Network Working Group
Internet-Draft
Intended status: Informational
Expires: June 9, 2017

G. Fairhurst, Ed.
University of Aberdeen
B. Trammell, Ed.
M. Kuehlewind, Ed.
ETH Zurich

December 06, 2016

Services provided by IETF transport protocols and congestion control mechanisms

draft-ietf-taps-transports-14

Abstract

This document describes, surveys, and classifies the protocol mechanisms provided by existing IETF protocols, as background for determining a common set of transport services. It examines the Transmission Control Protocol (TCP), Multipath TCP, the Stream Control Transmission Protocol (SCTP), the User Datagram Protocol (UDP), UDP-Lite, the Datagram Congestion Control Protocol (DCCP), the Internet Control Message Protocol (ICMP), the Realtime Transport Protocol (RTP), File Delivery over Unidirectional Transport/ Asynchronous Layered Coding Reliable Multicast (FLUTE/ALC), and NACK-Oriented Reliable Multicast (NORM), Transport Layer Security (TLS), Datagram TLS (DTLS), and the Hypertext Transport Protocol (HTTP), when HTTP is used as a pseudotransport. This survey provides background for the definition of transport services within the TAPS working group.

Copyright Notice

Table of Contents

1. Introduction

Internet applications make use of the Services provided by a Transport protocol, such as TCP (a reliable, in-order stream protocol) or UDP (an unreliable datagram protocol). We use the term "Transport Service" to mean the end-to-end service provided to an application by the transport layer. That service can only be provided correctly if information about the intended usage is supplied from the application. The application may determine this information at design time, compile time, or run time, and may include guidance on whether a feature is required, a preference by the application, or something in between. Examples of features of Transport Services are reliable delivery, ordered delivery, content privacy to in-path devices, and integrity protection.

The IETF has defined a wide variety of transport protocols beyond TCP and UDP, including SCTP, DCCP, MPTCP, and UDP-Lite. Transport services may be provided directly by these transport protocols, or layered on top of them using protocols such as WebSockets (which runs over TCP), RTP (over TCP or UDP) or WebRTC data channels (which run over SCTP over DTLS over UDP or TCP). Services built on top of UDP or UDP-Lite typically also need to specify additional mechanisms, including a congestion control mechanism (such as NewReno [RFC6582], TFRC [RFC5348] or LEDBAT [RFC6817]). This extends the set of available Transport Services beyond those provided to applications by TCP and UDP.

The transport protocols described in this document provide a basis for the definition of transport services provided by common protocols, as background for the TAPS working group. The protocols listed here were chosen to help expose as many potential transport services as possible, and are not meant to be a comprehensive survey or classification of all transport protocols.

1.1. Overview of Transport Features

Transport protocols can be differentiated by the features of the services they provide.

Some of these provided features are closely related to basic control function that a protocol needs to work over a network path, such as addressing. The number of participants in a given association also determines its applicability: if a connection is between endpoints (unicast), to one of multiple endpoints (anycast), or simultaneously to multiple endpoints (multicast). Unicast protocols usually support bidirectional communication, while multicast is generally unidirectional. Another feature is whether a transport requires a control exchange across the network at setup (e.g., TCP), or whether it is connection-less (e.g., UDP).

For packet delivery itself, reliability and integrity protection, ordering, and framing are basic features. However, these features are implemented with different levels of assurance in different protocols. As an example, a transport service may provide full reliability, providing detection of loss and retransmission (e.g., TCP). SCTP offers a message-based service that can provide full or partial reliability, and allows the protocol to minimize the head of line blocking due to the support of ordered and unordered message delivery within multiple streams. UDP-Lite and DCCP can provide partial integrity protection to enable corruption tolerance.

Usually a protocol has been designed to support one specific type of delivery/framing: data either needs to be divided into transmission units based on network packets (datagram service), a data stream is segmented and re-combined across multiple packets (stream service), or whole objects such as files are handled accordingly. This decision strongly influences the interface that is provided to the upper layer.

In addition, transport protocols offer a certain support for transmission control. For example, a transport service can provide flow control to allow a receiver to regulate the transmission rate of a sender. Further a transport service can provide congestion control (see Section 4). As an example TCP and SCTP provide congestion control for use in the Internet, whereas UDP leaves this function to the upper layer protocol that uses UDP.

Security features are often provided independent of the transport protocol, via Transport Layer Security (TLS, see Section 3.7) or by the application layer protocol itself. The security properties TLS provides to the application (such as confidentiality, integrity, and authenticity) are also features of the transport layer, even though they are often presently implemented in a separate protocol.

2. Terminology

The following terms are used throughout this document, and in subsequent documents produced by TAPS that describe the composition and decomposition of transport services.

Transport Service 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 different transport services using a specific framing and header format on the wire.

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

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

3. Existing Transport Protocols

This section provides a list of known IETF transport protocols and transport protocol frameworks. It does not make an assessment about whether specific implementations of protocols are fully compliant to current IETF specifications.

3.1. Transport Control Protocol (TCP)

TCP is an IETF standards track transport protocol. [RFC0793] introduces TCP as follows: "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." Since its introduction, TCP has become the default connection- oriented, stream-based transport protocol in the Internet. It is widely implemented by endpoints and widely used by common applications.

3.1.1. Protocol Description

TCP is a connection-oriented protocol, providing a three way handshake to allow a client and server to set up a connection and negotiate features, and mechanisms for orderly completion and immediate teardown of a connection. TCP is defined by a family of RFCs [RFC7414].

TCP provides multiplexing to multiple sockets on each host using port numbers. A similar approach is adopted by other IETF-defined transports. An active TCP session is identified by its four-tuple of local and remote IP addresses and local port and remote port numbers. The destination port during connection setup is often used to indicate the requested service.

TCP partitions a continuous stream of bytes into segments, sized to fit in IP packets based on a negotiated maximum segment size and further constrained by the effective Maximum Transmission Unit (MTU) from Path MTU Discovery (PMTUD). ICMP-based Path MTU discovery [RFC1191][RFC1981] as well as Packetization Layer Path MTU Discovery (PMTUD) [RFC4821] have been defined by the IETF.

Each byte in the stream is identified by a sequence number. The sequence number is used to order segments on receipt, to identify segments in acknowledgments, and to detect unacknowledged segments for retransmission. This is the basis of the reliable, ordered delivery of data in a TCP stream. TCP Selective Acknowledgment (SACK) [RFC2018] extends this mechanism by making it possible to provide earlier identification of which segments are missing, allowing faster retransmission. SACK-based methods (e.g. Duplicate Selective ACK) can also result in less spurious retransmission.

Receiver flow control is provided by a sliding window: limiting the amount of unacknowledged data that can be outstanding at a given time. The window scale option [RFC7323] allows a receiver to use windows greater than 64KB.

All TCP senders provide congestion control, such as described in [RFC5681]. TCP uses a sequence number with a sliding receiver window for flow control. The TCP congestion control mechanism also utilises this TCP sequence number to manage a separate congestion window [RFC5681]. The sending window at a given point in time is the minimum of the receiver window and the congestion window. The congestion window is increased in the absence of congestion and, respectively, decreased if congestion is detected. Often loss is implicitly handled as a congestion indication which is detected in TCP (also as input for retransmission handling) based on two mechanisms: A retransmission timer with exponential back-off or the reception of three acknowledgment for the same segment, so called duplicated ACKs (Fast retransmit). In addition, Explicit Congestion Notification (ECN) [RFC3168] can be used in TCP, if supported by both endpoints, that allows a network node to signal congestion without inducing loss. Alternatively, a delay-based congestion control scheme can be used in TCP that reacts to changes in delay as an early indication of congestion as also further described in Section 4. Examples for different kind of congestion control schemes are given in Section 4.

TCP protocol instances can be extended [RFC7414] and tuned. Some features are sender-side only, requiring no negotiation with the receiver; some are receiver-side only, some are explicitly negotiated during connection setup.

TCP may buffer data, e.g., to optimize processing or capacity usage. TCP can therefore provides mechanisms to control this, including an optional "PUSH" function [RFC0793] that explicitly requests the transport service not to delay data. By default, TCP segment partitioning uses Nagle’s algorithm [RFC0896] to buffer data at the sender into large segments, potentially incurring sender-side buffering delay; this algorithm can be disabled by the sender to transmit more immediately, e.g., to reduce latency for interactive sessions.

TCP provides an "urgent data" function for limited out-of-order delivery of the data. This function is deprecated [RFC6093].

A TCP Reset (RST) control message may be used to force a TCP endpoint to close a session [RFC0793], aborting the connection.

A mandatory checksum provides a basic integrity check against misdelivery and data corruption over the entire packet. Applications that require end to end integrity of data are recommended to include a stronger integrity check of their payload data. The TCP checksum [RFC1071] [RFC2460] does not support partial payload protection (as in DCCP/UDP-Lite).

TCP supports only unicast connections.

3.1.2. Interface description

A User/TCP Interface is defined in [RFC0793] providing six user commands: Open, Send, Receive, Close, Status. This interface does not describe configuration of TCP options or parameters beside use of the PUSH and URGENT flags.

[RFC1122] describes extensions of the TCP/application layer interface for:

- o reporting soft errors such as reception of ICMP error messages, extensive retransmission or urgent pointer advance,
- o providing a possibility to specify the Differentiated Services Code Point (DSCP) [RFC3260] (formerly, the Type-of-Service, TOS) for segments,
- o providing a flush call to empty the TCP send queue, and
- o multihoming support.

In API implementations derived from the BSD Sockets API, TCP sockets are created using the "SOCK_STREAM" socket type as described in the IEEE Portable Operating System Interface (POSIX) Base Specifications [POSIX]. The features used by a protocol instance may be set and tuned via this API. There are currently no documents in the RFC Series that describe this interface.

3.1.3. Transport Features

The transport features provided by TCP are:

- o connection-oriented transport with feature negotiation and application-to-port mapping (implemented using SYN segments and the TCP option field to negotiate features),

- unicast transport (though anycast TCP is implemented, at risk of instability due to rerouting),
- port multiplexing,
- uni- or bidirectional communication,
- stream-oriented delivery in a single stream,
- fully reliable delivery (implemented using ACKs sent from the receiver to confirm delivery),
- error detection (implemented using a segment checksum to verify delivery to the correct endpoint and integrity of the data and options),
- segmentation,
- data bundling (optional; uses Nagle’s algorithm to coalesce data sent within the same RTT into full-sized segments),
- flow control (implemented using a window-based mechanism where the receiver advertises the window that it is willing to buffer),
- congestion control (usually implemented using a window-based mechanism and four algorithms for different phases of the transmission: slow start, congestion avoidance, fast retransmit, and fast recovery [RFC5681]).

3.2. Multipath TCP (MPTCP)

Multipath TCP [RFC6824] is an extension for TCP to support multi-homing for resilience, mobility and load-balancing. It is designed to be as indistinguishable to middleboxes from non-multipath TCP as possible. It does so by establishing regular TCP flows between a pair of source/destination endpoints, and multiplexing the application’s stream over these flows. Sub- flows can be started over IPv4 or IPv6 for the same session.

3.2.1. Protocol Description

MPTCP uses TCP options for its control plane. They are used to signal multipath capabilities, as well as to negotiate data sequence numbers, and advertise other available IP addresses and establish new sessions between pairs of endpoints.

By multiplexing one byte stream over separate paths, MPTCP can achieve a higher throughput than TCP in certain situations. However, if coupled congestion control [RFC6356] is used, it might limit this benefit to maintain fairness to other flows at the bottleneck. When aggregating capacity over multiple paths, and depending on the way packets are scheduled on each TCP subflow, additional delay and higher jitter might be observed observed before in-order delivery of data to the applications.

3.2.2. Interface Description

By default, MPTCP exposes the same interface as TCP to the application. [RFC6897] however describes a richer API for MPTCP-aware applications.

This Basic API describes how an application can:

- enable or disable MPTCP.
- bind a socket to one or more selected local endpoints.
- query local and remote endpoint addresses.
- get a unique connection identifier (similar to an address-port pair for TCP).

The document also recommends the use of extensions defined for SCTP [RFC6458] (see next section) to support multihoming for resilience and mobility.

3.2.3. Transport features

As an extension to TCP, MPTCP provides mostly the same features. By establishing multiple sessions between available endpoints, it can additionally provide soft failover solutions in the case that one of the paths become unusable.

The transport features provided by MPTCP in addition to TCP therefore are:

- multihoming for load-balancing, with endpoint multiplexing of a single byte stream, using either coupled congestion control or for throughput maximization,
- address family multiplexing (using IPv4 and IPv6 for the same session),
- resilience to network failure and/or handover.

3.3. User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) [RFC0768] [RFC2460] is an IETF standards track transport protocol. It provides a unidirectional datagram protocol that preserves message boundaries. It provides no error correction, congestion control, or flow control. It can be used to send broadcast datagrams (IPv4) or multicast datagrams (IPv4 and IPv6), in addition to unicast and anycast datagrams. IETF guidance on the use of UDP is provided in [I-D.ietf-tsvwg-rfc5405bis]. UDP is widely implemented and widely used by common applications, including DNS.

3.3.1. Protocol Description

UDP is a connection-less protocol that maintains message boundaries, with no connection setup or feature negotiation. The protocol uses independent messages, ordinarily called datagrams. It provides detection of payload errors and misdelivery of packets to an unintended endpoint, either of which result in discard of received datagrams, with no indication to the user of the service.

It is possible to create IPv4 UDP datagrams with no checksum, and while this is generally discouraged [RFC1122] [I-D.ietf-tsvwg-rfc5405bis], certain special cases permit this use. These datagrams rely on the IPv4 header checksum to protect from misdelivery to an unintended endpoint. IPv6 does not permit UDP datagrams with no checksum, although in certain cases [RFC6936] this rule may be relaxed [RFC6935].

UDP does not provide reliability and does not provide retransmission. Messages may be re-ordered, lost, or duplicated in transit. Note that due to the relatively weak form of checksum used by UDP, applications that require end to end integrity of data are recommended to include a stronger integrity check of their payload data.

Because UDP provides no flow control, a receiving application that is unable to run sufficiently fast, or frequently, may miss messages. The lack of congestion handling implies UDP traffic may experience loss when using an overloaded path, and may cause the loss of messages from other protocols (e.g., TCP) when sharing the same network path.

On transmission, UDP encapsulates each datagram into a single IP packet or several IP packet fragments. This allows a datagram to be larger than the effective path MTU. Fragments are reassembled before delivery to the UDP receiver, making this transparent to the user of the transport service. When the jumbograms are supported, larger messages may be sent without performing fragmentation.

UDP on its own does not provide support for segmentation, receiver flow control, congestion control, PathMTU discovery/PLPMTUD, or ECN. Applications that require these features need to provide them on their own, or by using a protocol over UDP that provides them [I-D.ietf-tsvwg-rfc5405bis].

3.3.2. Interface Description

[RFC0768] describes basic requirements for an API for UDP. Guidance on use of common APIs is provided in [I-D.ietf-tsvwg-rfc5405bis].

A UDP endpoint consists of a tuple of (IP address, port number). De-multiplexing using multiple abstract endpoints (sockets) on the same IP address is supported. The same socket may be used by a single server to interact with multiple clients (note: this behavior differs from TCP, which uses a pair of tuples to identify a connection). Multiple server instances (processes) that bind to the same socket can cooperate to service multiple clients. The socket implementation arranges to not duplicate the same received unicast message to multiple server processes.

Many operating systems also allow a UDP socket to be "connected", i.e., to bind a UDP socket to a specific (remote) UDP endpoint. Unlike TCP's connect primitive, for UDP, this is only a local operation that serves to simplify the local send/receive functions and to filter the traffic for the specified addresses and ports [I-D.ietf-tsvwg-rfc5405bis].

3.3.3. Transport Features

The transport features provided by UDP are:

o unicast, multicast, anycast, or IPv4 broadcast transport,

o port multiplexing (where a receiving port can be configured to receive datagrams from multiple senders),

o message-oriented delivery,

o uni- or bidirectional communication where the transmissions in each direction are independent,

o non-reliable delivery,

o unordered delivery,

- error detection (implemented using a segment checksum to verify delivery to the correct endpoint and integrity of the data; optional for IPv4 and optional under specific conditions for IPv6 where all or none of the payload data is protected),

3.4. Lightweight User Datagram Protocol (UDP-Lite)

The Lightweight User Datagram Protocol (UDP-Lite) [RFC3828] is an IETF standards track transport protocol. It provides a unidirectional, datagram protocol that preserves message boundaries. IETF guidance on the use of UDP- Lite is provided in [I-D.ietf-tsvwg-rfc5405bis]. A UDP-Lite service may support IPv4 broadcast, multicast, anycast and unicast, and IPv6 multicast, anycast and unicast.

Examples of use include a class of applications that can derive benefit from having partially-damaged payloads delivered, rather than discarded. One use is to provider header integrity checks but allow delivery of corrupted payloads to error-tolerant applications, or when payload integrity is provided by some other mechanism (see [RFC6936]).

3.4.1. Protocol Description

Like UDP, UDP-Lite is a connection-less datagram protocol, with no connection setup or feature negotiation. It changes the semantics of the UDP "payload length" field to that of a "checksum coverage length" field, and is identified by a different IP protocol/next-header value. The "checksum coverage length" field specifies the intended checksum coverage, with the remaining unprotected part of the payload called the "error-insensitive part". Applications using UDP-Lite therefore cannot make assumptions regarding the correctness of the data received in the insensitive part of the UDP-Lite payload.

Otherwise, UDP-Lite is semantically identical to UDP. In the same way as for UDP, mechanisms for receiver flow control, congestion control, PMTU or PLPMTU discovery, support for ECN, etc. needs to be provided by upper layer protocols [I-D.ietf-tsvwg-rfc5405bis].

3.4.2. Interface Description

There is no API currently specified in the RFC Series, but guidance on use of common APIs is provided in [I-D.ietf-tsvwg-rfc5405bis].

The interface of UDP-Lite differs from that of UDP by the addition of a single (socket) option that communicates a checksum coverage length value. The checksum coverage may also be made visible to the application via the UDP-Lite MIB module [RFC5097].

3.4.3. Transport Features

The transport features provided by UDP-Lite are:

- o unicast, multicast, anycast, or IPv4 broadcast transport (as for UDP),
- o port multiplexing (as for UDP),
- o message-oriented delivery (as for UDP),
- o Uni- or bidirectional communication where the transmissions in each direction are independent (as for UDP),
- o non-reliable delivery (as for UDP),
- o non-ordered delivery (as for UDP),
- o partial or full payload error detection (where the checksum coverage field indicates the size of the payload data covered by the checksum).

3.5. Stream Control Transmission Protocol (SCTP)

SCTP is a message-oriented IETF standards track transport protocol. The base protocol is specified in [RFC4960]. It supports multi-homing and path failover to provide resilience to path failures. An SCTP association has multiple streams in each direction, providing in-sequence delivery of user messages within each stream. This allows it to minimize head of line blocking. SCTP supports multiple stream scheduling schemes controlling stream multiplexing, including priority and fair weighting schemes.

SCTP was originally developed for transporting telephony signaling messages and is deployed in telephony signaling networks, especially in mobile telephony networks. It can also be used for other services, for example, in the WebRTC framework for data channels.

3.5.1. Protocol Description

SCTP is a connection-oriented protocol using a four way handshake to establish an SCTP association, and a three way message exchange to gracefully shut it down. It uses the same port number concept as DCCP, TCP, UDP, and UDP-Lite. SCTP only supports unicast.

SCTP uses the 32-bit CRC32c for protecting SCTP packets against bit errors and misdelivery of packets to an unintended endpoint. This is stronger than the 16-bit checksums used by TCP or UDP. However, partial payload checksum coverage as provided by DCCP or UDP-Lite is not supported.

SCTP has been designed with extensibility in mind. A common header is followed by a sequence of chunks. [RFC4960] defines how a receiver processes chunks with an unknown chunk type. The support of extensions can be negotiated during the SCTP handshake. Currently defined extensions include mechanisms for dynamic re-configuration of streams [RFC6525] and IP addresses [RFC5061]. Furthermore, the extension specified in [RFC3758] introduces the concept of partial reliability for user messages.

SCTP provides a message-oriented service. Multiple small user messages can be bundled into a single SCTP packet to improve efficiency. For example, this bundling may be done by delaying user messages at the sender, similar to Nagle's algorithm used by TCP. User messages which would result in IP packets larger than the MTU will be fragmented at the sender and reassembled at the receiver. There is no protocol limit on the user message size. For MTU discovery the same mechanism than for TCP can be used [RFC1981][RFC4821], as well as utilizing probe packets with padding chunks, as defined in [RFC4820].

[RFC4960] specifies TCP-friendly congestion control to protect the network against overload. SCTP also uses sliding window flow control to protect receivers against overflow. Similar to TCP, SCTP also supports delaying acknowledgments. [RFC7053] provides a way for the sender of user messages to request the immediate sending of the corresponding acknowledgments.

Each SCTP association has between 1 and 65536 uni-directional streams in each direction. The number of streams can be different in each direction. Every user message is sent on a particular stream. User messages can be sent un- ordered, or ordered upon request by the upper layer. Un-ordered messages can be delivered as soon as they are completely received. For user messages not requiring fragmentation, this minimizes head of line blocking. On the other hand, ordered messages sent on the same stream are delivered at the receiver in the same order as sent by the sender.

The base protocol defined in [RFC4960] does not allow interleaving of user- messages. Large messages on one stream can therefore block the sending of user messages on other streams. [I-D.ietf-tsvwg-sctp-ndata] overcomes this limitation. This draft also specifies multiple algorithms for the sender side selection of which streams to send data from, supporting a variety of scheduling algorithms including priority based methods. The stream re-configuration extension defined in [RFC6525] allows streams to be reset during the lifetime of an association and to increase the number of streams, if the number of streams negotiated in the SCTP handshake becomes insufficient.

Each user message sent is either delivered to the receiver or, in case of excessive retransmissions, the association is terminated in a non-graceful way [RFC4960], similar to TCP behavior. In addition to this reliable transfer, the partial reliability extension [RFC3758] allows a sender to abandon user messages. The application can specify the policy for abandoning user messages.

SCTP supports multi-homing. Each SCTP endpoint uses a list of IP-addresses and a single port number. These addresses can be any mixture of IPv4 and IPv6 addresses. These addresses are negotiated during the handshake and the address re-configuration extension specified in [RFC5061] in combination with [RFC4895] can be used to change these addresses in an authenticated way during the lifetime of an SCTP association. This allows for transport layer mobility. Multiple addresses are used for improved resilience. If a remote address becomes unreachable, the traffic is switched over to a reachable one, if one exists.

For securing user messages, the use of TLS over SCTP has been specified in [RFC3436]. However, this solution does not support all services provided by SCTP, such as un-ordered delivery or partial reliability. Therefore, the use of DTLS over SCTP has been specified in [RFC6083] to overcome these limitations. When using DTLS over SCTP, the application can use almost all services provided by SCTP.

[I-D.ietf-tsvwg-natsupp] defines methods for endpoints and middleboxes to provide NAT traversal for SCTP over IPv4. For legacy NAT traversal, [RFC6951] defines the UDP encapsulation of SCTP-packets. Alternatively, SCTP packets can be encapsulated in DTLS packets as specified in [I-D.ietf-tsvwg-sctp-dtls-encaps]. The latter encapsulation is used within the WebRTC [I-D.ietf-rtcweb-transports] context.

An SCTP ABORT chunk may be used to force a SCTP endpoint to close a session [RFC4960], aborting the connection.

SCTP has a well-defined API, described in the next subsection.

3.5.2. Interface Description

[RFC4960] defines an abstract API for the base protocol. This API describes the following functions callable by the upper layer of SCTP: Initialize, Associate, Send, Receive, Receive Unsent Message, Receive Unacknowledged Message, Shutdown, Abort, SetPrimary, Status, Change Heartbeat, Request Heartbeat, Get SRTT Report, Set Failure Threshold, Set Protocol Parameters, and Destroy. The following notifications are provided by the SCTP stack to the upper layer: COMMUNICATION UP, DATA ARRIVE, SHUTDOWN COMPLETE, COMMUNICATION LOST, COMMUNICATION ERROR, RESTART, SEND FAILURE, NETWORK STATUS CHANGE.

An extension to the BSD Sockets API is defined in [RFC6458] and covers:

- the base protocol defined in [RFC4960]. The API allows control over local addresses and port numbers and the primary path. Furthermore the application has fine control about parameters like retransmission thresholds, the path supervision parameters, the delayed acknowledgment timeout, and the fragmentation point. The API provides a mechanism to allow the SCTP stack to notify the application about events if the application has requested them. These notifications provide information about status changes of the association and each of the peer addresses. In case of send failures, including drop of messages sent unreliably, the application can also be notified and user messages can be returned to the application. When sending user messages, the stream id, a payload protocol identifier, an indication whether ordered delivery is requested or not. These parameters can also be provided on message reception. Additionally a context can be provided when sending, which can be use in case of send failures. The sending of arbitrary large user messages is supported.

- the SCTP Partial Reliability extension defined in [RFC3758] to specify for a user message the PR-SCTP policy and the policy specific parameter. Examples of these policies defined in [RFC3758] and [RFC7496] are:

- Limiting the time a user message is dealt with by the sender.

- Limiting the number of retransmissions for each fragment of a user message. If the number of retransmissions is limited to 0, one gets a service similar to UDP.

- Abandoning messages of lower priority in case of a send buffer shortage.

- the SCTP Authentication extension defined in [RFC4895] allowing to manage the shared keys, the HMAC to use, set the chunk types which are only accepted in an authenticated way, and get the list of chunks which are accepted by the local and remote end point in an authenticated way.

- the SCTP Dynamic Address Reconfiguration extension defined in [RFC5061]. It allows to manually add and delete local addresses for SCTP associations and the enabling of automatic address addition and deletion. Furthermore the peer can be given a hint for choosing its primary path.

A BSD Sockets API extension has been defined in the documents that specify the following SCTP protocol extensions:

- the SCTP Stream Reconfiguration extension defined in [RFC6525]. The API allows to trigger the reset operation for incoming and outgoing streams and the whole association. It provides also a way to notify the association about the corresponding events. Furthermore the application can increase the number of streams.

- the UDP Encapsulation of SCTP packets extension defined in [RFC6951]. The API allows the management of the remote UDP encapsulation port.

- the SCTP SACK-IMMEDIATELY extension defined in [RFC7053]. The API allows the sender of a user message to request the receiver to send the corresponding acknowledgment immediately.

- the additional PR-SCTP policies defined in [RFC7496]. The API allows to enable/disable the PR-SCTP extension, choose the PR-SCTP policies defined in the document and provide statistical information about abandoned messages.

Future documents describing SCTP protocol extensions are expected to describe the corresponding BSD Sockets API extension in a "Socket API Considerations" section.

The SCTP socket API supports two kinds of sockets:

- one-to-one style sockets (by using the socket type "SOCK_STREAM").

- one-to-many style socket (by using the socket type "SOCK_SEQPACKET").

One-to-one style sockets are similar to TCP sockets, there is a 1:1 relationship between the sockets and the SCTP associations (except for listening sockets). One-to-many style SCTP sockets are similar to unconnected UDP sockets, where there is a 1:n relationship between the sockets and the SCTP associations.

The SCTP stack can provide information to the applications about state changes of the individual paths and the association whenever they occur. These events are delivered similar to user messages but are specifically marked as notifications.

New functions have been introduced to support the use of multiple local and remote addresses. Additional SCTP-specific send and receive calls have been defined to permit SCTP-specific information to be sent without using ancillary data in the form of additional cmsgs. These functions provide support for detecting partial delivery of user messages and notifications.

The SCTP socket API allows a fine-grained control of the protocol behavior through an extensive set of socket options.

The SCTP kernel implementations of FreeBSD, Linux and Solaris follow mostly the specified extension to the BSD Sockets API for the base protocol and the corresponding supported protocol extensions.

3.5.3. Transport Features

The transport features provided by SCTP are:

- connection-oriented transport with feature negotiation and application-to-port mapping,
- unicast transport,
- port multiplexing,
- uni- or bidirectional communication,
- message-oriented delivery with durable message framing supporting multiple concurrent streams,
- fully reliable, partially reliable, or unreliable delivery (based on user specified policy to handle abandoned user messages) with drop notification,
- ordered and unordered delivery within a stream,
- support for stream scheduling prioritization,
- segmentation,
- user message bundling,
- flow control using a window-based mechanism,
- congestion control using methods similar to TCP,

- o strong error detection (CRC32c),
- o transport layer multihoming for resilience and mobility.

3.6. Datagram Congestion Control Protocol (DCCP)

Datagram Congestion Control Protocol (DCCP) [RFC4340] is an IETF standards track bidirectional transport protocol that provides unicast connections of congestion-controlled messages without providing reliability.

The DCCP Problem Statement describes the goals that DCCP sought to address [RFC4336]: It is suitable for applications that transfer fairly large amounts of data and that can benefit from control over the trade off between timeliness and reliability [RFC4336].

DCCP offers low overhead, and many characteristics common to UDP, but can avoid "re-inventing the wheel" each time a new multimedia application emerges. Specifically it includes core transport functions (feature negotiation, path state management, RTT calculation, PMTUD, etc.): DCCP applications select how they send packets and, where suitable, choose common algorithms to manage their functions. Examples of applications that can benefit from such transport services include interactive applications, streaming media, or on-line games [RFC4336].

3.6.1. Protocol Description

DCCP is a connection-oriented datagram protocol, providing a three-way handshake to allow a client and server to set up a connection, and mechanisms for orderly completion and immediate teardown of a connection.

A DCCP protocol instance can be extended [RFC4340] and tuned using additional features. Some features are sender-side only, requiring no negotiation with the receiver; some are receiver-side only; and some are explicitly negotiated during connection setup.

DCCP uses a Connect packet to initiate a session, and permits each endpoint to choose the features it wishes to support. Simultaneous open [RFC5596], as in TCP, can enable interoperability in the presence of middleboxes. The Connect packet includes a Service Code [RFC5595] that identifies the application or protocol using DCCP, providing middleboxes with information about the intended use of a connection.

The DCCP service is unicast-only.

It provides multiplexing to multiple sockets at each endpoint using port numbers. An active DCCP session is identified by its four-tuple of local and remote IP addresses and local port and remote port numbers.

The protocol segments data into messages, typically sized to fit in IP packets, but which may be fragmented providing they are smaller than the maximum packet size. A DCCP interface allows applications to request fragmentation for packets larger than PMTU, but not larger than the maximum packet size allowed by the current congestion control mechanism (CCMPS) [RFC4340].

Each message is identified by a sequence number. The sequence number is used to identify segments in acknowledgments, to detect unacknowledged segments, to measure RTT, etc. The protocol may support unordered delivery of data, and does not itself provide retransmission. DCCP supports reduced checksum coverage, a partial payload protection mechanism similar to UDP-Lite. There is also a Data Checksum option, which when enabled, contains a strong CRC, to enable endpoints to detect application data corruption.

Receiver flow control is supported, which limits the amount of unacknowledged data that can be outstanding at a given time.

A DCCP Reset packet may be used to force a DCCP endpoint to close a session [RFC4340], aborting the connection.

DCCP supports negotiation of the congestion control profile between endpoints, to provide plug-and-play congestion control mechanisms. Examples of specified profiles include "TCP-like" [RFC4341], "TCP-friendly" [RFC4342], and "TCP-friendly for small packets" [RFC5622]. Additional mechanisms are recorded in an IANA registry.

A lightweight UDP-based encapsulation (DCCP-UDP) has been defined [RFC6773] that permits DCCP to be used over paths where DCCP is not natively supported. Support for DCCP in NAPT/NATs is defined in [RFC4340] and [RFC5595]. Upper layer protocols specified on top of DCCP include DTLS [RFC5595], RTP [RFC5672], ICE/SDP [RFC6773].

3.6.2. Interface Description

Functions expected for a DCCP API include: Open, Close and Management of the progress a DCCP connection. The Open function provides feature negotiation, selection of an appropriate CCID for congestion control and other parameters associated with the DCCP connection. A function allows an application to send DCCP datagrams, including setting the required checksum coverage, and any required options. (DCCP permits sending datagrams with a zero-length payload.) A function allows reception of data, including indicating if the data was used or dropped. Functions can also make the status of a connection visible to an application, including detection of the maximum packet size and the ability to perform flow control by detecting a slow receiver at the sender.

There is no API currently specified in the RFC Series.

3.6.3. Transport Features

The transport features provided by DCCP are:

- o unicast transport,
- o connection-oriented communication with feature negotiation and application-to-port mapping,
- o signaling of application class for middlebox support (implemented using Service Codes),
- o port multiplexing,
- o uni-or bidirectional communication,
- o message-oriented delivery,
- o unreliable delivery with drop notification,
- o unordered delivery,
- o flow control (implemented using the slow receiver function)
- o partial and full payload error detection (with optional strong integrity check).

3.7. Transport Layer Security (TLS) and Datagram TLS (DTLS) as a pseudotransport

Transport Layer Security (TLS) [RFC5246] and Datagram TLS (DTLS) [RFC6347] are IETF protocols that provide several security-related features to applications. TLS is designed to run on top of a reliable streaming transport protocol (usually TCP), while DTLS is designed to run on top of a best-effort datagram protocol (UDP or DCCP [RFC5238]). At the time of writing, the current version of TLS is 1.2, defined in [RFC5246]; work on TLS version 1.3 [I-D.ietf-tls-tls13] nearing completion. DTLS provides nearly identical functionality to applications; it is defined in [RFC6347] and its current version is also 1.2. The TLS protocol evolved from the Secure Sockets Layer (SSL) [RFC6101] protocols developed in the mid-1990s to support protection of HTTP traffic.

While older versions of TLS and DTLS are still in use, they provide weaker security guarantees. [RFC7457] outlines important attacks on TLS and DTLS. [RFC7525] is a Best Current Practices (BCP) document that describes secure configurations for TLS and DTLS to counter these attacks. The recommendations are applicable for the vast majority of use cases.

3.7.1. Protocol Description

Both TLS and DTLS provide the same security features and can thus be discussed together. The features they provide are:

- Confidentiality

- Data integrity

- Peer authentication (optional)

- Perfect forward secrecy (optional)

The authentication of the peer entity can be omitted; a common web use case is where the server is authenticated and the client is not. TLS also provides a completely anonymous operation mode in which neither peer's identity is authenticated. It is important to note that TLS itself does not specify how a peering entity's identity should be interpreted. For example, in the common use case of authentication by means of an X.509 certificate, it is the application's decision whether the certificate of the peering entity is acceptable for authorization decisions.

Perfect forward secrecy, if enabled and supported by the selected algorithms, ensures that traffic encrypted and captured during a session at time t0 cannot be later decrypted at time t1 (t1 > t0), even if the long-term secrets of the communicating peers are later compromised.

As DTLS is generally used over an unreliable datagram transport such as UDP, applications will need to tolerate lost, re-ordered, or duplicated datagrams. Like TLS, DTLS conveys application data in a sequence of independent records. However, because records are mapped to unreliable datagrams, there are several features unique to DTLS that are not applicable to TLS:

- Record replay detection (optional).

- Record size negotiation (estimates of PMTU and record size expansion factor).

- Conveyance of IP don't fragment (DF) bit settings by application.

- An anti-DoS stateless cookie mechanism (optional).

Generally, DTLS follows the TLS design as closely as possible. To operate over datagrams, DTLS includes a sequence number and limited forms of retransmission and fragmentation for its internal operations. The sequence number may be used for detecting replayed information, according to the windowing procedure described in Section 4.1.2.6 of [RFC6347]. DTLS forbids the use of stream ciphers, which are essentially incompatible when operating on independent encrypted records.

3.7.2. Interface Description

TLS is commonly invoked using an API provided by packages such as OpenSSL, wolfSSL, or GnuTLS. Using such APIs entails the manipulation of several important abstractions, which fall into the following categories: long-term keys and algorithms, session state, and communications/connections.

Considerable care is required in the use of TLS APIs to ensure creation of a secure application. The programmer should have at least a basic understanding of encryption and digital signature algorithms and their strengths, public key infrastructure (including X.509 certificates and certificate revocation), and the sockets API. See [RFC7525] and [RFC7457], as mentioned above.

As an example, in the case of OpenSSL, the primary abstractions are the library itself and method (protocol), session, context, cipher and connection. After initializing the library and setting the method, a cipher suite is chosen and used to configure a context object. Session objects may then be minted according to the parameters present in a context object and associated with individual connections. Depending on how precisely the programmer wishes to select different algorithmic or protocol options, various levels of details may be required.

3.7.3. Transport Features

Both TLS and DTLS employ a layered architecture. The lower layer is commonly called the record protocol. It is responsible for:

- message fragmentation,

- authentication and integrity via message authentication codes (MAC),
- data encryption,
- scheduling transmission using the underlying transport protocol.

DTLS augments the TLS record protocol with:

- ordering and replay protection, implemented using sequence numbers.

Several protocols are layered on top of the record protocol. These include the handshake, alert, and change cipher spec protocols. There is also the data protocol, used to carry application traffic. The handshake protocol is used to establish cryptographic and compression parameters when a connection is first set up. In DTLS, this protocol also has a basic fragmentation and retransmission capability and a cookie-like mechanism to resist DoS attacks. (TLS compression is not recommended at present). The alert protocol is used to inform the peer of various conditions, most of which are terminal for the connection. The change cipher spec protocol is used to synchronize changes in cryptographic parameters for each peer.

The data protocol, when used with an appropriate cipher, provides:

- authentication of one end or both ends of a connection,
- confidentiality,
- cryptographic integrity protection.

Both TLS and DTLS are unicast-only.

3.8. Realtime Transport Protocol (RTP)

RTP provides an end-to-end network transport service, suitable for applications transmitting real-time data, such as audio, video or data, over multicast or unicast transport services, including TCP, UDP, UDP-Lite, DCCP, TLS and DTLS.

3.8.1. Protocol Description

The RTP standard [RFC3550] defines a pair of protocols, RTP and the RTP control protocol, RTCP. The transport does not provide connection setup, instead relying on out-of-band techniques or associated control protocols to setup, negotiate parameters or tear down a session.

An RTP sender encapsulates audio/video data into RTP packets to transport media streams. The RFC-series specifies RTP payload formats that allow packets to carry a wide range of media, and specifies a wide range of multiplexing, error control and other support mechanisms.

If a frame of media data is large, it will be fragmented into several RTP packets. Likewise, several small frames may be bundled into a single RTP packet.

An RTP receiver collects RTP packets from the network, validates them for correctness, and sends them to the media decoder input-queue. Missing packet detection is performed by the channel decoder. The play-out buffer is ordered by time stamp and is used to reorder packets. Damaged frames may be repaired before the media payloads are decompressed to display or store the data. Some uses of RTP are able to exploit the partial payload protection features offered by DCCP and UDP-Lite.

RTCP is a control protocol that works alongside an RTP flow. Both the RTP sender and receiver will send RTCP report packets. This is used to periodically send control information and report performance. Based on received RTCP feedback, an RTP sender can adjust the transmission, e.g., perform rate adaptation at the application layer in the case of congestion.

An RTCP receiver report (RTCP RR) is returned to the sender periodically to report key parameters (e.g, the fraction of packets lost in the last reporting interval, the cumulative number of packets lost, the highest sequence number received, and the inter-arrival jitter). The RTCP RR packets also contain timing information that allows the sender to estimate the network round trip time (RTT) to the receivers.

The interval between reports sent from each receiver tends to be on the order of a few seconds on average, although this varies with the session rate, and sub-second reporting intervals are possible for high rate sessions. The interval is randomized to avoid synchronization of reports from multiple receivers.

3.8.2. Interface Description

There is no standard application programming interface defined for RTP or RTCP. Implementations are typically tightly integrated with a particular application, and closely follow the principles of application level framing and integrated layer processing [ClarkArch] in media processing [RFC2736], error recovery and concealment, rate adaptation, and security [RFC7202]. Accordingly, RTP implementations tend to be targeted at particular application domains (e.g., voice-over-IP, IPTV, or video conferencing), with a feature set optimized for that domain, rather than being general purpose implementations of the protocol.

3.8.3. Transport Features

The transport features provided by RTP are:

- o unicast, multicast or IPv4 broadcast (provided by lower layer protocol),
- o port multiplexing (provided by lower layer protocol),
- o uni- or bidirectional communication (provided by lower layer protocol),
- o message-oriented delivery with support for media types and other extensions,
- o reliable delivery when using erasure coding or unreliable delivery with drop notification (if supported by lower layer protocol),
- o connection setup with feature negotiation (using associated protocols) and application-to-port mapping (provided by lower layer protocol),
- o segmentation,
- o performance metric reporting (using associated protocols).

3.9. Hypertext Transport Protocol (HTTP) over TCP as a pseudotransport

The Hypertext Transfer Protocol (HTTP) is an application-level protocol widely used on the Internet. It provides object-oriented delivery of discrete data or files. Version 1.1 of the protocol is specified in [RFC7230] [RFC7231] [RFC7232] [RFC7233] [RFC7234] [RFC7235], and version 2 in [RFC7540]. HTTP is usually transported over TCP using port 80 and 443, although it can be used with other transports. When used over TCP it inherits TCP's properties.

Application layer protocols may use HTTP as a substrate with an existing method and data formats, or specify new methods and data formats. There are various reasons for this practice listed in [RFC3205]; these include being a well-known and well-understood protocol, reusability of existing servers and client libraries, easy use of existing security mechanisms such as HTTP digest authentication [RFC2617] and TLS [RFC5246], the ability of HTTP to traverse firewalls makes it work over many types of infrastructure, and in cases where an application server often needs to support HTTP anyway.

Depending on application need, the use of HTTP as a substrate protocol may add complexity and overhead in comparison to a special-purpose protocol (e.g., HTTP headers, suitability of the HTTP security model, etc.). [RFC3205] addresses this issue and provides some guidelines and identifies concerns about the use of HTTP standard port 80 and 443, the use of HTTP URL scheme and interaction with existing firewalls, proxies and NATs.

Representational State Transfer (REST) [REST] is another example of how applications can use HTTP as transport protocol. REST is an architecture style that may be used to build applications using HTTP as a communication protocol.

3.9.1. Protocol Description

Hypertext Transfer Protocol (HTTP) is a request/response protocol. A client sends a request containing a request method, URI and protocol version followed message whose design is inspired by MIME (see [RFC7231] for the differences between an HTTP object and a MIME message), containing information about the client and request modifiers. The message can also contain a message body carrying application data. The server responds with a status or error code followed by a message containing information about the server and information about the data. This may include a message body. It is possible to specify a data format for the message body using MIME media types [RFC2045]. The protocol has additional features, some relevant to pseudo-transport are described below.

Content negotiation, specified in [RFC7231], is a mechanism provided by HTTP to allow selection of a representation for a requested resource. The client and server negotiate acceptable data formats, character sets, data encoding (e.g., data can be transferred compressed using gzip). HTTP can accommodate exchange of messages as well as data streaming (using chunked transfer encoding [RFC7230]). It is also possible to request a part of a resource using an object range request [RFC7233]. The protocol provides powerful cache control signaling defined in [RFC7234].

The persistent connections of HTTP 1.1 and HTTP 2.0 allow multiple request- response transactions (streams) during the life-time of a single HTTP connection. This reduces overhead during connection establishment and mitigates transport layer slow-start that would have otherwise been incurred for each transaction. HTTP 2.0 connections can multiplex many request/response pairs in parallel on a single transport connection. Both are important to reduce latency for HTTP’s primary use case.

HTTP can be combined with security mechanisms, such as TLS (denoted by HTTPS). This adds protocol properties provided by such a mechanism (e.g., authentication, encryption). The TLS Application-Layer Protocol Negotiation (ALPN) extension [RFC7301] can be used to negotiate the HTTP version within the TLS handshake, eliminating the latency incurred by additional round-trip exchanges. Arbitrary cookie strings, included as part of the request headers, are often used as bearer tokens in HTTP.

3.9.2. Interface Description

There are many HTTP libraries available exposing different APIs. The APIs provide a way to specify a request by providing a URI, a method, request modifiers and optionally a request body. For the response, callbacks can be registered that will be invoked when the response is received. If HTTPS is used, the API exposes a registration of callbacks for a server that requests client authentication and when certificate verification is needed.

The World Wide Web Consortium (W3C) has standardized the XMLHttpRequest API [XHR]. This API can be used for sending HTTP/HTTPS requests and receiving server responses. Besides the XML data format, the request and response data format can also be JSON, HTML, and plain text. JavaScript and XMLHttpRequest are ubiquitous programming models for websites, and more general applications, where native code is less attractive.

3.9.3. Transport features

The transport features provided by HTTP, when used as a pseudo-transport, are:

- unicast transport (provided by the lower layer protocol, usually TCP),

- uni- or bidirectional communication,

- transfer of objects in multiple streams with object content type negotiation, supporting partial transmission of object ranges,

- ordered delivery (provided by the lower layer protocol, usually TCP),

- fully reliable delivery (provided by the lower layer protocol, usually TCP),

- flow control (provided by the lower layer protocol, usually TCP).

- congestion control (provided by the lower layer protocol, usually TCP).

HTTPS (HTTP over TLS) additionally provides the following features (as provided by TLS):

- authentication (of one or both ends of a connection),

- confidentiality,

- integrity protection.

3.10. File Delivery over Unidirectional Transport/Asynchronous Layered Coding Reliable Multicast (FLUTE/ALC)

FLUTE/ALC is an IETF standards track protocol specified in [RFC6726] and [RFC5775]. It provides object-oriented delivery of discrete data or files. Asynchronous Layer Coding (ALC) provides an underlying reliable transport service and FLUTE a file-oriented specialization of the ALC service (e.g., to carry associated metadata). The [RFC6726] and [RFC5775] protocols are non-backward-compatible updates of the [RFC3926] and [RFC3450] experimental protocols; these experimental protocols are currently largely deployed in the 3GPP Multimedia Broadcast and Multicast Services (MBMS) (see [MBMS], section 7) and similar contexts (e.g., the Japanese ISDB-Tmm standard).

The FLUTE/ALC protocol has been designed to support massively scalable reliable bulk data dissemination to receiver groups of arbitrary size using IP Multicast over any type of delivery network, including unidirectional networks (e.g., broadcast wireless channels). However, the FLUTE/ALC protocol also supports point-to-point unicast transmissions.

FLUTE/ALC bulk data dissemination has been designed for discrete file or memory-based "objects". Although FLUTE/ALC is not well adapted to byte- and message-streaming, there is an exception: FLUTE/ALC is used to carry 3GPP Dynamic Adaptive Streaming over HTTP (DASH) when scalability is a requirement (see [MBMS], section 5.6).

FLUTE/ALC's reliability, delivery mode, congestion control, and flow/rate control mechanisms can be separately controlled to meet different application needs. Section 4.1 of [I-D.ietf-tsvwg-rfc5405bis] describes multicast congestion control requirements for UDP.

3.10.1. Protocol Description

The FLUTE/ALC protocol works on top of UDP (though it could work on top of any datagram delivery transport protocol), without requiring any connectivity from receivers to the sender. Purely unidirectional networks are therefore supported by FLUTE/ALC. This guarantees scalability to an unlimited number of receivers in a session, since the sender behaves exactly the same regardless of the number of receivers.

FLUTE/ALC supports the transfer of bulk objects such as file or in-memory content, using either a push or an on-demand mode. in push mode, content is sent once to the receivers, while in on-demand mode, content is sent continuously during periods of time that can greatly exceed the average time required to download the session objects (see [RFC5651], section 4.2).

This enables receivers to join a session asynchronously, at their own discretion, receive the content and leave the session. In this case, data content is typically sent continuously, in loops (also known as "carousels"). FLUTE/ALC also supports the transfer of an object stream, with loose real-time constraints. This is particularly useful to carry 3GPP DASH when scalability is a requirement and unicast transmissions over HTTP cannot be used ([MBMS], section 5.6). In this case, packets are sent in sequence using push mode. FLUTE/ALC is not well adapted to byte- and message-streaming and other solutions could be preferred (e.g., FECFRAME [RFC6363] with real-time flows).

The FLUTE file delivery instantiation of ALC provides a metadata delivery service. Each object of the FLUTE/ALC session is described in a dedicated entry of a File Delivery Table (FDT), using an XML format (see [RFC6726], section 3.2). This metadata can include, but is not restricted to, a URI attribute (to identify and locate the object), a media type attribute, a size attribute, an encoding attribute, or a message digest attribute. Since the set of objects sent within a session can be dynamic, with new objects being added and old ones removed, several instances of the FDT can be sent and a mechanism is provided to identify a new FDT Instance.

Error detection and verification of the protocol control information relies on the on the underlying transport (e.g., UDP checksum).

To provide robustness against packet loss and improve the efficiency of the on-demand mode, FLUTE/ALC relies on packet erasure coding (AL-FEC). AL-FEC encoding is proactive (since there is no feedback and therefore no (N)ACK-based retransmission) and ALC packets containing repair data are sent along with ALC packets containing source data.

Several FEC Schemes have been standardized; FLUTE/ALC does not mandate the use of any particular one. Several strategies concerning the transmission order of ALC source and repair packets are possible, in particular in on-demand mode where it can deeply impact the service provided (e.g., to favor the recovery of objects in sequence, or at the other extreme, to favor the recovery of all objects in parallel), and FLUTE/ALC does not mandate nor recommend the use of any particular one.

A FLUTE/ALC session is composed of one or more channels, associated to different destination unicast and/or multicast IP addresses. ALC packets are sent in those channels at a certain transmission rate, with a rate that often differs depending on the channel. FLUTE/ALC does not mandate nor recommend any strategy to select which ALC packet to send on which channel. FLUTE/ALC can use a multiple rate congestion control building block (e.g., WEBRC) to provide congestion control that is feedback free, where receivers adjust their reception rates individually by joining and leaving channels associated with the session. To that purpose, the ALC header provides a specific field to carry congestion control specific information. However FLUTE/ALC does not mandate the use of a particular congestion control mechanism although WEBRC is mandatory to support for the Internet ([RFC6726], section 1.1.4). FLUTE/ALC is often used over a network path with pre-provisioned capacity [I-D.ietf-tsvwg-rfc5405bis] where there are no flows competing for capacity. In this case, a sender-based rate control mechanism and a single channel is sufficient.

[RFC6584] provides per-packet authentication, integrity, and anti-replay protection in the context of the ALC and NORM protocols. Several mechanisms are proposed that seamlessly integrate into these protocols using the ALC and NORM header extension mechanisms.

3.10.2. Interface Description

The FLUTE/ALC specification does not describe a specific application programming interface (API) to control protocol operation. Although open source and commercial implementations have specified APIs, there is no IETF- specified API for FLUTE/ALC.

3.10.3. Transport Features

The transport features provided by FLUTE/ALC are:

- unicast, multicast, anycast or IPv4 broadcast transmission,

- object-oriented delivery of discrete data or files and associated metadata,

- fully reliable or partially reliable delivery (of file or in-memory objects), using proactive packet erasure coding (AL-FEC) to recover from packet erasures,
- ordered or unordered delivery (of file or in-memory objects),
- error detection (based on the UDP checksum),
- per-packet authentication,
- per-packet integrity,
- per-packet replay protection,
- congestion control for layered flows (e.g., with WEBRC).

3.11. NACK-Oriented Reliable Multicast (NORM)

NORM is an IETF standards track protocol specified in [RFC5740]. It provides object-oriented delivery of discrete data or files.

The protocol was designed to support reliable bulk data dissemination to receiver groups using IP Multicast but also provides for point-to-point unicast operation. Support for bulk data dissemination includes discrete file or computer memory-based "objects" as well as byte- and message-streaming.

NORM can incorporate packet erasure coding as a part of its selective ARQ in response to negative acknowledgments from the receiver. The packet erasure coding can also be proactively applied for forward protection from packet loss. NORM transmissions are governed by TCP-friendly multicast congestion control (TFMCC, [RFC4654]). The reliability, congestion control and flow control mechanisms can be separately controlled to meet different application needs.

3.11.1. Protocol Description

The NORM protocol is encapsulated in UDP datagrams and thus provides multiplexing for multiple sockets on hosts using port numbers. For loosely coordinated IP Multicast, NORM is not strictly connection-oriented although per-sender state is maintained by receivers for protocol operation. [RFC5740] does not specify a handshake protocol for connection establishment. Separate session initiation can be used to coordinate port numbers. However, in-band "client-server" style connection establishment can be accomplished with the NORM congestion control signaling messages using port binding techniques like those for TCP client-server connections.

NORM supports bulk "objects" such as file or in-memory content but also can treat a stream of data as a logical bulk object for purposes of packet erasure coding. In the case of stream transport, NORM can support either byte streams or message streams where application-defined message boundary information is carried in the NORM protocol messages. This allows the receiver(s) to join/re-join and recover message boundaries mid-stream as needed. Application content is carried and identified by the NORM protocol with encoding symbol identifiers depending upon the Forward Error Correction (FEC) Scheme [RFC3452] configured. NORM uses NACK-based selective ARQ to reliably deliver the application content to the receiver(s). NORM proactively measures round-trip timing information to scale ARQ timers appropriately and to support congestion control. For multicast operation, timer-based feedback suppression is uses to achieve group size scaling with low feedback traffic levels. The feedback suppression is not applied for unicast operation.

NORM uses rate-based congestion control based upon the TCP-Friendly Rate Control (TFRC) [RFC4324] principles that are also used in DCCP [RFC4340]. NORM uses control messages to measure RTT and collect congestion event information (e.g., reflecting a loss event or ECN event) from the receiver(s) to support dynamic adjustment or the rate. The TCP-Friendly Multicast Congestion Control (TFMCC) [RFC4654] provides extra features to support multicast, but is functionally equivalent to TFRC for unicast.

Error detection and verification of the protocol control information relies on the on the underlying transport(e.g., UDP checksum).

The reliability mechanism is decoupled from congestion control. This allows invocation of alternative arrangements of transport services. For example, to support, fixed-rate reliable delivery or unreliable delivery (that may optionally be "better than best effort" via packet erasure coding) using TFRC. Alternative congestion control techniques may be applied. For example, TFRC rate control with congestion event detection based on ECN.

While NORM provides NACK-based reliability, it also supports a positive acknowledgment (ACK) mechanism that can be used for receiver flow control. This mechanism is decoupled from the reliability and congestion control, supporting applications with different needs. One example is use of NORM for quasi-reliable delivery, where timely delivery of newer content may be favored over completely reliable delivery of older content within buffering and RTT constraints.

3.11.2. Interface Description

The NORM specification does not describe a specific application programming interface (API) to control protocol operation. A freely-available, open source reference implementation of NORM is available at https://www.nrl.navy.mil/itd/ncs/products/norm, and a documented API is provided for this implementation. While a sockets-like API is not currently documented, the existing API supports the necessary functions for that to be implemented.

3.11.3. Transport Features

The transport features provided by NORM are:

- unicast or multicast transport,
- unidirectional communication,
- stream-oriented delivery in a single stream or object-oriented delivery of in-memory data or file bulk content objects,
- fully reliable (NACK-based) or partially reliable (using erasure coding both proactively and as part of ARQ) delivery,
- unordered delivery,
- error detection (relies on UDP checksum),
- segmentation,
- data bundling (using Nagle’s algorithm),
- flow control (timer-based and/or ack-based),
- congestion control (also supporting fixed rate reliable or unreliable delivery).

3.12. Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol (ICMP) [RFC0792] for IPv4 and ICMP for IPv6 [RFC4443] are IETF standards track protocols. It is a connection-less unidirectional protocol that delivers individual messages, without error correction, congestion control, or flow control. Messages may be sent as unicast, IPv4 broadcast or multicast datagrams (IPv4 and IPv6), in addition to anycast datagrams.

While ICMP is not typically described as a transport protocol, it does position itself over the network layer, and the operation of other transport protocols can be closely linked to the functions provided by ICMP.

Transport Protocols and upper layer protocols can use received ICMP messages to help them take appropriate decisions when network or endpoint errors are reported. For example, to implement, ICMP-based Path MTU discovery [RFC1191][RFC1981] or assist in Packetization Layer Path MTU Discovery (PMTUD) [RFC4821]. Such reactions to received messages need to protect from off-path data injection [I-D.ietf-tsvwg-rfc5405bis], to avoid an application receiving packets created by an unauthorized third party. An application therefore needs to ensure that all messages are appropriately validated, by checking the payload of the messages to ensure these are received in response to actually transmitted traffic (e.g., a reported error condition that corresponds to a UDP datagram or TCP segment was actually sent by the application). This requires context [RFC6056], such as local state about communication instances to each destination (e.g., in the TCP, DCCP, or SCTP protocols). This state is not always maintained by UDP-based applications [I-D.ietf-tsvwg-rfc5405bis].

3.12.1. Protocol Description

ICMP is a connection-less unidirectional protocol, It delivers independent messages, called datagrams. Each message is required to carry a checksum as an integrity check and to protect from mis-delivery to an unintended endpoint.

ICMP messages typically relay diagnostic information from an endpoint [RFC1122] or network device [RFC1812] addressed to the sender of a flow. This usually contains the network protocol header of a packet that encountered a reported issue. Some formats of messages can also carry other payload data. Each message carries an integrity check calculated in the same way as for UDP, this checksum is not optional.

The RFC series defines additional IPv6 message formats to support a range of uses. In the case of IPv6 the protocol incorporates neighbor discovery [RFC2461] [RFC3971] (provided by ARP for IPv4) and the Multicast Listener Discovery (MLD) [RFC2710] group management functions (provided by IGMP for IPv4).

Reliable transmission can not be assumed. A receiving application that is unable to run sufficiently fast, or frequently, may miss messages since there is no flow or congestion control. In addition some network devices rate-limit ICMP messages.

3.12.2. Interface Description

ICMP processing is integrated in many connection-oriented transports, but like other functions needs to be provided by an upper-layer protocol when using UDP and UDP-Lite.

On some stacks, a bound socket also allows a UDP application to be notified when ICMP error messages are received for its transmissions [I-D.ietf-tsvwg-rfc5405bis].

Any response to ICMP error messages ought to be robust to temporary routing failures (sometimes called "soft errors"), e.g., transient ICMP "unreachable" messages ought to not normally cause a communication abort [RFC5461] [I-D.ietf-tsvwg-rfc5405bis].

3.12.3. Transport Features

ICMP does not provide any transport service directly to applications. Used together with other transport protocols, it provides transmission of control, error, and measurement data between endpoints, or from devices along the path to one endpoint.

4. Congestion Control

Congestion control is critical to the stable operation of the Internet. A variety of mechanisms are used to provide the congestion control needed by many Internet transport protocols. Congestion is detected based on sensing of network conditions, whether through explicit or implicit feedback. The congestion control mechanisms that can be applied by different transport protocols are largely orthogonal to the choice of transport protocol. This section provides an overview of the congestion control mechanisms available to the protocols described in Section 3.

Many protocols use a separate window to determine the maximum sending rate that is allowed by the congestion control. The used congestion control mechanism will increase the congestion window if feedback is received that indicates that the currently used network path is not congested, and will reduce the window otherwise. Window-based mechanisms often increase their window slowing over multiple RTTs, while decreasing strongly when the first indication of congestion is received. One example is an Additive Increase Multiplicative Decrease (AIMD) scheme, where the window is increased by a certain number of packets/bytes for each data segment that has been successfully transmitted, while the window decreases multiplicatively on the occurrence of a congestion event. This can lead to a rather unstable, oscillating sending rate, but will resolve a congestion situation quickly. TCP New Reno [RFC5681] which is one of the initial proposed schemes for TCP as well as TCP Cubic [I-D.ietf-tcpm-cubic] which is the default mechanism for TCP in Linux are two examples for window-based AIMD schemes. This approach is also used by DCCP CCID-2 for datagram congestion control.

Some classes of applications prefer to use a transport service that allows sending at a more stable rate, that is slowly varied in response to congestion. Rate-based methods offer this type of congestion control and have been defined based on the loss ratio and observed round trip time, such as TFRC [RFC5348] and TFRC-SP [RFC4828]. These methods utilize a throughput equation to determine the maximum acceptable rate. Such methods are used with DCCP CCID-3 [RFC4342] and CCID-4 [RFC5622], WEBRC [RFC3738], and other applications.

Another class of applications prefer a transport service that yields to other (higher-priority) traffic, such as interactive transmissions. While most traffic in the Internet uses loss-based congestion control and therefore tends to fill the network buffers (to a certain level if Active Queue Management (AQM) is used), low-priority congestion control methods often react to changes in delay as an earlier indication of congestion. This approach tends to induce less loss than a loss-based method but does generally not compete well with loss-based traffic across shared bottleneck links. Therefore, methods such as LEDBAT [RFC6824], are deployed in the Internet for scavenger traffic that aim to only utilize otherwise unused capacity.

5. Transport Features

The transport protocol features described in this document can be used as a basis for defining common transport features, listed below with the protocols supporting them:

- o Control Functions
  - * Addressing
    - + unicast (TCP, MPTCP, UDP, UDP-Lite, SCTP, DCCP, TLS, RTP, HTTP, ICMP)
    - + multicast (UDP, UDP-Lite, RTP, ICMP, FLUTE/ALC, NORM). Note that, as TLS and DTLS are unicast-only, there is no widely deployed mechanism for supporting the features in the Security section below when using multicast addressing.
    - + IPv4 broadcast (UDP, UDP-Lite, ICMP)

    + anycast (UDP, UDP-Lite). Connection-oriented protocols such as TCP and DCCP have also been deployed using anycast addressing, with the risk that routing changes may cause connection failure.

  * Association type

    + connection-oriented (TCP, MPTCP, DCCP, SCTP, TLS, RTP, HTTP, NORM)

    + connectionless (UDP, UDP-Lite, FLUTE/ALC)

  * Multihoming support

    + resilience and mobility (MPTCP, SCTP)

    + load-balancing (MPTCP)

    + address family multiplexing (MPTCP, SCTP)

  * Middlebox cooperation

    + application-class signaling to middleboxes (DCCP)

    + error condition signaling from middleboxes and routers to endpoints (ICMP)

  * Signaling

    + control information and error signaling (ICMP)

    + application performance reporting (RTP)

o Delivery

  * Reliability

    + fully reliable delivery (TCP, MPTCP, SCTP, TLS, HTTP, FLUTE/ALC, NORM)

    + partially reliable delivery (SCTP, NORM)

      - using packet erasure coding (RTP, FLUTE/ALC, NORM)

      - with specified policy for dropped messages (SCTP)

    + unreliable delivery (SCTP, UDP, UDP-Lite, DCCP, RTP)

            - with drop notification to sender (SCTP, DCCP, RTP)

         + error detection

            - checksum for error detection (TCP, MPTCP, UDP, UDP-Lite, SCTP, DCCP, TLS, DTLS, FLUTE/ALC, NORM, ICMP)

            - partial payload checksum protection (UDP-Lite, DCCP). Some uses of RTP can exploit partial payload checksum protection feature to provide a corruption tolerant transport service.

            - checksum optional (UDP). Possible with IPv4 and in certain cases with IPv6.

      * Ordering

         + ordered delivery (TCP, MPTCP, SCTP, TLS, RTP, HTTP, FLUTE)

         + unordered delivery permitted (UDP, UDP-Lite, SCTP, DCCP, RTP, NORM)

      * Type/framing

         + stream-oriented delivery (TCP, MPTCP, SCTP, TLS, HTTP)

            - with multiple streams per association (SCTP, HTTP2)

         + message-oriented delivery (UDP, UDP-Lite, SCTP, DCCP, DTLS, RTP)

         + object-oriented delivery of discrete data or files and associated metadata (HTTP, FLUTE/ALC, NORM)

            - with partial delivery of object ranges (HTTP)

      * Directionality

         + unidirectional (UDP, UDP-Lite, DCCP, RTP, FLUTE/ALC, NORM)

         + bidirectional (TCP, MPTCP, SCTP, TLS, HTTP)

   o Transmission control

      * flow control (TCP, MPTCP, SCTP, DCCP, TLS, RTP, HTTP)

   * congestion control (TCP, MPTCP, SCTP, DCCP, RTP, FLUTE/ALC, NORM). Congestion control can also provided by the transport supporting an upper later transport (e.g., TLS, RTP, HTTP).

   * segmentation (TCP, MPTCP, SCTP, TLS, RTP, HTTP, FLUTE/ALC, NORM)

   * data/message bundling (TCP, MPTCP, SCTP, TLS, HTTP)

   * stream scheduling prioritization (SCTP, HTTP2)

   * endpoint multiplexing (MPTCP)

- Security

   * authentication of one end of a connection (TLS, DTLS, FLUTE/ALC)

   * authentication of both ends of a connection (TLS, DTLS)

   * confidentiality (TLS, DTLS)

   * cryptographic integrity protection (TLS, DTLS)

   * replay protection (TLS, DTLS, FLUTE/ALC)

6. IANA Considerations

This document has no considerations for IANA.

7. Security Considerations

This document surveys existing transport protocols and protocols providing transport-like services. Confidentiality, integrity, and authenticity are among the features provided by those services. This document does not specify any new features or mechanisms for providing these features. Each RFC referenced by this document discusses the security considerations of the specification it contains.

8. Contributors

In addition to the editors, this document is the work of Brian Adamson, Dragana Damjanovic, Kevin Fall, Simone Ferlin-Oliviera, Ralph Holz, Olivier Mehani, Karen Nielsen, Colin Perkins, Vincent Roca, and Michael Tuexen.

- Section 3.2 on MPTCP was contributed by Simone Ferlin-Oliviera (ferlin@simula.no) and Olivier Mehani (olivier.mehani@nicta.com.au)

- Section 3.3 on UDP was contributed by Kevin Fall (kfall@kfall.com)

- Section 3.5 on SCTP was contributed by Michael Tuexen (tuexen@fh-muenster.de) and Karen Nielsen (karen.nielsen@tieto.com)

- Section 3.7 on TLS and DTLS was contributed by Ralph Holz (ralph.holz@nicta.com.au) and Olivier Mehani (olivier.mehani@nicta.com.au)

- Section 3.8 on RTP contains contributions from Colin Perkins (csp@csperkins.org)

- Section 3.9 on HTTP was contributed by Dragana Damjanovic (ddamjanovic@mozilla.com)

- Section 3.10 on FLUTE/ALC was contributed by Vincent Roca (vincent.roca@inria.fr)

- Section 3.11 on NORM was contributed by Brian Adamson (brian.adamson@nrl.navy.mil)

9. Acknowledgments

Thanks to Joe Touch, Michael Welzl, Spencer Dawkins, and the TAPS Working Group for the comments, feedback, and discussion. This work is supported by the European Commission under grant agreement No. 318627 mPlane and from the Horizon 2020 research and innovation program under grant agreements No. 644334 (NEAT) and No. 688421 (MAMI). This support does not imply endorsement.

10. Informative References

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

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <http://www.rfc-editor.org/info/rfc792>.

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

[RFC0896] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <http://www.rfc-editor.org/info/rfc896>.

[RFC1071] Braden, R., Borman, D., and C. Partridge, "Computing the Internet checksum", RFC 1071, DOI 10.17487/RFC1071, September 1988, <http://www.rfc-editor.org/info/rfc1071>.

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

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

[RFC1812] Baker, F., Ed., "Requirements for IP Version 4 Routers", RFC 1812, DOI 10.17487/RFC1812, June 1995, <http://www.rfc-editor.org/info/rfc1812>.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, DOI 10.17487/RFC1981, August 1996, <http://www.rfc-editor.org/info/rfc1981>.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <http://www.rfc-editor.org/info/rfc2018>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <http://www.rfc-editor.org/info/rfc2045>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <http://www.rfc-editor.org/info/rfc2460>.

[RFC2461] Narten, T., Nordmark, E., and W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)", RFC 2461, DOI 10.17487/RFC2461, December 1998, <http://www.rfc-editor.org/info/rfc2461>.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, DOI 10.17487/RFC2617, June 1999, <http://www.rfc-editor.org/info/rfc2617>.

[RFC2710] Deering, S., Fenner, W., and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6", RFC 2710, DOI 10.17487/RFC2710, October 1999, <http://www.rfc-editor.org/info/rfc2710>.

[RFC2736] Handley, M. and C. Perkins, "Guidelines for Writers of RTP Payload Format Specifications", BCP 36, RFC 2736, DOI 10.17487/RFC2736, December 1999, <http://www.rfc-editor.org/info/rfc2736>.

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

[RFC3205] Moore, K., "On the use of HTTP as a Substrate", BCP 56, RFC 3205, DOI 10.17487/RFC3205, February 2002, <http://www.rfc-editor.org/info/rfc3205>.

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

[RFC3436] Jungmaier, A., Rescorla, E., and M. Tuexen, "Transport Layer Security over Stream Control Transmission Protocol", RFC 3436, DOI 10.17487/RFC3436, December 2002, <http://www.rfc-editor.org/info/rfc3436>.

[RFC3450] Luby, M., Gemmell, J., Vicisano, L., Rizzo, L., and J. Crowcroft, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 3450, DOI 10.17487/RFC3450, December 2002, <http://www.rfc-editor.org/info/rfc3450>.

[RFC3452] Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley, M., and J. Crowcroft, "Forward Error Correction (FEC) Building Block", RFC 3452, DOI 10.17487/RFC3452, December 2002, <http://www.rfc-editor.org/info/rfc3452>.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <http://www.rfc-editor.org/info/rfc3550>.

[RFC3738] Luby, M. and V. Goyal, "Wave and Equation Based Rate Control (WEBRC) Building Block", RFC 3738, DOI 10.17487/RFC3738, April 2004, <http://www.rfc-editor.org/info/rfc3738>.

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

[RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., Ed., and G. Fairhurst, Ed., "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, DOI 10.17487/RFC3828, July 2004, <http://www.rfc-editor.org/info/rfc3828>.

[RFC3926] Paila, T., Luby, M., Lehtonen, R., Roca, V., and R. Walsh, "FLUTE - File Delivery over Unidirectional Transport", RFC 3926, DOI 10.17487/RFC3926, October 2004, <http://www.rfc-editor.org/info/rfc3926>.

[RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005, <http://www.rfc-editor.org/info/rfc3971>.

[RFC4324] Royer, D., Babics, G., and S. Mansour, "Calendar Access Protocol (CAP)", RFC 4324, DOI 10.17487/RFC4324, December 2005, <http://www.rfc-editor.org/info/rfc4324>.

[RFC4336] Floyd, S., Handley, M., and E. Kohler, "Problem Statement for the Datagram Congestion Control Protocol (DCCP)", RFC 4336, DOI 10.17487/RFC4336, March 2006, <http://www.rfc-editor.org/info/rfc4336>.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <http://www.rfc-editor.org/info/rfc4340>.

[RFC4341] Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control", RFC 4341, DOI 10.17487/RFC4341, March 2006, <http://www.rfc-editor.org/info/rfc4341>.

[RFC4342] Floyd, S., Kohler, E., and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)", RFC 4342, DOI 10.17487/RFC4342, March 2006, <http://www.rfc-editor.org/info/rfc4342>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, DOI 10.17487/RFC4443, March 2006, <http://www.rfc-editor.org/info/rfc4443>.

[RFC4654] Widmer, J. and M. Handley, "TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification", RFC 4654, DOI 10.17487/RFC4654, August 2006, <http://www.rfc-editor.org/info/rfc4654>.

[RFC4820] Tuexen, M., Stewart, R., and P. Lei, "Padding Chunk and Parameter for the Stream Control Transmission Protocol (SCTP)", RFC 4820, DOI 10.17487/RFC4820, March 2007, <http://www.rfc-editor.org/info/rfc4820>.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <http://www.rfc-editor.org/info/rfc4821>.

[RFC4828] Floyd, S. and E. Kohler, "TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant", RFC 4828, DOI 10.17487/RFC4828, April 2007, <http://www.rfc-editor.org/info/rfc4828>.

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

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <http://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, <http://www.rfc-editor.org/info/rfc5061>.

[RFC5097] Renker, G. and G. Fairhurst, "MIB for the UDP-Lite protocol", RFC 5097, DOI 10.17487/RFC5097, January 2008, <http://www.rfc-editor.org/info/rfc5097>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5238] Phelan, T., "Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)", RFC 5238, DOI 10.17487/RFC5238, May 2008, <http://www.rfc-editor.org/info/rfc5238>.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <http://www.rfc-editor.org/info/rfc5348>.

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

[RFC5595] Fairhurst, G., "The Datagram Congestion Control Protocol (DCCP) Service Codes", RFC 5595, DOI 10.17487/RFC5595, September 2009, <http://www.rfc-editor.org/info/rfc5595>.

[RFC5596] Fairhurst, G., "Datagram Congestion Control Protocol (DCCP) Simultaneous-Open Technique to Facilitate NAT/Middlebox Traversal", RFC 5596, DOI 10.17487/RFC5596, September 2009, <http://www.rfc-editor.org/info/rfc5596>.

[RFC5622] Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion ID 4: TCP-Friendly Rate Control for Small Packets (TFRC-SP)", RFC 5622, DOI 10.17487/RFC5622, August 2009, <http://www.rfc-editor.org/info/rfc5622>.

[RFC5651] Luby, M., Watson, M., and L. Vicisano, "Layered Coding Transport (LCT) Building Block", RFC 5651, DOI 10.17487/RFC5651, October 2009, <http://www.rfc-editor.org/info/rfc5651>.

[RFC5672] Crocker, D., Ed., "RFC 4871 DomainKeys Identified Mail (DKIM) Signatures -- Update", RFC 5672, DOI 10.17487/RFC5672, August 2009, <http://www.rfc-editor.org/info/rfc5672>.

[RFC5740] Adamson, B., Bormann, C., Handley, M., and J. Macker, "NACK-Oriented Reliable Multicast (NORM) Transport Protocol", RFC 5740, DOI 10.17487/RFC5740, November 2009, <http://www.rfc-editor.org/info/rfc5740>.

[RFC5775] Luby, M., Watson, M., and L. Vicisano, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 5775, DOI 10.17487/RFC5775, April 2010, <http://www.rfc-editor.org/info/rfc5775>.

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

[RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, DOI 10.17487/RFC6056, January 2011, <http://www.rfc-editor.org/info/rfc6056>.

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

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

[RFC6101] Freier, A., Karlton, P., and P. Kocher, "The Secure Sockets Layer (SSL) Protocol Version 3.0", RFC 6101, DOI 10.17487/RFC6101, August 2011, <http://www.rfc-editor.org/info/rfc6101>.

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

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6356] Raiciu, C., Handley, M., and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols", RFC 6356, DOI 10.17487/RFC6356, October 2011, <http://www.rfc-editor.org/info/rfc6356>.

[RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, DOI 10.17487/RFC6363, October 2011, <http://www.rfc-editor.org/info/rfc6363>.

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

[RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, DOI 10.17487/RFC6582, April 2012, <http://www.rfc-editor.org/info/rfc6582>.

[RFC6584] Roca, V., "Simple Authentication Schemes for the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols", RFC 6584, DOI 10.17487/RFC6584, April 2012, <http://www.rfc-editor.org/info/rfc6584>.

[RFC6726] Paila, T., Walsh, R., Luby, M., Roca, V., and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport", RFC 6726, DOI 10.17487/RFC6726, November 2012, <http://www.rfc-editor.org/info/rfc6726>.

[RFC6773] Phelan, T., Fairhurst, G., and C. Perkins, "DCCP-UDP: A Datagram Congestion Control Protocol UDP Encapsulation for NAT Traversal", RFC 6773, DOI 10.17487/RFC6773, November 2012, <http://www.rfc-editor.org/info/rfc6773>.

[RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, DOI 10.17487/RFC6817, December 2012, <http://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, <http://www.rfc-editor.org/info/rfc6824>.

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

[RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", RFC 6935, DOI 10.17487/RFC6935, April 2013, <http://www.rfc-editor.org/info/rfc6935>.

[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, DOI 10.17487/RFC6936, April 2013, <http://www.rfc-editor.org/info/rfc6936>.

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

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

[RFC7202] Perkins, C. and M. Westerlund, "Securing the RTP Framework: Why RTP Does Not Mandate a Single Media Security Solution", RFC 7202, DOI 10.17487/RFC7202, April 2014, <http://www.rfc-editor.org/info/rfc7202>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <http://www.rfc-editor.org/info/rfc7232>.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <http://www.rfc-editor.org/info/rfc7233>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <http://www.rfc-editor.org/info/rfc7234>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <http://www.rfc-editor.org/info/rfc7235>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <http://www.rfc-editor.org/info/rfc7301>.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <http://www.rfc-editor.org/info/rfc7323>.

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

[RFC7457] Sheffer, Y., Holz, R., and P. Saint-Andre, "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)", RFC 7457, DOI 10.17487/RFC7457, February 2015, <http://www.rfc-editor.org/info/rfc7457>.

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

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

[I-D.ietf-tsvwg-rfc5405bis]
Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", draft-ietf-tsvwg-rfc5405bis-16 (work in progress), July 2016.

[I-D.ietf-tsvwg-sctp-dtls-encaps]
Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "DTLS Encapsulation of SCTP Packets", draft-ietf-tsvwg-sctp-dtls-encaps-09 (work in progress), January 2015.

[I-D.ietf-tsvwg-sctp-ndata]
Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol", draft-ietf-tsvwg-sctp-ndata-07 (work in progress), July 2016.

[I-D.ietf-tsvwg-natsupp]
Stewart, R., Tuexen, M., and I. Ruengeler, "Stream Control Transmission Protocol (SCTP) Network Address Translation Support", draft-ietf-tsvwg-natsupp-09 (work in progress), May 2016.

[I-D.ietf-tcpm-cubic]
Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", draft-ietf-tcpm-cubic-02 (work in progress), August 2016.

[I-D.ietf-rtcweb-transports]
Alvestrand, H., "Transports for WebRTC", draft-ietf-rtcweb-transports-15 (work in progress), August 2016.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-14 (work in progress), July 2016.

[XHR] van Kesteren, A., Aubourg, J., Song, J., and H. Steen, "XMLHttpRequest working draft (http://www.w3.org/TR/XMLHttpRequest/)", 2000.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures, Ph. D. (UC Irvine), Chapter 5: Representational State Transfer", 2000.

[POSIX] 1-2008, IEEE., "IEEE Standard for Information Technology -- Portable Operating System Interface (POSIX) Base Specifications, Issue 7", n.d..

[MBMS] 3GPP TSG WS S4, ., "3GPP TS 26.346: Multimedia Broadcast/ Multicast Service (MBMS); Protocols and codecs, release 13 (http://www.3gpp.org/DynaReport/26346.htm).", 2015.

[ClarkArch] Clark, D. and D. Tennenhouse, "Architectural Considerations for a New Generation of Protocols (Proc. ACM SIGCOMM)", 1990.

Authors' Addresses

Godred Fairhurst (editor)
University of Aberdeen
School of Engineering, Fraser Noble Building
Aberdeen AB24 3UE

Email: gorry@erg.abdn.ac.uk

Brian Trammell (editor)
ETH Zurich
Gloriastrasse 35
8092 Zurich
Switzerland

Email: ietf@trammell.ch

Mirja Kuehlewind (editor)
ETH Zurich
Gloriastrasse 35
8092 Zurich
Switzerland

Email: mirja.kuehlewind@tik.ee.ethz.ch

TAPS M. Welzl
Internet-Draft University of Oslo
Intended status: Informational M. Tuexen
Expires: March 24, 2016 Muenster Univ. of Appl. Sciences
N. Khademi
University of Oslo
September 21, 2015

An Approach to Identify Services Provided by IETF Transport Protocols and Congestion Control Mechanisms

draft-welzl-taps-transports-00

Abstract

This document describes a method to identify services in transport protocols and congestion control mechanisms. It shows the approach using TCP and SCTP (base protocol) as examples.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This document considers every form of defined interaction between a transport protocol and its user ("upper layer protocol" or "application") as a "service". Here, the term "service" is NOT the same as the term used to specify the entire "above transport" protocol that maps to a port number or service name (which is another common meaning of the term "service" in the context of transport protocols).

The list of services in this document is strictly based on the parts of relevant protocol specifications that relate to what the protocol provides to an application using it and how the application interacts with it. It is based on text that describes what a protocol provides to the upper layer and how it is used (abstract API descriptions), given for the base protocols in [RFC0793], [RFC1122] and [RFC4960]. It does not cover API instances, for example the one given for SCTP in [RFC6458]. It also does not cover parts of the protocol that are explicitly stated as optional to implement.

The document presents a three-pass process to arrive at a list of transport protocol services. In the first pass, the relevant RFC text is discussed per protocol. In the second pass, this discussion is used to derive a list of services that are uniformly categorized across protocols. Here, an attempt is made to present services in a slightly generalized form to highlight similarities. This is, for example, achieved by renaming "commands" (or "transport primitives") of protocols or by avoiding a strict 1:1-mapping between these commands and services in the list. Finally, the third pass presents all services from pass 2, identifying which protocol implements them.

In the list resulting from the second pass, some services are missing because they are implicit in some protocols, and they only become explicit when we consider the superset of all services offered by all protocols. For example, TCP's reliability includes integrity via a checksum, but we have to include a protocol like UDP-Lite as specified in [RFC3828] (which has a configurable checksum) in the list to consider an always-on checksum as a service (it would not be a service if no protocol would allow to disable / configure the checksum). Similar arguments apply to other protocol functions (e.g. congestion control). The complete list of services across all protocols is therefore only available after pass 3.

2. General Considerations

This document discusses unicast [AUTHOR'S NOTE: for simplicity, for now. Hopefully forever, for simplicity.] transport protocols.

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

Some protocols are connection-oriented. Connection-orientation, to the user of an application, means that there is state shared between the endpoints that persists across messages. Connection-oriented protocols often use an initial call to "open" a connection before communication can progress, and require communication to be explicitly terminated by issuing a "close" call. Moreover, a "connection" is the common state that some transport primitives refer to, e.g. to adjust general configuration settings. Connections establishment, maintenance and termination are therefore used to categorize certain services of connection-oriented transport protocols in pass 2 and 3.

3. Pass 1

In this first iteration, the relevant text parts of the RFCs describing the protocols are summarized, focusing on what a protocol provides to the upper layer and how it is used (abstract API descriptions). The resulting text is somewhat heterogeneous in terminology - e.g. the user of the protocol is called "Application" in TCP and "Upper-Layer Protocol (ULP)" in SCTP, and TCP's "user commands" are called "ULP primitives" in SCTP.

3.1. Services Provided by TCP

[RFC0793] states: "TCP is a connection-oriented, end-to-end reliable protocol (..)". Section 3.8 in [RFC0793] further specifies the interaction with the application by listing several user commands. It is also assumed that the Operating System provides a means for TCP to asynchronously signal the user program. Here, we describe the relevant user commands and notifications to the application.

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

Also optional, for multihomed hosts, the local IP address can be provided [RFC1122]. If it is not provided, a default choice will be made in case of active open calls. A passive open call will await incoming connection requests to all local addresses and then maintain usage of the local IP address where the incoming connection request has arrived. Finally, the 'options' parameter is explained in [RFC1122] to let the application specify IP options such as source route, record route, or timestamp. (It is not stated on which segments of a connection these options should be applied, but probably all segments, as this is also stated in a specification given for the usage of source route (section 4.2.3.8 of [RFC1122]). As the only non-optional IP option in this parameter, an application can specify a source route when it actively opens a TCP connection.

send: this command hands over a provided number of bytes that TCP should reliably send to the other side of the connection. The PUSH flag, if set, requires data to be promptly transmitted to the receiver without delaying it. Conversely, not using PUSH can reduce the number of unnecessary wakeup calls to the receiving application process. [RFC1122] states that "Generally, an interactive application protocol must set the PUSH flag at least in the last SEND call in each command or response sequence. A bulk transfer protocol like FTP should set the PUSH flag on the last segment of a file or when necessary to prevent buffer deadlock." An optional timeout parameter can be provided that updates the connection's timeout (see "open").

receive: This command 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.

close: This command 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 handed over to TCP (and if that fails, 'close' becomes 'abort'). Close also implies push function.

abort: This command causes all pending SENDs and RECEIVES to be aborted, the TCB to be removed, and a special RESET message to be sent to the TCP on the other side of the connection. See [RFC0793].

close event: TCP will signal a user, even if no RECEIVEs are outstanding, that the other side has closed, so the user can terminate his/her side gracefully. See [RFC0793], Section 3.5.

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

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

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

Type-of-Service: Section 4.2.4.2 of [RFC1122] states that the application layer MUST be able to specify the Type-of-Service (TOS) for segments that are sent on a connection. The application should be able to change the TOS during the connection lifetime, and the TOS value should be passed to the IP layer unchanged. Since then, parts of the TOS field have been assigned to ECN [RFC3168] and the six most significant bits have been assigned to DiffServ by the name of DSField [RFC3260]. Staying with the intention behind the application's ability to specify the "Type of Service", this should probably be interpreted to mean the value in the DSField, which is the Differentiated Services Codepoint (DSCP). [AUTHOR's NOTE: text trying to "read between the lines" of RFCs here... this perhaps calls for an update to [RFC1122]?]

Nagle: An application can disable the Nagle algorithm on an individual connection. This algorithm delays sending data for some time to increase the likelihood of sending a full-sized segment.

3.1.1. Excluded Services

The 'send' and 'receive' commands include usage of an "URGENT" mechanism, which SHOULD NOT be implemented according to [RFC6093] and is therefore not described here. This also concerns the notification "Urgent pointer advance" in the ERROR_REPORT described in Section 4.2.4.1 of [RFC1122].

The 'open' command specified in [RFC0793] can be handed optional Precedence or security/compartment information according to [RFC0793], but this was not incuded here because it is mostly irrelevant today, as explained in [RFC7414]. The 'open' command also includes a parameter "options" that is explained in [RFC1122] to let the application specify IP options such as source route, record route, or timestamp. This parameter was not included here because it is not clear which segments of a connection (all?) these options would then be applied to.

The 'status' command was not included because [RFC0793] calls this command "implementation dependent" and states that it "could be excluded without adverse effect". Moreover, while a data block containing specific information is described, it is also stated that not all of this information may always be available. The 'receive' command can (under some conditions) yield the status of the PUSH flag according to [RFC0793], but this TCP functionality is made optional in [RFC1122] and hence not considered here. Generally, section 4.2.2.2 of [RFC1122] says that PUSH on send calls MAY be implemented, which could be a reason not to consider it here. However, the text then explains that "an interactive application protocol must set the PUSH flag at least in the last SEND call in each command or response sequence", and most implementations provide some option to cause a behavior that is in some way similar to PUSH. Therefore PUSH is described as a part of SEND here. [RFC1122] also introduces keep-alives to TCP, but these are optional and hence not considered here. [RFC1122] describes that "some TCP implementations have included a FLUSH call", indicating that this call is optional to implement. It is therefore not considered here.

3.2. Services Provided by SCTP

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

Section 10 of [RFC4960] further specifies the interaction with the application (which RFC [RFC4960] calls the "Upper Layer Protocol" (ULP)). It is assumed that the Operating System provides a means for SCTP to asynchronously signal the user program. Here, we describe the relevant ULP primitives and notifications to the ULP process:

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

Associate: This creates an association (the SCTP equivalent of a connection) between the local SCTP instance and a remote SCTP instance. Most primitives are associated with a specific association, which is assumed to first have been created. Associate can return a list of destination transport addresses so that multiple paths can later be used. One of the transport addresses from the returned destination addresses will be selected by the local endpoint as default primary path for sending SCTP packets to this peer, but this choice can be changed by the ULP using the list of destination addresses. Associate is also given the number of outgoing streams to request and optionally returns the number of outgoing streams negotiated.

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). An optional maximum life time can specify the time after which the message should be discarded rather than sent. A choice (advisory, i.e. not guaranteed) of the preferred path can be made by providing a destination transport address, and the message can be delivered out-of-order if the unordered flag is set. Another advisory flag indicates the ULP’s preference to avoid bundling user data with other outbound DATA chunks (i.e., in the same packet). The handling of this no-bundle flags is similar to the sender side handling of the TCP PUSH flag. A payload protocol-id can be provided to pass a value that indicates the type of payload protocol data to the peer.

Receive: Messages are received from an association, and optionally a stream within the association, with their size returned. The ULP 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 ULP.

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

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

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

Set Protocol Parameters: This allows to set values for protocol parameters per association; for some parameters, a setting can be made per transport address. The set listed in [RFC4960] is: RTO.Initial; RTO.Min; RTO.Max; Max.Burst; RTO.Alpha; RTO.Beta; Valid.Cookie.Life; Association.Max.Retrans; Path.Max.Retrans; Max.Init.Retransmits; HB.interval; HB.Max.Burst.

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

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

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

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

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

NETWORK STATUS CHANGE notification: The NETWORK STATUS CHANGE notification informs the ULP about a transport address becoming active/inactive.

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

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

3.2.1. Excluded Services

For the 'Set Primary' primitive, an optional possibility to specify the source transport address to be used in outgoing IP datagrams is described, but the RFC text says "some implementations may allow you to", indicating that implementing this in SCTP is optional. This functionality is therefore not considered here. The 'Receive' primitive can also return certain additional information, but this is also left up to the implementation and therefore not considered. With a COMMUNICATION LOST notification, some more information may optionally be passed to the ULP (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 what 'Status' provides and is therefore not discussed. Similarly, 'Set Failure Threshold' sets only one out of various possible parameters included in 'Set Protocol Parameters'. The 'Destroy SCTP Instance' primitive was excluded: it erases the SCTP instance that was created by 'Initialize', but this does not translate into a service for the ULP.

4. Pass 2

Here we categorize the services from pass 1 based on whether they relate to a connection or to data transmission. Services are presented following the nomenclature "CATEGORY.[SUBCATEGORY].SERVICENAME.PROTOCOL". We present "connection" as a general protocol-independent concept and use it to refer to both TCP's connections (which are identifiable by a unique socket pair, where a socket is defined as an IP address and TCP port) and SCTP's associations (which are identifiable by multiple IP address and port number pairs). We define the "transport address" as "the combination of an IP address and a transport protocol's port number". The "application" is the user of the protocol (called "Upper-Level Protocol (ULP)" in SCTP).

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

4.1. CONNECTION Related Services

ESTABLISHMENT:
Active creation of a connection from one transport address to one or more transport addresses.

- CONNECT.TCP:
  Command / event: 'open' (active) or 'open' (passive) with destination transport address, followed by 'send'
  Parameters: 1 local IP address (optional); 1 destination transport address (for active open; else the destination transport address and the local IP address of the succeeding incoming connection request will be maintained); timeout (optional); options (optional)
  Comments: If the local IP address is not provided, a default choice will automatically be made. [AUTHOR'S NOTE: [RFC1122] does not clearly state this, but it seems to be the implication of some text there.] The timeout can also be a retransmission count. The

options are IP options to be used on all segments of the connection. At least the Source Route option is mandatory for TCP to provide.

- CONNECT.SCTP:
  Command / event: 'initialize', followed by 'associate'
  Parameters: list of local transport addresses (initialize); 1 destination transport address; outbound stream count
  Returns: destination transport address list
  Comments: 'initialize' needs to be called only once per local transport address list. One destination transport address will automatically be chosen; it can later be changed in MAINTENANCE.

AVAILABILITY:
Preparing to receive incoming connection requests.

- LISTEN.TCP:
  Command / event: 'open' (passive)
  Parameters: 1 local IP address (optional); 1 destination transport address (optional); timeout (optional)
  Comments: if the transport address 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 restraint(s). ESTABLISHMENT can later be done with 'send'.

- LISTEN.SCTP:
  Command / event: 'initialize', followed by 'COMMUNICATION UP' notification
  Parameters: list of local transport addresses (initialize)
  Returns: destination transport address list; outbound stream count; inbound stream count
  Comments: initialize needs to be called only once per local transport address list. COMMUNICATION UP can also follow a COMMUNICATION LOST notification, indicating that the lost communication is restored.

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 when new data are handed over for sending).

o CHANGE-TIMEOUT.TCP:
Command / event: 'send'
Parameters: timeout value
Comments: when sending data, the connection's timeout value (time after which the connection will be aborted if data cannot be delivered) can be adjusted.

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

o DISABLE-NAGLE.TCP:
Command / event: not specified
Parameters: one boolean value
Comments: the Nagle algorithm delays data transmission to increase the chance to send a full-sized segment. An application must be able to disable this algorithm for a connection. This is related to the no-bundle flag in DATA.SEND.SCTP.

o REQUESTHEARTBEAT.SCTP:
Command / event: 'Request HeartBeat'
Parameters: destination transport address
Returns: success or failure
Comments: requests a heartbeat to be immediately carried out on a path, returning success or failure.

o SETPROTOCOLPARAMETERS.SCTP:
Command / event: 'Set Protocol Parameters'
Parameters: RTO.Initial; RTO.Min; RTO.Max; Max.Burst; RTO.Alpha; RTO.Beta; Valid.Cookie.Life; Association.Max.Retrans; Path.Max.Retrans; Max.Init.Retransmits; HB.interval; HB.Max.Burst

o SETPRIMARY.SCTP:
Command / event: 'Set Primary'
Parameters: destination transport address
Returns: result of attempting this operation
Comments: update the current primary address to be used, based on the set of available destination transport addresses of the association.

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

- STATUS.SCTP:
  Command / event: 'Status' 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 receiver window size; current congestion window sizes; number of unacknowledged DATA chunks; number of DATA chunks pending receipt; primary path; most recent SRTT on primary path; RTO on primary path; SRTT and RTO on other destination addresses. The NETWORK STATUS CHANGE notification informs the application about a transport address becoming active/inactive.

- CHANGE-DSCP.TCP:
  Command / event: not specified
  Parameters: DSCP value
  Comments: This allows an application to change the DSCP value. It was only specified for the TOS field in [RFC1122], which is here interpreted to refer to the DSField as per [RFC3260].

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

- CLOSE.TCP:
  Command / event: 'close'
  Comments: this terminates the sending side of a connection after reliably delivering all remaining data. Close also implies push function (see DATA.SEND.TCP).

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

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

- o ABORT.SCTP:
  Command / 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.

- o TIMEOUT.TCP:
  Command / event: 'USER TIMEOUT' event
  Comments: the application is informed that the connection is aborted. This event is executed when the timeout set in CONNECTION.ESTABLISHMENT.CONNECT.TCP (and possibly adjusted in CONNECTION.MAINTENANCE.CHANGE-TIMEOOUT.TCP) expires.

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

- o ABORT-EVENT.TCP:
  Command / event: not specified

- o ABORT-EVENT.SCTP:
  Command / 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.

- o CLOSE-EVENT.TCP:
  Command / event: not specified

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

4.2. DATA Transfer Related Services

All commands in this section refer to an existing connection, i.e. a connection that was either established or made available for receiving data. In addition to the listed parameters, all sending commands contain a reference to a data block and all receiving commands contain a reference to available buffer space for the data.

o SEND.TCP:
Command / event: 'send'
Parameters: PUSH flag (optional); timeout (optional)
Comments: If the push flag is set, the data block should promptly be transmitted to the receiver without waiting. The timeout can be configured with this call whenever data are sent (see also CONNECTION.MAINTENANCE.CHANGE-TIMEOUT.TCP).

o SEND.SCTP:
Command / event: 'Send'
Parameters: stream number; context (optional); life time (optional); destination transport address (optional); unordered flag (optional); no-bundle flag (optional); payload protocol-id (optional)
Comments: 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 'life time' specifies a time after which this data block will not be sent. The 'destination transport address' 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 (this is related to CONNECTION.MAINTENANCE.DISABLE-NAGLE.TCP). The 'payload protocol-id' is a number that will, if it was provided, be handed over to the receiving application.

o RECEIVE.TCP:
Command / event: 'receive'

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

o SENDFAILURE-EVENT.SCTP:
Command / event: 'SEND FAILURE' notification, optionally followed by 'Receive Unsent Message' or 'Receive Unacknowledged Message'
Returns: cause code; context; unsent or unacknowledged message (optional)
Comments: 'cause code' indicates the reason of the failure, and

'context' is the context number if such a number has been provided in DATA.SEND.SCTP, for later use with 'Receive Unsent Message' or 'Receive Unacknowledged Message', respectively. These commands can be used to retrieve the complete unsent or unacknowledged message if desired.

5. Pass 3

Here we present the superset of all services in all protocols, based on the list in pass 2 but also on text in pass 1 to include services that can be configured in one protocol and are static properties in another. Again, some minor details are omitted for the sake of generalization -- e.g., TCP may provide various different IP options but only supporting source route is mandatory to implement, and this detail is no longer visible in "Specify IP Options". The detail was removed because no other protocols provide this features. [AUTHOR'S NOTE: and if we find another one that does, we need that detail again.]

[AUTHOR'S NOTE: the list here looks pretty similar to the list in pass 2 for now. This will change as more protocols are added. For example, if we add UDP, we will find that UDP does not do congestion control, which is relevant to the application using it. This will have to be reflected in pass 1 and pass 2, only for UDP. In pass 3, we can derive "congestion control" as a service of TCP and SCTP because it probably does not make much sense to write that only UDP provides a congestion control related service: the "service" of not doing it -- meaning that it may require more work from the application developer.]

5.1. CONNECTION Related Services

ESTABLISHMENT:
Active creation of a connection from one transport address to one or more transport addresses.

- o Specify IP Options
  Protocols: TCP

- o Request multiple streams
  Protocols: SCTP

- o Obtain multiple destination transport addresses
  Protocols: SCTP

AVAILABILITY:
Preparing to receive incoming connection requests.

- Listen, 1 specified local interface
  Protocols: TCP, SCTP

- Listen, N specified local interfaces
  Protocols: SCTP

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

- Obtain requested number of streams
  Protocols: SCTP

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

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

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

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

- Set protocol parameters
  Protocols: SCTP
  SCTP parameters: RTO.Initial; RTO.Min; RTO.Max; Max.Burst; RTO.Alpha; RTO.Beta; Valid.Cookie.Life; Association.Max.Retrans; Path.Max.Retrans; Max.Init.Retransmits; HB.interval; HB.Max.Burst
  Comments: in future versions of this document, it might make sense to split out some of these parameters -- e.g., if a different protocol provides means to adjust the RTO calculation there could be a common service for them called "adjust RTO calculation".

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

- o Notification of ICMP error message arrival
  Protocols: TCP

- o Status (query or notification)
  Protocols: SCTP
  SCTP parameters: association connection state; destination transport address list; destination transport address reachability states; current receiver window size; current congestion window sizes; number of unacknowledged DATA chunks; number of DATA chunks pending receipt; primary path; most recent SRTT on primary path; RTO on primary path; SRTT and RTO on other destination addresses; transport address becoming active / inactive

- o Set primary path
  Protocols: SCTP

- o Change DSCP
  Protocols: TCP
  Comments: This is described to be changeable for SCTP too in [RFC6458].

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

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

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

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

5.2. DATA Transfer Related Services

All services in this section refer to an existing connection, i.e. a connection that was either established or made available for receiving data. In addition to the listed parameters, all sending commands contain a reference to a data block and all receiving commands contain a reference to available buffer space for the data. Reliable data transfer entails delay -- e.g. for the sender to wait until it can transmit data, or due to retransmission in case of packet loss.

5.2.1. Sending Data

All services in this section are provided by DATA.SEND from pass 2. DATA.SEND is given a data block from the application, which we here call a "message".

- o Reliably transfer data
  Protocols: TCP, SCTP

- o Notifying the receiver to promptly hand over data to application
  Protocols: TCP
  Comments: This seems unnecessary in SCTP, where data arrival causes an event for the application.

- o Message identification
  Protocols: SCTP

- o Choice of stream
  Protocols: SCTP

- o Choice of path (destination address)
  Protocols: SCTP

- o Message lifetime
  Protocols: SCTP

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

- o Request not to bundle messages
  Protocols: SCTP

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

5.2.2. Receiving Data

All services in this section are provided by DATA.RECEIVE from pass 2. DATA.RECEIVE fills a buffer provided to the application, with what we here call a "message".

- Receive data
  Protocols: TCP, SCTP

- Choice of stream to receive on
  Protocols: SCTP

- Message identification
  Protocols: SCTP
  Comments: In SCTP, this is optionally achieved with a "stream sequence number". The stream sequence number is always provided in case of partial message arrival.

- 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 unsent messages
  Protocols: SCTP

- Notification of unacknowledged messages
  Protocols: SCTP

6. Acknowledgements

The authors would like to thank Joe Touch for comments on the TCP part. This work has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT). The views expressed are solely those of the author(s).

7. IANA Considerations

XX RFC ED - PLEASE REMOVE THIS SECTION XXX

This memo includes no request to IANA.

8. Security Considerations

Security will be considered in future versions of this document.

9. References

9.1. Normative References

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

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

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

9.2. Informative References

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

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

[RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., Ed., and G. Fairhurst, Ed., "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, DOI 10.17487/RFC3828, July 2004, <http://www.rfc-editor.org/info/rfc3828>.

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

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

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

Authors' Addresses

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

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

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

Email: tuexen@fh-muenster.de

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

Email: naeemk@ifi.uio.no