Network Working Group   G. Fairhurst, Ed.
Internet-Draft   University of Aberdeen
Intended status: Informational   B. Trammell, Ed.
Expires: July 31, 2016   M. Kuehlewind, Ed.
                                                               ETH Zurich
                                                         January 28, 2016

       **Services provided by IETF transport protocols and congestion control
                                mechanisms
                      draft-ietf-taps-transports-09**

Abstract

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

Table of Contents

1. Introduction

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

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

1.1. Overview of Transport Features

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

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

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

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

In addition, transport protocols offer a certain support on
transmission control. For example, a transport service can provide

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

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

2. Terminology

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

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

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

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

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

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

3. Existing Transport Protocols

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

3.1. **Transport Control Protocol (TCP)**

   TCP is an IETF standards track transport protocol. [RFC0793]
   introduces TCP as follows: "The Transmission Control Protocol (TCP)
   is intended for use as a highly reliable host-to-host protocol
   between hosts in packet-switched computer communication networks, and
   in interconnected systems of such networks." Since its introduction,
   TCP has become the default connection- oriented, stream-based
   transport protocol in the Internet. It is widely implemented by
   endpoints and widely used by common applications.

3.1.1. **Protocol Description**

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

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

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

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

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

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

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

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

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

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

TCP supports only unicast connections.

3.1.2. Interface description

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

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

   o reporting soft errors such as reception of ICMP error messages,
      extensive retransmission or urgent pointer advance,

   o providing a possibility to specify the Differentiated Services
      Code Point (DSCP) [RFC3260] (formerly, the Type-of-Service, TOS)
      for segments,

   o providing a flush call to empty the TCP send queue, and

   o multihoming support.

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

3.1.3. Transport Features

   The transport features provided by TCP are:

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

   o unicast transport (though anycast TCP is implemented, at risk of
      instability due to rerouting),

   o port multiplexing,

   o uni- or bidirectional communication,

   o stream-oriented delivery in a single stream,

   o fully reliable delivery (implemented using ACKs sent from the
      receiver to confirm delivery),

o error detection (implemented using a segment checksum to verify
   delivery to the correct endpoint and integrity of the data and
   options),

o segmentation,

o data bundling (optional; uses Nagle's algorithm to coalesce data
   sent within the same RTT into full-sized segments),

o flow control (implemented using a window-based mechanism where the
   receiver advertises the window that it is willing to buffer),

o congestion control (usually implemented using a window-based
   mechanism and four algorithm for different phases of the
   transmission: slow start, congestion avoidance, fast retransmit,
   and fast recovery [RFC5681]).

3.2. Multipath TCP (MPTCP)

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

3.2.1. Protocol Description

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

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

3.2.2. Interface Description

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

This Basic API describes how an application can:

o enable or disable MPTCP.

o bind a socket to one or more selected local endpoints.

o query local and remote endpoint addresses.

o get a unique connection identifier (similar to an address-port
   pair for TCP).

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

3.2.3. Transport features

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

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

o multihoming for load-balancing, with endpoint multiplexing of a
   single byte stream, using either coupled congestion control or for
   throughput maximization,

o address family multiplexing (using IPv4 and IPv6 for the same
   session),

o resilience to network failure and/or handover.

3.3. Stream Control Transmission Protocol (SCTP)

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

SCTP was originally developed for transporting telephony signaling
messages and is deployed in telephony signaling networks, especially

in mobile telephony networks. It can also be used for other
services, for example, in the WebRTC framework for data channels.

3.3.1. 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. SCTP only supports unicast.

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

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

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

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

Each SCTP association has between 1 and 65536 uni-directional streams
in each direction. The number of streams can be different in each
direction. Every user message is sent on a particular stream. User
messages can be sent un-ordered, or ordered upon request by the upper
layer. Un-ordered messages can be delivered as soon as they are

completely received. 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 minimizes head of
line blocking.

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

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

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

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

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

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

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

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

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

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

o   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.

o   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:

o   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.

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

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

o   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:

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

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

One-to-one style sockets are similar to TCP sockets, there is a 1:1
relationship between the sockets and the SCTP associations (except

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

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

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

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

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

3.3.3. Transport Features

The transport features provided by SCTP are:

o connection-oriented transport with feature negotiation and
   application-to-port mapping,

o unicast transport,

o port multiplexing,

o uni- or bidirectional communication,

o message-oriented delivery with durable message framing supporting
   multiple concurrent streams,

o fully reliable, partially reliable, or unreliable delivery (based
   on user specified policy to handle abandoned user messages) with
   drop notification,

o ordered and unordered delivery within a stream,

o support for stream scheduling prioritization,

o segmentation,

   o user message bundling,

   o flow control using a window-based mechanism,

   o congestion control using methods similar to TCP,

   o strong error detection (CRC32c),

   o transport layer multihoming for resilience and 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 no
   error correction, congestion control, or flow control. It can be
   used to send broadcast datagrams (IPv4) or multicast datagrams (IPv4
   and IPv6), in addition to unicast and anycast datagrams. IETF
   guidance on the use of UDP is provided in
   [I-D.ietf-tsvwg-rfc5405bis]. UDP is widely implemented and widely
   used by common applications, including DNS.

3.4.1. 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. It provides
   detection of payload errors and misdelivery of packets to an
   unintended endpoint, either of which result in discard of received
   datagrams, with no indication to the user of the service.

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

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

   Because UDP provides no flow control, a receiving application that is
   unable to run sufficiently fast, or frequently, may miss messages.

The lack of congestion handling implies UDP traffic may experience
loss when using an overloaded path, and may cause the loss of
messages from other protocols (e.g., TCP) when sharing the same
network path.

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

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

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

3.4.3. Transport Features

The transport features provided by UDP are:

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

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

o   message-oriented delivery,

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

o   non-reliable delivery,

o   unordered delivery,

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

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

Examples of use include a class of applications that can derive
benefit from having partially-damaged payloads delivered, rather than
discarded. One use is to 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]).

3.5.1. Protocol Description

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

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

3.5.2. Interface Description

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

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

3.5.3. Transport Features

The transport features provided by UDP-Lite are:

o unicast, multicast, anycast, or IPv4 broadcast transport (as for UDP),

o port multiplexing (as for UDP),

o message-oriented delivery (as for UDP),

o Uni- or bidirectional communication where the transmissions in each direction are independent (as for UDP),

o non-reliable delivery (as for UDP),

o non-ordered delivery (as for UDP),

o partial or full payload error detection (where the checksum coverage field indicates the size of the payload data covered by the checksum).

3.6. Datagram Congestion Control Protocol (DCCP)

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

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

DCCP offers low overhead, and many characteristics common to UDP, but can avoid "re-inventing the wheel" each time a new multimedia application emerges. Specifically it includes core transport functions (feature negotiation, path state management, RTT

calculation, PMTUD, etc.): DCCP applications select how they send
packets and, where suitable, choose common algorithms to manage their
functions. Examples of applications that can benefit from such
transport services include interactive applications, streaming media,
or on-line games [RFC4336].

3.6.1. Protocol Description

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

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

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

DCCP service is unicast-only.

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

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

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

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

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

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

3.6.2. Interface Description

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

There is no API currently specified in the RFC Series.

3.6.3. Transport Features

The transport features provided by DCCP are:

o unicast transport,

o connection-oriented communication with feature negotiation and
   application-to-port mapping,

o signaling of application class for middlebox support (implemented
   using Service Codes),

o port multiplexing,

o uni-or bidirectional communication,

o   message-oriented delivery,

o   unreliable delivery with drop notification,

o   unordered delivery,

o   flow control (implemented using the slow receiver function)

o   partial and full payload error detection (with optional strong
    integrity check).

3.7. Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol (ICMP) [RFC0792] for IPv4 and
ICMP for IPv6 [RFC4433] are IETF standards track protocols. It is a
connection-less unidirectional protocol that delivers individual
messages, without error correction, congestion control, or flow
control. Messages may be sent as unicast, IPv4 broadcast or
multicast datagrams (IPv4 and IPv6), in addition to anycast
datagrams.

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

3.7.1. Protocol Description

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

ICMP messages typically relay diagnostic information from an endpoint
[RFC1122] or network device [RFC1716] addressed to the sender of a

flow. This usually contains the network protocol header of a packet
that encountered a reported issue. Some formats of messages can also
carry other payload data. Each message carries an integrity check
calculated in the same way as for UDP, this checksum is not optional.

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

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

3.7.2. Interface Description

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

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

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

3.7.3. Transport Features

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

3.8. Realtime Transport Protocol (RTP)

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

[3.8.1](#). **Protocol Description**

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

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

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

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

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

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

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

3.8.2. Interface Description

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

3.8.3. Transport Features

The transport features provided by RTP are:

o unicast, multicast or IPv4 broadcast (provided by lower layer
   protocol),

o port multiplexing (provided by lower layer protocol),

o uni- or bidirectional communication (provided by lower layer
   protocol),

o message-oriented delivery with support for media types and other
   extensions,

o reliable delivery when using erasure coding or unreliable delivery
   with drop notification (if supported by lower layer protocol),

o connection setup with feature negotiation (using associated
   protocols) and application-to-port mapping (provided by lower
   layer protocol),

o segmentation,

o performance metric reporting (using associated protocols).

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

FLUTE/ALC is an IETF standards track protocol specified in [RFC6726]
and [RFC5775]. It provides object-oriented delivery of discrete data
or files. Asynchronous Layer Coding (ALC) provides an underlying
reliable transport service and FLUTE a file-oriented specialization
of the ALC service (e.g., to carry associated metadata). The
[RFC6726] and [RFC5775] protocols are non-backward-compatible updates

of the [RFC3926] and [RFC3450] experimental protocols; these
experimental protocols are currently largely deployed in the 3GPP
Multimedia Broadcast and Multicast Services (MBMS) (see [MBMS],
section 7) and similar contexts (e.g., the Japanese ISDB-Tmm
standard).

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

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

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

3.9.1. Protocol Description

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

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

This enables receivers to join a session asynchronously, at their own
discretion, receive the content and leave the session. In this case,
data content is typically sent continuously, in loops (also known as
"carousels"). FLUTE/ALC also supports the transfer of an object
stream, with loose real-time constraints. This is particularly
useful to carry 3GPP DASH when scalability is a requirement and

unicast transmissions over HTTP cannot be used ([MBMS], section 5.6).
In this case, packets are sent in sequence using push mode. FLUTE/
ALC is not well adapted to byte- and message-streaming and other
solutions could be preferred (e.g., FECFRAME [RFC6363] with real-time
flows).

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

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

To provide robustness against packet loss and improve the efficiency
of the on-demand mode, FLUTE/ALC relies on packet erasure coding (AL-
FEC). AL-FEC encoding is proactive (since there is no feedback and
therefore no (N)ACK-based retransmission) and ALC packets containing
repair data are sent along with ALC packets containing source data.
Several FEC Schemes have been standardized; FLUTE/ALC does not
mandate the use of any particular one. Several strategies concerning
the transmission order of ALC source and repair packets are possible,
in particular in on-demand mode where it can deeply impact the
service provided (e.g., to favor the recovery of objects in sequence,
or at the other extreme, to favor the recovery of all objects in
parallel), and FLUTE/ALC does not mandate nor recommend the use of
any particular one.

A FLUTE/ALC session is composed of one or more channels, associated
to different destination unicast and/or multicast IP addresses. ALC
packets are sent in those channels at a certain transmission rate,
with a rate that often differs depending on the channel. FLUTE/ALC
does not mandate nor recommend any strategy to select which ALC
packet to send on which channel. FLUTE/ALC can use a multiple rate
congestion control building block (e.g., WEBRC) to provide congestion
control that is feedback free, where receivers adjust their reception
rates individually by joining and leaving channels associated with
the session. To that purpose, the ALC header provides a specific
field to carry congestion control specific information. However
FLUTE/ALC does not mandate the use of a particular congestion control
mechanism although WEBRC is mandatory to support for the Internet
([RFC6726], section 1.1.4). FLUTE/ALC is often used over a network

   path with pre-provisioned capacity [I-D.ietf-tsvwg-rfc5405bis] where
   there are no flows competing for capacity. In this case, a sender-
   based rate control mechanism and a single channel is sufficient.

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

3.9.2. 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.

3.9.3. Transport Features

   The transport features provided by FLUTE/ALC are:

   o unicast, multicast, anycast or IPv4 broadcast transmission,

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

   o fully reliable or partially reliable delivery (of file or in-
      memory objects), using proactive packet erasure coding (AL-FEC) to
      recover from packet erasures,

   o ordered or unordered delivery (of file or in-memory objects),

   o error detection (based on the UDP checksum),

   o per-packet authentication,

   o per-packet integrity,

   o per-packet replay protection,

   o congestion control for layered flows (e.g., with WEBRC).

3.10. NACK-Oriented Reliable Multicast (NORM)

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

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

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

3.10.1. Protocol Description

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

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

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

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

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

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

3.10.2. 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.

3.10.3. Transport Features

   The transport features provided by NORM are:

   o unicast or multicast transport,

   o unidirectional communication,

   o stream-oriented delivery in a single stream or object-oriented
      delivery of in-memory data or file bulk content objects,

   o fully reliable (NACK-based) or partially reliable (using erasure
      coding both proactively and as part of ARQ) delivery,

   o unordered delivery,

   o error detection (relies on UDP checksum),

   o segmentation,

   o data bundling (using Nagle's algorithm),

   o flow control (timer-based and/or ack-based),

   o congestion control (also supporting fixed rate reliable or
      unreliable delivery).

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

   Transport Layer Security (TLS) [RFC5246]} and Datagram TLS (DTLS)
   [RFC6347]} are IETF protocols that provide several security-related
   features to applications. TLS is designed to run on top of a
   reliable streaming transport protocol (usually TCP), while DTLS is
   designed to run on top of a best-effort datagram protocol (UDP or
   DCCP [RFC5238]). At the time of writing, the current version of TLS
   is 1.2; which 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-1990s
   to support protection of HTTP traffic.

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

3.11.1. Protocol Description

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

   o Confidentiality

   o Data integrity

   o Peer authentication (optional)

o Perfect forward secrecy (optional)

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

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

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

o Record replay detection (optional).

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

o Coveyance of IP don't fragment (DF) bit settings by application.

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

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

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

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

3.11.3. Transport Features

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

o message fragmentation,

o authentication and integrity via message authentication codes
   (MAC),

o data encryption,

o scheduling transmission using the underlying transport protocol.

DTLS augments the TLS record protocol with:

o ordering and replay protection, implemented using sequence
   numbers.

Several protocols are layered on top of the record protocol. These
include the handshake, alert, and change cipher spec protocols.
There is also the data protocol, used to carry application traffic.
The handshake protocol is used to establish cryptographic and
compression parameters when a connection is first set up. In DTLS,
this protocol also has a basic fragmentation and retransmission
capability and a cookie-like mechanism to resist DoS attacks. (TLS

compression is not recommended at present). The alert protocol is
used to inform the peer of various conditions, most of which are
terminal for the connection. The change cipher spec protocol is used
to synchronize changes in cryptographic parameters for each peer.

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

o authentication of one end or both ends of a connection,

o confidentiality,

o cryptographic integrity protection.

Both TLS and DTLS are unicast-only.

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

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

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

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

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

3.12.1. 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
also contain a message body carrying application data. The server
responds with a status or error code followed by a MIME-like message
containing information about the server and information about the
data. This may include a message body. It is possible to specify a
data format for the message body using MIME media types [RFC2045].
The protocol has additional features, some relevant to pseudo-
transport are described below.

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

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

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

3.12.2. 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, the API exposes a registration of
callbacks for a server that requests client authentication and when
certificate verification is needed.

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

3.12.3. Transport features

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

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

o uni- or bidirectional communication,

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

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

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

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

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

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

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

o confidentiality,

o integrity protection.

4. Congestion Control

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

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

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

   Another class of applications prefer a transport service that yields
   to other (higher-priority) traffic, such as interactive
   transmissions. While most traffic in the Internet uses loss-based
   congestion control and therefore need to fill the network buffers (to
   a certain level if Active Queue Management (AQM) is used), low-
   priority congestion control methods often react to changes in delay
   as an earlier indication of congestion. This approach tends to

induce less loss than a loss-based method but does generally not
compete well with loss-based traffic across shared bottleneck links.
Therefore, methods such as LEDBAT [RFC6824], are deployed in the
Internet for scavenger traffic that aim to only utilize otherwise
unused capacity.

5. Transport Features

The tables below summarize some key features to illustrate the range
of functions provided across the IETF-specified transports. Figure 1
considers transports that may be directly layered over the network,
and Figure 2 considers transports layered over another transport
service. Features that are permitted, but not required, are marked
as "Poss" indicating that it is possible for the transport service to
offer this feature.

Feature	TCP	MPTCP	SCTP	UDP	UDP-L	DCCP	ICMP
Datagram	No	No	Yes	Yes	Yes	Yes	Yes
Conn. Oriented	Yes	Yes	Yes	No	No	Yes	No
Reliability	Yes	Yes	Yes	No	No	No	No
Partial Rel.	No	No	Poss	N/A	N/A	Yes	N/A
Corupt. Tol	No	No	No	No	Yes	Yes	No
Cong.Control	Yes	Yes	Yes	No	No	Yes	No
Endpoint	1	>=1	>=1	1	1	1	1
Multicast Cap.	No	No	No	Yes	Yes	No	No

Figure 1: Summary comparison: Transport protocols

```
   +---------------+------+------+------+------+------+
   | Feature       | RTP  | FLUTE| NORM |(D)TLS| HTTP |
   +---------------+------+------+------+------+------+
   | Datagram      | Yes  | No   | Both | Both | No   |
   +---------------+------+------+------+------+------+
   | Conn. Oriented| No   | Yes  | Yes  | Yes  | Yes  |
   +---------------+------+------+------+------+------+
   | Reliability   | No   | Yes  | Poss | Poss | Yes  |
   +---------------+------+------+------+------+------+
   | Partial R     | Poss | No   | Poss | No   | No   |
   +---------------+------+------+------+------+------+
   | Corupt. Tol   | Poss | No   | No   | No   | No   |
   +---------------+------+------+------+------+------+
   | Cong.Control  | Poss | Poss | Poss | N/A  | N/A  |
   +---------------+------+------+------+------+------+
   | Endpoint      | >=1  | >=1  | >=1  | 1    | 1    |
   +---------------+------+------+------+------+------+
   | Multicast Cap.| Yes  | Yes  | Yes  | No   | No   |
   +---------------+------+------+------+------+------+
```

              Figure 2: Upper layer transports and frameworks

   The transport protocol features described in this document could be
   used as a basis for defining common transport features:

   o Control Functions

      * Addressing

         + unicast (TCP, MPTCP, SCTP, UDP, UDP-Lite, DCCP, ICMP, RTP,
            TLS, HTTP)

         + multicast (UDP, UDP-Lite, DCCP, ICMP, RTP, FLUTE/ALC, NORM).
            Note that, as TLS and DTLS are unicast-only, there is no
            widely deployed mechanism for supporting the features in the
            Security section below when using multicast addressing.

         + IPv4 broadcast (UDP, UDP-Lite, ICMP)

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

      * Association type

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

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

   * Multihoming support

      + resilience and mobility (MPTCP, SCTP)

      + load-balancing (MPTCP)

      + address family multiplexing (MPTCP, SCTP)

   * Middlebox cooperation

      + application-class signaling to middleboxes (DCCP)

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

   * Signaling

      + control information and error signaling (ICMP)

      + performance metric reporting (RTP)

o Delivery

   * Reliability

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

      + partially reliable delivery (SCTP, NORM)

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

         - with specified policy for dropped messages (SCTP)

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

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

      + error detection

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

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

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

      * Ordering

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

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

      * Type/framing

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

            - with multiple streams per association (SCTP, HTTP2)

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

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

            - with partial delivery of object ranges (HTTP)

      * Directionality

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

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

   o Transmission control

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

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

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

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

   * stream scheduling prioritization (SCTP, HTTP2)

   * endpoint multiplexing (MPTCP)

   o Security

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

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

   * confidentiality (TLS, DTLS)

   * cryptographic integrity protection (TLS, DTLS)

   * replay protection (FLUTE/ALC, DTLS)

[6]. IANA Considerations

   This document has no considerations for IANA.

[7]. Security Considerations

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

[8]. Contributors

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

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

   o Section 3.4 on UDP was contributed by Kevin Fall (kfall@kfall.com)

   o Section 3.3 on SCTP was contributed by Michael Tuexen (tuexen@fh-
      muenster.de) and Karen Nielsen (karen.nielsen@tieto.com)

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

   o Section 3.9 on FLUTE/ALC was contributed by Vincent Roca
      (vincent.roca@inria.fr)

   o Section 3.10 on NORM was contributed by Brian Adamson
      (brian.adamson@nrl.navy.mil)

   o Section 3.11 on TLS and DTLS was contributed by Ralph Holz
      (ralph.holz@nicta.com.au) and Olivier Mehani
      (olivier.mehani@nicta.com.au)

   o Section 3.12 on HTTP was contributed by Dragana Damjanovic
      (ddamjanovic@mozilla.com)

9. Acknowledgments

10. Informative References

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

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

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

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

   [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>.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

   [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>.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

   [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>.

   [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>.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

   [I-D.ietf-aqm-ecn-benefits]
              Fairhurst, G. and M. Welzl, "The Benefits of using
              Explicit Congestion Notification (ECN)", draft-ietf-aqm-
              ecn-benefits-08 (work in progress), November 2015.

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

   [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-sctp-failover]
              Nishida, Y., Natarajan, P., Caro, A., Amer, P., and K.
              Nielsen, "SCTP-PF: Quick Failover Algorithm in SCTP",
              draft-ietf-tsvwg-sctp-failover-14 (work in progress),
              December 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.

   [I-D.ietf-tcpm-cubic]
              Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and
              R. Scheffenegger, "CUBIC for Fast Long-Distance Networks",
              draft-ietf-tcpm-cubic-00 (work in progress), June 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.

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

Authors' Addresses

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

   Email: gorry@erg.abdn.ac.uk

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

   Email: ietf@trammell.ch

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

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