TAPS
Internet-Draft
Intended status: Informational
Expires: May 4, 2017

M. Welzl
University of Oslo
M. Tuexen
Muenster Univ. of Appl. Sciences
N. Khademi
University of Oslo

October 31, 2016

On the Usage of Transport Service Features Provided by IETF Transport Protocols
draft-ietf-taps-transports-usage-02

Abstract

This document describes how transport protocols expose services to applications and how an application can configure and use the features of a transport service.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 4, 2017.

Table of Contents

1. Terminology

Transport Service Feature: a specific end-to-end feature that a transport service provides to its clients. Examples include confidentiality, reliable delivery, ordered delivery, message-versus-stream orientation, etc.

Transport Service: a set of transport service features, without an association to any given framing protocol, which provides a complete service to an application.

Transport Protocol: an implementation that provides one or more different transport services using a specific framing and header format on the wire.

Transport Protocol Component: an implementation of a transport service feature within a protocol.

Transport Service Instance: an arrangement of transport protocols with a selected set of features and configuration parameters that implements a single transport service, e.g., a protocol stack (RTP over UDP).

Application: an entity that uses the transport layer for end-to-end delivery of data across the network (this may also be an upper layer protocol or tunnel encapsulation).

Endpoint: an entity that communicates with one or more other endpoints using a transport protocol.

Connection: shared state of two or more endpoints that persists across messages that are transmitted between these endpoints.

Primitive: a function call that is used to locally communicate between an application and a transport endpoint and is related to one or more Transport Service Features.

Parameter: a value passed between an application and a transport protocol by a primitive.

Socket: the combination of a destination IP address and a destination port number.

Transport Address: the combination of an IP address, transport protocol and the port number used by the transport protocol.

2. Introduction

This document presents defined interactions between transport protocols and applications in the form of 'primitives' (function calls). Primitives can be invoked by an application or a transport protocol; the latter type is called an "event". The list of transport service features and primitives in this document is strictly based on the parts of protocol specifications that relate to what the protocol provides to an application using it and how the application interacts with it. It does not cover parts of a protocol that are explicitly stated as optional to implement.

The document presents a three-pass process to arrive at a list of transport service features. In the first pass, the relevant RFC text is discussed per protocol. In the second pass, this discussion is used to derive a list of primitives that are uniformly categorized across protocols. Here, an attempt is made to present or -- where text describing primitives does not yet exist -- construct primitives in a slightly generalized form to highlight similarities. This is, for example, achieved by renaming primitives of protocols or by avoiding a strict 1:1-mapping between the primitives in the protocol specification and primitives in the list. Finally, the third pass presents transport service features based on pass 2, identifying which protocols implement them.

In the list resulting from the second pass, some transport service features are missing because they are implicit in some protocols, and they only become explicit when we consider the superset of all features offered by all protocols. For example, TCP's reliability includes integrity via a checksum, but we have to include a protocol like UDP-Lite as specified in [RFC3828] (which has a configurable checksum) in the list before we can consider an always-on checksum as a transport service feature. Similar arguments apply to other protocol functions (e.g. congestion control). The complete list of features across all protocols is therefore only available after pass 3.

This document discusses unicast transport protocols. [AUTHOR'S NOTE: we skip "congestion control mechanisms" for now. This simplifies the discussion; the congestion control mechanisms part is about LEDBAT, which should be easy to add later.] Transport protocols provide communication between processes that operate on network endpoints, which means that they allow for multiplexing of communication between the same IP addresses, and normally this multiplexing is achieved using port numbers. Port multiplexing is therefore assumed to be always provided and not discussed in this document.

Some protocols are connection-oriented. Connection-oriented protocols often use an initial call to a specific transport primitive to open a connection before communication can progress, and require communication to be explicitly terminated by issuing another call to a transport primitive (usually called "close"). A "connection" is the common state that some transport primitives refer to, e.g., to adjust general configuration settings. Connection establishment, maintenance and termination are therefore used to categorize transport primitives of connection-oriented transport protocols in pass 2 and pass 3.

3. Pass 1

This first iteration summarizes the relevant text parts of the RFCs describing the protocols, focusing on what each transport protocol provides to the application and how it is used (abstract API descriptions, where they are available).

3.1. Primitives Provided by TCP

[RFC0793] states: "The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks". Section 3.8 in [RFC0793] further specifies the interaction with the application by listing several transport primitives. It is also assumed that an Operating System provides a means for TCP to asynchronously signal the application; the primitives representing such signals are called 'events' in this section. This section describes the relevant primitives.

open: this is either active or passive, to initiate a connection or listen for incoming connections. All other primitives are associated with a specific connection, which is assumed to first have been opened. An active open call contains a socket. A passive open call with a socket waits for a particular connection; alternatively, a passive open call can leave the socket unspecified to accept any incoming connection. A fully specified passive call can later be made active by calling 'send'. Optionally, a timeout can be specified, after which TCP will abort the connection if data has not been successfully delivered to the destination (else a default timeout value is used). [RFC1122] describes a procedure for aborting the connection that must be used to avoid excessive retransmissions, and states that an application must be able to control the threshold used to determine the condition for aborting -- and that this threshold may be measured in time units or as a count of retransmission. This indicates that the timeout could also be specified as a count of retransmission.

Also optional, for multihomed hosts, the local IP address can be provided [RFC1122]. If it is not provided, a default choice will be made in case of active open calls. A passive open call will await incoming connection requests to all local addresses and then maintain usage of the local IP address where the incoming connection request has arrived. Finally, the 'options' parameter is explained in [RFC1122] to allow the application to specify IP options such as source route, record route, or timestamp. It is not stated on which segments of a connection these options should

be applied, but probably all segments, as this is also stated in a specification given for the usage of source route (section 4.2.3.8 of [RFC1122]). Source route is the only non-optional IP option in this parameter, allowing an application to specify a source route when it actively opens a TCP connection.

send: this is the primitive that an application uses to give the local TCP transport endpoint a number of bytes that TCP should reliably send to the other side of the connection. The URGENT flag, if set, states that the data handed over by this send call is urgent and this urgency should be indicated to the receiving process in case the receiving application has not yet consumed all non-urgent data preceding it. An optional timeout parameter can be provided that updates the connection's timeout (see 'open').

receive: This primitive allocates a receiving buffer for a provided number of bytes. It returns the number of received bytes provided in the buffer when these bytes have been received and written into the buffer by TCP. The application is informed of urgent data via an URGENT flag: if it is on, there is urgent data. If it is off, there is no urgent data or this call to 'receive' has returned all the urgent data.

close: This primitive closes one side of a connection. It is semantically equivalent to "I have no more data to send" but does not mean "I will not receive any more", as the other side may still have data to send. This call reliably delivers any data that has already been given to TCP (and if that fails, 'close' becomes 'abort').

abort: This primitive causes all pending 'send' and 'receive' calls to be aborted. A TCP RESET message is sent to the TCP endpoint on the other side of the connection [RFC0793].

close event: TCP uses this primitive to inform an application that the application on the other side has called the 'close' primitive, so the local application can also issue a 'close' and terminate the connection gracefully. See [RFC0793], Section 3.5.

abort event: When TCP aborts a connection upon receiving a "Reset" from the peer, it "advises the user and goes to the CLOSED state." See [RFC0793], Section 3.4.

USER TIMEOUT event: This event, described in Section 3.9 of [RFC0793], is executed when the user timeout expires (see 'open'). All queues are flushed and the application is informed that the connection had to be aborted due to user timeout.

ERROR_REPORT event: This event, described in Section 4.2.4.1 of [RFC1122], informs the application of "soft errors" that can be safely ignored [RFC5461], including the arrival of an ICMP error message or excessive retransmissions (reaching a threshold below the threshold where the connection is aborted).

Type-of-Service: Section 4.2.4.2 of [RFC1122] states that the application layer MUST be able to specify the Type-of-Service (TOS) for segments that are sent on a connection. The application should be able to change the TOS during the connection lifetime, and the TOS value should be passed to the IP layer unchanged. Since then the TOS field has been redefined. A part of the field has been assigned to ECN [RFC3168] and the six most significant bits have been assigned to carry the DiffServ CodePoint, DSField [RFC3260]. Staying with the intention behind the application's ability to specify the "Type of Service", this should probably be interpreted to mean the value in the DSField, which is the Differentiated Services Codepoint (DSCP).

Nagle: The Nagle algorithm, described in Section 4.2.3.4 of [RFC1122], delays sending data for some time to increase the likelihood of sending a full-sized segment. An application can disable the Nagle algorithm for an individual connection.

User Timeout Option: The User Timeout Option (UTO) [RFC5482] allows one end of a TCP connection to advertise its current user timeout value so that the other end of the TCP connection can adapt its own user timeout accordingly. In addition to the configurable value of the User Timeout (see 'send'), [RFC5482] introduces three per-connection state variables that an application can adjust to control the operation of the User Timeout Option (UTO): ADV_UTO is the value of the UTO advertised to the remote TCP peer (default: system-wide default user timeout); ENABLED (default false) is a boolean-type flag that controls whether the UTO option is enabled for a connection. This applies to both sending and receiving. CHANGEABLE is a boolean-type flag (default true) that controls whether the user timeout may be changed based on a UTO option received from the other end of the connection. CHANGEABLE becomes false when an application explicitly sets the user timeout (see 'send').

3.1.1. Excluded Primitives or Parameters

The 'open' primitive specified in [RFC0793] can be handed optional Precedence or security/compartment information according to [RFC0793], but this was not included here because it is mostly irrelevant today, as explained in [RFC7414].

The 'status' primitive was not included because [RFC0793] describes this primitive as "implementation dependent" and states that it "could be excluded without adverse effect". Moreover, while a data block containing specific information is described, it is also stated that not all of this information may always be available. The 'send' primitive described in [RFC0793] includes an optional PUSH flag which, if set, requires data to be promptly transmitted to the receiver without delay; the 'receive' primitive described in [RFC0793] can (under some conditions) yield the status of the PUSH flag. Because PUSH functionality is made optional to implement for both the 'send' and 'receive' primitives in [RFC1122], this functionality is not included here. [RFC1122] also introduces keep-alives to TCP, but these are optional to implement and hence not considered here. [RFC1122] describes that "some TCP implementations have included a FLUSH call", indicating that this call is also optional to implement. It is therefore not considered here.

3.2. Primitives Provided by MPTCP

Multipath TCP (MPTCP) is an extension to TCP that allows the use of multiple paths for a single data-stream. It achieves this by creating different so-called TCP subflows for each of the interfaces and scheduling the traffic across these TCP subflows. The service provided by MPTCP is described in [RFC6182] "Multipath TCP MUST follow the same service model as TCP [RFC0793]: in- order, reliable, and byte-oriented delivery. Furthermore, a Multipath TCP connection SHOULD provide the application with no worse throughput or resilience than it would expect from running a single TCP connection over any one of its available paths."

Further, [RFC6182] states constraints on the API exposed by MPTCP: "A multipath-capable equivalent of TCP MUST retain some level of backward compatibility with existing TCP APIs, so that existing applications can use the newer merely by upgrading the operating systems of the end hosts." As such, the primitives provided by MPTCP are equivalent to the ones provided by TCP. Nevertheless, [RFC6824] and [RFC6897] clarify some parts of TCP's primitives with respect to MPTCP and add some extensions for better control on MPTCP's subflows. Hereafter is a list of the clarifications and extensions the above cited RFCs provide to TCP's primitives.

open: [RFC6897] states "An application should be able to request to turn on or turn off the usage of MPTCP.". The RFC states that this functionality can be provided through a socket-option called TCP_MULTIPATH_ENABLE. Further, [RFC6897] says that MPTCP must be disabled in case the application is binding to a specific address.

send/receive: [RFC6824] states that the sending and receiving of data does not require any changes to the application when MPTCP is being used. The MPTCP-layer will "take one input data stream from an application, and split it into one or more subflows, with sufficient control information to allow it to be reassembled and delivered reliably and in order to the recipient application." The use of the Urgent-Pointer is special in MPTCP and [RFC6824] says "a TCP subflow MUST NOT use the Urgent Pointer to interrupt an existing mapping."

address and subflow management: MPTCP uses different addresses and allows a host to announce these addresses as part of the protocol. [RFC6897] says "An application should be able to restrict MPTCP to binding to a given set of addresses." and thus allows applications to limit the set of addresses that are being used by MPTCP. Further, "An application should be able to obtain information on the pairs of addresses used by the MPTCP subflows.".

3.3. Primitives Provided by SCTP

Section 1.1 of [RFC4960] lists limitations of TCP that SCTP removes. Three of the four mentioned limitations directly translate into a transport service features that are visible to an application using SCTP: 1) it allows for preservation of message delineations; 2) these messages, while reliably transferred, do not require to be in order unless the application wants it; 3) multi-homing is supported. In SCTP, connections are called "association" and they can be between not only two (as in TCP) but multiple addresses at each endpoint.

Section 10 of [RFC4960] further specifies the interaction with the application (which RFC [RFC4960] calls the "Upper Layer Protocol" (ULP)). It is assumed that the Operating System provides a means for SCTP to asynchronously signal the application; the primitives representing such signals are called 'events' in this section. Here, we describe the relevant primitives. In addition to the abstract API described in Section 10 of [RFC4960], an extension to the socket API is described in [RFC6458] covering the functionality of the base protocol specified in [RFC4960] and its extensions specified in [RFC3758], [RFC4895], and [RFC5061]. For the protocol extensions specified in [RFC6525], [RFC6951], [RFC7053], [RFC7496], and [RFC7829] the corresponding extensions of the socket API are specified in these protocol specifications. The functionality exposed to the ULP through this socket API is considered here in addition to the abstract API specified in Section 10 of [RFC4960].

Initialize: Initialize creates a local SCTP instance that it binds to a set of local addresses (and, if provided, port number). Initialize needs to be called only once per set of local addresses.

Associate: This creates an association (the SCTP equivalent of a connection) between the local SCTP instance and a remote SCTP instance. Most primitives are associated with a specific association, which is assumed to first have been created. Associate can return a list of destination transport addresses so that multiple paths can later be used. One of the returned sockets will be selected by the local endpoint as default primary path for sending SCTP packets to this peer, but this choice can be changed by the application using the list of destination addresses. Associate is also given the number of outgoing streams to request and optionally returns the number of outgoing streams negotiated. An optional parameter of 32-bits, the adaptation layer indication, can be provided, as specified in [RFC5061]. If the extension specified in [RFC4895] is used, the chunk types required to be sent authenticated by the peer can be provided.

Send: This sends a message of a certain length in bytes over an association. A number can be provided to later refer to the correct message when reporting an error, and a stream id is provided to specify the stream to be used inside an association (we consider this as a mandatory parameter here for simplicity: if not provided, the stream id defaults to 0). A condition to abandon the message can be specified (for example limiting the number of retransmissions or the lifetime of the user message). This allows to control the partial reliability extension specified in [RFC3758] and [RFC7496]. An optional maximum life time can specify the time after which the message should be discarded rather than sent. A choice (advisory, i.e. not guaranteed) of the preferred path can be made by providing a socket, and the message can be delivered out-of-order if the unordered flag is set. An advisory flag indicates that the peer should not delay the acknowledgement of the user message provided by making use of the I-bit specified in [RFC7053]. Another advisory flag indicates whether the application prefers to avoid bundling user data with other outbound DATA chunks (i.e., in the same packet). A payload protocol-id can be provided to pass a value that indicates the type of payload protocol data to the peer. If the extension specified in [RFC4895] is used, the key identifier used for authenticating the DATA chunks can be provided.

Receive: Messages are received from an association, and optionally a stream within the association, with their size returned. The application is notified of the availability of data via a DATA ARRIVE notification. If the sender has included a payload protocol-id, this value is also returned. If the received message is only a partial delivery of a whole message, a partial flag will indicate so, in which case the stream id and a stream sequence number are provided to the application. A delivery number lets the application detect reordering.

Shutdown: This primitive gracefully closes an association, reliably delivering any data that has already been handed over to SCTP. A return code informs about success or failure of this procedure.

Abort: This ungracefully closes an association, by discarding any locally queued data and informing the peer that the association was aborted. Optionally, an abort reason to be passed to the peer may be provided by the application. A return code informs about success or failure of this procedure.

Change Heartbeat / Request Heartbeat: This allows the application to enable/disable heartbeats and optionally specify a heartbeat frequency as well as requesting a single heartbeat to be carried out upon a function call, with a notification about success or failure of transmitting the HEARTBEAT chunk to the destination.

Set Protocol Parameters: This allows to set values for protocol parameters per association; for some parameters, a setting can be made per socket. The set listed in [RFC4960] is: RTO.Initial; RTO.Min; RTO.Max; Max.Burst; RTO.Alpha; RTO.Beta; Valid.Cookie.Life; Association.Max.Retrans; Path.Max.Retrans; Max.Init.Retransmits; HB.interval; HB.Max.Burst. In addition to these, the Quick Failover Algorithm specified in [RFC7829] can be controlled by the PotentiallyFailed.Max.Retrans and Primary.Switchover.Max.Retrans parameter. A remote UDP encapsulation port can be set for using UDP encapsulation as specified in [RFC6951].

Set Primary: This allows to set a new primary default path for an association by providing a socket. Optionally, a default source address to be used in IP datagrams can be provided.

Set / Get Authentication Parameters: This allows an endpoint to add/remove key material to/from an association. In addition, the chunk types being authenticated can be queried. This is provided by the protocol extension defined in [RFC4895].

Change Local Address / Set Peer Primary: This allows an endpoint to add/remove local addresses to/from an association. In addition, the peer can be given a hint which address to use as the primary address. This is provided by the protocol extension defined in [RFC5061].

Add / Reset Streams, Reset Association: This allows an endpoint to add streams to an existing association or or to reset them individually. Additionally, the association can be reset. This is provided by the protocol extension defined in [RFC6525].

Status: The 'Status' primitive returns a data block with information about a specified association, containing: association connection state; socket list; destination transport address reachability states; current receiver window size; current congestion window sizes; number of unacknowledged DATA chunks; number of DATA chunks pending receipt; primary path; most recent SRTT on primary path; RTO on primary path; SRTT and RTO on other destination addresses.

COMMUNICATION UP notification: When a lost communication to an endpoint is restored or when SCTP becomes ready to send or receive user messages, this notification informs the application process about the affected association, the type of event that has occurred, the complete set of sockets of the peer, the maximum number of allowed streams and the inbound stream count (the number of streams the peer endpoint has requested).

DATA ARRIVE notification: When a message is ready to be retrieved via the Receive primitive, the application is informed by this notification.

SEND FAILURE notification / Receive Unsent Message / Receive Unacknowledged Message: When a message cannot be delivered via an association, the sender can be informed about it and learn whether the message has just not been acknowledged or (e.g. in case of lifetime expiry) if it has not even been sent.

NETWORK STATUS CHANGE notification: The NETWORK STATUS CHANGE notification informs the application about a socket becoming active/inactive.

COMMUNICATION LOST notification: When SCTP loses communication to an endpoint (e.g. via Heartbeats or excessive retransmission) or detects an abort, this notification informs the application process of the affected association and the type of event (failure OR termination in response to a shutdown or abort request).

SHUTDOWN COMPLETE notification: When SCTP completes the shutdown procedures, this notification is passed to the upper layer, informing it about the affected assocation.

AUTHENICATION notification: When SCTP wants to notify the upper layer regarding the key management related to the extension defined in [RFC4895], this notification is passed to the upper layer.

ADAPTATION LAYER INDICATION notification: When SCTP completes the association setup and the peer provided an adaptation layer indication, this is passed to the upper layer. This extension is defined in [RFC5061] and [RFC6458].

STREAM RESET notification: When SCTP completes the procedure for resetting streams as specified in [RFC6525], this notification is passed to the upper layer, informing it about the result.

ASSOCIATION RESET notification: When SCTP completes the association reset procedure as specified in [RFC6525], this notification is passed to the upper layer, informing it about the result.

STREAM CHANGE notification: When SCTP completes the procedure used to increase the number of streams as specified in [RFC6525], this notification is passed to the upper layer, informing it about the result.

3.3.1. Excluded Primitives or Parameters

The 'Receive' primitive can return certain additional information, but this is optional to implement and therefore not considered. With a COMMUNICATION LOST notification, some more information may optionally be passed to the application (e.g., identification to retrieve unsent and unacknowledged data). SCTP "can invoke" a COMMUNICATION ERROR notification and "may send" a RESTART notification, making these two notifications optional to implement. The list provided under 'Status' includes "etc", indicating that more information could be provided. The primitive 'Get SRTT Report' returns information that is included in the information that 'Status' provides and is therefore not discussed. Similarly, 'Set Failure Threshold' sets only one out of various possible parameters included in 'Set Protocol Parameters'. The 'Destroy SCTP Instance' API function was excluded: it erases the SCTP instance that was created by 'Initialize', but is not a Primitive as defined in this document because it does not relate to a Transport Service Feature.

3.4. Primitives Provided by UDP and UDP-Lite

The primitives provided by UDP and UDP-Lite are described in [FJ16].

4. Pass 2

This pass categorizes the primitives from pass 1 based on whether they relate to a connection or to data transmission. Primitives are presented following the nomenclature "CATEGORY.[SUBCATEGORY].PRIMITIVENAME.PROTOCOL". The CATEGORY can be CONNECTION or DATA. Within the CONNECTION category, ESTABLISHMENT, AVAILABILITY, MAINTENANCE and TERMINATION subcategories can be considered. The DATA category does not have any SUBCATEGORY (as of now). The PROTOCOL name "UDP(-Lite)" is used when primitives are equivalent for UDP and UDP-Lite; the PROTOCOL name "TCP" refers to both TCP and MPTCP. We present "connection" as a general protocol-independent concept and use it to refer to, e.g., TCP connections (identifiable by a unique pair of IP addresses and TCP port numbers), SCTP associations (identifiable by multiple IP address and port number pairs), as well UDP and UDP-Lite connections (identifiable by a unique socket pair).

Some minor details are omitted for the sake of generalization -- e.g., SCTP's 'close' [RFC4960] returns success or failure, whereas this is not described in the same way for TCP in [RFC0793], but this detail plays no significant role for the primitives provided by either TCP or SCTP.

The TCP 'send' and 'receive' primitives include usage of an "URGENT" mechanism. This mechanism is required to implement the "synch signal" used by telnet [RFC0854], but SHOULD NOT be used by new applications [RFC6093]. Because pass 2 is meant as a basis for the creation of TAPS systems, the "URGENT" mechanism is excluded. This also concerns the notification "Urgent pointer advance" in the ERROR_REPORT described in Section 4.2.4.1 of [RFC1122].

4.1. CONNECTION Related Primitives

ESTABLISHMENT:
Active creation of a connection from one transport endpoint to one or more transport endpoints.
Interfaces to UDP and UDP-Lite allow both connection-oriented and connection-less usage of the API [I-D.ietf-tsvwg-rfc5405bis]

- CONNECT.TCP:
  Pass 1 primitive / event: 'open' (active) or 'open' (passive) with socket, followed by 'send'

Parameters: 1 local IP address (optional); 1 destination transport address (for active open; else the socket and the local IP address of the succeeding incoming connection request will be maintained); timeout (optional); options (optional)
Comments: If the local IP address is not provided, a default choice will automatically be made. The timeout can also be a retransmission count. The options are IP options to be used on all segments of the connection. At least the Source Route option is mandatory for TCP to provide.

- CONNECT.SCTP:
  Pass 1 primitive / event: 'initialize', followed by 'associate'
  Parameters: list of local SCTP port number / IP address pairs (initialize); 1 socket; outbound stream count; adaptation layer indication; chunk types required to be authenticated
  Returns: socket list
  Comments: 'initialize' needs to be called only once per list of local SCTP port number / IP address pairs. One socket will automatically be chosen; it can later be changed in MAINTENANCE.

- CONNECT.MPTCP:
  This is similar to CONNECT.TCP except for one additional boolean parameter that allows to enable or disable MPTCP for a particular connection or socket (default: enabled).

- CONNECT.UDP(-Lite):
  Pass 1 primitive / event: 'connect' followed by 'send'.
  Parameters: 1 local IP address (default (ANY), or specified); 1 destination transport address; 1 local port (default (OS chooses), or specified); 1 destination port (default (OS chooses), or specified).
  Comments: Associates a transport address creating a UDP(-Lite) socket connection. This can be called again with a new transport address to create a new connection. The CONNECT function allows an application to receive errors from messages sent to a transport address.

AVAILABILITY:
Preparing to receive incoming connection requests.

- LISTEN.TCP:
  Pass 1 primitive / event: 'open' (passive)
  Parameters: 1 local IP address (optional); 1 socket (optional); timeout (optional)
  Comments: if the socket and/or local IP address is provided, this waits for incoming connections from only and/or to only the provided address. Else this waits for incoming connections

without this / these constraint(s). ESTABLISHMENT can later be performed with 'send'.

- LISTEN.SCTP:
  Pass 1 primitive / event: 'initialize', followed by 'COMMUNICATION UP' notification and possibly 'ADAPTATION LAYER' notification
  Parameters: list of local SCTP port number / IP address pairs (initialize)
  Returns: socket list; outbound stream count; inbound stream count; adaptation layer indication; chunks required to be authenticated
  Comments: initialize needs to be called only once per list of local SCTP port number / IP address pairs. COMMUNICATION UP can also follow a COMMUNICATION LOST notification, indicating that the lost communication is restored. If the peer has provided an adaptation layer indication, an 'ADAPTATION LAYER' notification is issued.

- LISTEN.MPTCP:
  This is similar to LISTEN.TCP except for one additional boolean parameter that allows to enable or disable MPTCP for a particular connection or socket (default: enabled).

- LISTEN.UDP(-Lite):
  Pass 1 primitive / event: 'receive'.
  Parameters: 1 local IP address (default (ANY), or specified); 1 destination transport address; local port (default (OS chooses), or specified); destination port (default (OS chooses), or specified).
  Comments: The receive function registers the application to listen for incoming UDP(-Lite) datagrams at an endpoint.

MAINTENANCE:
Adjustments made to an open connection, or notifications about it. These are out-of-band messages to the protocol that can be issued at any time, at least after a connection has been established and before it has been terminated (with one exception: CHANGE-TIMEOUT.TCP can only be issued for an open connection when DATA.SEND.TCP is called). In some cases, these primitives can also be immediately issued during ESTABLISHMENT or AVAILABILITY, without waiting for the connection to be opened (e.g. CHANGE-TIMEOUT.TCP can be done using TCP's 'open' primitive). For UDP and UDP-Lite, these functions may establish a setting per connection, but may also be changed per datagram message.

- CHANGE-TIMEOUT.TCP:
  Pass 1 primitive / event: 'open' or 'send' combined with unspecified control of per-connection state variables
  Parameters: timeout value (optional); ADV_UTO (optional); boolean

UTO_ENABLED (optional, default false); boolean CHANGEABLE (optional, default true)
Comments: when sending data, an application can adjust the connection's timeout value (time after which the connection will be aborted if data could not be delivered). If UTO_ENABLED is true, the user timeout value (or, if provided, the value ADV_UTO) will be advertised for the TCP on the other side of the connection to adapt its own user timeout accordingly. UTO_ENABLED controls whether the UTO option is enabled for a connection. This applies to both sending and receiving. CHANGEABLE controls whether the user timeout may be changed based on a UTO option received from the other end of the connection; it becomes false when 'timeout value' is used.

- CHANGE-TIMEOUT.SCTP:
  Pass 1 primitive / event: 'Change HeartBeat' combined with 'Set Protocol Parameters'
  Parameters: 'Change HeartBeat': heartbeat frequency; 'Set Protocol Parameters': Association.Max.Retrans (whole association) or Path.Max.Retrans (per socket)
  Comments: Change Heartbeat can enable / disable heartbeats in SCTP as well as change their frequency. The parameter Association.Max.Retrans defines after how many unsuccessful heartbeats the connection will be terminated; thus these two primitives / parameters together can yield a similar behavior to CHANGE-TIMEOUT.TCP.

- DISABLE-NAGLE.TCP:
  Pass 1 primitive / event: not specified
  Parameters: one boolean value
  Comments: the Nagle algorithm delays data transmission to increase the chance to send a full-sized segment. An application must be able to disable this algorithm for a connection.

- REQUESTHEARTBEAT.SCTP:
  Pass 1 primitive / event: 'Request HeartBeat'
  Parameters: socket
  Returns: success or failure
  Comments: requests an immediate heartbeat on a path, returning success or failure.

- SETPROTOCOLPARAMETERS.SCTP:
  Pass 1 primitive / event: 'Set Protocol Parameters'
  Parameters: RTO.Initial; RTO.Min; RTO.Max; Max.Burst; RTO.Alpha; RTO.Beta; Valid.Cookie.Life; Association.Max.Retrans; Path.Max.Retrans; Max.Init.Retransmits; HB.interval; HB.Max.Burst; PotentiallyFailed.Max.Retrans; Primary.Switchover.Max.Retrans; Remote.UDPEncapsPort.

- SETPRIMARY.SCTP:
  Pass 1 primitive / event: 'Set Primary'
  Parameters: socket
  Returns: result of attempting this operation
  Comments: update the current primary address to be used, based on the set of available sockets of the association.

- SETPEERPRIMARY.SCTP:
  Pass 1 primitive / event: Change Local Address / Set Peer Primary
  Parameters: local IP address
  Comments: this is only advisory for the peer.

- SETAUTH.SCTP:
  Pass 1 primitive / event: Set / Get Authentication Parameters
  Parameters: key_id, key, hmac_id

- GETAUTH.SCTP:
  Pass 1 primitive / event: Set / Get Authentication Parameters
  Parameters: key_id, chunk_list

- RESETSTREAM.SCTP:
  Pass 1 primitive / event: Add / Reset Streams, Reset Association
  Parameters: sid, direction

- RESETSTREAM-EVENT.SCTP:
  Pass 1 primitive / event: STREAM RESET notification
  Parameters: information about the result of RESETSTREAM.SCTP.
  Comments: This is issued when the procedure for resetting streams has completed.

- RESETASSOC.SCTP:
  Pass 1 primitive / event: Add / Reset Streams, Reset Association
  Parameters: information related to the extension defined in [RFC3260].

- RESETASSOC-EVENT.SCTP:
  Pass 1 primitive / event: ASSOCIATION RESET notification
  Parameters: information about the result of RESETASSOC.SCTP.
  Comments: This is issued when the procedure for resetting an association has completed.

- ADDSTREAM.SCTP:
  Pass 1 primitive / event: Add / Reset Streams, Reset Association
  Parameters: number if outgoing and incoming streams to be added

- ADDSTREAM-EVENT.SCTP:
  Pass 1 primitive / event: STREAM CHANGE notification
  Parameters: information about the result of ADDSTREAM.SCTP.

Comments: This is issued when the procedure for adding a stream has completed.

- ERROR.TCP:
  Pass 1 primitive / event: 'ERROR_REPORT'
  Returns: reason (encoding not specified); subreason (encoding not specified)
  Comments: soft errors that can be ignored without harm by many applications; an application should be able to disable these notifications. The reported conditions include at least: ICMP error message arrived; Excessive Retransmissions.

- ERROR.UDP(-Lite):
  Pass 1 primitive / event: 'ERROR_REPORT'.
  Returns: Error report
  Comments: This returns soft errors that may be ignored without harm by many applications; An application must connect to be able receive these notifications.

- STATUS.SCTP:
  Pass 1 primitive / event: 'Status' and 'NETWORK STATUS CHANGE' notification
  Returns: data block with information about a specified association, containing: association connection state; socket list; destination transport address reachability states; current receiver window size; current congestion window sizes; number of unacknowledged DATA chunks; number of DATA chunks pending receipt; primary path; most recent SRTT on primary path; RTO on primary path; SRTT and RTO on other destination addresses. The NETWORK STATUS CHANGE notification informs the application about a socket becoming active/inactive.

- STATUS.MPTCP:
  Pass 1 primitive / event: not specified
  Returns: list of pairs of tuples of IP address and TCP port number of each subflow. The first of the pair is the local IP and port number, while the second is the remote IP and port number.

- SET_DSCP.TCP:
  Pass 1 primitive / event: not specified
  Parameters: DSCP value
  Comments: this allows an application to change the DSCP value for outgoing segments. For TCP this was originally specified for the TOS field [RFC1122], which is here interpreted to refer to the DSField [RFC3260].

- SET_DSCP.UDP(-Lite):
  Pass 1 primitive / event: 'SET_DSCP'
  Parameter: DSCP value
  Comments: This allows an application to change the DSCP value for outgoing UDP(-Lite) datagrams. [RFC7657] and [I-D.ietf-tsvwg-rfc5405bis] provide current guidance on using this value with UDP.

- ADD_SUBFLOW.MPTCP:
  Pass 1 primitive / event: not specified
  Parameters: local IP address and optionally the local port number
  Comments: the application specifies the local IP address and port number that must be used for a new subflow.

- ADD_ADDR.SCTP:
  Pass 1 primitive / event: Change Local Address / Set Peer Primary
  Parameters: local IP address

- REM_SUBFLOW.MPTCP:
  Pass 1 primitive / event: not specified
  Parameters: local IP address, local port number, remote IP address, remote port number
  Comments: the application removes the subflow specified by the IP/port-pair. The MPTCP implementation must trigger a removal of the subflow that belongs to this IP/port-pair.

- REM_ADDR.SCTP:
  Pass 1 primitive / event: Change Local Address / Set Peer Primary
  Parameters: local IP address

- CHECKSUM.UDP:
  Pass 1 primitive / event: 'DISABLE_CHECKSUM'.
  Parameters: 0 when no checksum is used at sender, 1 for checksum at sender (default).

- CHECKSUM_REQUIRED.UDP:
  Pass 1 primitive / event: 'REQUIRE_CHECKSUM'.
  Parameter: 0 when checksum is required at receiver, 1 to allow zero checksum at receiver (default).

- SET_CHECKSUM_COVERAGE.UDP-Lite:
  Pass 1 primitive / event: 'SET_CHECKSUM_COVERAGE'.
  Parameters: Coverage length at sender (default maximum coverage)

- SET_MIN_CHECKSUM_COVERAGE.UDP-Lite:
  Pass 1 primitive / event: 'SET_MIN_COVERAGE'.
  Parameter: Coverage length at receiver (default minimum coverage)

- SET_DF.UDP(-Lite):
  Pass 1 primitive event: 'SET_DF'.
  Parameter: 0 when DF is not set (default), 1 when DF is set.

- SET_TTL.UDP(-Lite) (IPV6_UNICAST_HOPS):
  Pass 1 primitive / event: 'SET_TTL' and 'SET_IPV6_UNICAST_HOPS'
  Parameters: IPv4 TTL value or IPv6 Hop Count value
  Comments: This allows an application to change the IPv4 TTL of
  IPv6 Hop count value for outgoing UDP(-Lite) datagrams.

- GET_TTL.UDP(-Lite) (IPV6_UNICAST_HOPS):
  Pass 1 primitive / event: 'GET_TTL' and 'GET_IPV6_UNICAST_HOPS'
  Returns: IPv4 TTL value or IPv6 Hop Count value
  Comments: This allows an application to read the the IPv4 TTL of
  IPv6 Hop count value from a received UDP(-Lite) datagram.

- SET_ECN.UDP(-Lite):
  Pass 1 primitive / event: 'SET_ECN'
  Parameters: ECN value
  Comments: This allows a UDP(-Lite) application to set the ECN
  codepoint field for outgoing UDP(-Lite) datagrams.

- GET_ECN.UDP(-Lite):
  Pass 1 primitive / event: 'GET_ECN'
  Parameters: ECN value
  Comments: This allows a UDP(-Lite) application to read the ECN
  codepoint field from a received UDP(-Lite) datagram.

- SET_IP_OPTIONS.UDP(-Lite):
  Pass 1 primitive / event: 'SET_IP_OPTIONS'
  Parameters: options
  Comments: This allows a UDP(-Lite) application to set IP Options
  for outgoing UDP(-Lite) datagrams. These options can at least be
  the Source Route, Record Route, and Time Stamp option.

- GET_IP_OPTIONS.UDP(-Lite):
  Pass 1 primitive / event: 'GET_IP_OPTIONS'
  Returns: options
  Comments: This allows a UDP(-Lite) application to receive any IP
  options that are contained in a received UDP(-Lite) datagram.

- AUTHENTICATION_NOTIFICATION-EVENT.SCTP:
  Pass 1 primitive / event: 'AUTHENTICATION notification'
  Returns: information regarding key management.

TERMINATION:
Gracefully or forcefully closing a connection, or being informed

about this event happening.

- CLOSE.TCP:
  Pass 1 primitive / event: 'close'
  Comments: this terminates the sending side of a connection after reliably delivering all remaining data.

- CLOSE.SCTP:
  Pass 1 primitive / event: 'Shutdown'
  Comments: this terminates a connection after reliably delivering all remaining data.

- CLOSE.UDP(-Lite):
  Pass 1 primitive event: 'CLOSE'
  Comments: No further UDP(-Lite) datagrams are sent/received on this connection.

- ABORT.TCP:
  Pass 1 primitive / event: 'abort'
  Comments: this terminates a connection without delivering remaining data and sends an error message to the other side.

- ABORT.SCTP:
  Pass 1 primitive / event: 'abort'
  Parameters: abort reason to be given to the peer (optional)
  Comments: this terminates a connection without delivering remaining data and sends an error message to the other side.

- TIMEOUT.TCP:
  Pass 1 primitive / event: 'USER TIMEOUT' event
  Comments: the application is informed that the connection is aborted. This event is executed on expiration of the timeout set in CONNECTION.ESTABLISHMENT.CONNECT.TCP (possibly adjusted in CONNECTION.MAINTENANCE.CHANGE-TIMEOUT.TCP).

- TIMEOUT.SCTP:
  Pass 1 primitive / event: 'COMMUNICATION LOST' event
  Comments: the application is informed that the connection is aborted. this event is executed on expiration of the timeout that should be enabled by default (see beginning of section 8.3 in [RFC4960]) and was possibly adjusted in CONNECTION.MAINTENANCE.CHANGE-TIMEOOUT.SCTP.

- ABORT-EVENT.TCP:
  Pass 1 primitive / event: not specified.

- ABORT-EVENT.SCTP:
  Pass 1 primitive / event: 'COMMUNICATION LOST' event
  Returns: abort reason from the peer (if available)
  Comments: the application is informed that the other side has aborted the connection using CONNECTION.TERMINATION.ABORT.SCTP.

- CLOSE-EVENT.TCP:
  Pass 1 primitive / event: not specified.

- CLOSE-EVENT.SCTP:
  Pass 1 primitive / event: 'SHUTDOWN COMPLETE' event
  Comments: the application is informed that CONNECTION.TERMINATION.CLOSE.SCTP was successfully completed.

4.2. DATA Transfer Related Primitives

All primitives in this section refer to an existing connection, i.e. a connection that was either established or made available for receiving data (although this is optional for the primitives of UDP(-Lite)). In addition to the listed parameters, all sending primitives contain a reference to a data block and all receiving primitives contain a reference to available buffer space for the data.

- SEND.TCP:
  Pass 1 primitive / event: 'send'
  Parameters: timeout (optional)
  Comments: this gives TCP a data block for reliable transmission to the TCP on the other side of the connection. The timeout can be configured with this call whenever data are sent (see also CONNECTION.MAINTENANCE.CHANGE-TIMEOUT.TCP).

- SEND.SCTP:
  Pass 1 primitive / event: 'Send'
  Parameters: stream number; context (optional); socket (optional); unordered flag (optional); no-bundle flag (optional); payload protocol-id (optional); pr-policy (optional) pr-value (optional); sack-immediately flag (optional); key-id (optional)
  Comments: this gives SCTP a data block for transmission to the SCTP on the other side of the connection (SCTP association). The 'stream number' denotes the stream to be used. The 'context' number can later be used to refer to the correct message when an error is reported. The 'socket' can be used to state which path should be preferred, if there are multiple paths available (see also CONNECTION.MAINTENANCE.SETPRIMARY.SCTP). The data block can be delivered out-of-order if the 'unordered flag' is set. The 'no-bundle flag' can be set to indicate a preference to avoid bundling. The 'payload protocol-id' is a number that will, if

provided, be handed over to the receiving application. Using pr-policy and pr-value the level of reliability can be controlled. The sack-immediately flag can be used to indicate that the peer should not delay the sending of a SACK corresponding to the provided user message. If specified, the provided key-id is used for authenticating the user message.

- SEND.UDP(-Lite):
  Pass 1 primitive / event: 'SEND'
  Parameters: IP Address and Port Number of the destination endpoint (optional if connected).
  Comments: This provides a message for unreliable transmission using UDP(-Lite) to the specified transport address. IP address and Port may be omitted for connected UDP(-Lite) sockets. All CONNECTION.MAINTENANCE.SET_*.UDP(-Lite) primitives apply per message sent.

- RECEIVE.TCP:
  Pass 1 primitive / event: 'receive'.

- RECEIVE.SCTP:
  Pass 1 primitive / event: 'DATA ARRIVE' notification, followed by 'Receive'
  Parameters: stream number (optional)
  Returns: stream sequence number (optional), partial flag (optional)
  Comments: if the 'stream number' is provided, the call to receive only receives data on one particular stream. If a partial message arrives, this is indicated by the 'partial flag', and then the 'stream sequence number' must be provided such that an application can restore the correct order of data blocks that comprise an entire message. Additionally, a delivery number lets the application detect reordering.

- RECEIVE.UDP(-Lite):
  Pass 1 primitive / event: 'RECEIVE',
  Parameters: Buffer for received datagram.
  Comments: All CONNECTION.MAINTENANCE.GET_*.UDP(-Lite) primitives apply per message received.

- SENDFAILURE-EVENT.SCTP:
  Pass 1 primitive / event: 'SEND FAILURE' notification, optionally followed by 'Receive Unsent Message' or 'Receive Unacknowledged Message'
  Returns: cause code; context; unsent or unacknowledged message (optional)
  Comments: 'cause code' indicates the reason of the failure, and 'context' is the context number if such a number has been provided

in DATA.SEND.SCTP, for later use with 'Receive Unsent Message' or 'Receive Unacknowledged Message', respectively. These primitives can be used to retrieve the complete unsent or unacknowledged message if desired.

- SEND_FAILURE.UDP(-Lite):
  Pass 1 primitive / event: 'SEND'
  Comment: This may be used to probe for the effective PMTU when using in combination with the 'MAINTENANCE.SET_DF' primitive.

5. Pass 3

This section presents the superset of all transport service features in all protocols that were discussed in the preceding sections, based on the list of primitives in pass 2 but also on text in pass 1 to include features that can be configured in one protocol and are static properties in another (congestion control, for example). Again, some minor details are omitted for the sake of generalization -- e.g., TCP may provide various different IP options, but only source route is mandatory to implement, and this detail is not visible in the Pass 3 feature "Specify IP Options".

5.1. CONNECTION Related Transport Service Features

ESTABLISHMENT:
Active creation of a connection from one transport endpoint to one or more transport endpoints.

- Connect
  Protocols: TCP, SCTP, UDP(-Lite)

- Specify which IP Options must always be used
  Protocols: TCP

- Request multiple streams
  Protocols: SCTP

- Obtain multiple sockets
  Protocols: SCTP

- Disable MPTCP
  Protocols: MPTCP

- Specify which chunk types must always be authenticated
  Protocols: SCTP
  Comments: DATA, ACK etc. are different 'chunks' in SCTP; one or

more chunks may be included in a single packet.

- Indicate an Adaptation Layer (via an adaptation code point)
  Protocols: SCTP

AVAILABILITY:
Preparing to receive incoming connection requests.

- Listen, 1 specified local interface
  Protocols: TCP, SCTP, UDP(-Lite)

- Listen, N specified local interfaces
  Protocols: SCTP, UDP(-Lite)

- Listen, all local interfaces
  Protocols: TCP, SCTP, UDP(-Lite)

- Obtain requested number of streams
  Protocols: SCTP

- Specify which IP Options must always be used
  Protocols: TCP

- Disable MPTCP
  Protocols: MPTCP

- Specify which chunk types must always be authenticated
  Protocols: SCTP
  Comments: DATA, ACK etc. are different 'chunks' in SCTP; one or more chunks may be included in a single packet.

- Indicate an Adaptation Layer (via an adaptation code point)
  Protocols: SCTP

MAINTENANCE:
Adjustments made to an open connection, or notifications about it.
NOTE: all features except "set primary path" in this category apply to one out of multiple possible paths (identified via sockets) in SCTP, whereas TCP uses only one path (one socket).

- Change timeout for aborting connection (using retransmit limit or time value)
  Protocols: TCP, SCTP

- Control advertising timeout for aborting connection to remote endpoint
  Protocols: TCP

- Disable Nagle algorithm
  Protocols: TCP, SCTP
  Comments: This is not specified in [RFC4960] but in [RFC6458].

- Request an immediate heartbeat, returning success/failure
  Protocols: SCTP

- Set protocol parameters
  Protocols: SCTP
  SCTP parameters: RTO.Initial; RTO.Min; RTO.Max; Max.Burst; RTO.Alpha; RTO.Beta; Valid.Cookie.Life; Association.Max.Retrans; Path.Max.Retrans; Max.Init.Retransmits; HB.interval; HB.Max.Burst; PotentiallyFailed.Max.Retrans; Primary.Switchover.Max.Retrans; Remote.UDPEncapsPort
  Comments: as transport layer features from other protocols are added, it might make sense to separate out some of these parameters -- e.g., if a different protocol provides means to adjust the RTO calculation there could be a common feature for them called "adjust RTO calculation".

- Notification of Excessive Retransmissions (early warning below abortion threshold)
  Protocols: TCP

- Notification of ICMP error message arrival
  Protocols: TCP, UDP(-Lite)

- Obtain status (query or notification)
  Protocols: SCTP, MPTCP
  SCTP parameters: association connection state; socket list; socket reachability states; current receiver window size; current congestion window sizes; number of unacknowledged DATA chunks; number of DATA chunks pending receipt; primary path; most recent SRTT on primary path; RTO on primary path; SRTT and RTO on other destination addresses; socket becoming active / inactive
  MPTCP parameters: subflow-list (identified by source-IP; source-Port; destination-IP; destination-Port)

- Change authentication parameters
  Protocols: SCTP

- Obtain authentication information
  Protocols: SCTP

- Set primary path
  Protocols: SCTP

- Reset Stream
  Protocols: SCTP

- Notification of Stream Reset
  Protocols: STCP

- Reset Association
  Protocols: SCTP

- Notification of Association Reset
  Protocols: STCP

- Add Streams
  Protocols: SCTP

- Notification of Added Stream
  Protocols: STCP

- Set peer primary path
  Protocols: SCTP

- Specify DSCP field
  Protocols: TCP, SCTP, UDP(-Lite)

- Add subflow
  Protocols: MPTCP
  MPTCP Parameters: source-IP; source-Port; destination-IP;
  destination-Port

- Remove subflow
  Protocols: MPTCP
  MPTCP Parameters: source-IP; source-Port; destination-IP;
  destination-Port

- Add local address
  Protocols: SCTP

- Remove local address
  Protocols: SCTP

- Disable checksum when sending
  Protocols: UDP

- Disable checksum requirement when receiving
  Protocols: UDP

- Specify checksum coverage used by the sender
  Protocols: UDP-Lite

- Specify minimum checksum coverage required by receiver
  Protocols: UDP-Lite

- Specify DF field
  Protocols: UDP(-Lite)

- Specify TTL/Hop count field
  Protocols: UDP(-Lite)

- Obtain TTL/Hop count field
  Protocols: UDP(-Lite)

- Specify ECN field
  Protocols: UDP(-Lite)

- Obtain ECN field
  Protocols: UDP(-Lite)

- Specify IP Options
  Protocols: UDP(-Lite)

- Obtain IP Options
  Protocols: UDP(-Lite)

TERMINATION:
Gracefully or forcefully closing a connection, or being informed about this event happening.

- Close after reliably delivering all remaining data, causing an event informing the application on the other side
  Protocols: TCP, SCTP
  Comments: A TCP endpoint locally only closes the connection for sending; it may still receive data afterwards.

- Abort without delivering remaining data, causing an event informing the application on the other side
  Protocols: TCP, SCTP
  Comments: In SCTP a reason can optionally be given by the application on the aborting side, which can then be received by the application on the other side.

- Timeout event when data could not be delivered for too long
  Protocols: TCP, SCTP
  Comments: the timeout is configured with CONNECTION.MAINTENANCE "Change timeout for aborting connection (using retransmit limit or time value)".

5.2. DATA Transfer Related Transport Service Features

All features in this section refer to an existing connection, i.e. a connection that was either established or made available for receiving data. Reliable data transfer entails delay -- e.g. for the sender to wait until it can transmit data, or due to retransmission in case of packet loss.

5.2.1. Sending Data

All features in this section are provided by DATA.SEND from pass 2. DATA.SEND is given a data block from the application, which we here call a "message" if the beginning and end of the data block can be identified at the receiver, and "data" otherwise.

- Reliably transfer data, with congestion control
  Protocols: TCP

- Reliably transfer a message, with congestion control
  Protocols: SCTP

- Unreliably transfer a message, with congestion control
  Protocols: SCTP

- Unreliably transfer a message, without congestion control
  Protocols: UDP(-Lite)

- Configurable Message Reliability
  Protocols: SCTP

- Choice of stream
  Protocols: SCTP

- Choice of path (destination address)
  Protocols: SCTP

- Choice between unordered (potentially faster) or ordered delivery of messages
  Protocols: SCTP

- Request not to bundle messages
  Protocols: SCTP

- Specifying a "payload protocol-id" (handed over as such by the receiver)
  Protocols: SCTP

- Specifying a key id to be used to authenticate a message
  Protocols: SCTP

- Request not to delay the acknowledgement (SACK) of a message
  Protocols: SCTP

5.2.2. Receiving Data

All features in this section are provided by DATA.RECEIVE from pass 2. DATA.RECEIVE fills a buffer provided by the application, with what we here call a "message" if the beginning and end of the data block can be identified at the receiver, and "data" otherwise.

- Receive data
  Protocols: TCP

- Receive a message
  Protocols: SCTP, UDP(-Lite)

- Choice of stream to receive from
  Protocols: SCTP

- Information about partial message arrival
  Protocols: SCTP
  Comments: In SCTP, partial messages are combined with a stream sequence number so that the application can restore the correct order of data blocks an entire message consists of.

- Obtain a message delivery number
  Protocols: SCTP
  Comments: This number can let applications detect and, if desired, correct reordering.

5.2.3. Errors

This section describes sending failures that are associated with a specific call to DATA.SEND from pass 2.

- Notification of unsent messages
  Protocols: SCTP, UDP(-Lite)

- Notification of unacknowledged messages
  Protocols: SCTP

6. Acknowledgements

The authors would like to thank (in alphabetical order) Bob Briscoe, Gorry Fairhurst, David Hayes, Tom Jones, Karen Nielsen and Joe Touch for providing valuable feedback on this document. We especially thank to Christoph Paasch for providing input related to Multipath TCP. This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT). The views expressed are solely those of the author(s).

7. IANA Considerations

XX RFC ED - PLEASE REMOVE THIS SECTION XXX

This memo includes no request to IANA.

8. Security Considerations

Security will be considered in future versions of this document.

9. References

9.1. Normative References

[I-D.ietf-tsvwg-rfc5405bis]
Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", draft-ietf-tsvwg-rfc5405bis-07 (work in progress), November 2015.

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

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989,

<http://www.rfc-editor.org/info/rfc1122>.

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

[RFC5482] Eggert, L. and F. Gont, "TCP User Timeout Option", RFC 5482, DOI 10.17487/RFC5482, March 2009, <http://www.rfc-editor.org/info/rfc5482>.

9.2. Informative References

[FA16] Fairhurst, Ed., G., Trammell, Ed., B., and M. Kuehlewind, Ed., "Services provided by IETF transport protocols and congestion control mechanisms", draft-ietf-taps-transports-12.txt (work in progress), October 2016.

[FJ16] Fairhurst, G. and T. Jones, "Features of the User Datagram Protocol (UDP) and Lightweight UDP (UDP-Lite) Transport Protocols", draft-fairhurst-taps-transports-usage-udp-03 (work in progress), October 2016.

[RFC0854] Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, DOI 10.17487/RFC0854, May 1983, <http://www.rfc-editor.org/info/rfc854>.

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

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

[RFC3260] Grossman, D., "New Terminology and Clarifications for Diffserv", RFC 3260, DOI 10.17487/RFC3260, April 2002, <http://www.rfc-editor.org/info/rfc3260>.

[RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, DOI 10.17487/RFC3758, May 2004, <http://www.rfc-editor.org/info/rfc3758>.

[RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., Ed.,

and G. Fairhurst, Ed., "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, DOI 10.17487/RFC3828, July 2004, <http://www.rfc-editor.org/info/rfc3828>.

[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, <http://www.rfc-editor.org/info/rfc4895>.

[RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", RFC 5061, DOI 10.17487/RFC5061, September 2007, <http://www.rfc-editor.org/info/rfc5061>.

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

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

[RFC6182] Ford, A., Raiciu, C., Handley, M., Barre, S., and J. Iyengar, "Architectural Guidelines for Multipath TCP Development", RFC 6182, DOI 10.17487/RFC6182, March 2011, <http://www.rfc-editor.org/info/rfc6182>.

[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, <http://www.rfc-editor.org/info/rfc6458>.

[RFC6525] Stewart, R., Tuexen, M., and P. Lei, "Stream Control Transmission Protocol (SCTP) Stream Reconfiguration", RFC 6525, DOI 10.17487/RFC6525, February 2012, <http://www.rfc-editor.org/info/rfc6525>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <http://www.rfc-editor.org/info/rfc6824>.

[RFC6897] Scharf, M. and A. Ford, "Multipath TCP (MPTCP) Application Interface Considerations", RFC 6897, DOI 10.17487/RFC6897, March 2013, <http://www.rfc-editor.org/info/rfc6897>.

[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, <http://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, <http://www.rfc-editor.org/info/rfc7053>.

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

[RFC7496] Tuexen, M., Seggelmann, R., Stewart, R., and S. Loreto, "Additional Policies for the Partially Reliable Stream Control Transmission Protocol Extension", RFC 7496, DOI 10.17487/RFC7496, April 2015, <http://www.rfc-editor.org/info/rfc7496>.

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

[RFC7829] Nishida, Y., Natarajan, P., Caro, A., Amer, P., and K. Nielsen, "SCTP-PF: A Quick Failover Algorithm for the Stream Control Transmission Protocol", RFC 7829, DOI 10.17487/RFC7829, April 2016, <http://www.rfc-editor.org/info/rfc7829>.

Appendix A. Overview of RFCs used as input for pass 1

TCP: [RFC0793], [RFC1122], [RFC5482]
MPTCP: [RFC6182], [RFC6824], [RFC6897]
SCTP: RFCs without a socket API specification: [RFC3758], [RFC4895], [RFC4960], [RFC5061]. RFCs that include a socket API specification: [RFC6458], [RFC6525], [RFC6951], [RFC7053], [RFC7496] [RFC7829].
UDP(-Lite): See [FJ16]

Appendix B. How to contribute

This document is only concerned with transport service features that are explicitly exposed to applications via primitives. It also strictly follows RFC text: if a feature is truly relevant for an application, the RFCs better say so and in some way describe how to use and configure it. Thus, the approach to follow for contributing to this document is to identify the right RFCs, then analyze and process their text.

Experimental RFCs are excluded, and so are primitives that MAY be implemented (by the transport protocol). To be included, the minimum requirement level for a primitive to be implemented by a protocol is SHOULD. If [RFC2119]-style requirements levels are not used, primitives should be excluded when they are described in conjunction with statements like, e.g.: "some implementations also provide" or "an implementation may also". Briefly describe excluded primitives in a subsection called "excluded primitives".

Pass 1: Identify text that talks about primitives. An API specification, abstract or not, obviously describes primitives -- but note that we are not *only* interested in API specifications. The text describing the 'send' primitive in the API specified in [RFC0793], for instance, does not say that data transfer is reliable. TCP's reliability is clear, however, from this text in Section 1 of [RFC0793]: "The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks."

For the new pass 1 subsection about the protocol you're describing, it is recommendable to begin by copy+pasting all the relevant text parts from the relevant RFCs, then adjust terminology to match the terminology in Section 1 and adjust (shorten!) phrasing to match the general style of the document. Try to formulate everything as a primitive description to make the primitive description as complete as possible (e.g., the "SEND.TCP" primitive in pass 2 is explicitly described as reliably transferring data); if there is text that is relevant for the primitives presented in this pass but still does not fit directly under any primitive, use it as an introduction for your subsection. However, do note that document length is a concern and all the protocols and their services / features are already described in [FA16].

Pass 2: The main goal of this pass is unification of primitives. As input, use your own text from Pass 1, no exterior sources. If you find that something is missing there, fix the text in Pass 1. The list in pass 2 is not done by protocol ("first protocol X, here are

all the primitives; then protocol Y, here are all the primitives, ..") but by primitive ("primitive A, implemented this way in protocol X, this way in protocol Y, ..."). We want as many similar pass 2 primitives as possible. This can be achieved, for instance, by not always maintaining a 1:1 mapping between pass 1 and pass 2 primitives, renaming primitives etc. Please consider the primitives that are already there and try to make the ones of the protocol you are describing as much in line with the already existing ones as possible. In other words, we would rather have a primitive with new parameters than a new primitive that allows to send in a particular way.

Please make primitives fit within the already existing categories and subcategories. For each primitive, please follow the style:

- PRIMITIVENAME.PROTOCOL:
  Pass 1 primitive / event:
  Parameters:
  Returns:
  Comments:

The entries "Parameters", "Returns" and "Comments" may be skipped if a primitive has no parameters, no described return value or no comments seem necessary, respectively. Optional parameters must be followed by "(optional)". If a default value is known, provide it too.

Pass 3: the main point of this pass is to identify features that are the result of static properties of protocols, for which all protocols have to be listed together; this is then the final list of all available features. For this, we need a list of features per category (similar categories as in pass 2) along with the protocol supporting it. This should be primarily based on text from pass 2 as input, but text from pass 1 can also be used. Do not use external sources.

Appendix C. Revision information

XXX RFC-Ed please remove this section prior to publication.

-00 (from draft-welzl-taps-transports): this now covers TCP based on all TCP RFCs (this means: if you know of something in any TCP RFC that you think should be addressed, please speak up!) as well as SCTP, exclusively based on [RFC4960]. We decided to also incorporate [RFC6458] for SCTP, but this hasn't happened yet. Terminology made in line with [FA16]. Addressed comments by Karen Nielsen and Gorry Fairhurst; various other fixes. Appendices (TCP overview and how-to-

contribute) added.

-01: this now also covers MPTCP based on [RFC6182], [RFC6824] and [RFC6897].

-02: included UDP, UDP-Lite, and all extensions of SCTPs. This includes fixing the [RFC6458] omission from -00.

TODO: security considerations (see review in ML); the "how to contribute" section (which, at some point, should be updated to reflect how the document WAS created, not how it SHOULD BE created) still says "Experimental RFCs are excluded". This is wrong, and accordingly, Experimental RFCs must also be considered - thus, TFO (are there more Experimental ones for TCP?). Also, include LEDBAT. SCTP: DSCP and SCTP_NODELAY (equivalent to Nagle) are missing in pass 1 and 2. Are we missing more (DF, TTL, ..)? What about e.g. "notification of ICMP error message arrival"? Also consider draft-ietf-tsvwg-sctp-ndata.

Authors' Addresses

Michael Welzl
University of Oslo
PO Box 1080 Blindern
Oslo, N-0316
Norway

Phone: +47 22 85 24 20
Email: michawe@ifi.uio.no

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstrasse 39
Steinfurt 48565
Germany

Email: tuexen@fh-muenster.de

Naeem Khademi
University of Oslo
PO Box 1080 Blindern
Oslo, N-0316
Norway

Email: naeemk@ifi.uio.no