Services provided by IETF transport protocols and congestion control mechanisms
draft-ietf-taps-transports-07

Abstract

This document describes services provided by existing IETF protocols and congestion control mechanisms. It is designed to help application and network stack programmers and to inform the work of the IETF TAPS Working Group.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 9, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
# Table of Contents

1. Introduction .................................................. 3
2. Terminology ...................................................... 4
3. Existing Transport Protocols ................................. 5
   3.1. Transport Control Protocol (TCP) ............................. 5
      3.1.1. Protocol Description .................................... 5
      3.1.2. Interface description .................................... 6
      3.1.3. Transport Features ....................................... 7
   3.2. Multipath TCP (MPTCP) ........................................ 8
      3.2.1. Protocol Description .................................... 8
      3.2.2. Interface Description .................................... 8
      3.2.3. Transport features ....................................... 8
   3.3. Stream Control Transmission Protocol (SCTP) ............... 9
      3.3.1. Protocol Description .................................... 9
      3.3.2. Interface Description .................................... 12
      3.3.3. Transport Features ....................................... 14
   3.4. User Datagram Protocol (UDP) ............................... 15
      3.4.1. Protocol Description .................................... 15
      3.4.2. Interface Description .................................... 16
      3.4.3. Transport Features ....................................... 16
   3.5. Lightweight User Datagram Protocol (UDP-Lite) ............. 17
      3.5.1. Protocol Description .................................... 17
      3.5.2. Interface Description .................................... 18
      3.5.3. Transport Features ....................................... 18
   3.6. Datagram Congestion Control Protocol (DCCP) .............. 19
      3.6.1. Protocol Description .................................... 19
      3.6.2. Interface Description .................................... 20
      3.6.3. Transport Features ....................................... 21
   3.7. Lightweight User Datagram Protocol (UDP-Lite) ............. 21
      3.7.1. Protocol Description .................................... 21
      3.7.2. Interface Description .................................... 22
      3.7.3. Transport Features ....................................... 22
   3.8. Internet Control Message Protocol (ICMP) .................. 23
      3.8.1. Protocol Description .................................... 23
      3.8.2. Interface Description .................................... 24
      3.8.3. Transport Features ....................................... 24
   3.9. Realtime Transport Protocol (RTP) .......................... 25
      3.9.1. Protocol Description .................................... 25
      3.9.2. Interface Description .................................... 26
      3.9.3. Transport Features ....................................... 26
   3.10. File Delivery over Unidirectional Transport/Asynchronous
        Layered Coding Reliable Multicast (FLUTE/ALC) ............ 26
      3.10.1. Protocol Description .................................... 26
      3.10.2. Interface Description .................................... 27
1. Introduction

Most Internet applications make use of the Transport Services provided by 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, MP-TCP, 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, TFRC or
LEDBAT). This extends the set of available Transport Services beyond those provided to applications by TCP and UDP.

[GF: Ledbat is a mechanism, not protocol – hence use the work "support" in para below.]

Transport protocols can also be differentiated by the features of the services they provide: for instance, SCTP offers a message-based service providing full or partial reliability and allowing to minimize the head of line blocking due to the support of unordered and unordered message delivery within multiple streams, UDP-Lite and DCCP provide partial integrity protection, and LEDBAT can support low-priority "scavenger" communication.

2. Terminology

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

[EDITOR’S NOTE: we may want to add definitions for the different kinds of interfaces that are important here.]

[GF: Interfaces may be covered by Michael Welzl’s companion document?]

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

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

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

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

Transport Service Instance: an arrangement of transport protocols with a selected set of features and configuration parameters that implements a single transport service, e.g. a protocol stack (RTP over UDP).
Application: an entity that uses the transport layer for end-to-end delivery 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 protocol and transport protocol frameworks.

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 [RFC4614].

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. ICMP-based PathMTU discovery [RFC1191][RFC1981] as well as Packetization Layer Path MTU Discovery (PMTUD) [RFC4821] are supported.

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 [RFC2018] extends this mechanism by making it possible to identify missing segments more precisely, reducing 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 [RFC5681]: This uses a separate window, where each time congestion is detected, this congestion window is reduced. Most of the used congestion control mechanisms use one of three mechanisms to detect congestion: A retransmission timer (with exponential back-up), detection of loss (interpreted as a congestion signal), or Explicit Congestion Notification (ECN) [RFC3168] to provide early signaling (see [I-D.ietf-aqm-ecn-benefits]). In addition, a congestion control mechanism may react to changes in delay as an early indication for congestion.

A TCP protocol instance can be extended [RFC4614] 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.

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 a push and a urgent function to enable data to be directly accessed by the receiver without having to wait for in-order delivery of the data. However, [RFC6093] does not recommend the use of the urgent flag due to the range of TCP implementations that process TCP urgent indications differently.

A checksum provides an Integrity Check and is mandatory across the entire packet. This check protects from delivery of corrupted data and misdelivery of packets to the wrong endpoint. This check is relatively weak, applications that require end to end integrity of data are recommended to include a stronger integrity check of their payload data. The TCP checksum does not support partial corruption protection (as in DCCP/UDP-Lite).

TCP only supports 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 1) reporting soft errors such as reception of ICMP error
messages, extensive retransmission or urgent pointer advance, 2)
providing a possibility to specify the Type-of-Service (TOS) for
segments, 3) providing a flush call to empty the TCP send queue, and
4) 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. However, there is no RFC that documents this
interface.

3.1.3. Transport Features

The transport features provided by TCP are:

[EDITOR’S NOTE: expand each of these slightly]

- unicast transport
- connection setup with feature negotiation and application-to-port
  mapping, implemented using SYN segments and the TCP option field
to negotiate features.
- port multiplexing: each TCP session is uniquely identified by a
  combination of the ports and IP address fields.
- 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: a segment checksum verifies delivery to the
  correct endpoint and integrity of the data and options.
- segmentation: packets are fragmented to a negotiated maximum
  segment size, further constrained by the effective MTU from PMTUD.
- data bundling, an optional mechanism that uses Nagle’s algorithm
to coalesce data sent within the same RTT into full-sized
segments.
3. flow control using a window-based mechanism, where the receiver advertises the window that it is willing to buffer.

- congestion control: a window-based method that uses AIMD to control the sending rate and to conservatively choose a rate after congestion is detected.

3.2. Multipath TCP (MPTCP)

Multipath TCP [RFC6824] is an extension for TCP to support multi-homing. It is designed to be as transparent as possible to middle-boxes. It does so by establishing regular TCP flows between a pair of source/destination endpoints, and multiplexing the application’s stream over these flows.

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.

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.

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 should one of the paths become unusable. In addition, by multiplexing one byte stream over
separate paths, it can achieve a higher throughput than TCP in certain situations (note however that coupled congestion control [RFC6356] 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, an additional delay and higher jitter might be observed before in-order delivery of data to the applications.

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

- congestion control with load balancing over multiple connections.
- endpoint multiplexing of a single byte stream (higher throughput).
- address family multiplexing: sub-flows can be started over IPv4 or IPv6 for the same session.
- resilience to network failure and/or handover.

[AUTHOR’S NOTE: it is unclear whether MPTCP has to provide data bundling.]

3.3. Stream Control Transmission Protocol (SCTP)

SCTP is a message-oriented standards track transport protocol. The base protocol is specified in [RFC4960]. It supports multi-homing to handle path failures. It also optionally supports path failover to provide resilience to path failures. An SCTP association has multiple unidirectional streams in each direction and provides in-sequence delivery of user messages only within each stream. This allows it to minimize head of line blocking. SCTP is extensible and the currently defined extensions include mechanisms for dynamic re-configurations of streams [RFC6525] and IP-addresses [RFC5061]. Furthermore, the extension specified in [RFC3758] introduces the concept of partial reliability for user messages.

SCTP was originally developed for transporting telephony signalling messages and is deployed in telephony signalling networks, especially in mobile telephony networks. It can also be used for other services, for example in the WebRTC framework for data channels and is therefore deployed in all WEB-browsers supporting WebRTC.

3.3.1. 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, and only supports unicast.

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

SCTP has been designed with extensibility in mind. Each SCTP packet starts with a single common header containing the port numbers, a verification tag and the CRC32c checksum. This common header is followed by a sequence of chunks. Each chunk consists of a type field, flags, a length field and a value. [RFC4960] defines how a receiver processes chunks with an unknown chunk type. The support of extensions can be negotiated during the SCTP handshake.

SCTP provides a message-oriented service. Multiple small user messages can be bundled into a single SCTP packet to improve the efficiency. For example, this bundling may be done by delaying user messages at the sender similar to the Nagle 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. ICMP-based path MTU discovery as specified for IPv4 in [RFC1191] and for IPv6 in [RFC1981] as well as packetization layer path MTU discovery as specified in [RFC4821] with probe packets using the padding chunks defined the [RFC4820] are supported.

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

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. Ordered messages sent on the same stream are delivered at the receiver in the same order as sent by the sender. For user messages not requiring fragmentation, this minimises head of line blocking.

The base protocol defined in [RFC4960] does not allow interleaving of user-messages, which results in sending a large message on one stream
can block the sending of user messages on other streams. [I-D.ietf-tsvwg-sctp-ndata] overcomes this limitation. Furthermore, [I-D.ietf-tsvwg-sctp-ndata] 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 behaviour. 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. Examples for 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.

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 livetime 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. Each SCTP end-point supervises continuously the reachability of all peer addresses using a heartbeat mechanism.

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 (for example un-ordered delivery or partial reliability), and 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 support NAT 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 context.

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

3.3.2. 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 to control the 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 acknowledgement timeout, and the fragmentation point. The API provides a mechanism to allow the SCTP stack to notify the application about event 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 that 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.

- 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.

For the following SCTP protocol extensions the BSD Sockets API extension is defined in the document specifying the 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 acknowledgement 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 behaviour 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.

3.3.3. Transport Features

The transport features provided by SCTP are:

[GF: This needs to be harmonised with the components for TCP]

- unicast.
- connection setup with feature negotiation and application-to-port mapping.
- port multiplexing.
- message-oriented delivery.
- fully reliable or partially reliable delivery.
- ordered and unordered delivery within a stream.
- support for multiple concurrent streams.
- support for stream scheduling prioritization.
- flow control.
- congestion control.
o user message bundling.
o user message fragmentation and reassembly.
o strong error/misdelivery detection (CRC32c).
o transport layer multihoming for resilience.
o transport layer mobility.

3.4. 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 none of the following transport features: 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.

3.4.1. 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. Each stream of messages is independently managed, therefore retransmission does not hold back data sent using other logical streams. It provides detection of payload errors and misdelivery of packets to the wrong endpoint, either of which result in discard of received datagrams.

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 its use. These datagrams rely on the IPv4 header checksum to protect from misdelivery to the wrong endpoint. IPv6 does not by permit UDP datagrams with no checksum, although in certain cases this rule may be relaxed [RFC6935]. The checksum support considerations for omitting the checksum are defined in [RFC6936]. 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.

It does not provide reliability and does not provide retransmission. This implies messages may be re-ordered, lost, or duplicated in transit.
A receiving application that is unable to run sufficiently fast, or frequently, may miss messages since there is no flow control. 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 an IP packet, which may in turn be fragmented by IP and are reassembled before delivery to the UDP receiver.

Applications that need to provide fragmentation or that have other requirements such as receiver flow control, congestion control, PathMTU discovery/PLPMTUD, support for ECN, etc need these to be provided by protocols operating over UDP [I-D.ietf-tsvwg-rfc5405bis].

3.4.2. 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). Demultiplexing using multiple abstract endpoints (sockets) on the same IP address are 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 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].

3.4.3. Transport Features

The transport features provided by UDP are:

- unicast.
o multicast, anycast, or IPv4 broadcast.

o port multiplexing. A receiving port can be configured to receive datagrams from multiple senders.

o message-oriented delivery.

o unidirectional or bidirectional. Transmission in each direction is independent.

o non-reliable delivery.

o non-ordered delivery.

o IPv6 jumbograms.

o error and misdelivery detection (checksum).

o optional checksum. All or none of the payload data is protected.

3.5. 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].

3.5.1. Protocol Description

UDP-Lite is a connection-less datagram protocol, with no connection setup or feature negotiation. The protocol use messages, rather than a byte-stream. Each stream of messages is independently managed, therefore retransmission does not hold back data sent using other logical streams.

It provides multiplexing to multiple sockets on each host using port numbers, and its operation follows that for UDP. An active UDP-Lite session is identified by its four-tuple of local and remote IP addresses and local port and remote port numbers.

UDP-Lite 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. Otherwise, UDP-Lite is semantically identical to UDP. Applications using UDP-Lite therefore can not make assumptions regarding the correctness of the data received in the insensitive part of the UDP-Lite payload.
As for UDP, mechanisms for receiver flow control, congestion control, PMTU or PLPMTU discovery, support for ECN, etc need to be provided by upper layer protocols [I-D.ietf-tsvwg-rfc5405bis].

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 support error tolerate payload corruption when used over paths that include error-prone links, another application is when header integrity checks are required, but payload integrity is provided by some other mechanism (e.g., [RFC6936]).

A UDP-Lite service may support IPv4 broadcast, multicast, anycast and unicast, and IPv6 multicast, anycast and unicast.

3.5.2. Interface Description

There is no current API 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: at the sender, this specifies the intended checksum coverage, with the remaining unprotected part of the payload called the "error-insensitive part". The checksum coverage may also be made visible to the application via the UDP-Lite MIB module [RFC5097].

3.5.3. Transport Features

The transport features provided by UDP-Lite are:

- unicast.
- multicast, anycast, or IPv4 broadcast.
- port multiplexing (as for UDP).
- message-oriented delivery (as for UDP).
- non-reliable delivery (as for UDP).
- non-ordered delivery (as for UDP).
- error and misdelivery detection (checksum).
- partial or full integrity protection. The checksum coverage field indicates the size of the payload data covered by the checksum.
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].

It 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 functions (feature negotiation, path state management, RTT calculation, PMTUD, etc): This allows applications to use a compatible method defining how they send packets and where suitable to choose common algorithms to manage their functions. Examples of suitable applications 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. The protocol is defined by a family of RFCs.

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. At connection setup, DCCP also exchanges the service code [RFC5595], a mechanism that allows transport instantiations to indicate the service treatment that is expected from the network.

The protocol segments data into messages, typically sized to fit in IP packets, but which may be fragmented providing they are less 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 ordered or unordered delivery of data, and does not itself provide retransmission. DCCP supports reduced checksum coverage, a partial integrity mechanisms similar to UDP-1Lite. There is also a
Data Checksum option that when enabled, contains a strong CRC, to enable endpoints to detect application data corruption.

Receiver flow control is supported: limiting the amount of unacknowledged data that can be outstanding at a given time.

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

A DCCP service is unicast.

DCCP supports negotiation of the congestion control profile, to provide Plug and Play congestion control mechanisms. Examples of specified profiles include [RFC4341] [RFC4342] [RFC5662]. All IETF-defined methods provide Congestion Control.

DCCP use a Connect packet to initiate a session, and permits half-connections that allow each client 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 field [RFC5595] designed to allow middle boxes and endpoints to identify the characteristics required by a session.

A lightweight UDP-based encapsulation (DCCP-UDP) has been defined [RFC6773] that permits DCCP to be used over paths where it is not natively supported. Support 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].

A common packet format has allowed tools to evolve that can read and interpret DCCP packets (e.g. Wireshark).

3.6.2. Interface Description

API characteristics include: - Datagram transmission. - Notification of the current maximum packet size. - Send and reception of zero-length payloads. - Slow Receiver flow control at a receiver. - Detect a Slow receiver at the sender.

There is no current API currently specified in the RFC Series.
3.6.3. Transport Features

The transport features provided by DCCP are:

- unicast.
- connection setup with feature negotiation and application-to-port mapping.
- Service Codes. Identifies the upper layer service to the endpoint and network.
- port multiplexing.
- message-oriented delivery.
- non-reliable delivery.
- ordered delivery.
- flow control. The slow receiver function allows a receiver to control the rate of the sender.
- drop notification. Allows a receiver to notify which datagrams were not delivered to the peer upper layer protocol.
- timestamps.
- partial and full integrity protection (with optional strong integrity check).

3.7. 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].

3.7.1. Protocol Description

UDP-Lite is a connection-less datagram protocol, with no connection setup or feature negotiation. The protocol uses messages, rather than a byte-stream. Each stream of messages is independently managed, therefore retransmission does not hold back data sent using other logical streams.
It provides multiplexing to multiple sockets on each host using port numbers, and its operation follows that for UDP. An active UDP-Lite session is identified by its four-tuple of local and remote IP addresses and local port and remote port numbers.

UDP-Lite 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. Otherwise, UDP-Lite is semantically identical to UDP. Applications using UDP-Lite therefore cannot make assumptions regarding the correctness of the data received in the insensitive part of the UDP-Lite payload.

As for UDP, mechanisms for receiver flow control, congestion control, PMTU or PLPMTU discovery, support for ECN, etc need to be provided by upper layer protocols [I-D.ietf-tsvwg-rfc5405bis].

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 support error tolerant payload corruption when used over paths that include error-prone links, another application is when header integrity checks are required, but payload integrity is provided by some other mechanism (e.g., [RFC6936]).

A UDP-Lite service may support IPv4 broadcast, multicast, anycast and unicast, and IPv6 multicast, anycast and unicast.

3.7.2. Interface Description

There is no current API 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: at the sender, this specifies the intended checksum coverage, with the remaining unprotected part of the payload called the "error-insensitive part". The checksum coverage may also be made visible to the application via the UDP-Lite MIB module [RFC5097].

3.7.3. Transport Features

The transport features provided by UDP-Lite are:

- unicast
- multicast, anycast, or IPv4 broadcast.
- port multiplexing (as for UDP).
o message-oriented delivery (as for UDP).

o non-reliable delivery (as for UDP).

o non-ordered delivery (as for UDP).

o partial or full integrity protection.

3.8. Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol (ICMP) [RFC0792] for IPv4 and [RFC4433] for IPv6 are IETF standards track protocols.

It provides a connection-less unidirectional protocol that delivers individual messages. It provides none of the following transport features: error correction, congestion control, or flow control. Some messages may be sent as broadcast datagrams (IPv4) or multicast datagrams (IPv4 and IPv6), in addition to unicast (and anycast) datagrams.

3.8.1. Protocol Description

ICMP is a connection-less unidirectional protocol that delivers individual messages. The protocol uses independent messages, ordinarily called datagrams. Each message is required to carry a checksum as an integrity check and to protect from misdelivery to the wrong endpoint.

ICMP messages typically relay diagnostic information from an endpoint [RFC1122] or network device [RFC1716] addressed to the sender of a flow. This usually contains the network protocol header of a packet that encountered the reported issue. Some formats of messages may also carry other payload data. Each message carries an integrity check calculated in the same way as UDP.

The RFC series defines additional IPv6 message formats to support a range of uses. In the case of IPv6 the protocol incorporates neighbour 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.

Transport Protocols and upper layer protocols can use ICMP messages to help them take appropriate decisions when network or endpoint
errors are reported. For example to implement, ICMP-based PathMTU
discovery [RFC1191][RFC1981] or assist in Packetization Layer Path
MTU Discovery (PMTUD) [RFC4821]. Such reactions to received messages
needs to protects from off-path data injection
[I-D.ietf-tsvwg-rfc5405bis], avoiding an application receiving
packets that were 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].

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].

3.8.2. Interface Description

ICMP processing is integrated into 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].

3.8.3. Transport Features

The transport features provided by ICMP are:

- unidirectional.
- multicast, anycast and IP4 broadcast.
- message-oriented delivery.
- non-reliable delivery.
- non-ordered delivery.
- error and misdelivery detection (checksum).
3.9. 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 network services, including TCP, UDP, UDP-Lite, or DCCP.

[EDITOR’S NOTE: Varun Singh signed up as contributor for this section. Given the complexity of RTP, suggest to have an abbreviated section here contrasting RTP with other transports, and focusing on those features that are RTP-unique. Gorry Fairhurst contributed this stub section]

3.9.1. Protocol Description

The RTP standard [RFC3550] defines a pair of protocols, RTP and the Real Time Control Protocol, RTCP. The transport does not provide connection setup, but relies 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 media formats 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 fragment this into several RTP packets. If small, several frames may be bundled into a single RTP packet. RTP may runs over a congestion-controlled or non-congestion-controlled transport protocol.

An RTP receiver collects RTP packets from 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.

RTCP is an associated control protocol that works with RTP. Both the RTP sender and receiver can 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 randomised to avoid synchronization of reports from multiple receivers.

3.9.2. Interface Description

[EDITOR’S NOTE: to do]

3.9.3. Transport Features

The transport features provided by RTP are:

- unicast.
- multicast, anycast or IPv4 broadcast.
- port multiplexing.
- message-oriented delivery.
- associated protocols for connection setup with feature negotiation and application-to-port mapping.
- support for media types and other extensions.
- segmentation and aggregation.
- performance reporting.
- drop notification.
- timestamps.

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]. 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". Transmissions happen either in push mode, where content is sent once, or in on-demand mode, where content is continuously sent during periods of time that can largely exceed the average time required to download the session objects (see [RFC5651], section 4.2).

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). In that case, each Audio/Video segment is transmitted as a distinct FLUTE/ALC object in push mode. FLUTE/ALC uses packet erasure coding (also known as Application-Level Forward Erasure Correction, or AL-FEC) in a proactive way. The goal of using AL-FEC is both to increase the robustness in front of packet erasures and to improve the efficiency of the on-demand service. FLUTE/ALC transmissions can be governed by a congestion control mechanism such as the "Wave and Equation Based Rate Control" (WEBRC) [RFC3738] when FLUTE/ALC is used in a layered transmission manner, with several session channels over which ALC packets are sent. However many FLUTE/ALC deployments involve only Constant Bit Rate (CBR) channels with no competing flows, for which a sender-based rate control mechanism is sufficient. In any case, FLUTE/ALC’s reliability, delivery mode, congestion control, and flow/rate control mechanisms are distinct components that can be separately controlled to meet different application needs.

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 regardness 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.

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.

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 in case of Internet ([RFC6726], section 1.1.4). FLUTE/ALC is often used over a network path with pre-provisioned capacity [RFC5404] 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. Open source reference implementations of FLUTE/ALC are available at http://planete-bcast.inrialpes.fr/ (no longer maintained) and http://mad.cs.tut.fi/ (no longer maintained), and these implementations specify and document their own APIs. Commercial versions are also available, some derived from the above implementations, with their own API.

3.10.3. Transport Features

The transport features provided by FLUTE/ALC are:

- unicast
- multicast, anycast or IPv4 broadcast.
- per-object dynamic meta-data delivery.
- push delivery or on-demand delivery service.
- fully reliable or partially reliable delivery (of file or in-memory objects).
- ordered or unordered delivery (of file or in-memory objects).
- per-packet authentication, integrity, and anti-replay services.
3.11. NACK-Oriented Reliable Multicast (NORM)

NORM is an IETF standards track protocol specified in [RFC5740]. 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. Its support for bulk data dissemination
includes discrete file or computer memory-based "objects" as well as
byte- and message-streaming. NORM is designed to incorporate packet
erasure coding as an inherent part of its selective ARQ in response
to receiver negative acknowledgements. The packet erasure coding can
also be proactively applied for forward protection from packet loss.
NORM transmissions are governed by the TCP-friendly congestion
control. NORM’s reliability, congestion control, and flow control
mechanism are distinct components and 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
purposes of 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 and 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 used 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 (e.g., loss event, ECN event, etc) information from the receiver(s) to support dynamic rate control adjustment. The TCP-Friendly Multicast Congestion Control (TFMCC) [RFC4654] used provides some extra features to support multicast but is functionally equivalent to TFRC in the unicast case.

NORM’s reliability mechanism is decoupled from congestion control. This allows alternative arrangements of transport services to be invoked. For example, fixed-rate reliable delivery can be supported or unreliable (but optionally "better than best effort" via packet erasure coding) delivery with rate-control per TFRC can be achieved. Additionally, alternative congestion control techniques may be applied. For example, TFRC rate control with congestion event detection based on ECN for links with high packet loss (e.g., wireless) has been implemented and demonstrated with NORM.

While NORM is NACK-based for reliability transfer, it also supports a positive acknowledgment (ACK) mechanism that can be used for receiver flow control. Again, since this mechanism is decoupled from the reliability and congestion control, applications that have different needs in this aspect can use the protocol differently. One example is the 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.
- stream-oriented delivery in a single stream.
- object-oriented delivery of discrete data or file items.
- reliable delivery.
- unordered unidirectional delivery (of in-memory data or file bulk content objects).
- error detection (UDP checksum).
- segmentation.
- data bundling (Nagle’s algorithm).
- flow control (Nagle-based and/or ack-based).
- congestion control.
- packet erasure coding (both proactively and as part of ARQ).

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

Transport Layer Security (TLS) and Datagram TLS (DTLS) 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; it is defined in [RFC5246]. 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) protocols developed in the mid 90s 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.
3.12.1. 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 loss, 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).
- Coveyance of IP don’t fragment (DF) bit settings by application.
o 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]. Note also that DTLS forbids the use of stream ciphers, which are essentially incompatible when operating on independent encrypted records.

3.12.2. 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. There may also be special APIs required to deal with time and/or random numbers, both of which are needed by a variety of encryption algorithms and protocols.

Considerable care is required in the use of TLS APIs in order to create 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.

3.12.3. Transport Features

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

o message fragmentation

o authentication and integrity via message authentication codes (MAC)
o data encryption

o scheduling transmission using the underlying transport protocol

DTLS augments the TLS record protocol with:

o 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.

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

Hypertext Transfer Protocol (HTTP) is an application-level protocol widely used on the Internet. Version 1.1 of the protocol is specified in [RFC7230] [RFC7231] [RFC7232] [RFC7233] [RFC7234] [RFC7235], and version 2 in [RFC7540]. Furthermore, HTTP is used as a substrate for other application-layer protocols. 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 which makes it work with a lot of infrastructure, and cases where a application server often needs to support HTTP anyway.

Depending on application’s needs, 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] address this issues and provides some guidelines and 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.

Though not strictly bound to TCP, HTTP is almost exclusively run over TCP, and therefore inherits its properties when used in this way.
3.13.1. 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 by a MIME-like message (see [RFC7231] for the differences between an HTTP object and a MIME message), containing information about the client and request modifiers. The message can contain a message body carrying application data as well. The server responds with a status or error code followed by a MIME-like message containing information about the server and information about carried data and it can include a message body. It is possible to specify a data format for the message body using MIME media types [RFC2045]. Furthermore, the protocol has numerous additional features; features relevant to pseudotransport are described below.

Content negotiation, specified in [RFC7231], is a mechanism provided by HTTP for selecting a representation on a requested resource. The client and server negotiate acceptable data formats, charsets, data encoding (e.g. data can be transferred compressed, gzip), etc. 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 range requests specified in [RFC7233]. The protocol provides powerful cache control signalling defined in [RFC7234].

HTTP 1.1’s and HTTP 2.0’s persistent connections can be use to perform multiple request-response transactions during the life-time of a single HTTP connection. Moreover, HTTP 2.0 connections can multiplex many request/response pairs in parallel on a single connection. This reduces connection establishment overhead and the effect of TCP slow-start on each transaction, important for HTTP’s primary use case.

It is possible to combine HTTP with security mechanisms, like TLS (denoted by HTTPS), which adds protocol properties provided by such a mechanism (e.g. authentication, encryption, etc.). TLS’s Application-Layer Protocol Negotiation (ALPN) extension [RFC7301] can be used for HTTP version negotiation within TLS handshake which eliminates addition round-trip. Arbitrary cookie strings, included as part of the MIME headers, are often used as bearer tokens in HTTP.

Application layer protocols using HTTP as substrate may use existing method and data formats, or specify new methods and data formats. Furthermore some protocols may not fit a request/response paradigm and instead rely on HTTP to send messages (e.g. [RFC6546]). Because HTTP is working in many restricted infrastructures, it is also used to tunnel other application-layer protocols.
3.13.2. 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 TLS is used, API expose a registration of callbacks in case a server requests client authentication and when certificate verification is needed.

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

Representational State Transfer (REST) [REST] is another example how applications can use HTTP as transport protocol. REST is an architecture style for building application on the Internet. It uses HTTP as a communication protocol.

3.13.3. Transport features

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

- unicast.
- message and stream-oriented transfer.
- bi- or unidirectional transmission.
- ordered delivery.
- fully reliable delivery.
- object range request.
- message content type negotiation.
- flow control.

HTTPS (HTTP over TLS) additionally provides the following components:

- authentication (of one or both ends of a connection).
4. Transport Service Features

[EDITOR’S NOTE: This section is still work-in-progress. This list is probably not complete and/or too detailed.]

The transport protocol components analyzed in this document which can be used as a basis for defining common transport service features, normalized and separated into categories, are as follows:

- Control Functions
  * Addressing
    + unicast
    + multicast, anycast and IPv4 broadcast
    + use of NAPT-compatible port numbers
  * Multihoming support
    + multihoming for resilience
    + multihoming for mobility
      - specify handover latency?
    + multihoming for load-balancing
      - specify interleaving delay?
  * Multiplexing
    + application to port mapping
    + single vs. multiple streaming

- Delivery
  * reliability
    + fully reliable delivery
    + partially reliable delivery
- packet erasure coding
+ unreliable delivery
- drop notification
- Integrity protection
  o checksum for error detection
  o partial payload checksum protection
  o checksum optional
* ordering
+ ordered delivery
+ unordered delivery
  - unordered delivery of in-memory data
* type/framing
+ stream-oriented delivery
+ message-oriented delivery
+ object-oriented delivery of discrete data or file items
  - object content type negotiation
+ range-based partial object transmission
+ file bulk content objects
o Transmission control
* rate control
  + timer-based
  + ACK-based
* congestion control
* flow control
* segmentation
* data/message bundling (Nagle’s algorithm)
* stream scheduling prioritization

---

Security

* authentication of one end of a connection
* authentication of both ends of a connection
* confidentiality
* cryptographic integrity protection

A future revision of this document will define transport service features based upon this list.

[EDITOR’S NOTE: this section will draw from the candidate features provided by protocol components in the previous section – please discuss on taps@ietf.org list]

4.1. Complete Protocol Feature Matrix

[EDITOR’S NOTE: Dave Thaler has signed up as a contributor for this section. Michael Welzl also has a beginning of a matrix which could be useful here.]

[EDITOR’S NOTE: The below is a strawman proposal below by Gorry Fairhurst for initial discussion]

The table below summarises protocol mechanisms that have been standardised. It does not make an assessment on whether specific implementations are fully compliant to these specifications.
<table>
<thead>
<tr>
<th>Mechanism</th>
<th>UDP</th>
<th>UDP-L</th>
<th>DCCP</th>
<th>SCTP</th>
<th>TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unicast</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Mcast/IPv4Bcast</td>
<td>Yes (2)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Port Mux</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Mode</td>
<td>Dgram</td>
<td>Dgram</td>
<td>Dgram</td>
<td>Dgram</td>
<td>Stream</td>
</tr>
<tr>
<td>Connected</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Data bundling</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Feature Nego</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Options</td>
<td>No</td>
<td>No</td>
<td>Support</td>
<td>Support</td>
<td>Support</td>
</tr>
<tr>
<td>Data priority</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Data bundling</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Reliability</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Select</td>
<td>Yes</td>
</tr>
<tr>
<td>Ordered deliv</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Stream</td>
<td>Yes</td>
</tr>
<tr>
<td>Corruption Tol.</td>
<td>No</td>
<td>Support</td>
<td>Support</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Flow Control</td>
<td>No</td>
<td>No</td>
<td>Support</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PMTU/PLPMTU</td>
<td>(1)</td>
<td>(1)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Cong Control</td>
<td>(1)</td>
<td>(1)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ECN Support</td>
<td>(1)</td>
<td>(1)</td>
<td>Yes</td>
<td>TBD</td>
<td>Yes</td>
</tr>
<tr>
<td>NAT support</td>
<td>Limited</td>
<td>Limited</td>
<td>Support</td>
<td>TBD</td>
<td>Support</td>
</tr>
<tr>
<td>Security</td>
<td>DTLS</td>
<td>DTLS</td>
<td>DTLS</td>
<td>DTLS</td>
<td>TLS, AO</td>
</tr>
<tr>
<td>UDP encaps</td>
<td>N/A</td>
<td>None</td>
<td>Yes</td>
<td>Yes</td>
<td>None</td>
</tr>
<tr>
<td>RTP support</td>
<td>Support</td>
<td>Support</td>
<td>Support</td>
<td>?</td>
<td>Support</td>
</tr>
</tbody>
</table>

Note (1): this feature requires support in an upper layer protocol.
Note (2): this feature requires support in an upper layer protocol when used with IPv6.

5. IANA Considerations

This document has no considerations for IANA.

6. 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 components or mechanisms for providing these features. Each RFC listed in this document discusses the security considerations of the specification it contains.

7. Contributors

[Editor’s Note: turn this into a real contributors section with addresses once we figure out how to trick the toolchain into doing so]

- Section 3.2 on MPTCP was contributed by Simone Ferlin-Oliviera (ferlin@simula.no) and Olivier Mehani (olivier.mehani@nicta.com.au)
- Section 3.4 on UDP was contributed by Kevin Fall (kfall@kfall.com)
- Section 3.3 on SCTP was contributed by Michael Tuexen (tuexen@fh-muenster.de)
- 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)
- Section 3.12 on TLS and DTLS was contributed by Ralph Holz (ralph.holz@nicta.com.au) and Olivier Mehani (olivier.mehani@nicta.com.au)
- Section 3.13 on HTTP was contributed by Dragana Damjanovic (ddamjanovic@mozilla.com)
8. Acknowledgments

Thanks to Karen Nielsen, Joe Touch, and Michael Welzl for the comments, feedback, and discussion. This work is partially supported by the European Commission under grant agreements FP7-ICT-318627 mPlane and from the Horizon 2020 research and innovation program under grant agreement No. 644334 (NEAT); support does not imply endorsement.

9. Informative References


[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,

[RFC5097]  Renker, G. and G. Fairhurst, "MIB for the UDP-Lite
protocol", RFC 5097, DOI 10.17487/RFC5097, January 2008,

RFC5246, August 2008,

[RFC5238]  Phelan, T., "Datagram Transport Layer Security (DTLS) over
the Datagram Congestion Control Protocol (DCCP)", RFC
5238, DOI 10.17487/RFC5238, May 2008,

G.719", RFC 5404, DOI 10.17487/RFC5404, January 2009,

10.17487/RFC5461, February 2009,

(DCCP) Service Codes", RFC 5595, DOI 10.17487/RFC5595,

(DCCP) Simultaneous-Open Technique to Facilitate NAT/
Middlebox Traversal", RFC 5596, DOI 10.17487/RFC5596,

Transport (LCT) Building Block", RFC 5651, DOI 10.17487/
RFC5651, October 2009,

"Network File System (NFS) Version 4 Minor Version 1
External Data Representation Standard (XDR) Description",
RFC 5662, DOI 10.17487/RFC5662, January 2010,


Internet-Draft               TAPS Transports                October 2015

[I-D.ietf-tsvwg-sctp-dtls-encaps]

[I-D.ietf-tsvwg-sctp-ndata]

[I-D.ietf-tsvwg-natsupp]


Authors’ Addresses

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

Email: gorry@erg.abdn.ac.uk
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

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

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

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

This Internet-Draft will expire on March 24, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ............................................. 3
2. General Considerations ................................. 3
3. Pass 1 .................................................. 4
   3.1. Services Provided by TCP .......................... 4
       3.1.1. Excluded Services ............................ 7
   3.2. Services Provided by SCTP ........................ 7
       3.2.1. Excluded Services ............................ 10
4. Pass 2 .................................................. 11
   4.1. CONNECTION Related Services ....................... 11
   4.2. DATA Transfer Related Services ............... 15
5. Pass 3 .................................................. 17
   5.1. CONNECTION Related Services ....................... 17
   5.2. DATA Transfer Related Services ............... 20
       5.2.1. Sending Data ................................ 20
       5.2.2. Receiving Data ............................... 21
       5.2.3. Errors ........................................ 21
6. Acknowledgements ......................................... 21
7. IANA Considerations ..................................... 22
8. Security Considerations ................................ 22
9. References ............................................... 22
   9.1. Normative References ............................ 22
   9.2. Informative References .......................... 22
Authors’ Addresses ........................................ 23
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.
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 included 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 association.

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.

o 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.
o 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.

o 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.

o 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.

o 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).

o CLOSE.SCTP:
   Command / event: ‘Shutdown’
   Comments: this terminates a connection after reliably delivering all remaining data.

o 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-TIMEOUT.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-
  TIMEOUT.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.

- Specify IP Options
  Protocols: TCP

- Request multiple streams
  Protocols: SCTP

- 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
\begin{itemize}
\item Notification of ICMP error message arrival
  
  Protocols: TCP

\item 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

\item Set primary path
  
  Protocols: SCTP

\item Change DSCP
  
  Protocols: TCP
  
  Comments: This is described to be changeable for SCTP too in [RFC6458].
\end{itemize}

**TERMINATION:**

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

\begin{itemize}
\item 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.

\item 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.

\item 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)".
\end{itemize}
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".

- Reliably transfer data
  Protocols: TCP, SCTP

- 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.

- Message identification
  Protocols: SCTP

- Choice of stream
  Protocols: SCTP

- Choice of path (destination address)
  Protocols: SCTP

- Message lifetime
  Protocols: SCTP

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

- Request not to bundle messages
  Protocols: SCTP

- 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

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


9.2. Informative References


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