Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 9, 2016

G. Fairhurst, Ed.
University of Aberdeen
B. Trammell, Ed.
M. Kuehlewind, Ed.
ETH Zurich
October 07, 2015

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.

Table of Contents

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 Micahel 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 wihout 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 miselivery 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 fo 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 fush 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.

- 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 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 mutiple 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 resilliance 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 miselivery 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 snet 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.

- user message bundling.
- user message fragmentation and reassembly.
- strong error/misdelivery detection (CRC32c).
- transport layer multihoming for resilience.
- 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 relie 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 overlaoded path and may cause the loss of messages from other protocols (e.g., TCP) when sharing the same network path.

[GF: This para isn't needed": Messages with payload errors are ordinarily detected by an invalid end- to-end checksum and are discarded before being delivered to an application. UDP-Lite (see [RFC3828], and below) provides the ability for portions of the message contents to be exempt from checksum coverage.]

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:

o unicast.

- multicast, anycast, or IPv4 broadcast.
- port multiplexing. A receiving port can be configured to receive datagrams from multiple senders.
- message-oriented delivery.
- unidirectional or bidirectional. Transmission in each direction is independent.
- non-reliable delivery.
- non-ordered delivery.
- IPv6 jumbograms.
- error and misdelivery detection (checksum).
- 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).
- partialor 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-lIte. 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 curremntly 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 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.

3.7.2. 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).

- message-oriented delivery (as for UDP).
- non-reliable delivery(as for UDP).
- non-ordered delivery (as for UDP).
- 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 conection-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 conection-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 messaged 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 mulriplexing, 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).

Altough 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 regardless of the number of receivers.

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

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-provisoned capacity [RFC5404] whete theres 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.

- proactive packet erasure coding (AL-FEC) to recover from packet erasures and improve the on-demand delivery service,
- error detection (through UDP and lower level checksums).
- congestion control for layered flows (e.g., with WEBRC).
- rate control transmission in a given channel.

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

[EDITOR'S NOTE: needs to be more clear about the application of FEC and packet erasure coding; expand ARQ.]

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 uses to achieve group size scaling with low feedback traffic levels. The feedback suppression is not applied for unicast operation.

NORM uses rate-based congestion control based upon the TCP-Friendly Rate Control (TFRC) [RFC4324] principles that are also used in DCCP [RFC4340]. NORM uses control messages to measure RTT and collect congestion event (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 (timer-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.

[NOTE: The Logjam authors (weakdh.org) give (inconclusive) evidence that one of the recommendations of [RFC7525], namely the use of DHE-1024 as a fallback, may not be sufficient in all cases to counter an attacker with the resources of a nation-state. It is unclear at this time if the RFC is going to be updated as a result, or whether there will be an RFC7525bis.]

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.

- 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:

- message fragmentation
- authentication and integrity via message authentication codes (MAC)

- data encryption
- scheduling transmission using the underlying transport protocol

DTLS augments the TLS record protocol with:

- ordering and replay protection, implemented using sequence numbers.

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

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).

- o confidentiality.
- o integrity protection.

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:

- o 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
- o 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 partical 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 drawn 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.

Mechanism	UDP	UDP-L	DCCP	SCTP	TCP
Unicast	Yes	Yes	Yes	Yes	Yes
Mcast/IPv4Bcast	Yes(2)	Yes	No	No	No
Port Mux	Yes	Yes	Yes	Yes	Yes
Mode	Dgram	Dgram	Dgram	Dgram	Stream
Connected	No	No	Yes	Yes	Yes
Data bundling	No	No	No	Yes	Yes
Feature Nego	No	No	Yes	Yes	Yes
Options	No	No	Support	Support	Support
Data priority	*	*	*	Yes	No
Data bundling	No	No	No	Yes	Yes
Reliability	None	None	None	Select	Full
Ordered deliv	No	No	No	Stream	Yes
Corruption Tol.	No	Support	Support	No	No
Flow Control	No	No	Support	Yes	Yes
PMTU/PLPMTU	(1)	(1)	Yes	Yes	Yes
Cong Control	(1)	(1)	Yes	Yes	Yes
ECN Support	(1)	(1)	Yes	TBD	Yes
NAT support	Limited	Limited	Support	TBD	Support
Security	DTLS	DTLS	DTLS	DTLS	TLS, AO
UDP encaps	N/A	None	Yes	Yes	None
RTP support	Support	Support	Support	?	Support

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

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

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

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

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

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

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

[RFC1716] Almquist, P. and F. Kastenholz, "Towards Requirements for IP Routers", RFC 1716, DOI 10.17487/RFC1716, November 1994, <http://www.rfc-editor.org/info/rfc1716>.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[RFC4433] Kulkarni, M., Patel, A., and K. Leung, "Mobile IPv4 Dynamic Home Agent (HA) Assignment", RFC 4433, DOI 10.17487/RFC4433, March 2006, <http://www.rfc-editor.org/info/rfc4433>.

[RFC4614] Duke, M., Braden, R., Eddy, W., and E. Blanton, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", RFC 4614, DOI 10.17487/RFC4614, September 2006, <http://www.rfc-editor.org/info/rfc4614>.

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

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

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

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

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

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

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

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

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

[RFC5404] Westerlund, M. and I. Johansson, "RTP Payload Format for G.719", RFC 5404, DOI 10.17487/RFC5404, January 2009, <http://www.rfc-editor.org/info/rfc5404>.

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

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

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

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

[RFC5662] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 External Data Representation Standard (XDR) Description", RFC 5662, DOI 10.17487/RFC5662, January 2010, <http://www.rfc-editor.org/info/rfc5662>.

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

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

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

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

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

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

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

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

[RFC6546] Trammell, B., "Transport of Real-time Inter-network Defense (RID) Messages over HTTP/TLS", RFC 6546, DOI 10.17487/RFC6546, April 2012, <http://www.rfc-editor.org/info/rfc6546>.

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

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

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

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <http://www.rfc-editor.org/info/rfc6455>.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[I-D.ietf-tsvwg-rfc5405bis] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", draft-ietf-tsvwg-rfc5405bis-05 (work in progress), August 2015.

[I-D.ietf-aqm-ecn-benefits] Fairhurst, G. and M. Welzl, "The Benefits of using Explicit Congestion Notification (ECN)", draft-ietf-aqm-ecn-benefits-06 (work in progress), July 2015.

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

[I-D.ietf-tsvwg-sctp-ndata]
Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol", draft-ietf-tsvwg-sctp-ndata-04 (work in progress), July 2015.

[I-D.ietf-tsvwg-natsupp]
Stewart, R., Tuexen, M., and I. Ruengeler, "Stream Control Transmission Protocol (SCTP) Network Address Translation Support", draft-ietf-tsvwg-natsupp-08 (work in progress), July 2015.

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

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

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

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

Authors' Addresses

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

Email: gorry@erg.abdn.ac.uk

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

Email: ietf@trammell.ch

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

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