодомным, конечной точке **следует** выбирать размер блока DATA, не превышающий значение AMDCS.

После фрагментации сообщения оно не может быть фрагментировано ещё раз. Если же значение PMTU уменьшилось, **должна** использоваться фрагментация IP. Поэтому реализация SCTP может сталкиваться в отказом, если фрагментация IP не работает на каком-либо пути. Определение значений PMTU 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 на несколько блоков DATA. Отправителю **следует** выбирать размер блоков DATA, не превышающий AMDCS.

2) Передающая сторона **должна** выделять 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 для битов B/E в первом блоке DATA значение 10, в последнем - 01, а в остальных - 00.

Конечная точка **должна** распознавать 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.

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. раздел 11) 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 PMTU.

Если партнёр является многодомным, передающей стороне **следует** выбирать размер пакета так, чтобы он не превышал значение PMTU для текущего основного пути.

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

Примечание. Поскольку управляющие блоки размещаются в пакете первыми, 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). Неполным является блок, которым не помещается в пакет SCTP (пакет слишком мал для включения всех байтов, указанных в поле размера блока).

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 перегрузки

Контроль перегрузки является одной из базовых функций SCTP. Для контроля перегрузок применяются механизмы, описанные в этом разделе.

Примечание для разработчиков. При условии удовлетворения заданных требований к производительности реализации могут выбрать более консервативный алгоритм контроля перегрузки, нежели описано ниже.

Алгоритмы контроля перегрузки протокола SCTP основаны на [RFC5681]. В этом разделе описано, как алгоритмы, определённые в [RFC5681], адаптированы для использования в SCTP. Сначала рассматриваются различия между протоколами TCP и SCTP, а после этого описывается схема контроля перегрузки в SCTP. В описании используется та же терминология, которая принята для протокола TCP.

Контроль перегрузки в SCTP всегда применяется ко всей ассоциации (соединению), а не к отдельным потокам.

7.1. Различия в контроле перегрузки 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 и в SCTP сведения из Gap Ack Blocks блоков SACK являются лишь консультативными. В SCTP любой блок DATA, подтверждённый с помощью SACK (включая блоки DATA, полученные с нарушением порядка), не считается доставленным 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). Поэтому 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 перегрузки. 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 ассоциаций между конечными точками, каждая из которых может быть доступна по нескольким транспортным адресам. Использование различных адресов может вести к организации разных путей между парой конечных точек и в идеальном случае для каждого пути нужно применять свои параметры контроля перегрузки. Приведённая здесь трактовка контроля перегрузки для многодомных получателей является новинкой протокола SCTP и может быть уточнена в будущем. Текущие алгоритмы основаны на приведённых ниже 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 перегрузки 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. В [RFC5681] этот период указан как тайм-аут повторной передачи.

- Для каждого адреса получателя конечная точка выполняет процедуру замедленного старта (slow-start) при первой передаче 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 и Congestion Avoidance

Алгоритмы замедленного старта и предотвращения перегрузки **должны** использоваться конечной точкой для контроля количества данных, передаваемых в сеть. Контроль перегрузки 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 алгоритмы, однако для отправителя **недопустимо** быть более агрессивным, нежели позволяют алгоритмы.

Подобно TCP, конечная точка SCTP использует 3 указанных ниже переменных для управления скоростью передачи.

- Анонсируемый получателем размер окна приёма (rwnd[1], в байтах) устанавливается получателем данных с учётом возможностей выделения буферов для принимаемых пакетов.

  Примечание. Эта переменная имеет общее значение для всей ассоциации.

- Окно контроля перегрузки (cwnd[2], в байтах) устанавливается отправителем с учётом условий в сети.

  Примечание. Эта переменная поддерживается для каждого адреса получателя.

- Порог замедленного старта (ssthresh[3], в байтах) используется отправителем для принятия решения о выборе используемого алгоритма контроля перегрузки (slow start или congestion avoidance).

  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 у партнёра (если тот является многодомным). При расчёте этих переменных **следует** использовать размер блоков DATA с учётом заполнения.

Только переменная rwnd имеет общее 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 в начале передачи или при восстановлении потерь, обнаруженных по таймеру повтора передачи.

- В качестве стартового значения cwnd до передачи блоков DATA или после достаточно долгого бездействия **должно** выбираться значение min(4*PMDS, max (2*PMDS, 4404) байт для адреса партнёра IPv4 и min(4*PMDS, max (2*PMDS, 4344) байт для IPv6.

- Начальное значение cwnd после тайм-аута повтора передачи **должно** быть не более PMDS и разрешается лишь один находящийся в пути пакет без подтверждения.

- Начальному значению ssthresh **следует** быть произвольно большим (например, размер наибольшего возможного анонсируемого окна).

- При положительном значении cwnd конечная точка может иметь cwnd ожидающих подтверждения байтов данных для этого транспортного адреса. **Следует** поддерживать ограниченное превышение, как указано в правиле B параграфа 6.1.

- Если cwnd не больше ssthresh, конечная точка SCTP **должна** использовать алгоритм slow-start для увеличения размера cwnd только в том случае, когда текущее окно перегрузки полностью использовано и отправитель данных не находится в состоянии Fast Recovery. Размер окна cwnd можно увеличивать только при выполнении обох условий, в противном случае увеличение **недопустимо**. Если все условия выполнены, значение cwnd **должно** увеличиваться не более чем на меньшее из двух значений:

  1) общий размер подтверждённых блоков DATA

  2) L раз по PMDS получателя.

  Первое ограничение обеспечивает защиту от атак ACK-Splitting, описанных в [SAVAGE99]. Положительному целому числу L **следует** иметь значение 1 и **можно** использовать большее значение. Выбор L описан в [RFC3465].

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

- В случаях, когда партнёр является многодомным и конечная точка получает подтверждение SACK, ведущее к обновлению cwnd, ей **следует** обновить значение cwnd (или несколько значений cwnd) для адреса получателя, по которому были переданы подтверждённые данные.

- Когда конечная точка не передаёт по данному транспортному адресу, в качестве cwnd для этого адреса **следует** установить max(cwnd/2, 4*PMDS) один раз за RTO. До первого обновления cwnd для параметра ssthresh **следует** установить значение cwnd.

7.2.2. Предотвращение перегрузки

При cwnd > ssthresh, значение cwnd **следует** увеличивать на PMDS за каждый период RTT, если у отправителя имеется не менее cwnd байтов неподтвержденных данных для соответствующего транспортного адреса. Базовые рекомендации по увеличению cwnd в случае предотвращения перегрузки приведены ниже.

- SCTP **может** увеличивать cwnd на PMDS.

- SCTP **следует** увеличивать cwnd на PMDS 1 раз за период RTT, когда у отправителя есть не менее cwnd остающихся байтов данных для соответствующего транспортного адреса.

- SCTP **недопустимо** увеличивать cwnd более чем на PMDS за период RTT.

На практике эта процедура может быть реализована описанным ниже способом.

- Устанавливается partial_bytes_acked = 0.

- Если cwnd > ssthresh, всякий раз при получении SACK значение partial_bytes_acked увеличивается на общее число байтов (включая заголовок блока и заполнение) во всех новых блоках DATA, подтверждённых этим SACK, включая блоки, подтверждённые новым значением Cumulative TSN Ack, Gap Ack Block и и число байтов в блоках-дубликатах, указанных в Duplicate TSN.

- Когда (1) значение partial_bytes_acked становится больше cwnd и (2) до прибытия SACK у отправителя имеется менее cwnd остающихся данных (т. е., до прибытия SACK, объем переданных, но ещё не подтверждённых данных меньше cwnd), устанавливается partial_bytes_acked = cwnd.

- Когда (1) 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 еще не подтверждённых данных не меньше cwnd), значение partial_bytes_acked уменьшается на cwnd, а затем cwnd увеличивается на PMDS.

- Как и при замедленном старте в качестве значения cwnd для адреса получателя следует установить max(cwnd/2, 4*PMDS) за период 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 перегрузки

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

```
ssthresh = max(cwnd/2, 4*PMDS)
cwnd = ssthresh
partial_bytes_acked = 0
```

Обычно потеря пакета приводит к уменьшению cwnd вдвое.

По завершении отсчёта таймера T3-rtx для адреса, SCTP **следует** выполнить процедуру slow start, установив

```
ssthresh = max(cwnd/2, 4*PMDS)
cwnd = PMDS
partial_bytes_acked = 0
```

и обеспечить 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 использует отложенные подтверждения. Однако при обнаружении пропуска в TSN **следует** начать передачу подтверждений SACK сразу после доставки каждого пакета, пока пропуск в порядковых номерах не будет заполнен.

Всякий раз при получении SACK, указывающего на пропуск TSN, конечной точке **следует** дождаться 2 последующих индикаций потери (в следующих подтверждениях SACK) для тех же номеров, прежде, чем начать реализацию механизма ускоренного повтора (Fast Retransmit).

Для индикации пропусков **следует** применять алгоритм HTNA[1]. Для каждого входящего блока SACK значение счётчика индикации пропуска увеличивается только для пропущенных TSN до HTNA в SACK. Недавно подтверждённым блоком DATA считается блок, который не был ранее указан в SACK. Если конечная точка в состоянии Fast Recovery получает блок SACK, указывающий за пределы Cumulative TSN Ack Point, значения индикации пропуска увеличиваются для всех TSN, пропуск которых указывает этот блок SACK.

При получении третьей индикации отсутствующих TSN отправитель выполняет указанные ниже действия.

1) Пометить указанные блоки DATA для повторной передачи.

2) Если не введён режим Fast Recovery, 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.

[1]Highest TSN Newly Acknowledged - максимальный недавно подтверждённый номер.

3) Если не введён режим Fast Recovery, определить, сколько наиболее ранних (т. е., с меньшими TSN) блоков DATA может быть включено в один пакет с учётом значения PMTU для транспортного адреса получателя, по которому будут передаваться данные (обозначим это количество буквой K). Повторно передать эти K блоков DATA в одном пакете. В режиме Fast Retransmit отправителю **следует** игнорировать значение cwnd и **не следует** задерживать повтор передачи для одного данного пакета.

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

5) Пометить блок(и) DATA, как ускоренно повторённый и не приемлемый для последующего режима Fast Retransmit. Эти TSN маркируются для повторной передачи по причине того, что алгоритм Fast Retransmit не подходит для дейтаграммы, содержащей K других TSN, которые также отмечены неприемлемыми для последующего ускоренного повтора. Однако в результате маркировки для повтора они будут переданы заново, как только позволит cwnd.

6) Перейти в режим Fast Recovery (если он ещё не введён) и указать старший остающийся номер TSN как точку выхода из режима быстрого восстановления. Когда блок SACK подтвердит все TSN, вплоть до указанной точки выхода, режим быстрого восстановления завершается. В режиме Fast Recovery значения ssthresh и cwnd **не следует** менять для каких-либо получателей в результате следующего события Fast Recovery (т. е. **не следует** снижать значение cwnd в результате последующего ускоренного повтора).

Примечание. Если полученный блок SACK также подтверждает новые блоки DATA за пределами Cumulative TSN Ack Point, перед выполнением перечисленных в пп. 1-6 операций сначала **должно** быть изменено значение cwnd в соответствии с правилами, приведёнными в параграфах 7.2.1 и 7.2.2.

7.2.5. Повторная инициализация

В течение срока действия ассоциации SCTP могут происходить события, вдещие к использованию сети в новых изменившихся условиях. При обнаружении этого реализацией SCTP контроль перегрузок **должен** инициализироваться заново.

7.2.5.1. Смена кодов дифференцированного обслуживания

Реализации SCTP **могут** разрешать приложениям настраивать коды DSCP (Differentiated Services Code Point) для передаваемых пакетов. Если смена DSCP может приводить к размещению пакетов в разных очередях, параметры контроля перегрузок для всех затрагиваемых адресов получателя **должны** быть сброшены к начальным значениям.

7.2.5.2. Смена маршрутов

Реализации SCTP **могут** знать об изменении маршрутов, влияющем на пакеты, отправленные получателю. В частности, это включает выбор другого адреса отправителя в передаваемых пакетах. При таких изменениях маршрутов параметры контроля перегрузок для всех затрагиваемых адресов получателя **должны** быть сброшены к начальным значениям.

7.3. Определение PMTU

В [RFC8899], [RFC8201] и [RFC1191] описан алгоритм определения MTU на пути для уровня пакетизации (Packetization Layer Path MTU Discovery), позволяющий конечным точкам оценить PMTU для данного пути через Internet и воздерживаться от передачи по этому пути пакетов, размер которых превышает PMTU, без использования методов зондирования с целью определения изменений PMTU. В [RFC8899] подробно рассмотрен механизм определения PMTU и стратегия установки сквозного значения PMTU, а также детектирования изменений PMTU.

Конечной точке **следует** применять эти методы определения MTU, а также **следует** выполнять такое определение PMTU независимо для каждого адреса получателя.

При определении значения PMTU для протокола SCTP существуют несколько специфических аспектов.

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

2) Отправителю следует отслеживать значение AMDCS, которое будет меньшим из PMDCS среди всех адресов партнёра. При фрагментации сообщений значение AMDCS **следует** использовать для определении размера каждого блока DATA. Такой подход позволит передавать фрагменты по любому из возможных путей без дополнительной фрагментации на уровне IP.

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) с учётом неподтвержденных блоков HEARTBEAT, наблюдаемых на текущем пути передачи данных. Неподтвержденные блоки HEARTBEAT на других путях **не следует** учитывать в счетчике ошибок, поскольку это ведёт к закрытию ассоциации даже если текущий путь передачи данных доступен (но бездействует). Если значение этого счётчика превысит порог, указанный параметром протокола Association.Max.Retrans[1], конечной точке **следует** рассмотреть вопрос о доступности партнёра и **нужно** прекратить передачу ему каких-либо данных (т. е. перевести ассоциацию 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 **должен** сбрасываться всякий раз, когда переданный партнёру блок DATA подтверждается (приём SACK) и его **следует** сбрасывать при получении от удалённого партнёра блока HEARTBEAT ACK. Получатель блока

[1]Максимальное число повторов передачи для ассоциации.

HEARTBEAT ACK **может** не сбрасывать счётчик, если имеет в ассоциации остающиеся данные. Это позволяет обрабатывать возможные различия в доступности на основе блоков DATA и HEARTBEAT.

8.2. Обнаружение отказов на 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 TSN подтверждаются или блок HEARTBEAT, переданный по бездействующему адресу, подтверждается блоком HEARTBEAT ACK, конечной точке **следует** сбрасывать счётчик ошибок для транспортного адреса получателя, по которому был отправлен последний блок DATA (или HEARTBEAT), а также **следует** информировать вышележащий уровень при активизации неактивного адреса. 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 пользователь должен избегать установки для параметра Association.Max.Retrans значения, превышающего любое из значений Path.Max.Retrans для адресов партнёра. 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 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 конечной точке SCTP следует вести мониторинг доступности бездействующих транспортных адресов своего партнёра путём периодической отправки по таким адресам блоков HEARTBEAT. Передача HEARTBEAT **может** начинаться с момента перехода в состояние ESTABLISHED и завершаться после отправки блока SHUTDOWN или SHUTDOWN-ACK. Получатель HEARTBEAT **должен** отвечать на него блоком HEARTBEAT-ACK после перехода в состояние COOKIE-ECHOED (отправитель INIT) или ESTABLISHED (получатель INIT), пока не перейдёт в состояние SHUTDOWN-SENT (отправитель SHUTDOWN) или SHUTDOWN-ACK-SENT (получатель SHUTDOWN).

Транспортный адрес рассматривается как бездействующий (idle), если нет новых блоков, которые можно использовать для обновления периода RTT на пути (обычно, первая передача 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 перечисленные ниже функции.

A) Запрет HEARTBEAT для определённого транспортного адреса в данной ассоциации.

B) Изменение 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 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, конечной точке **следует** уведомить вышележащий уровень об изменении состояния доступности данного адреса. После этого конечной точке **следует** продолжать передачу блоков HEARTBEAT по этому адресу, но без увеличения значения счётчика ошибок.

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

Примечание для разработчиков. Можно использовать 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 TLV 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). Конечной точке **следует** сообщать вышележащему уровню об

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

активизации транспортного адреса в результате получения последнего блока HEARTBEAT ACK. Получателю блока HEARTBEAT ACK **следует** 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.

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

Для пользователей SCTP обеспечивается примитив, позволяющий изменять значение HB.interval и включать/выключать передачу HEARTBEAT для данного адреса. Интервал HB.interval, заданный пользователем SCTP, добавляется к RTO для данного адресата (с учётом экспоненциального отката 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.

При подстройке HB.interval может возникать побочный эффект, который **следует** принимать во внимание. Когда этот интервал возрастает (блоки HEARTBEAT передаются реже), обнаружение потери блоков 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 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 неожиданных пакетов

Пакет SCTP называется неожиданным (OOTB[1]), если он правильно сформирован (т. е. включает корректное значение CRC32c, см. параграф 6.8), но получатель не может идентифицировать ассоциацию, к которой этот пакет относится.

Получатель пакета OOTB выполняет перечисленные ниже операции с соблюдением их порядка.

1) Если в пакете OOTB указан отличный от индивидуального (non-unicast) адрес отправителя или получателя, такой пакет **следует** отбрасывать без уведомления.

2) Если пакет OOTB содержит блок ABORT, получатель **должен** отбросить пакет без уведомления и выполнения каких-либо иных действий.

3) Если пакет содержит блок INIT с Verification Tag = 0, он обрабатывается, как описано в параграфе 5.1. Если по той или иной причине нормальная обработка INIT невозможна и в ответ передаётся блок ABORT, поле Verification Tag в пакете с блоком ABORT **должно** иметь значение Initiate Tag из полученного блока INIT, а бит T в блоке ABORT должен быть сброшен (0) для индикации того, что значение Verification Tag не отражено.

4) Если пакет содержит COOKIE ECHO в первом блоке, он **должен** обрабатываться 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 отражения Verification Tag.

6) Если пакет содержит блок SHUTDOWN COMPLETE, получателю **следует** отбросить пакет без уведомления и выполнения каких-либо иных действий.

7) Если пакет содержит блок ERROR с причиной ошибки Stale cookie или блок COOKIE ACK, пакет SCTP **следует** отбросить без уведомления.

8) В остальных случаях получателю **следует** ответить отправителю 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 отражения Verification Tag. После передачи блока ABORT получатель пакета OOTB **должен** отбросить этот пакет, не выполняя каких-либо иных действий.

8.5. Тег 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, получатель **должен** отбросить пакет без уведомления и **недопустимо** выполнять каких-либо иные операции 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.

[1]Out of the blue - совершенно неожиданный.

- 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.
- Получатель блока ABORT **должен** воспринять пакет, если значение Verification Tag соответствует его собственному тегу при сброшенном бите T или тегу партнёра при установленном бите T. В иных случаях 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 TCB, в поле верификации **должен** использоваться тег адресата, а установка флага T **недопустима**. При отсутствии TCB отправителю **следует** использовать значение Verification Tag из блока SHUTDOWN ACK и он **должен** установить флаг T.
- Получатель SHUTDOWN COMPLETE **должен** воспринять пакет, если поле Verification Tag соответствует его собственному тегу при сброшенном флаге T или тегу партнёра при установленном бите 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 Initiate Tag из принятого блока INIT ACK.
- Получателю COOKIE ECHO **следует** выполнить процедуры, описанные в разделе 5.

E) Правило для пакетов с блоками SHUTDOWN ACK.

A) 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 партнёра прекращают приём новых данных от пользователя и доставляются лишь данные, которые были 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. Если ассоциация прерывается по запросу вышележащего уровня, в блоке ABORT **следует** указать причину User-Initiated Abort (см. параграф 3.3.10.12).

Для конечной точки **недопустима** передача откликов на любой принятый пакет с блоком ABORT (см. 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 ABORT конечная точка **должна** удалить ассоциацию из своих записей и ей **следует** также сообщить о разрыве ассоциации на вышележащий уровень. Если в блоке ABORT указана причина User-Initiated Abort, её **следует** сделать доступной вышележащему уровню.

9.2. Завершение ассоциации (Shutdown)

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

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 партнёру, если в этом возникает необходимость (заполнение пропуска в порядковых номерах).

После того, как все остающиеся данные будут подтверждены партнёром, конечная точка передаёт партнёру блок SHUTDOWN, содержащий в поле Cumulative TSN Ack последний номер TSN, полученный от партнёра. После этого ей **следует** 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 TSN конечная точка **может** группировать блоки SACK и 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, а также **следует** 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 остающихся блоков DATA из его очередей.

При получении блока SHUTDOWN конечная точка:

- переходит в состояние SHUTDOWN-RECEIVED;
- прекращает приём новых данных от своего пользователя SCTP;
- проверяет по полю Cumulative TSN Ack, что все блоки DATA приняты отправителем блока SHUTDOWN.

После перехода конечной точки в состояние SHUTDOWN-RECEIVED она **должна** игнорировать запросы ULP на завершение и **должна** отвечать на блоки SHUTDOWN от партнёра.

При наличии остающихся блоков DATA получатель SHUTDOWN **должен** продолжать обычные процедуры передачи, описанные в разделе 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. Если блок SHUTDOWN сам по себе не может подтвердить все принятые блоки DATA (т. е., имеются номера TSN, которые могут быть подтверждены, но больше кумулятивного значения TSN и в последовательности TSN возникает пропуск) или получены дубликаты TSN, **должен** передаваться также блок SACK.

Отправитель блока SHUTDOWN **может** также запустить таймер T5-shutdown-guard для ограничения общей продолжительности процедуры завершения ассоциации. По завершению отсчёта этого таймера отправителю **следует** разорвать ассоциацию передачей блока ABORT. При использовании таймера T5-shutdown-guard для него **следует** устанавливать **рекомендуемое** значение 5 * RTO.Max.

Если у получателя блока SHUTDOWN больше не остаётся блоков DATA, он **должен** передать блок SHUTDOWN ACK и запустить таймер T2-shutdown, перейдя в состояние SHUTDOWN-ACK-SENT. По завершении отсчёта таймера конечная точка **должна** повторить передачу 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, а также **следует** 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 проверяет, что она находится в состоянии SHUTDOWN-ACK-SENT и ей **следует** отбросить 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 сведения об ассоциацией (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 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 пакета с блоком INIT с теми же адресами IP отправителя и получателя, которые назначены конечной точке, 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 с 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 T1-init будет приводить к повтору передачи блока INIT или COOKIE и 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. Обработка ICMP

При получении конечной точкой SCTP сообщения ICMP **должны** выполняться указанные ниже процедуры для подобающего использования информации, предоставленной уровнем 3.

ICMP1)

Реализация **может** игнорировать все сообщения ICMPv4 с полем типа, отличным от Destination Unreachable.

ICMP2)

Реализация **может** игнорировать все сообщения ICMPv6 с полем типа, отличным от Destination Unreachable, Parameter Problem, Packet Too Big.

ICMP3)

Реализации **следует** игнорировать все сообщения ICMP с кодом Port Unreachable.

ICMP4)

Реализация **может** игнорировать все сообщения ICMPv6 с полем типа Parameter Problem, если код отличается от Unrecognized Next Header Type Encountered.

ICMP5)

Реализация **должна** использовать содержимое сообщения ICMP (v4 или v6) для нахождения ассоциации, передавшей сообщение, на которое получен отклик ICMP. Если ассоциация не найдена реализации следует игнорировать сообщение ICMP.

ICMP6)

Реализация **должна** проверить совпадение Verification Tag в сообщении ICMP значению Verification Tag партнёра. Если Verification Tag отличается от 0 и не совпадает, сообщение ICMP отбрасывается. Если тег имеет значени 0 и сообщение ICMP содержит достаточно байтов для проверки того, что блок имеет тип INIT, а значение Initiate Tag совпадает с тегом партнёра, выполняется ICMP7. Если сообщение ICMP слишком короткое, тип блока или Initiate Tag не совпадает, пакет просто отбрасывается.

ICMP7)

Если сообщение является ICMPv6 типа Packet Too Big или ICMPv4 типа Destination Unreachable с кодом Fragmentation Needed, реализации **следует** обработать эти сведения как задано для определения PMTU.

ICMP8)

Если сообщение ICMP имеет код Unrecognized Next Header Type Encountered или Protocol Unreachable, реализация **должна** трактовать его как прерывание с установленным битом T, когда в нем не содержится блока INIT. Если сообщение включает блок INIT и ассоциация находится в состоянии COOKIE-WAIT, сообщение ICMP обрабатывается подобно блоку ABORT.

ICMP9)

Если сообщение ICMP имеет тип Destination Unreachable, реализация **может** перевести получателя в число недоступных или увеличить счётчик ошибок для пути. SCTP **может** передавать вышележащему уровню сведения о приёме сообщений ICMP при указании смены состояния сети.

Эти процедуры отличаются от [RFC1122], требований по обработке сообщений о недоступности порта и требований, в соответствии с которыми реализация **должна** разрывать ассоциации в ответ на сообщение о недоступности протокола. Сообщения о недоступности порта не обрабатываются, поскольку реализации передают блок ABORT вместо port-unreachable. Более строгая обработка сообщений о недоступности протокола обусловлена соображениями безопасности хостов, не поддерживающих SCTP.

11. Интерфейс с вышележащим уровнем

Протоколы вышележащих уровней (ULP) запрашивают сервис путём передачи примитивов протоколу SCTP и получают 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. Реализации SCTP могут использовать разные интерфейсы с ULP, однако все реализации должны обеспечивать некий минимальный сервис, гарантирующий поддержку одной иерархии протоколов всеми реализациями SCTP.

Отметим, что этот раздел предназначен лишь для информации (не нормативный).

[RFC6458] и раздел 7 (Socket API Considerations) в [RFC7053] определяют расширение API сокетов для SCTP, как описано в этом документе.

11.1. ULP -> SCTP

В последующих параграфах приведены функциональные характеристики интерфейса ULP-SCTP. Используемая 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.

11.1.1. Initialize - инициализация

```
INITIALIZE ([local port],[local eligible address list]) -> local SCTP instance name
```

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 другими конечными точками без повторного использования данного примитива.

SCTP будет возвращать ULP имя локального экземпляра SCTP.

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 данной точке.

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.

11.1.2. Associate - создать ассоциацию

```
ASSOCIATE(local SCTP instance name, initial destination transport addr list, outbound stream count)
```

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

Этот примитив позволяет инициировать создание ассоциации с указанным партнёром.

Конечная точка партнёра указывается одним или несколькими из её транспортных адресов (см. параграф 1.3). Если локальный экземпляр SCTP ещё не инициализирован, вызов ASSOCIATE считается ошибкой.

При успешном создании ассоциации возвращается идентификатор 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 число исходящих потоков локальной точки. Один из транспортных адресов удалённого партнёра выбирается локальной точкой в качестве первичного (используемого по умолчанию) транспортного адреса для передачи пакетов SCTP 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.
- initial destination transport addr list - непустой список транспортных адресов партнёра, с которым создаётся ассоциация.
- outbound stream count - число исходящих потоков, которые ULP будет открывать для обмена данными с партнёром.

11.1.3. Shutdown - завершение ассоциации

```
SHUTDOWN(association id) -> result
```

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.

Обязательный атрибут

association id - локальный идентификатор ассоциации SCTP.

11.1.4. Abort - разрыв ассоциации

```
ABORT(association id [, cause code]) -> result
```

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.

Обязательный атрибут

association id - локальный идентификатор ассоциации SCTP.

Необязательный атрибут

Upper Layer Abort Reason - причина разрыва ассоциации, передаваемая партнёру.

11.1.5. Send - передача данных

```
SEND(association id, buffer address, byte count [,context] [,stream id] [,life time]
    [,destination transport address] [,unorder flag] [,no-bundle flag]
    [,payload protocol-id] [,sack-immediately flag]) -> result
```

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 SEND FAILURE 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 действия 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 примитива SEND в течение заданного этим параметром времени. Однако пользовательские данные будут переданы, если протокол SCTP начал попытки отправки блока данных до истечения заданного времени.

  Примечание для разработчиков. Для более эффективной поддержки опции времени действия данных отправитель может присваивать передаваемому блоку DATA 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 действия, заданное для блока DATA.
- destination transport address - задаёт один из транспортных адресов получателя, по которому пакет должен передаваться. По возможности протокол SCTP использетn заданный адрес вместо адреса первичного пути.
- unorder flag - этот флаг указывает что пользовательские данные доставляются партнёру без соблюдения порядка (т. е., устанавливается U = 1 во всех блоках DATA, содержащих это сообщение).
- no-bundle flag - указывает протоколу SCTP, что эти данные не должны группироваться с другими 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. Отметим, что вышележащий уровень отвечает за преобразование порядка байтов этого поля из хостового в сетевой; SCTP передаёт поле как 4 необрабатываемых (opaque) байта.

- sack-immediately flag - установить флаг I в последнем блоке DATA передаваемого пользовательского сообщения.

11.1.6. Set Primary - выбор основного пути

```
SETPRIMARY(association id, destination transport address, [source transport address]) -> result
```

Задаёт 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 destination transport address list, возвращённом ранее примитивом ASSOCIATE или уведомлением COMMUNICATION UP, возвращается 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.

Необязательный атрибут

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

11.1.7. Receive - приём данных

```
RECEIVE(association id, buffer address, buffer size [,stream id])
   -> byte count [,transport address] [,stream id] [,stream sequence number]
      [,partial flag] [,payload protocol-id]
```

Этот примитив считывает 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 (Stream Sequence Number).

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 примитив возвращает лишь часть данных сообщения. Установка этого флага сопровождается возвратом идентификатора потока и порядкового номера в потоке. Нулевое значение флага показывает, что для этого порядкового номера в потоке больше нет данных.
- payload protocol-id - 32-битовое целое число без знака, полученное от партнёра и указывающее тип данных в принятом сообщении. Отметим, что вышележащий уровень отвечает за преобразование порядка байтов этого поля из хостового в сетевой; SCTP передаёт поле как 4 необрабатываемых (opaque) байта.

11.1.8. Status - статус

```
STATUS(association id) -> данные о состоянии
```

Этот примитив возвращает блок данных, содержащий:

- состояние соединения для ассоциации;
- список адресов транспортного уровня для получателя;
- состояния доступности транспортных адресов получателя;
- текущий размер окна приёма;
- текущий размер окна перегрузки;
- 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.

Обязательный атрибут

association id - локальный идентификатор ассоциации SCTP.

11.1.9. Change Heartbeat - смена режима Heartbeat

```
CHANGEHEARTBEAT(association id, destination transport address, new state [,interval]) -> result
```

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).

Необязательный атрибут

interval - определяет частоту передачи heartbeat, если эта функция включена для транспортного адреса партнёра. Значение параметра добавляется к RTO транспортного адреса. Параметр действует для всех транспортных адресов получателя.

11.1.10. Request Heartbeat - запрос на выполнение Heartbeat

```
REQUESTHEARTBEAT(association id, destination transport address) -> result
```

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

Обязательный атрибут

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

11.1.11. Get SRTT Report - запрос значения SRTT

```
GETSRTTREPORT(association id, destination transport address) -> srtt result
```

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, для которого определяется значение SRTT.

11.1.12. Set Failure Threshold - задать порог детектирования отказа

```
SETFAILURETHRESHOLD(association id, destination transport address, failure threshold) -> result
```

Этот примитив позволяет локальному модулю SCTP задать значение порога детектирования отказа Path.Max.Retrans для заданного транспортного адреса. Отметим, что это можно сделать с помощью примитива SETPROTOCOLPARAMETERS (параграф 11.1.13).

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 транспортного адреса.

11.1.13. Set Protocol Parameters - установить параметры протокола

```
SETPROTOCOLPARAMETERS(association id, [,destination transport address,]
                      protocol parameter list) -> result
```

Этот примитив позволяет локальному модулю SCTP установить параметры протокола.

Обязательные атрибуты

- association id - локальный идентификатор ассоциации SCTP;
- protocol parameter list - список имён и значений протокольных параметров (например, Association.Max.Retrans, см. раздел 16 или таких параметров как DSCP), которые пользователь SCTP желает установить.

Необязательный атрибут

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

11.1.14. Receive unsent message - получить неотправленное сообщение

```
RECEIVE_UNSENT(data retrieval id, buffer address, buffer size [,stream id]
               [, stream sequence number] [,partial flag] [,payload protocol-id])
```

Обязательные атрибуты

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

Необязательные атрибуты

- stream id - идентификатор потока, в который были переданы данные.
- Stream Sequence Number - порядковый номер в потоке, связанный с сообщением.
- partial flag - указывает на частичную доставку сообщения. При установке этого флага также возвращаются идентификатор потока и порядковый номер в потоке. Нулевое значение флага показывает, что для данного порядкового номера в потоке больше не будет получено данных.
- payload protocol-id - 32-битовое целое число без знака, которое было задано для передачи партнёру с целью идентификации протокола полученных данных.

11.1.15. Receive Unacknowledged Message - приём неподтвержденного сообщения

```
RECEIVE_UNACKED(data retrieval id, buffer address, buffer size, [,stream id]
                [, stream sequence number] [,partial flag] [,payload protocol-id])
```

Обязательные атрибуты

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

Необязательные атрибуты

- stream id - идентификатор потока, в который были переданы данные.
- Stream Sequence Number - порядковый номер в потоке, связанный с сообщением.
- partial flag - указывает на частичную доставку сообщения. При установке этого флага также возвращаются идентификатор потока и порядковый номер в потоке. Нулевое значение флага показывает, что для данного порядкового номера в потоке больше не будет получено данных.

- payload protocol-id - это 32-битовое целое число без знака, которое было передано для идентификации протокола полученных данных.

11.1.16. Destroy SCTP instance - уничтожить экземпляр SCTP

`DESTROY(local SCTP instance name)`

Обязательный атрибут

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

11.2. SCTP -> ULP

Предполагается, что операционная система или прикладная среда обеспечивает асинхронную передачу 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.

11.2.1. Уведомление DATA ARRIVE

SCTP передаёт такое уведомление ULP, 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.

11.2.2. Уведомление SEND FAILURE

Если сообщение не может быть доставлено, протокол SCTP передаёт это уведомление ULP. Уведомление может включать следующие необязательные параметры:

- association id - локальный идентификатор ассоциации SCTP;
- data retrieval id - идентификатор, используемый для считывания неотправленных или неподтвержденных данных;
- cause code - указывает причину отказа (например, слишком большой размер сообщения, завершение срока действия сообщения и т. п.);
- mode - указывает, была ли какая-либо часть сообщения отправлена, но не подтверждена полностью;
- context - дополнительная информация, связанная с сообщением (см. параграф 11.1.5).

11.2.3. Уведомление NETWORK STATUS CHANGE

Когда транспортный адрес получателя помечается, как неактивный (например, при обнаружении отказа) или, наоборот, становится активным (например, при обнаружении восстановления), SCTP передаёт такое уведомление ULP. В уведомлении содержится следующая информация:

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

11.2.4. 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 содержится 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 (может отличаться от outbound stream count).

11.2.5. Уведомление COMMUNICATION LOST

Это уведомление передаётся протоколом SCTP в случае полной утраты связи с удалённой точкой (например, через Heartbeat) или обнаружения вызова удалённой точкой операции прерывания ассоциации (abort). В уведомлении содержится 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:

- last-acked - последний номер TSN, подтверждённый партнёром;
- last-sent - последний номер TSN, переданный партнёру;
- Upper Layer Abort Reason - причина прерывания указывается в случае разрыва по инициативе пользователя.

11.2.6. Уведомление COMMUNICATION ERROR

Когда SCTP получает блок ERROR от 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 из блока ERROR.

11.2.7. Уведомление RESTART

При обнаружении рестарта на стороне партнёра SCTP может использовать этот тип для передачи уведомления ULP. Уведомление может включать параметр association id - локальный идентификатор ассоциации SCTP.

11.2.8. Уведомление SHUTDOWN COMPLETE

Это уведомление SCTP передаёт на вышележащий уровень при завершении процедуры SHUTDOWN (параграф 9.2). Уведомление может включать параметр association id - локальный идентификатор ассоциации SCTP.

12. Вопросы безопасности

12.1. Цели защиты

Для протоколов 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.

12.2. 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, применяющих SCTP, следует использовать [RFC2196] в качестве руководства по обеспечению безопасности своего сайта.

12.2.1. Учёт 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.

12.2.2. Защита от повреждения данных в сети

Если риск возникновения незаметных ошибок в дейтаграммах, доставляемых с помощью транспорта нижележащих уровней, слишком велик, нужны дополнительные меры по обеспечению целостности данных. Если дополнительная защита обеспечивается на прикладном уровне, заголовок SCTP остаётся уязвимым для преднамеренных атак с целью повреждения данных. Хотя существующих механизмов обнаружения 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.

Можно использовать расширение SCTP-AUTH [RFC4895], если среда с угрозами требует сильной защиты целостности, не требуя защиты конфиденциальности.

12.2.3. Защита 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. Для этого **можно** воспользоваться [RFC6083]. Как вариант, пользовательское приложение **может** применять специфические для реализации API для запроса сервиса IP ESP [RFC4303], 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.

Независимо от способа защиты конфиденциальности **следует** использовать механизмы IKEv2 [RFC7296] для управления ключами ESP.

Операторы могут обратиться к [RFC4301] для получения дополнительной информации по средствам обеспечения безопасности непосредственно поверх уровня IP.

12.2.4. 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.

12.2.4.1. Лавинная атака (Flooding)

Целью лавинной атаки является выведение сервиса из строя и некорректное поведение системы из-за истощения ресурсов, интерференции с легитимными транзакциями или использования связанных с буферами программных ошибок. Лавинная атака может быть направлена на узел SCTP, каналы доступа или Internet. В последних случаях лавинная атака будет проявляться как прекращение работы сетевых служб при наличии межсетевых экранов.

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, переданных по адресам, не являющимся индивидуальными (non-unicast).

Сетевое оборудование должно обеспечивать возможность генерации сигналов и записи в журнал сведений о подозрительном трафике. В журнальный файл следует включать информацию, которая позволит идентифицировать входящий канал и адреса источников, что может помочь администратору сети или системы SCTP принять меры по защите. Предполагается, что оператор будет действовать в соответствии с сигналами тревоги при возникновении чёткой схемы злоупотребления.

Протокол SCTP устойчив к лавинным атакам, в частности, благодаря использованию четырёхэтапного 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.

Механизм ESP может также оказаться полезным для снижения риска при некоторых типах атак на службы.

Поддержка параметра Host Name исключена из протокола. Конечная точка, получившая блок INIT или INIT ACK с параметром Host Name, **должна** передать в ответ блок ABORT и **может** включить в него код причины Unresolvable Address.

12.2.4.2. 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 ролевым узлом SCTP. Замаскированный атакующий может попытаться организовать ассоциацию и прикинуться ролевым узлом, чтобы впоследствии настоящий ролевой узел не мог подключиться к сервису.
- Посредством преднамеренного раскрытия подмены для провоцирования блокировки ролевого узла целевым узлом SCTP.
- Путём взаимодействия с действующей ассоциацией посредством вставки в поток добавочного содержимого (например, блока SHUTDOWN).

SCTP снижает риск слепого маскирования с подменой адресов (IP spoofing) за счёт использования четырёхэтапного согласования при старте ассоциации. Поскольку начальный обмен не занимает памяти, атаки с маскированием вслепую не вызывают никакой блокировки. 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 ассоциации за счёт использования тегов верификации (Verification Tag).

Запись (log) полученных запросов INIT и аномалий вроде неожиданных блоков INIT ACK может быть полезна 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. Запись не имеет смысла без создания рабочих процедур повседневного просмотра и анализа журнальных файлов.

12.2.4.3. 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 обнаружения передачи больших объёмов информации или сообщений по-ор в данной ассоциации, а также другие механизмы фиксации и разрыва ассоциаций в соответствии с принятой локальной политикой.

12.3. Взаимодействие SCTP с межсетевыми экранами

Для некоторых межсетевых экранов будет полезна возможность проверки первого фрагмента фрагментированного пакета SCTP и однозначного определения его соответствия блоку INIT (см. дополнительную информацию в [RFC1858]). Поэтому ещё раз подчёркиваются требования, приведённые в параграфе 3.1, - (1) блок INIT **недопустимо** группировать с каким-либо другим блоком в одном пакете, (2) пакет с блоком INIT **должен** иметь Verification Tag = 0.

Получатель INIT **должен** отбрасывать этот блок и все последующие блоки, если INIT сгруппирован с другими блоками или имеет отличный от нуля тег верификации.

12.4. Защита хостов, не поддерживающих SCTP

Для обеспечения не поддерживающим SCTP хостам такого же уровня защиты от атак, как для хостов SCTP, все реализации протокола SCTP **должны** поддерживать обработку ICMP, описанную в разделе 10.

Когда стек SCTP получает пакет, содержащий несколько блоков управления или DATA, и обработка такого пакета ведёт к передаче в ответ нескольких блоков, отправителю этих блоков **недопустимо** передавать более одного пакета с блоками, отличными от DATA. Это защищает сеть от передачи «пика» пакетов в ответ на единственный пакет. Если поддерживается группировка, несколько блоков отклика **можно** собрать в один пакет отклика. Если группировка блоков не поддерживается, отправителю **недопустимо** передавать более одного блока-отклика, а остальные он **должен** отбросить. Отметим, что это правило не применимо к блокам SACK, поскольку они сами по себе являются откликами на блоки DATA, и SACK не требует передачи в ответ блоков DATA.

Реализация SCTP **должна** прерывать ассоциацию при получении блока SACK с подтверждением номера TSN, который не был передан.

Реализация SCTP, получающая блок INIT, для отклика на который требуется большой пакет по причине включения в него множества параметров Unrecognized Parameter, **может** (по своему усмотрению) опустить часть или все такие параметры для снижения размера INIT ACK. С учётом размера параметра State Cookie и множества адресов, которые получатель INIT может указывать своему партнёру, размер блока INIT ACK может превышать размер исходного блока INIT. Реализациям SCTP **следует** пытаться максимально сократить размер блоков INIT ACK для снижения возможности организации «атак с усилением» (byte amplification attack).

13. Вопросы управления сетью

Модуль MIB для протокола SCTP, определённый в [RFC3873], применим для описанной здесь версии протокола.

14. Рекомендуемые параметры TCB

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

14.1. Параметры, требуемые для экземпляра SCTP

Associations

Список существующих ассоциаций и отображений на потребители 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 дескрипторы, указатели на именованные каналы или таблицы указателей) в зависимости от конкретной реализации SCTP.

Secret Key

Секретный 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 [RFC4086].

Address List

Список адресов IP, с которыми связан данный экземпляр. Эти сведения передаются партнёру в блоках INIT и INIT ACK.

SCTP Port

Локальный номер порта, с которым связана конечная точка SCTP.

14.2. 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

COOKIE-WAIT, COOKIE-ECHOED, ESTABLISHED, SHUTDOWN-PENDING, SHUTDOWN-SENT, SHUTDOWN-RECEIVED, SHUTDOWN-ACK-SENT.

Примечание. Состояние CLOSED не указано, поскольку для ассоциации в этом состоянии 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 основного транспортного адреса партнёра. Это значение может также указывать адрес источника пакетов от этой точки.

Overall Error Count

Общий счётчик ошибок для всей ассоциации.

Overall Error Threshold

Значение числа 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 выделяется 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 в блоке INIT или INIT ACK, путём вычитания 1 из полученного значения.

Mapping Array
Массив битов или байтов, показывающий, какие переупорядоченных 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 нули. Эта структура может быть кольцевым буферои или битовым массивом.

Ack State
Этот флаг показывает, будет ли передано подтверждение SACK при получении следующего пакета. Начальное значение равно 0. При получении пакета значение инкрементируется. При достижении значения 2 или более в ответ на получение пакета передаётся SACK и значение 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.

Local Transport Address List
Список локальных адресов IP, связанных с данной ассоциацией.

Association Maximum DATA Chunk Size
Наименьшее значение Path Maximum DATA Chunk Size среди всех адресов партнёра.

14.3. Данные для транспортного адреса

Для каждого транспортного адреса партнёра из списка, полученного в блоке INIT или INIT ACK, поддерживается группа параметров, включающая перечисленные ниже.

Error count
Текущее значение счётчика ошибок для данного получателя.

Error Threshold
Текущее значение порога для числа ошибок, связанных с данным получателем. При достижении порога получатель помечается как недоступный.

cwnd
Текущий размер окна перегрузки.

ssthresh
Текущее значение порога Slow Start (ssthresh).

RTO
Текущее значение тайм-аута повтора передачи.

SRTT
Текущее значение сглаженного времени кругового обхода.

RTTVAR
Вариации текущего значения RTT.

partial bytes acked
Метод слежения, используемый для увеличения размера cwnd в режиме предотвращения перегрузки (congestion avoidance, см. параграф 7.2.2).

state
Текущее состояние адресата (DOWN, UP, ALLOW-HB, NO-HEARTBEAT и т. п.).

PMTU
Текущее известное значение PMTU.

PMDCS
Текущее известное значение PMDCS.

Per Destination Timer
Таймер, используемый для каждого 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
Время передачи адресату последнего пакета. Это значение может использоваться для определения необходимости передачи HEARTBEAT.

14.4. Требуемые параметры общего назначения

Out Queue
Очередь исходящих блоков DATA.

In Queue
Очередь входящих блоков DATA.

15. Взаимодействие с IANA

SCTP определяет пять реестров, поддерживаемых агентством IANA:

- типы блоков;
- флаги блоков;

- типы параметров;
- коды причины ошибки в блоках ERROR;
- идентификаторы протоколов данных.

Агентство IANA внесло в упомянутые реестры перечисленные ниже изменения.

- В реестре Chunk Types ссылки на [RFC4960] и [RFC6096] заменены ссылками на этот документ. В разделе Notes ссылка на параграф 3.2 в [RFC6096] заменена ссылкой на параграф 15.2 этого документа. Кроме того, ссылки на [RFC4960] были заменены ссылками на данный документ для типов:
  - Payload Data (DATA);
  - Initiation (INIT);
  - Initiation Acknowledgement (INIT ACK);
  - Selective Acknowledgement (SACK);
  - Heartbeat Request (HEARTBEAT);
  - Heartbeat Acknowledgement (HEARTBEAT ACK);
  - Abort (ABORT);
  - Shutdown (SHUTDOWN);
  - Shutdown Acknowledgement (SHUTDOWN ACK);
  - Operation Error (ERROR);
  - State Cookie (COOKIE ECHO);
  - Cookie Acknowledgement (COOKIE ACK);
  - Reserved for Explicit Congestion Notification Echo (ECNE);
  - Reserved for Congestion Window Reduced (CWR);
  - Shutdown Complete (SHUTDOWN COMPLETE);
  - Reserved for IETF-defined Chunk Extensions.
- В реестре Chunk Parameter Types ссылки на [RFC4960] заменены ссылками на этот документ. Агентство IANA сменило имя Unrecognized Parameters для типа параметров блока на Unrecognized Parameter в реестре Chunk Parameter Types. Кроме того, ссылки на [RFC4960] были заменены ссылками на данный документ для типов:
  - Heartbeat Info;
  - IPv4 Address;
  - IPv6 Address;
  - State Cookie;
  - Unrecognized Parameter;
  - Cookie Preservative;
  - Host Name Address;
  - Supported Address Types

  Агентство IANA добавило ссылку на данный документ для типа параметров блока:
  - Reserved for ECN Capable (0x8000).

  Кроме того, агентство IANA добавило значение 65535 к зарезервированным для заданных IETF расширений.
- В реестре Chunk Flags ссылки на [RFC6096] заменены ссылками на данный документ. Кроме того, ссылки на [RFC4960] были заменены ссылками на данный документ для следующих флагов блока DATA:
  - E;
  - B;
  - U.

  Агентство IANA заменило ссылку на [RFC7053] ссылкой на этот документ для флага блока DATA:
  - I.

  Агентство IANA заменило ссылку на [RFC4960] ссылкой на этот документ для флага блока ABORT:
  - T.

  Агентство IANA заменило ссылку на [RFC4960] ссылкой на этот документ для флага блока SHUTDOWN COMPLETE:
  - T.

- В реестре Error Cause Codes ссылки на [RFC4960] заменены ссылками на этот документ. Имя причины ошибки User Initiated Abort заменено на User-Initiated Abort. A Stale Cookie Error - на Stale Cookie. Кроме того, ссылки на [RFC4960] были заменены ссылками на данный документ для следующих кодов причин:
  - Invalid Stream Identifier;
  - Missing Mandatory Parameter;
  - Stale Cookie;
  - Out of Resource;
  - Unresolvable Address;
  - Unrecognized Chunk Type;
  - Invalid Mandatory Parameter;
  - Unrecognized Parameters;
  - No User Data;
  - Cookie Received While Shutting Down;
  - Restart of an Association with New Addresses.

  Агентство IANA заменило ссылку на [RFC4460] ссылкой на этот документ для следующих кодов причин:
  - User-Initiated Abort;
  - Protocol Violation.
- В реестре SCTP Payload Protocol Identifiers ссылки на [RFC4960] заменены ссылками на этот документ. Агентство IANA заменило ссылку на [RFC4460] ссылкой на этот документ для следующих идентификаторов протоколов в данных SCTP:
  - Reserved by SCTP

Протокол SCTP требует, чтобы реестр IANA Port Numbers был открыт для регистрации портов SCTP, как описано в параграфе 15.6. Назначенный IESG эксперт поддерживает IANA при оценке запросов на выделение порта SCTP.

В реестре Service Name and Transport Protocol Port Number Registry ссылки на [RFC4960] заменены ссылками на этот документ для портов SCTP:

- 9 (discard);
- 20 (ftp-data);
- 21 (ftp);
- 22 (ssh);
- 80 (http);
- 179 (bgp);
- 443 (https).

В реестре Hypertext Transfer Protocol (HTTP) Digest Algorithm Values ссылка на приложение B к [RFC4960] заменена ссылкой на приложение A к данному документу.

В реестре ONC RPC Netids (Standards Action) каждая ссылка на [RFC4960] заменена ссылкой на данный документ для:

- sctp;
- sctp6.

В реестре IPFIX Information Elements каждая ссылка на [RFC4960] заменена ссылкой на данный документ для:

- sourceTransportPort;
- destinationTransportPort;
- collectorTransportPort;
- exporterTransportPort;
- postNAPTSourceTransportPort;
- postNAPTDestinationTransportPort.

15.1. Расширения для блоков, определяемые IETF

Выделение новых кодов для блоков SCTP выполняется по процедуре IETF Review, определённой в [RFC8126]. Документация для нового блока **должна** включать:

a) полное и сокращённое имя нового типа блоков;

b) подробное описание структуры блока, которое **должно** соответствовать базовой структуре, определённой в параграфе 3.2;

c) определение и подробное описание каждого поля блока, включая флаги, если они используются; определённые флаги блока будут служить начальными значениями таблицы флагов для нового типа блока;

d) подробное описание процедур использования нового типа блоков в работе протокола.

Последний номер типа (255) зарезервирован для будущих расширений.

Для каждого нового типа блоков IANA создаёт таблицу регистрации флагов нового блока. Процедура регистрации флагов конкретного блока описана в параграфе 15.2.

15.2. Регистрация определённых IETF флагов блока

Выделение новых флагов блоков выполняется по процедуре RFC Required [RFC8126]. Документация для флагов блока **должна** включать:

a) имя нового флага блока;

b) подробное процедурное описание использования нового в работе протокола; **должно** учитываться, что не поддерживающие флаг реализации будут устанавливать значение 0 при отправке и игнорировать флаг при получении.

IANA выбирает значение для нового флага. Это **должно** быть одно из значений 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, которое **должно** быть уникальным среди значений флагов для конкретного типа блока.

15.3. Определяемые IETF расширения для типа параметров блоков

Выделение новых кодов для типа параметров блоков SCTP (chunk parameter type code) выполняется по процедуре IETF Review, определённой в [RFC8126]. Документация для параметров блока **должна** включать:

a) имя типа параметра;

b) подробное описание структуры поля параметра, которая **должна** соответствовать базовому форматы TLV, определённому в параграфе 3.2.1;

c) подробное описание каждого элемента значения параметра;

d) подробное описание предполагаемого использования этого типа параметра и возможности присутствия в одном блоке множества экземпляров данного параметра;

e) каждый тип параметров **должен** быть уникальным для всех блоков.

15.4. Определяемые IETF дополнительные коды причин ошибок

Дополнительные значения кодов ошибок выделяются по процедуре Specification Required, определённой в [RFC8126]. Представляемая документация **должна** включать:

a) имя ошибки;

b) подробное описание условий, при которых конечной точке SCTP следует генерировать блок ERROR (или ABORT) с этим кодом ошибки;

c) ожидаемые действия конечной точки SCTP при получении блока ERROR (или ABORT) с этим кодом ошибки;

d) подробное описание структуры и содержимого полей данных, сопровождающих этот код.

Первое слово кода причина (32 бита) **должно** соответствовать формату, определённому в параграфе 3.3.10:

- первые 2 байта содержат значение кода причины ошибки;
- последние два байта указывают размер причины ошибки.

15.5. Идентификаторы протоколов (Payload)

Идентификаторы протокола выделяются по процедуре First Come First Served [RFC8126].

За исключением значения 0, зарезервированного в SCTP для индикации неуказанного протокола в блоке DATA, реализация SCTP не отвечает за стандартизацию или проверку идентификаторов протоколов. SCTP просто получает идентификатор от вышележащего уровня и передаёт его с соответствующими элементами данных.

Вышележащему уровню (пользователю SCTP) **следует** стандартизовать SHOULD идентификаторы конкретных протоколов через IANA, если такая стандартизация желательна. Использование каких-либо конкретных идентификаторов протоколов в элементах данных выходит за рамки спецификации.

15.6. Реестр номеров портов

Службы SCTP могут использовать контактные номера портов для предоставления услуг незнакомым абонентам, как это делается в TCP и UDP. Назначенные IESG эксперты поддерживают запрос в IANA на выделение портов SCTP в соответствии с процедурами, описанными в [RFC8126]. Детали процесса описаны в [RFC6335].

16. Предлагаемые параметры протокола SCTP

Рекомендуется использовать следующие значения параметров протокола:

RTO.Initial	- 1 секунда
RTO.Min	- 1 секунда
RTO.Max	- 60 секунд
Max.Burst	- 4
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
HB.Max.Burst	- 1
SACK.Delay	- 200 миллисекунд

Примечание для разработчиков. Реализация SCTP может разрешать ULP изменение некоторых параметров протокола (см. раздел 11).

Для RTO.Min следует использовать приведённое выше значение.

17. Литература

17.1. Нормативные документы

[ITU.V42.1994] International Telecommunications Union, "Error-correcting Procedures for DCEs Using Asynchronous-to-Synchronous Conversion", ITU-T Recommendation V.42, 1994.

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

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

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

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, DOI 10.17487/RFC1982, August 1996, <https://www.rfc-editor.org/info/rfc1982>.

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

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <https://www.rfc-editor.org/info/rfc4303>.

[RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", RFC 4895, DOI 10.17487/RFC4895, August 2007, <https://www.rfc-editor.org/info/rfc4895>.

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

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC6083] Tuexen, M., Seggelmann, R., and E. Rescorla, "Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)", RFC 6083, DOI 10.17487/RFC6083, January 2011, <https://www.rfc-editor.org/info/rfc6083>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

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

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

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

[RFC8899] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <https://www.rfc-editor.org/info/rfc8899>.

17.2. Дополнительная литература

[FALL96] Fall, K. and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", SIGCOM 99, V. 26, N. 3, pp 5-21, July 1996.

[SAVAGE99] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP Congestion Control with a Misbehaving Receiver", ACM Computer Communications Review 29(5), October 1999.

[ALLMAN99] Allman, M. and V. Paxson, "On Estimating End-to-End Network Path Properties", SIGCOM 99, October 1999.

[WILLIAMS93] Williams, R., "A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS", SIGCOM 99, August 1993, <https://archive.org/stream/PainlessCRC/crc_v3.txt>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

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

[RFC1858] Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering", RFC 1858, DOI 10.17487/RFC1858, October 1995, <https://www.rfc-editor.org/info/rfc1858>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[RFC2196] Fraser, B., "Site Security Handbook", FYI 8, RFC 2196, DOI 10.17487/RFC2196, September 1997, <https://www.rfc-editor.org/info/rfc2196>.

[RFC2522] Karn, P. and W. Simpson, "Photuris: Session-Key Management Protocol", RFC 2522, DOI 10.17487/RFC2522, March 1999, <https://www.rfc-editor.org/info/rfc2522>.

[RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, DOI 10.17487/RFC2960, October 2000, <https://www.rfc-editor.org/info/rfc2960>.

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

[RFC3873] Pastor, J. and M. Belinchon, "Stream Control Transmission Protocol (SCTP) Management Information Base (MIB)", RFC 3873, DOI 10.17487/RFC3873, September 2004, <https://www.rfc-editor.org/info/rfc3873>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[RFC4460] Stewart, R., Arias-Rodriguez, I., Poon, K., Caro, A., and M. Tuexen, "Stream Control Transmission Protocol (SCTP) Specification Errata and Issues", RFC 4460, DOI 10.17487/RFC4460, April 2006, <https://www.rfc-editor.org/info/rfc4460>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[RFC6096] Tuexen, M. and R. Stewart, "Stream Control Transmission Protocol (SCTP) Chunk Flags Registration", RFC 6096, DOI 10.17487/RFC6096, January 2011, <https://www.rfc-editor.org/info/rfc6096>.

[RFC6458] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)", RFC 6458, DOI 10.17487/RFC6458, December 2011, <https://www.rfc-editor.org/info/rfc6458>.

[RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, DOI 10.17487/RFC6951, May 2013, <https://www.rfc-editor.org/info/rfc6951>.

[RFC7053] Tuexen, M., Ruengeler, I., and R. Stewart, "SACK-IMMEDIATELY Extension for the Stream Control Transmission Protocol", RFC 7053, DOI 10.17487/RFC7053, November 2013, <https://www.rfc-editor.org/info/rfc7053>.

[RFC8260] Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol", RFC 8260, DOI 10.17487/RFC8260, November 2017, <https://www.rfc-editor.org/info/rfc8260>.

[RFC8261] Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets", RFC 8261, DOI 10.17487/RFC8261, November 2017, <https://www.rfc-editor.org/info/rfc8261>.

[RFC8540] Stewart, R., Tuexen, M., and M. Proshin, "Stream Control Transmission Protocol: Errata and Issues in RFC 4960", RFC 8540, DOI 10.17487/RFC8540, February 2019, <https://www.rfc-editor.org/info/rfc8540>.

Приложение A. Расчет контрольной суммы CRC32c

Мы определяем «отраженное значение», как значение с порядком битов, обратным по отношению к используемому в машине. 32-битовое значение CRC рассчитывается, как описано для CRC32c и использует полиномиальный код 0x11EDC6F41 (Castagnoli93) или $x^{32}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{14}+x^{13}+x^{11}+x^{10}+x^{9}+x^{8}+x^{6}+x^{0}$. Значение CRC рассчитывается с помощью процедуры, похожей на ETHERNET CRC [ITU32], которая изменена с учетом применения на транспортном уровне.

Расчёт CRC использует полиномиальное деление. Битовая строка сообщения (M) преобразуется в полином M(X) и значение CRC рассчитывается по M(X) с использованием полиномиальной арифметики.

При использовании CRC на канальном уровне полином строится с использованием естественного порядка битов - первый бит сигнала в линии является коэффициентов старшего порядка. Поскольку SCTP является протоколом транспортного уровня, он не может знать порядка передачи битов в физическую среду. Более того, на разных участках пути между конечными точками SCTP может на канальном уровне применяться разный порядок битов.

Следовательно, требуется соглашение для отображения транспортных сообщений SCTP на полиномы с целью расчёта CRC. При отображении сообщений SCTP на полиномы сначала берётся старший байт, но в каждом байте биты берутся, начиная с младшего. Первый байт сообщения обеспечивает восемь старших коэффициентов. В каждом байте

младший бит SCTP дает самый старший коэффициент полинома в рамках данного байта, а старший бит SCTP - младший коэффициент в рамках байта (такой порядок иногда называют отраженным - mirrored или reflected [WILLIAMS93]). Полиномы CRC преобразуются обратно в значения байтов транспортного уровня SCTP с помощью соответствующего отображения.

Значение CRC для транспортного уровня SCTP следует рассчитывать в соответствии с приведённым ниже описанием.

- Входными данными CRC является поток байтов с номерами 0 - N-1.
- Байтовый поток транспортного уровня отображается на полиномиальное значение. PDU размером N байтов с номерами j от 0 до N-1 рассматривается, как коэффициенты полинома M(x) порядка 8N-1 и бит 0 байта j будет коэффициентом $x^{(8(N-j)-8)}$, а бит 7 этого байта - $x^{(8(N-j)-1)}$.
- Оставшаяся часть регистра CRC заполняется единицами и рассчитывается значение CRC с помощью умножения на x^{32} и деления на полином CRC.
- Полином умножается на x^{32} и делится на G(x), что порождает полином степени не более 31, образующий оставшуюся часть R(x).
- Коэффициенты R(x) рассматриваются, как 32-битовая последовательность.
- Последовательность битов дополняется и результат является полиномом CRC.
- Полином CRC отображается обратно на байты транспортного уровня SCTP. Коэффициент x^{31} даёт значение биты 7 в байте SCTP 0, а коэффициент x^{24} даёт значение бита 0 в байте 0. Коэффициент x^{7} даёт значение бита 7 в байте 3, а коэффициент x^{0} - значение бита 0 в байте 3. Результирующая 4-байтовая последовательность является последовательностью транспортного уровня для 32-битовой контрольной суммы SCTP.

Примечание для разработчиков. В стандартах, книгах и литературе от производителей для CRC зачастую приводятся иные формулировки, где используется регистр для хранения остатка алгоритма деления long-division, инициализируемый нулями, а не 1 и первые 32 бита сообщения дополняются. Алгоритм long-division, используемый в нашей формулировке совмещает начальное умножение на 232 и «длинное деление» в одной операции. Для таких алгоритмов и сообщений, размер которых превышает 64 бита, две спецификации полностью эквивалентны. Обеспечение эквивалентности является одной из целей данного документа.

Следует предупредить разработчиков SCTP о том, что в литературе можно найти обе спецификации и иногда без ограничений для алгоритма long-division. Выбор формулировки в этом документе обусловлен возможностью применения не только для SCTP, когда тот же алгоритм CRC может применяться для сообщений меньше 64 битов.

Можно несколько снизить вычислительные издержки, проверяя ассоциацию по значению Verification Tag до расчета контрольной суммы, чтобы не рассчитывать контрольные суммы для пакетов с некорректными тегами. Исключением из этого правила являются блоки INIT и некоторые обмены SHUTDOWN-COMPLETE, а также устаревшие блоки COOKIE ECHO. Однако в этих случаях пакеты достаточно малы и расчет контрольных сумм не требует значительных ресурсов.

Ниже приведён (ненормативный) пример кода, заимствованный из генератора CRC с открытым кодом [WILLIAMS93], использующего метод «отражения» и таблицу для SCTP CRC32c с 256 записями по 32 бита в каждой. Этот метод не слишком быстрый и не слишком медленный по сравнению с поиском в таблицах CRC, но его преиуществом является возможность работы с процессами, использующими порядок big-endian и little-endian, при просмотре общих таблиц (с хост-порядком), а также использование лишь предопределенных операций ntohl() и htonl(). Код несколько отличается от [WILLIAMS93] для обеспечения переносимости между архитектурами big-endian и little-endian (отметим, что в тех случаях, когда известно, что целевая архитектура использует порядок little-endian, финальные операции bit-reversal и byte-reversal могут быть объединены).

```
<CODE BEGINS>
   /*****************************************************************/
   /* Для генератора таблиц Ross Williams устанавливаются       */
   /* значения TB_WIDTH=4, TB_POLLY=0x1EDC6F41, TB_REVER=TRUE   */
   /* Для прямого расчета Mr. Williams используются значения    */
   /* cm_width=32, cm_poly=0x1EDC6F41, cm_init=0xFFFFFFFF,      */
   /* cm_refin=TRUE, cm_refot=TRUE, cm_xorort=0x00000000        */
   /*****************************************************************/

   /* Пример файла с таблицей crc */
   #ifndef __crc32cr_table_h__
   #define __crc32cr_table_h__

   #define CRC32C_POLY 0x1EDC6F41
   #define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

   unsigned long  crc_c[256] =
   {
   0x00000000L, 0xF26B8303L, 0xE13B70F7L, 0x1350F3F4L,
   0xC79A971FL, 0x35F1141CL, 0x26A1E7E8L, 0xD4CA64EBL,
   0x8AD958CFL, 0x78B2DBCCL, 0x6BE22838L, 0x9989AB3BL,
   0x4D43CFD0L, 0xBF284CD3L, 0xAC78BF27L, 0x5E133C24L,
   0x105EC76FL, 0xE235446CL, 0xF165B798L, 0x030E349BL,
   0xD7C45070L, 0x25AFD373L, 0x36FF2087L, 0xC494A384L,
   0x9A879FA0L, 0x68EC1CA3L, 0x7BBCEF57L, 0x89D76C54L,
   0x5D1D08BFL, 0xAF768BBCL, 0xBC267848L, 0x4E4DFB4BL,
   0x20BD8EDEL, 0xD2D60DDDL, 0xC186FE29L, 0x33ED7D2AL,
   0xE72719C1L, 0x154C9AC2L, 0x061C6936L, 0xF477EA35L,
   0xAA64D611L, 0x580F5512L, 0x4B5FA6E6L, 0xB93425E5L,
   0x6DFE410EL, 0x9F95C20DL, 0x8CC531F9L, 0x7EAEB2FAL,
   0x30E349B1L, 0xC288CAB2L, 0xD1D83946L, 0x23B3BA45L,
   0xF779DEAEL, 0x05125DADL, 0x1642AE59L, 0xE4292D5AL,
```

```
  0xBA3A117EL, 0x4851927DL, 0x5B016189L, 0xA96AE28AL,
  0x7DA08661L, 0x8FCB0562L, 0x9C9BF696L, 0x6EF07595L,
  0x417B1DBCL, 0xB3109EBFL, 0xA0406D4BL, 0x522BEE48L,
  0x86E18AA3L, 0x748A09A0L, 0x67DAFA54L, 0x95B17957L,
  0xCBA24573L, 0x39C9C670L, 0x2A993584L, 0xD8F2B687L,
  0x0C38D26CL, 0xFE53516FL, 0xED03A29BL, 0x1F682198L,
  0x5125DAD3L, 0xA34E59D0L, 0xB01EAA24L, 0x42752927L,
  0x96BF4DCCL, 0x64D4CECFL, 0x77843D3BL, 0x85EFBE38L,
  0xDBFC821CL, 0x2997011FL, 0x3AC7F2EBL, 0xC8AC71E8L,
  0x1C661503L, 0xEE0D9600L, 0xFD5D65F4L, 0x0F36E6F7L,
  0x61C69362L, 0x93AD1061L, 0x80FDE395L, 0x72966096L,
  0xA65C047DL, 0x5437877EL, 0x4767748AL, 0xB50CF789L,
  0xEB1FCBADL, 0x197448AEL, 0x0A24BB5AL, 0xF84F3859L,
  0x2C855CB2L, 0xDEEEDFB1L, 0xCDBE2C45L, 0x3FD5AF46L,
  0x7198540DL, 0x83F3D70EL, 0x90A324FAL, 0x62C8A7F9L,
  0xB602C312L, 0x44694011L, 0x5739B3E5L, 0xA55230E6L,
  0xFB410CC2L, 0x092A8FC1L, 0x1A7A7C35L, 0xE811FF36L,
  0x3CDB9BDDL, 0xCEB018DEL, 0xDDE0EB2AL, 0x2F8B6829L,
  0x82F63B78L, 0x709DB87BL, 0x63CD4B8FL, 0x91A6C88CL,
  0x456CAC67L, 0xB7072F64L, 0xA457DC90L, 0x563C5F93L,
  0x082F63B7L, 0xFA44E0B4L, 0xE9141340L, 0x1B7F9043L,
  0xCFB5F4A8L, 0x3DDE77ABL, 0x2E8E845FL, 0xDCE5075CL,
  0x92A8FC17L, 0x60C37F14L, 0x73938CE0L, 0x81F80FE3L,
  0x55326B08L, 0xA759E80BL, 0xB4091BFFL, 0x466298FCL,
  0x1871A4D8L, 0xEA1A27DBL, 0xF94AD42FL, 0x0B21572CL,
  0xDFEB33C7L, 0x2D80B0C4L, 0x3ED04330L, 0xCCBBC033L,
  0xA24BB5A6L, 0x502036A5L, 0x4370C551L, 0xB11B4652L,
  0x65D122B9L, 0x97BAA1BAL, 0x84EA524EL, 0x7681D14DL,
  0x2892ED69L, 0xDAF96E6AL, 0xC9A99D9EL, 0x3BC21E9DL,
  0xEF087A76L, 0x1D63F975L, 0x0E330A81L, 0xFC588982L,
  0xB21572C9L, 0x407EF1CAL, 0x532E023EL, 0xA145813DL,
  0x758FE5D6L, 0x87E466D5L, 0x94B49521L, 0x66DF1622L,
  0x38CC2A06L, 0xCAA7A905L, 0xD9F75AF1L, 0x2B9CD9F2L,
  0xFF56BD19L, 0x0D3D3E1AL, 0x1E6DCDEEL, 0xEC064EEDL,
  0xC38D26C4L, 0x31E6A5C7L, 0x22B65633L, 0xD0DDD530L,
  0x0417B1DBL, 0xF67C32D8L, 0xE52CC12CL, 0x1747422FL,
  0x49547E0BL, 0xBB3FFD08L, 0xA86F0EFCL, 0x5A048DFFL,
  0x8ECEE914L, 0x7CA56A17L, 0x6FF599E3L, 0x9D9E1AE0L,
  0xD3D3E1ABL, 0x21B862A8L, 0x32E8915CL, 0xC083125FL,
  0x144976B4L, 0xE622F5B7L, 0xF5720643L, 0x07198540L,
  0x590AB964L, 0xAB613A67L, 0xB831C993L, 0x4A5A4A90L,
  0x9E902E7BL, 0x6CFBAD78L, 0x7FAB5E8CL, 0x8DC0DD8FL,
  0xE330A81AL, 0x115B2B19L, 0x020BD8EDL, 0xF0605BEEL,
  0x24AA3F05L, 0xD6C1BC06L, 0xC5914FF2L, 0x37FACCF1L,
  0x69E9F0D5L, 0x9B8273D6L, 0x88D28022L, 0x7AB90321L,
  0xAE7367CAL, 0x5C18E4C9L, 0x4F48173DL, 0xBD23943EL,
  0xF36E6F75L, 0x0105EC76L, 0x12551F82L, 0xE03E9C81L,
  0x34F4F86AL, 0xC69F7B69L, 0xD5CF889DL, 0x27A40B9EL,
  0x79B737BAL, 0x8BDCB4B9L, 0x988C474DL, 0x6AE7C44EL,
  0xBE2DA0A5L, 0x4C4623A6L, 0x5F16D052L, 0xAD7D5351L,
  };

  #endif

   /* Пример программы построения таблицы */
  #include <stdio.h>
  #include <stdlib.h>

  #define OUTPUT_FILE   "crc32cr.h"
  #define CRC32C_POLY    0x1EDC6F41L
  FILE *tf;
  unsigned long
  reflect_32 (unsigned long b)
  {
    int i;
    unsigned long rw = 0L;

    for (i = 0; i < 32; i++){
        if (b & 1)
          rw |= 1 << (31 - i);
        b >>= 1;
    }
    return (rw);
  }

  unsigned long
  build_crc_table (int index)
  {
    int i;
    unsigned long rb;

    rb = reflect_32 (index);

    for (i = 0; i < 8; i++){
        if (rb & 0x80000000L)
         rb = (rb << 1) ^ CRC32C_POLY;
```

```
        else
         rb <<= 1;
    }
    return (reflect_32 (rb));
}

main ()
{
    int i;

    printf ("\nGenerating CRC-32c table file <%s>\n",
    OUTPUT_FILE);
    if ((tf = fopen (OUTPUT_FILE, "w")) == NULL){
        printf ("Unable to open %s\n", OUTPUT_FILE);
        exit (1);
    }
    fprintf (tf, "#ifndef __crc32cr_table_h__\n");
    fprintf (tf, "#define __crc32cr_table_h__\n\n");
    fprintf (tf, "#define CRC32C_POLY 0x%08lX\n",
    CRC32C_POLY);
    fprintf (tf,
    "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n");
    fprintf (tf, "\nunsigned long  crc_c[256] =\n{\n");
    for (i = 0; i < 256; i++){
        fprintf (tf, "0x%08lXL, ", build_crc_table (i));
        if ((i & 3) == 3)
          fprintf (tf, "\n");
    }
    fprintf (tf, "};\n\n#endif\n");

    if (fclose (tf) != 0)
      printf ("Unable to close <%s>." OUTPUT_FILE);
    else
      printf ("\nThe CRC-32c table has been written to <%s>.\n",
        OUTPUT_FILE);
}

/* Пример вставки crc */
#include "crc32cr.h"

unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = ~0L;
    unsigned long result;
    unsigned char byte0,byte1,byte2,byte3;

    for (i = 0; i < length; i++){
        CRC32C(crc32, buffer[i]);
    }

    result = ~crc32;

    /*  В result храниться «негативный» остаток полинома,
     *  поскольку таблица и алгоритм являются «зеркальными [williams95].
     *  Т. е., result имеет такое же значение, как будто мы отобразили сообщение
     *  на полином, рассчитали остаток полинома для хостового порядка битов
     *  выполнили финальную смену знака (negation) и сквозное обращение битов.
     *  Отметим, что 32-битовое обращение битов идентично 4 восьмибитовым обращениям
     *  с последующим обращением порядка байтов. На машинах little-endian
     *  такое обращение байтов и финальная операция ntohl cancel не нужны.
     */
    byte0 = result & 0xff;
    byte1 = (result>>8) & 0xff;
    byte2 = (result>>16) & 0xff;
    byte3 = (result>>24) & 0xff;
    crc32 = ((byte0 << 24) |
             (byte1 << 16) |
             (byte2 << 8)  |
             byte3);
    return ( crc32 );
}

int
insert_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned long crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer,length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return 1;
}
```

```
   int
   validate_crc32(unsigned char *buffer, unsigned int length)
   {
     SCTP_message *message;
     unsigned int i;
     unsigned long original_crc32;
     unsigned long crc32 = ~0L;

     /* сохраним и обнулим контрольную сумму */
     message = (SCTP_message *) buffer;
     original_crc32 = ntohl(message->common_header.checksum);
     message->common_header.checksum = 0L;
     crc32 = generate_crc32c(buffer,length);
     return ((original_crc32 == crc32)? 1 : -1);
   }
<CODE ENDS>
```

Благодарности

Представленный в этом документе протокол является серьёзным достижением, основанным на результатах работы авторов исходного [RFC2960] - Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson.

Кроме того, следует отметить всех, кто внес вклад в создание исходного RFC - 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 многих других, кто дал свои значимые комментарии.

Добавим в этот список соавторов [RFC4460] - I. Arias-Rodriguez, K. Poon, A. Caro.

Отметим усилия участников всех семи проверок совместимости SCTP и тех, кто представил комментарии к [RFC4460], как отмечено здесь - Barry Zuckerman, La Monte Yarroll, Qiaobing Xie, Wang Xiaopeng, Jonathan Wood, Jeff Waskow, Mike Turner, John Townsend, Sabina Torrente, Cliff Thomas, Yuji Suzuki, Manoj Solanki, Sverre Slotte, Keyur Shah, Jan Rovins, Ben Robinson, Renee Revis, Ian Periam, RC Monee, Sanjay Rao, Sujith Radhakrishnan, Heinz Prantner, Biren Patel, Nathalie Mouellic, Mitch Miers, Bernward Meyknecht, Stan McClellan, Oliver Mayor, Tomas Orti Martin, Sandeep Mahajan, David Lehmann, Jonathan Lee, Philippe Langlois, Karl Knutson, Joe Keller, Gareth Keily, Andreas Jungmaier, Janardhan Iyengar, Mutsuya Irie, John Hebert, Kausar Hassan, Fred Hasle, Dan Harrison, Jon Grim, Laurent Glaude, Steven Furniss, Atsushi Fukumoto, Ken Fujita, Steve Dimig, Thomas Curran, Serkan Cil, Melissa Campbell, Peter Butler, Rob Brennan, Harsh Bhondwe, Brian Bidulock, Caitlin Bestler, Jon Berger, Robby Benedyk, Stephen Baucke, Sandeep Balani и Ronnie Sellar.

Отдельная благодарность Mark Allman, который реально должен был стать соавтором [RFC4460] по max-burst, но сумел уклониться по причине занятости. Благодарим также Lyndon Ong и Phil Conrad за их полезный вклад в работу.

Отметим также тех, кто подготовил [RFC4460] и комментировал его, включая Alfred Hoenes и Ronnie Sellars.

Добавим сюда соавтора [RFC8540]: Maksim Proshin и тех, кто представил замечания к [RFC8540]: Pontus Andersson, Eric W. Biederman, Cedric Bonnet, Spencer Dawkins, Gorry Fairhurst, Benjamin Kaduk, Mirja Kühlewind, Peter Lei, Gyula Marosi, Lionel Morand, Jeff Morriss, Tom Petch, Kacheong Poon, Julien Pourtet, Irene Rüngeler, Michael Welzl, Qiaobing Xie.

Наконец, добавим тех, кто представил комментарии к этому документу, включая Gorry Fairhurst, Martin Duke, Benjamin Kaduk, Tero Kivinen, Eliot Lear, Marcelo Ricardo Leitner, David Mandelberg, John Preuß Mattsson, Claudio Porfiri, Maksim Proshin, Ines Robles, Timo Völker, Magnus Westerlund, Zhouming.

Благодарность в адрес участников кодирования, тестирования и обновления документа непросто выразить словами. Спасибо Вам!

Адреса авторов

Randall R. Stewart
Netflix, Inc.
2455 Heritage Green Ave
Davenport, FL 33837
United States of America
Email: randall@lakerest.net

Michael Tüxen
Münster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany
Email: tuexen@fh-muenster.de

Karen E. E. Nielsen
Kamstrup A/S
Industrivej 28
DK-8660 Skanderborg
Denmark
Email: kee@kamstrup.com

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru