Internet Engineering Task Force (IETF)
Request for Comments: 8303
Category: Informational
ISSN: 2070-1721

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

February 2018

On the Usage of Transport Features Provided by IETF Transport Protocols

Abstract

This document describes how the transport protocols Transmission Control Protocol (TCP), MultiPath TCP (MPTCP), Stream Control Transmission Protocol (SCTP), User Datagram Protocol (UDP), and Lightweight User Datagram Protocol (UDP-Lite) expose services to applications and how an application can configure and use the features that make up these services. It also discusses the service provided by the Low Extra Delay Background Transport (LEDBAT) congestion control mechanism. The description results in a set of transport abstractions that can be exported in a transport services (TAPS) API.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8303.

Table of Contents

1. Introduction

This specification describes how transport protocols offer transport services, such that applications using them are no longer directly tied to a specific protocol. Breaking this strict connection can reduce the effort for an application programmer, yet attain greater transport flexibility by pushing complexity into an underlying transport services (TAPS) system.

This design process has started with a survey of the services provided by IETF transport protocols and congestion control mechanisms [RFC8095]. The present document and [RFC8304] complement this survey with an in-depth look at the defined interactions between applications and the following unicast transport protocols: Transmission Control Protocol (TCP), MultiPath TCP (MPTCP), Stream Control Transmission Protocol (SCTP), User Datagram Protocol (UDP), and Lightweight User Datagram Protocol (UDP-Lite). We also define a primitive to enable/disable and configure the Low Extra Delay Background Transport (LEDBAT) unicast congestion control mechanism. For UDP and UDP-Lite, the first step of the protocol analysis -- a discussion of relevant RFC text -- is documented in [RFC8304].

This snapshot in time of the IETF transport protocols is published as an RFC to document the analysis by the authors and the TAPS Working Group; this generates a set of transport abstractions that can be exported in a TAPS API. It provides the basis for the minimal set of transport services that end systems supporting TAPS should implement [TAPS-MINSET].

The list of primitives, events, and transport features in this document is strictly based on the parts of protocol specifications that describe what the protocol provides to an application using it and how the application interacts with it. 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 this multiplexing is achieved using port numbers. Port multiplexing is therefore assumed to be always provided and not discussed in this document.

Parts of a protocol that are explicitly stated as optional to implement are not covered. Interactions between the application and a transport protocol that are not directly related to the operation of the protocol are also not covered. For example, there are various ways for an application to use socket options to indicate its interest in receiving certain notifications [RFC6458]. However, for the purpose of identifying primitives, events, and transport features, the ability to enable or disable the reception of notifications is irrelevant. Similarly, "one-to-many style sockets"

[RFC6458] just affect the application programming style, not how the underlying protocol operates, and they are therefore not discussed here. The same is true for the ability to obtain the unchanged value of a parameter that an application has previously set (e.g., via "get" in get/set operations [RFC6458]).

The document presents a three-pass process to arrive at a list of transport features. In the first pass (pass 1), the relevant RFC text is discussed per protocol. In the second pass (pass 2), this discussion is used to derive a list of primitives and events that are uniformly categorized across protocols. Here, an attempt is made to present or -- where text describing primitives or events does not yet exist -- construct primitives or events in a slightly generalized form to highlight similarities. This is, for example, achieved by renaming primitives or events of protocols or by avoiding a strict 1:1 mapping between the primitives or events in the protocol specification and primitives or events in the list. Finally, the third pass (pass 3) presents transport features based on pass 2, identifying which protocols implement them.

In the list resulting from the second pass, some transport features are missing because they are implicit in some protocols, and they only become explicit when we consider the superset of all transport features offered by all protocols. For example, TCP always carries out congestion control; we have to consider it together with a protocol like UDP (which does not have congestion control) before we can consider congestion control as a transport feature. The complete list of transport features across all protocols is therefore only available after pass 3.

Some protocols are connection oriented. Connection-oriented protocols often use an initial call to a specific primitive to open a connection before communication can progress and require communication to be explicitly terminated by issuing another call to a 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. For this purpose, UDP is assumed to be used with "connected" sockets, i.e., sockets that are bound to a specific pair of addresses and ports [RFC8304].

2. Terminology

Transport Feature: a specific end-to-end feature that the transport layer provides to an application. Examples include confidentiality, reliable delivery, ordered delivery, message-versus-stream orientation, etc.

Transport Service: a set of transport 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 transport services using a specific framing and header format on the wire.

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

Transport Service Instance: an arrangement of transport protocols with a selected set of features and configuration parameters that implement 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. A primitive is related to one or more transport features.

Event: a primitive that is invoked by a transport endpoint.

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.

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). When presenting primitives, events, and parameters, the use of lower- and upper-case characters is made uniform for the sake of readability.

3.1. Primitives Provided by TCP

The initial TCP specification [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 of [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). A procedure for aborting the connection is used to avoid excessive retransmissions, and an application is able to control the threshold used to determine the condition for aborting; this threshold may be measured in time units or as a count of retransmission [RFC1122]. 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 allows the application to specify IP options such as Source Route, Record Route, or Timestamp [RFC1122]. It is not stated on which segments of a connection these options should be applied, but probably on all segments, as this is also stated in a specification given for the usage of the Source Route IP option (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.

Master Key Tuples (MKTs) for authentication can optionally be configured when calling 'Open' (Section 7.1 of [RFC5925]). When authentication is in use, complete TCP segments are authenticated, including the TCP IPv4 pseudoheader, TCP header, and TCP data.

TCP Fast Open (TFO) [RFC7413] allows applications to immediately hand over a message from the active open to the passive open side of a TCP connection together with the first message establishment packet (the SYN). This can be useful for applications that are sensitive to TCP's connection setup delay. [RFC7413] states that "TCP implementations MUST NOT use TFO by default, but only use TFO if requested explicitly by the application on a per-service-port basis." The size of the message sent with TFO cannot be more than TCP's maximum segment size (minus options used in the SYN). For the active open side, it is recommended to change or replace the connect() call in order to support a user data buffer argument [RFC7413]. For the passive open side, the application needs to enable the reception of Fast Open requests, e.g., via a new TCP_FASTOPEN setsockopt() socket option before listen(). The receiving application must be prepared to accept duplicates of the TFO message, as the first data written to a socket can be delivered more than once to the application on the remote host.

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'). Additionally, optional parameters allow the ability to indicate the preferred outgoing MKT (current_key) and/or the preferred incoming MKT (rnext_key) of a connection (Section 7.1 of [RFC5925]).

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. The application is also informed about the current_key and rnext_key information carried in a recently received segment via an optional parameter (Section 7.1 of [RFC5925]).

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 is executed when the user timeout (Section 3.9 of [RFC0793]) expires (see the definition of 'Open' in this section). 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 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). See Section 4.2.4.1 of [RFC1122].

Type-of-Service: Section 4.2.4.2 of the requirements for Internet hosts [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. The Differentiated Services (Diffserv) model [RFC2475] [RFC3260] replaces this field in the IP header, assigning the six most significant bits to carry the Differentiated Services Code Point (DSCP) field [RFC2474].

Nagle: The Nagle algorithm delays sending data for some time to increase the likelihood of sending a full-sized segment (Section 4.2.3.4 of [RFC1122]). 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'), there are three per-connection state variables that an application can adjust to control the operation of the 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').

Set/Get Authentication Parameters: The preferred outgoing MKT (current_key) and/or the preferred incoming MKT (rnext_key) of a connection can be configured. Information about current_key and rnext_key carried in a recently received segment can be retrieved (Section 7.1 of [RFC5925]).

3.1.1. Excluded Primitives or Parameters

The 'Open' primitive can be handed optional precedence or security/compartment information [RFC0793], but this was not included here because it is mostly irrelevant today [RFC7414].

The 'Status' primitive was not included because the initial TCP specification describes this primitive as "implementation dependent" and states that it "could be excluded without adverse effect" [RFC0793]. Moreover, while a data block containing specific information is described, it is also stated that not all of this information may always be available. While [RFC5925] states that 'Status' "SHOULD be augmented to allow the MKTs of a current or pending connection to be read (for confirmation)", the same

information is also available via 'Receive', which, following [RFC5925], "MUST be augmented" with that functionality. The 'Send' primitive includes an optional 'push' flag which, if set, requires data to be promptly transmitted to the receiver without delay [RFC0793]; the 'Receive' primitive described in can (under some conditions) yield the status of the 'push' flag. Because "push" functionality is optional to implement for both the 'Send' and 'Receive' primitives [RFC1122], this functionality is not included here. The requirements for Internet hosts [RFC1122] also introduce keep-alives to TCP, but these are optional to implement and hence not considered here. The same document also describes that "some TCP implementations have included a FLUSH call", indicating that this call is also optional to implement; therefore, it is not considered here.

3.2. Primitives Provided by MPTCP

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 as follows 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, there are some constraints on the API exposed by MPTCP, as stated in [RFC6182]:

> 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 transport 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, the MPTCP RFCs [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: "An application should be able to request to turn on or turn off the usage of MPTCP" [RFC6897]. This functionality can be provided through a socket option called 'tcp_multipath_enable'. Further, MPTCP must be disabled in case the application is binding to a specific address [RFC6897].

Send/Receive: The sending and receiving of data does not require any changes to the application when MPTCP is being used [RFC6824]. 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 [RFC6824], which states: "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. The MPTCP API Considerations RFC [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

TCP has a number of limitations that SCTP removes (Section 1.1 of [RFC4960]). The following three removed limitations directly translate into transport features that are visible to an application using SCTP: 1) it allows for preservation of message delimiters; 2) it does not provide in-order or reliable delivery unless the application wants that; 3) multihoming is supported. In SCTP, connections are called "associations" and they can be between not only two (as in TCP) but multiple addresses at each endpoint.

Section 10 of the SCTP base protocol specification [RFC4960] specifies the interaction with the application (which SCTP 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 sockets API is described in [RFC6458]. This covers the functionality of the base protocol [RFC4960] and some of its extensions [RFC3758] [RFC4895] [RFC5061]. For other protocol extensions [RFC6525] [RFC6951] [RFC7053] [RFC7496] [RFC7829]

[RFC8260], the corresponding extensions of the sockets API are specified in these protocol specifications. The functionality exposed to the ULP through all these APIs is considered here.

The abstract API contains a 'SetProtocolParameters' primitive that allows elements of a parameter list [RFC4960] to be adjusted; it is stated that SCTP implementations "may allow ULP to customize some of these protocol parameters", indicating that none of the elements of this parameter list are mandatory to make ULP configurable. Thus, we only consider the parameters in the abstract API that are also covered in one of the other RFCs listed above, which leads us to exclude the parameters 'RTO.Alpha', 'RTO.Beta', and 'HB.Max.Burst'. For clarity, we also replace 'SetProtocolParameters' itself with primitives that adjust parameters or groups of parameters that fit together.

Initialize: Initialize creates a local SCTP instance that it binds to a set of local addresses (and, if provided, a local port number) [RFC4960]. Initialize needs to be called only once per set of local addresses. A number of per-association initialization parameters can be used when an association is created, but before it is connected (via the primitive 'Associate' below): the maximum number of inbound streams the application is prepared to support, the maximum number of attempts to be made when sending the INIT (the first message of association establishment), and the maximum retransmission timeout (RTO) value to use when attempting an INIT [RFC6458]. At this point, before connecting, an application can also enable UDP encapsulation by configuring the remote UDP encapsulation port number [RFC6951].

Associate: This creates an association (the SCTP equivalent of a connection) that connects the local SCTP instance and a remote SCTP instance. To identify the remote endpoint, it can be given one or multiple (using "connectx") sockets (Section 9.9 of [RFC6458]). 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 the 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 negotiated outgoing streams. An optional parameter of 32 bits, the adaptation layer indication, can be provided [RFC5061]. If authenticated chunks are used, the chunk types required to be sent authenticated by the peer can be provided [RFC4895]. An 'SCTP_Cant_Str_Assoc' notification is used to inform the

application of a failure to create an association [RFC6458]. An application could use sendto() or sendmsg() to implicitly set up an association, thereby handing over a message that SCTP might send during the association setup phase [RFC6458]. Note that this mechanism is different from TCP's TFO mechanism: the message would arrive only once, after at least one RTT, as it is sent together with the third message exchanged during association setup, the COOKIE-ECHO chunk).

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 control of the partial reliability extension [RFC3758] [RFC7496]. An optional maximum lifetime 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 [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 authenticated chunks are used, the key identifier for authenticating DATA chunks can be provided [RFC4895].

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.

Shutdown: This primitive gracefully closes an association, reliably delivering any data that has already been handed over to SCTP. A parameter lets the application control whether further receive or send operations or both are disabled when the call is issued. 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.

Configure Max. Retransmissions of an Association: The parameter 'Association.Max.Retrans' [RFC4960] (called "sasoc_maxrxt" in the SCTP sockets API extensions [RFC6458]) allows the configuration of the number of unsuccessful retransmissions after which an entire association is considered as failed; this should invoke a 'Communication Lost' notification.

Set Primary: This allows the ability 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.

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 for which address to use as the primary address [RFC5061].

Configure Path Switchover: The abstract API contains a primitive called 'Set Failure Threshold' [RFC4960]. This configures the parameter 'Path.Max.Retrans', which determines after how many retransmissions a particular transport address is considered as unreachable. If there are more transport addresses available in an association, reaching this limit will invoke a path switchover. An extension called "SCTP-PF" adds a concept of "Potentially Failed (PF)" paths to this method [RFC7829]. When a path is in PF state, SCTP will not entirely give up sending on that path, but it will preferably send data on other active paths if such paths are available. Entering the PF state is done upon exceeding a configured maximum number of retransmissions. Thus, for all paths where this mechanism is used, there are two configurable error thresholds: one to decide that a path is in PF state, and one to decide that the transport address is unreachable.

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 [RFC4895].

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

Status: The 'Status' primitive returns a data block with information about a specified association, containing: an association connection state; a destination transport address list; destination transport address reachability states; current local and peer receiver window sizes; current local congestion window sizes; number of unacknowledged DATA chunks; number of DATA chunks pending receipt; a primary path; the most recent Smoothed Round-Trip Time (SRTT) on a primary path; RTO on a primary path; SRTT and RTO on other destination addresses [RFC4960]; and an MTU per path [RFC6458].

Enable/Disable Interleaving: This allows the negotiation of user message interleaving support for future associations to be enabled or disabled. For existing associations, it is possible to query whether user message interleaving support was negotiated or not on a particular association [RFC8260].

Set Stream Scheduler: This allows the ability to select a stream scheduler per association, with a choice of: First-Come, First-Served; Round-Robin; Round-Robin per Packet; Priority-Based; Fair Bandwidth; and Weighted Fair Queuing [RFC8260].

Configure Stream Scheduler: This allows the ability to change a parameter per stream for the schedulers: a priority value for the Priority-Based scheduler and a weight for the Weighted Fair Queuing scheduler.

Enable/Disable NoDelay: This turns on/off any Nagle-like algorithm for an association [RFC6458].

Configure Send Buffer Size: This controls the amount of data SCTP may have waiting in internal buffers to be sent or retransmitted [RFC6458].

Configure Receive Buffer Size: This sets the receive buffer size in octets, thereby controlling the receiver window for an association [RFC6458].

Configure Message Fragmentation: If a user message causes an SCTP packet to exceed the maximum fragmentation size (which can be provided by the application and is otherwise the Path MTU (PMTU) size), then the message will be fragmented by SCTP. Disabling message fragmentation will produce an error instead of fragmenting the message [RFC6458].

Configure Path MTU Discovery: Path MTU Discovery (PMTUD) can be enabled or disabled per peer address of an association (Section 8.1.12 of [RFC6458]). When it is enabled, the current Path MTU value can be obtained. When it is disabled, the Path MTU to be used can be controlled by the application.

Configure Delayed SACK Timer: The time before sending a SACK can be adjusted; delaying SACKs can be disabled; and the number of packets that must be received before a SACK is sent without waiting for the delay timer to expire can be configured [RFC6458].

Set Cookie Life Value: The cookie life value can be adjusted (Section 8.1.2 of [RFC6458]). 'Valid.Cookie.Life' is also one of the parameters that is potentially adjustable with 'SetProtocolParameters' [RFC4960].

Set Maximum Burst: The maximum burst of packets that can be emitted by a particular association (default 4, and values above 4 are optional to implement) can be adjusted (Section 8.1.2 of [RFC6458]). 'Max.Burst' is also one of the parameters that is potentially adjustable with 'SetProtocolParameters' [RFC4960].

Configure RTO Calculation: The abstract API contains the following adjustable parameters: 'RTO.Initial'; 'RTO.Min'; 'RTO.Max'; 'RTO.Alpha'; and 'RTO.Beta'. Only the initial, minimum and maximum RTOs are also described as configurable in the SCTP sockets API extensions [RFC6458].

Set DSCP Value: The DSCP value can be set per peer address of an association (Section 8.1.12 of [RFC6458]).

Set IPv6 Flow Label: The flow label field can be set per peer address of an association (Section 8.1.12 of [RFC6458]).

Set Partial Delivery Point: This allows the ability to specify the size of a message where partial delivery will be invoked. Setting this to a lower value will cause partial deliveries to happen more often [RFC6458].

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). If interleaving is supported by both endpoints, this information is also included in this notification.

Restart Notification: When SCTP has detected that the peer has restarted, this notification is passed to the upper layer [RFC6458].

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. This can also inform the sender that a part of the message has been successfully delivered.

Network Status Change Notification: This informs the application about a socket becoming active/inactive [RFC4960] or "Potentially Failed" [RFC7829].

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 association.

Authentication Notification: When SCTP wants to notify the upper layer regarding the key management related to authenticated chunks [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 [RFC5061] [RFC6458].

Stream Reset Notification: When SCTP completes the procedure for resetting streams [RFC6525], this notification is passed to the upper layer, informing it about the result.

Association Reset Notification: When SCTP completes the association reset procedure [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 [RFC6525], this notification is passed to the upper layer, informing it about the result.

Sender Dry Notification: When SCTP has no more user data to send or retransmit on a particular association, this notification is passed to the upper layer [RFC6458].

Partial Delivery Aborted Notification: When a receiver has begun to receive parts of a user message but the delivery of this message is then aborted, this notification is passed to the upper layer (Section 6.1.7 of [RFC6458]).

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. 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 feature. The 'Shutdown' event informs an application that the peer has sent a SHUTDOWN, and hence no further data should be sent on this socket (Section 6.1 of [RFC6458]). However, if an application would try to send data on the socket, it would get an error message anyway; thus, this event is classified as "just affecting the application programming style, not how the underlying protocol operates" and is not included here.

3.4. Primitives Provided by UDP and UDP-Lite

The set of pass 1 primitives for UDP and UDP-Lite is documented in [RFC8304].

3.5. The Service of LEDBAT

The service of the LEDBAT congestion control mechanism is described as follows:

> LEDBAT is designed for use by background bulk-transfer applications to be no more aggressive than standard TCP congestion control (as specified in RFC 5681) and to yield in the presence of competing flows, thus limiting interference with the network performance of competing flows [RFC6817].

LEDBAT does not have any primitives, as LEDBAT is not a transport protocol. According to its specification [RFC6817]:

> LEDBAT can be used as part of a transport protocol or as part of an application, as long as the data transmission mechanisms are capable of carrying timestamps and acknowledging data frequently. LEDBAT can be used with TCP, Stream Control Transmission Protocol (SCTP), and Datagram Congestion Control Protocol (DCCP), with appropriate extensions where necessary; and it can be used with proprietary application protocols, such as those built on top of UDP for peer-to-peer (P2P) applications.

At the time of writing, the appropriate extensions for TCP, SCTP, or DCCP do not exist.

A number of configurable parameters exist in the LEDBAT specification: TARGET, which is the queuing delay target at which LEDBAT tries to operate, must be set to 100 ms or less. 'allowed_increase' (should be 1, must be greater than 0) limits the speed at which LEDBAT increases its rate. 'gain', which according to [RFC6817] "MUST be set to 1 or less" to avoid a faster ramp-up than TCP Reno, determines how quickly the sender responds to changes in queueing delay. Implementations may divide 'gain' into two parameters: one for increase and a possibly larger one for decrease. We call these parameters 'Gain_Inc' and 'Gain_Dec' here. 'Base_History' is the size of the list of measured base delays, and, according to [RFC6817], "SHOULD be 10". This list can be filtered using a 'Filter' function, which is not prescribed [RFC6817], that yields a list of size 'Current_Filter'. The initial and minimum congestion windows, 'Init_CWND' and 'Min_CWND', should both be 2.

Regarding which of these parameters should be under control of an application, the possible range goes from exposing nothing on the one hand to considering everything that is not prescribed with a "MUST" in the specification as a parameter on the other hand. Function implementations are not provided as a parameter to any of the transport protocols discussed here; hence, we do not regard the

'Filter' function as a parameter. However, to avoid unnecessarily limiting future implementations, we consider all other parameters above as tunable parameters that should be exposed.

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. 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 and lets the application control whether further receive or send operations, or both, are disabled [RFC6458]. This is not described in the same way for TCP [RFC0793], but these details play no significant role for the primitives provided by either TCP or SCTP (for the sake of being generic, it could be assumed that both receive and send operations are disabled in both cases).

The TCP 'Send' and 'Receive' primitives include usage of an 'urgent' parameter. This parameter controls a mechanism that is required to implement the "synch signal" used by telnet [RFC0854], but [RFC6093] states that "new applications SHOULD NOT employ the TCP urgent mechanism." Because pass 2 is meant as a basis for the creation of future systems, the "urgent" mechanism is excluded. This also concerns the notification 'Urgent Pointer Advance' in the 'Error_Report' (Section 4.2.4.1 of [RFC1122]).

Since LEDBAT is a congestion control mechanism and not a protocol, it is not currently defined when to enable/disable or configure the mechanism. For instance, it could be a one-time choice upon connection establishment or when listening for incoming connections, in which case it should be categorized under CONNECTION.ESTABLISHMENT or CONNECTION.AVAILABILITY, respectively. To avoid unnecessarily

limiting future implementations, it was decided to place it under CONNECTION.MAINTENANCE, with all parameters that are described in the specification [RFC6817] made configurable.

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 [RFC8085].

- 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); MKT configuration (optional); and user message (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. 'MKT configuration' refers to the ability to configure MKTs for authentication. The user message may be transmitted to the peer application immediately upon reception of the TCP SYN packet. To benefit from the lower latency this provides as part of the experimental TFO mechanism, its length must be at most the TCP's maximum segment size (minus TCP options used in the SYN). The message may also be delivered more than once to the application on the remote host.

- CONNECT.SCTP:

  Pass 1 primitive/event: 'Initialize', followed by 'Enable/Disable Interleaving' (optional), followed by 'Associate'

  Parameters: list of local SCTP port number / IP address pairs ('Initialize'); one or several sockets (identifying the peer); outbound stream count; maximum allowed inbound stream count; adaptation layer indication (optional); chunk types required to be authenticated (optional); request interleaving on/off; maximum

number of INIT attempts (optional); maximum init. RTO for INIT (optional); user message (optional); and remote UDP port number (optional)

Returns: socket list or failure

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. The user message may be transmitted to the peer application immediately upon reception of the packet containing the COOKIE-ECHO chunk. To benefit from the lower latency this provides, its length must be limited such that it fits into the packet containing the COOKIE-ECHO chunk. If a remote UDP port number is provided, SCTP packets will be encapsulated in UDP.

- CONNECT.MPTCP:

  This is similar to CONNECT.TCP except for one additional boolean parameter that allows the ability 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); and 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); buffer to receive a user message (optional); and MKT configuration (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'. If a buffer is provided to receive a user message, a user message can be received from a TFO-enabled sender before the TCP's connection handshake is completed. This message may arrive multiple times. 'MKT configuration' refers to the ability to configure MKTs for authentication.

- LISTEN.SCTP:

  Pass 1 primitive/event: 'Initialize', followed by the 'Communication Up' or 'Restart' notification and possibly the '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; and interleaving supported on both sides yes/no

  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 the ability 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); and 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); and boolean changeable (optional, default true)

  Comments: when sending data, an application can adjust the connection's timeout value (the time after which the connection will be aborted if data could not be delivered). If 'uto_enabled' is true, the '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 the 'timeout value' is used.

- CHANGE_TIMEOUT.SCTP:

  Pass 1 primitive/event: 'Change Heartbeat' combined with 'Configure Max. Retransmissions of an Association'

  Parameters: 'Change Heartbeat': heartbeat frequency and 'Configure Max. Retransmissions of an Association': Association.Max.Retrans

  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 transmissions of any packets (including heartbeats) the

association will be terminated; thus, these two primitives/ parameters together can yield a similar behavior for SCTP associations as CHANGE_TIMEOUT.TCP does for TCP connections.

- DISABLE_NAGLE.TCP:

  Pass 1 primitive/event: not specified

  Parameters: one boolean value

  Comments: the Nagle algorithm delays data transmission to increase the chance of sending a full-sized segment. An application must be able to disable this algorithm for a connection.

- DISABLE_NAGLE.SCTP:

  Pass 1 primitive/event: 'Enable/Disable NoDelay'

  Parameters: one boolean value

  Comments: Nagle-like algorithms delay data transmission to increase the chance of sending a full-sized packet.

- REQUEST_HEARTBEAT.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.

- ADD_PATH.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_PATH.SCTP:

  Pass 1 primitive/event: 'Change Local Address / Set Peer Primary'

  Parameters: local IP address

- REM_PATH.MPTCP:

  Pass 1 primitive/event: not specified

  Parameters: local IP address; local port number; remote IP address; and 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_PATH.SCTP:

  Pass 1 primitive/event: 'Change Local Address / Set Peer Primary'

  Parameters: local IP address

- SET_PRIMARY.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.

- SET_PEER_PRIMARY.SCTP:

  Pass 1 primitive/event: 'Change Local Address / Set Peer Primary'

  Parameters: local IP address

  Comments: this is only advisory for the peer.

- CONFIG_SWITCHOVER.SCTP:

  Pass 1 primitive/event: 'Configure Path Switchover'

  Parameters: primary max retrans (number of retransmissions after which a path is considered inactive) and PF max retrans (number of retransmissions after which a path is considered to be "Potentially Failed", and others will be preferably used) (optional)

- STATUS.SCTP:

  Pass 1 primitive/event: 'Status', 'Enable/Disable Interleaving', and 'Network Status Change' notification

  Returns: data block with information about a specified association, containing: association connection state; destination transport address list; destination transport address reachability states; current local and peer receiver window sizes; current local 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; MTU per path; and interleaving supported yes/no

  Comments: the 'Network Status Change' notification informs the application about a socket becoming active/inactive; this only affects the programming style, as the same information is also available via 'Status'.

- 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.

- SET_DSCP.SCTP:

  Pass 1 primitive/event: 'Set DSCP value'

  Parameters: DSCP value

  Comments: this allows an application to change the DSCP value for outgoing packets on a path.

- 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 [RFC8085] provide current guidance on using this value with UDP.

- ERROR.TCP:

  Pass 1 primitive/event: 'Error_Report'

  Returns: reason (encoding not specified) and 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 and 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.

- SET_AUTH.TCP:

  Pass 1 primitive/event: not specified

  Parameters: current_key and rnext_key

  Comments: current_key and rnext_key are the preferred outgoing MKT and the preferred incoming MKT, respectively, for a segment that is sent on the connection.

- SET_AUTH.SCTP:

  Pass 1 primitive/event: 'Set/Get Authentication Parameters'

  Parameters: key_id; key; and hmac_id

- GET_AUTH.TCP:

  Pass 1 primitive/event: not specified

  Parameters: current_key and rnext_key

  Comments: current_key and rnext_key are the preferred outgoing MKT and the preferred incoming MKT, respectively, that were carried on a recently received segment.

- GET_AUTH.SCTP:

  Pass 1 primitive/event: 'Set/Get Authentication Parameters'

  Parameters: key_id and chunk_list

- RESET_STREAM.SCTP:

  Pass 1 primitive/event: 'Add/Reset Streams, Reset Association'

  Parameters: sid and direction

- RESET_STREAM-EVENT.SCTP:

  Pass 1 primitive/event: 'Stream Reset' notification

  Parameters: information about the result of RESET_STREAM.SCTP

  Comments: this is issued when the procedure for resetting streams has completed.

- RESET_ASSOC.SCTP:

  Pass 1 primitive/event: 'Add/Reset Streams, Reset Association'

  Parameters: information related to the extension, as defined in [RFC3260]

- RESET_ASSOC-EVENT.SCTP:

  Pass 1 primitive/event: 'Association Reset' notification

  Parameters: information about the result of RESET_ASSOC.SCTP

  Comments: this is issued when the procedure for resetting an association has completed.

- ADD_STREAM.SCTP:

  Pass 1 primitive/event: 'Add/Reset Streams, Reset Association'

  Parameters: number of outgoing and incoming streams to be added

- ADD_STREAM-EVENT.SCTP:

  Pass 1 primitive/event: 'Stream Change' notification

  Parameters: information about the result of ADD_STREAM.SCTP

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

- SET_STREAM_SCHEDULER.SCTP:

  Pass 1 primitive/event: 'Set Stream Scheduler'

  Parameters: scheduler identifier

  Comments: choice of First-Come, First-Served; Round-Robin; Round-Robin per Packet; Priority-Based; Fair Bandwidth; and Weighted Fair Queuing.

- CONFIGURE_STREAM_SCHEDULER.SCTP:

  Pass 1 primitive/event: 'Configure Stream Scheduler'

  Parameters: priority

  Comments: the priority value only applies when Priority-Based or Weighted Fair Queuing scheduling is chosen with SET_STREAM_SCHEDULER.SCTP. The meaning of the parameter differs between these two schedulers, but in both cases, it realizes some form of prioritization regarding how bandwidth is divided among streams.

- SET_FLOWLABEL.SCTP:

  Pass 1 primitive/event: 'Set IPv6 Flow Label'

  Parameters: flow label

  Comments: this allows an application to change the IPv6 header's flow label field for outgoing packets on a path.

- AUTHENTICATION_NOTIFICATION-EVENT.SCTP:

  Pass 1 primitive/event: 'Authentication' notification

  Returns: information regarding key management

- CONFIG_SEND_BUFFER.SCTP:

  Pass 1 primitive/event: 'Configure Send Buffer Size'

  Parameters: size value in octets

- CONFIG_RECEIVE_BUFFER.SCTP:

  Pass 1 primitive/event: 'Configure Receive Buffer Size'

  Parameters: size value in octets

  Comments: this controls the receiver window.

- CONFIG_FRAGMENTATION.SCTP:

  Pass 1 primitive/event: 'Configure Message Fragmentation'

  Parameters: one boolean value (enable/disable) and maximum fragmentation size (optional; default: PMTU)

  Comments: if fragmentation is enabled, messages exceeding the maximum fragmentation size will be fragmented. If fragmentation is disabled, trying to send a message that exceeds the maximum fragmentation size will produce an error.

- CONFIG_PMTUD.SCTP:

  Pass 1 primitive/event: 'Configure Path MTU Discovery'

  Parameters: one boolean value (PMTUD on/off) and PMTU value (optional)

  Returns: PMTU value

  Comments: this returns a meaningful PMTU value when PMTUD is enabled (the boolean is true), and the PMTU value can be set if PMTUD is disabled (the boolean is false).

- CONFIG_DELAYED_SACK.SCTP:

  Pass 1 primitive/event: 'Configure Delayed SACK Timer'

  Parameters: one boolean value (delayed SACK on/off); timer value (optional); and number of packets to wait for (default 2)

  Comments: if delayed SACK is enabled, SCTP will send a SACK either upon receiving the provided number of packets or when the timer expires, whatever occurs first.

- CONFIG_RTO.SCTP:

  Pass 1 primitive/event: 'Configure RTO Calculation'

  Parameters: init (optional); min (optional); and max (optional)

  Comments: this adjusts the initial, minimum, and maximum RTO values.

- SET_COOKIE_LIFE.SCTP:

  Pass 1 primitive/event: 'Set Cookie Life Value'

  Parameters: cookie life value

- SET_MAX_BURST.SCTP:

  Pass 1 primitive/event: 'Set Maximum Burst'

  Parameters: max burst value

  Comments: not all implementations allow values above the default of 4.

- SET_PARTIAL_DELIVERY_POINT.SCTP:

  Pass 1 primitive/event: 'Set Partial Delivery Point'

  Parameters: partial delivery point (integer)

  Comments: this parameter must be smaller or equal to the socket receive buffer size.

- SET_CHECKSUM_ENABLED.UDP:

  Pass 1 primitive/event: 'Checksum_Enabled'

  Parameters: 0 when zero checksum is used at sender, 1 for checksum at sender (default)

- SET_CHECKSUM_REQUIRED.UDP:

  Pass 1 primitive/event: 'Require_Checksum'

  Parameter: 0 to allow zero checksum, 1 when a non-zero checksum is required (default) at the receiver

- 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) in the IPv4 header, 1 when DF is set

- GET_MMS_S.UDP(-Lite):

  Pass 1 primitive event: 'Get_MM_S'

  Comments: this retrieves the maximum transport-message size that may be sent using a non-fragmented IP packet from the configured interface.

- GET_MMS_R.UDP(-Lite):

  Pass 1 primitive event: 'Get_MMS_R'

  Comments: this retrieves the maximum transport-message size that may be received from the configured interface.

- 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 IPv4 TTL of the 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 Explicit Congestion Notification (ECN) code point field for outgoing UDP(-Lite) datagrams. It defaults to sending '00'.

- GET_ECN.UDP(-Lite):

  Pass 1 primitive/event: 'Get_ECN'

  Parameters: ECN value

  Comments: this allows a UDP(-Lite) application to read the ECN code point 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 Timestamp 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.

- CONFIGURE.LEDBAT:

  Pass 1 primitive/event: N/A

  Parameters: enable (boolean); target; allowed_increase; gain_inc; gain_dec; base_history; current_filter; init_cwnd; and min_cwnd

  Comments: 'enable' is a newly invented parameter that enables or disables the whole LEDBAT service.

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.

- 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.

- ABORT.UDP(-Lite):

  Pass 1 primitive event: 'Close'

  Comments: this terminates a connection without delivering remaining data. No further UDP(-Lite) datagrams are sent/received for this transport service instance.

- 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 the 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. Note that CONNECT.TCP and LISTEN.TCP in the ESTABLISHMENT and AVAILABILITY categories also allow to transfer data (an optional user message) before the connection is fully established.

- SEND.TCP:

  Pass 1 primitive/event: 'Send'

  Parameters: timeout (optional); current_key (optional); and rnext_key (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 (see also CONNECTION.MAINTENANCE.CHANGE_TIMEOUT.TCP). 'current_key' and 'rnext_key' are authentication parameters that can be configured with this call (see also CONNECTION.MAINTENANCE.SET_AUTH.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); and 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. The IP address and port number 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'

  Parameters: current_key (optional) and rnext_key (optional)

  Comments: 'current_key' and 'rnext_key' are authentication parameters that can be read with this call (see also CONNECTION.MAINTENANCE.GET_AUTH.TCP).

- RECEIVE.SCTP:

  Pass 1 primitive/event: 'Data Arrive' notification, followed by 'Receive'

  Parameters: stream number (optional)

  Returns: stream sequence number (optional) and 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.

- 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; and 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 unsent or unacknowledged message (or part of the message, in case a part was delivered) if desired.

- SEND_FAILURE.UDP(-Lite):

  Pass 1 primitive/event: 'Send'

  Comments: this may be used to probe for the effective PMTU when using in combination with the 'MAINTENANCE.SET_DF' primitive.

- SENDER_DRY-EVENT.SCTP:

  Pass 1 primitive/event: 'Sender Dry' notification

  Comments: this informs the application that the stack has no more user data to send.

- PARTIAL_DELIVERY_ABORTED-EVENT.SCTP:

  Pass 1 primitive/event: 'Partial Delivery Aborted' notification

  Comments: this informs the receiver of a partial message that the further delivery of the message has been aborted.

5. Pass 3

This section presents the superset of all transport 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 transport 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 transport feature "Specify IP options". As before, "UDP(-Lite)" represents both UDP and UDP-Lite, and "TCP" refers to both TCP and MPTCP.

5.1. CONNECTION-Related Transport Features

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

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

- Specify which IP options must always be used
  Protocols: TCP and UDP(-Lite)

- Request multiple streams
  Protocols: SCTP

- Limit the number of inbound streams
  Protocols: SCTP

- Specify number of attempts and/or timeout for the first establishment message
  Protocols: TCP and SCTP

- Obtain multiple sockets
  Protocols: SCTP

- Disable MPTCP
  Protocols: MPTCP

- Configure authentication
  Protocols: TCP and SCTP
  Comments: with TCP, this allows the configuration of MKTs. In SCTP, this allows the specification of which chunk types must always be authenticated. 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

- Request to negotiate interleaving of user messages
  Protocols: SCTP

- Hand over a message to reliably transfer (possibly multiple times) before connection establishment
  Protocols: TCP

- Hand over a message to reliably transfer during connection establishment
  Protocols: SCTP

- Enable UDP encapsulation with a specified remote UDP port number
  Protocols: SCTP

AVAILABILITY:

Preparing to receive incoming connection requests.

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

- Listen, N specified local interfaces
  Protocols: SCTP

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

- Obtain requested number of streams
  Protocols: SCTP

- Limit the number of inbound streams
  Protocols: SCTP

- Specify which IP options must always be used
  Protocols: TCP and UDP(-Lite)

- Disable MPTCP
  Protocols: MPTCP

- Configure authentication
  Protocols: TCP and SCTP
  Comments: with TCP, this allows the configuration of MKTs. In SCTP, this allows the specification of which chunk types must always be authenticated. 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.

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

- Suggest timeout to the peer
  Protocols: TCP

- Disable Nagle algorithm
  Protocols: TCP and SCTP

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

- Notification of excessive retransmissions (early warning below abortion threshold)
  Protocols: TCP

- Add path
  Protocols: MPTCP and SCTP
  MPTCP Parameters: source-IP; source-Port; destination-IP; and destination-Port
  SCTP Parameters: local IP address

- Remove path
  Protocols: MPTCP and SCTP
  MPTCP Parameters: source-IP; source-Port; destination-IP; and destination-Port
  SCTP Parameters: local IP address

- Set primary path
  Protocols: SCTP

- Suggest primary path to the peer
  Protocols: SCTP

- Configure Path Switchover
  Protocols: SCTP

- Obtain status (query or notification)
  Protocols: SCTP and MPTCP
  SCTP parameters: association connection state; destination transport address list; destination transport address reachability states; current local and peer receiver window sizes; current local 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; MTU per path; and interleaving supported yes/no
  MPTCP parameters: subflow-list (identified by source-IP; source-Port; destination-IP; and destination-Port)

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

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

- Change authentication parameters
  Protocols: TCP and SCTP

- Obtain authentication information
  Protocols: TCP and 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

- Choose a scheduler to operate between streams of an association
  Protocols: SCTP

- Configure priority or weight for a scheduler
  Protocols: SCTP

- Specify IPv6 flow label field
  Protocols: SCTP

- Configure send buffer size
  Protocols: SCTP

- Configure receive buffer (and rwnd) size
  Protocols: SCTP

- Configure message fragmentation
  Protocols: SCTP

- Configure PMTUD
  Protocols: SCTP

- Configure delayed SACK timer
  Protocols: SCTP

- Set Cookie life value
  Protocols: SCTP

- Set maximum burst
  Protocols: SCTP

- Configure size where messages are broken up for partial delivery
  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)

- Get max. transport-message size that may be sent using a non-fragmented IP packet from the configured interface
  Protocols: UDP(-Lite)

- Get max. transport-message size that may be received from the configured interface
  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)

- Enable and configure "Low Extra Delay Background Transfer"
  Protocols: A protocol implementing the LEDBAT congestion control mechanism

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 and 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 that informs the application on the other side
  Protocols: TCP and 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.

- Abort without delivering remaining data, not causing an event that informs the application on the other side
  Protocols: UDP(-Lite)

- Timeout event when data could not be delivered for too long
  Protocols: TCP and 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 Features

All transport features in this section refer to an existing connection, i.e., a connection that was either established or made available for receiving data. Note that TCP allows the transfer of data (a single optional user message, possibly arriving multiple times) before the connection is fully established. 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 transport features in this section are provided by DATA.SEND from pass 2. DATA.SEND is given a data block from the application, which here we 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

- Ordered message delivery (potentially slower than unordered)
  Protocols: SCTP

- Unordered message delivery (potentially faster than ordered)
  Protocols: SCTP and UDP(-Lite)

- Request not to bundle messages
  Protocols: SCTP

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

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

- Receive data (with no message delimiting)
  Protocols: TCP

- Receive a message
  Protocols: SCTP and 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.

5.2.3. Errors

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

- Notification of an unsent (part of a) message
  Protocols: SCTP and UDP(-Lite)

- Notification of an unacknowledged (part of a) message
  Protocols: SCTP

- Notification that the stack has no more user data to send
  Protocols: SCTP

- Notification to a receiver that a partial message delivery has been aborted
  Protocols: SCTP

6. IANA Considerations

This document does not require any IANA actions.

7. Security Considerations

Authentication, confidentiality protection, and integrity protection are identified as transport features [RFC8095]. These transport features are generally provided by a protocol or layer on top of the transport protocol; none of the transport protocols considered in this document provides these transport features on its own. Therefore, these transport features are not considered in this document, with the exception of native authentication capabilities of TCP and SCTP for which the security considerations in [RFC5925] and [RFC4895] apply.

Security considerations for the use of UDP and UDP-Lite are provided in the referenced RFCs. Security guidance for application usage is provided in the UDP Guidelines [RFC8085].

8. References

8.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://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, <https://www.rfc-editor.org/info/rfc1122>.

[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, <https://www.rfc-editor.org/info/rfc3758>.

[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>.

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

[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, <https://www.rfc-editor.org/info/rfc5061>.

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

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

[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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6525>.

[RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, DOI 10.17487/RFC6817, December 2012, <https://www.rfc-editor.org/info/rfc6817>.

[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, <https://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, <https://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, <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>.

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

[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, <https://www.rfc-editor.org/info/rfc7496>.

[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, <https://www.rfc-editor.org/info/rfc7829>.

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

[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>.

[RFC8304] Fairhurst, G. and T. Jones, "Transport Features of the User Datagram Protocol (UDP) and Lightweight UDP (UDP-Lite)", RFC 8304, DOI 10.17487/RFC8304, February 2018, <https://www.rfc-editor.org/info/rfc8304>.

8.2. Informative References

[RFC0854] Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, DOI 10.17487/RFC0854, May 1983, <https://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, <https://www.rfc-editor.org/info/rfc2119>.

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

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <https://www.rfc-editor.org/info/rfc2475>.

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

[RFC5461] Gont, F., "TCP's Reaction to Soft Errors", RFC 5461, DOI 10.17487/RFC5461, February 2009, <https://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, <https://www.rfc-editor.org/info/rfc6093>.

[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, <https://www.rfc-editor.org/info/rfc7414>.

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

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

[TAPS-MINSET] Welzl, M. and S. Gjessing, "A Minimal Set of Transport Services for TAPS Systems", Work in Progress, draft-ietf-taps-minset-01, February 2018.

Appendix A. Overview of RFCs Used as Input for Pass 1

TCP: [RFC0793], [RFC1122], [RFC5482], [RFC5925], and [RFC7413].

MPTCP: [RFC6182], [RFC6824], and [RFC6897].

SCTP: RFCs without a sockets API specification: [RFC3758], [RFC4895], [RFC4960], and [RFC5061].

RFCs that include a sockets API specification: [RFC6458], [RFC6525], [RFC6951], [RFC7053], [RFC7496], and [RFC7829].

UDP(-Lite): See [RFC8304].

LEDBAT: [RFC6817].

Appendix B. How This Document Was Developed

This section gives an overview of the method that was used to develop this document. It was given to contributors for guidance, and it can be helpful for future updates or extensions.

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

Primitives that "MAY" be implemented by a transport protocol were excluded. To be included, the minimum requirement level for a primitive to be implemented by a protocol was "SHOULD". Where style requirement levels as described in [RFC2119] are not used, primitives were excluded when they are described in conjunction with statements like, e.g., "some implementations also provide" or "an implementation may also". Excluded primitives or parameters were briefly described in a dedicated subsection.

Pass 1: This began by identifying text that talks about primitives. An API specification, abstract or not, obviously describes primitives -- but 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.

Some text for the pass 1 subsections was developed by copying and pasting all the relevant text parts from the relevant RFCs then adjusting the terminology to match that in Section 2 and shortening phrasing to match the general style of the document. An effort was made to formulate everything as a primitive description such that the primitive descriptions became as complete as possible (e.g., the 'SEND.TCP' primitive in pass 2 is explicitly described as reliably transferring data); text that is relevant for the primitives presented in this pass but still does not fit directly under any primitive was used in a subsection's introduction.

Pass 2: The main goal of this pass is unification of primitives. As input, only text from pass 1 was used (no exterior sources). The list in pass 2 is not arranged by protocol (i.e., "first protocol X, here are all the primitives; then protocol Y, here are all the primitives, ...") but by primitive (i.e., "primitive A, implemented this way in protocol X, this way in protocol Y, ..."). It was a goal to obtain as many similar pass 2 primitives as possible. For instance, this was sometimes achieved by not always maintaining a 1:1 mapping between pass 1 and pass 2 primitives, renaming primitives, etc. For every new primitive, the already-existing primitives were considered to try to make them as coherent as possible.

For each primitive, the following style was used:

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

The entries "Parameters", "Returns", and "Comments" were skipped when a primitive had no parameters, no described return value, or no comments seemed necessary, respectively. Optional parameters are followed by "(optional)". When known, default values were provided.

Pass 3: The main point of this pass is to identify transport 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 transport features. This list was primarily based on text from pass 2, with additional input from pass 1 (but no external sources).

Acknowledgements

The authors would like to thank (in alphabetical order) Bob Briscoe, Spencer Dawkins, Aaron Falk, David Hayes, Karen Nielsen, Tommy Pauly, Joe Touch, and Brian Trammell for providing valuable feedback on this document. We especially thank Christoph Paasch for providing input related to Multipath TCP and Gorry Fairhurst and Tom Jones for providing input related to UDP(-Lite). This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT).

Authors' Addresses

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

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