       Services provided by IETF transport protocols and congestion control
                                  mechanisms
                        draft-ietf-taps-transports-04

Abstract

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

the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction

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

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

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

2. Terminology

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

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

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

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

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

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

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

3. Existing Transport Protocols

   This section provides a list of known IETF transport protocol and
   transport protocol frameworks.

   [EDITOR'S NOTE: Contributions to the subsections below are welcome]

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
   mechanisms for orderly completion and immediate teardown of a
   connection. TCP is defined by a family of RFCs [RFC4614].

   TCP provides multiplexing to multiple sockets on each host using port
   numbers. 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 has a different role as
   it is often used to indicate the requested service.

   TCP partitions a continuous stream of bytes into segments, sized to
   fit in IP packets. ICMP-based PathMTU discovery [RFC1191][RFC1981]
   as well as Packetization Layer Path MTU Discovery (PMTUD) [RFC4821]
   are supported.

   Each byte in the stream is identified by a sequence number. The
   sequence number is used to order segments on receipt, to identify
   segments in acknowledgments, and to detect unacknowledged segments
   for retransmission. This is the basis of TCP's reliable, ordered
   delivery of data in a stream. TCP Selective Acknowledgment [RFC2018]
   extends this mechanism by making it possible to identify missing
   segments more precisely, reducing spurious retransmission.

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

   All TCP senders provide Congestion Control: This uses a separate
   window, where each time congestion is detected, this congestion
   window is reduced. A receiver detects congestion using one of three
   mechanisms: A retransmission timer, detection of loss (interpreted as
   a congestion signal), or Explicit Congestion Notification (ECN)
   [RFC3168] to provide early signaling (see
   [I-D.ietf-aqm-ecn-benefits])

   A TCP protocol instance can be extended [RFC4614] and tuned. Some
   features are sender-side only, requiring no negotiation with the
   receiver; some are receiver-side only, some are explicitly negotiated
   during connection setup.

   By default, TCP segment partitioning uses Nagle's algorithm [RFC0896]
   to buffer data at the sender into large segments, potentially
   incurring sender-side buffering delay; this algorithm can be disabled

by the sender to transmit more immediately, e.g. to enable smoother
interactive sessions.

[EDITOR'S NOTE: add URGENT and PUSH flag (note [RFC6093] says SHOULD
NOT use due to the range of TCP implementations that process TCP
urgent indications differently.)]

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

A TCP service is unicast.

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.

In API implementations derived from the BSD Sockets API, TCP sockets
are created using the "SOCK_STREAM" socket type.

The features used by a protocol instance may be set and tuned via
this API.

(more on the API goes here)

3.1.3. Transport Protocol Components

The transport protocol components provided by TCP are:

o unicast

o connection setup with feature negotiation and application-to-port
   mapping

o port multiplexing

o reliable delivery

o ordered delivery for each byte stream

o error detection (checksum)

o   segmentation

o   stream-oriented delivery in a single stream

o   data bundling (Nagle's algorithm)

o   flow control

o   congestion control

[EDITOR'S NOTE: discussion of how to map this to features and TAPS: what does the higher layer need to decide? what can the transport layer decide based on global settings? what must the transport layer decide based on network characteristics?]

3.2. Multipath TCP (MP-TCP)

[EDITOR'S NOTE: a few sentences describing Multipath TCP [RFC6824] go here. Note that this adds transport-layer multihoming to the components TCP provides]

3.3. Stream Control Transmission Protocol (SCTP)

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

SCTP was originally developed for transporting telephony signalling messages and is deployed in telephony signalling networks, especially in mobile telephony networks. Additionally, it is used in the WebRTC framework for data channels and is therefore deployed in all WEB-browsers supporting WebRTC.

[EDITOR'S NOTE: Michael Tuexen and Karen Nielsen signed up as contributors for these sections.]

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 do and only supports unicast.

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

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

SCTP provides a message-oriented service. Multiple small user
messages can be bundled into a single SCTP packet to improve the
efficiency. For example, this bundling may be done by delaying user
messages at the sender side similar to the Nagle algorithm used by
TCP. User messages which would result in IP packets larger than the
MTU will be fragmented at the sender side and reassembled at the
receiver side. There is no protocol limit on the user message size.
ICMP-based path MTU discovery as specified for IPv4 in [RFC1191] and
for IPv6 in [RFC1981] as well as packetization layer path MTU
discovery as specified in [RFC4821] with probe packets using the
padding chunks defined the [RFC4820] are supported.

[RFC4960] specifies a TCP friendly congestion control to protect the
network against overload. SCTP also uses a sliding window flow
control to protect receivers against overflow.

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. Only all ordered messages sent on the same
stream are delivered at the receiver in the same order as sent by the
sender. For user messages not requiring fragmentation, this
minimises head of line blocking. The base protocol defined in
[RFC4960] doesn't allow interleaving of user-messages, which results
in sending a large message on one stream can block the sending of
user messages on other streams. [I-D.ietf-tsvwg-sctp-ndata]
overcomes this limitation. Furthermore, [I-D.ietf-tsvwg-sctp-ndata]
specifies multiple algorithms for the sender side selection of which
streams to send data from supporting a variety of scheduling
algorithms including priority based ones. The stream re-

configuration extension defined in [RFC6525] allows to reset streams
during the lifetime of an association and to increase the number of
streams, if the number of streams negotiated in the SCTP handshake is
not sufficient.

According to [RFC4960], 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, similar to the TCP
behaviour. In addition to this reliable transfer, the partial
reliability extension defined in [RFC3758] allows the sender to
abandon user messages. The application can specify the policy for
abandoning user messages. Examples for these policies include:

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

o 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.

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

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

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

[I-D.ietf-tsvwg-natsupp] defines a methods for end-hosts and
middleboxes to provide for NAT support 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 with in the WebRTC context.

Having a well defined API is also a feature provided by SCTP as
described in the next subsection.

3.3.2. Interface Description

[RFC4960] defines an abstract API for the base protocol. An
extension to the BSD Sockets API is defined in [RFC6458] and covers:

o the base protocol defined in [RFC4960].

o the SCTP Partial Reliability extension defined in [RFC3758].

o the SCTP Authentication extension defined in [RFC4895].

o the SCTP Dynamic Address Reconfiguration extension defined in
   [RFC5061].

For the following SCTP protocol extensions the BSD Sockets API
extension is defined in the document specifying the protocol
extensions:

o the SCTP SACK-IMMEDIATELY extension defined in [RFC7053].

o the SCTP Stream Reconfiguration extension defined in [RFC6525].

o the UDP Encapsulation of SCTP packets extension defined in
   [RFC6951].

o the additional PR-SCTP policies defined in
   [I-D.ietf-tsvwg-sctp-prpolicies].

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 as 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.

A couple of 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 allow dealing with the SCTP specific information without using ancillary data in the form of additional cmsgs, which are also defined. These functions provide support for detecting partial delivery of user messages and notifications.

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

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

3.3.3. Transport Protocol Components

The transport protocol components provided by SCTP are:

o unicast

o connection setup with feature negotiation and application-to-port mapping

o port multiplexing

o reliable or partially reliable delivery

o ordered and unordered delivery within a stream

o support for multiple concurrent streams

o support for stream scheduling prioritization

o flow control

o message-oriented delivery

o congestion control

o user message bundling

o user message fragmentation and reassembly

o strong error detection (CRC32C)

o transport layer multihoming for resilience

o transport layer mobility

[EDITOR'S NOTE: update this list.]

3.4. User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) [RFC0768] [RFC2460] is an IETF
standards track transport protocol. It provides a uni-directional,
datagram protocol which preserves message boundaries. It provides
none of the following transport features: error correction,
congestion control, or flow control. It can be used to send
broadcast datagrams (IPv4) or multicast datagrams (IPv4 and IPv6), in
addition to unicast (and anycast) datagrams. IETF guidance on the
use of UDP is provided in[RFC5405]. UDP is widely implemented and
widely used by common applications, especially DNS.

3.4.1. Protocol Description

UDP is a connection-less protocol which maintains message boundaries,
with no connection setup or feature negotiation. The protocol uses
independent messages, ordinarily called datagrams. The lack of error
control and flow control implies messages may be damaged, re-ordered,
lost, or duplicated in transit. A receiving application unable to
run sufficiently fast or frequently may miss messages. The lack of
congestion handling implies UDP traffic may cause the loss of
messages from other protocols (e.g., TCP) when sharing the same
network paths. UDP traffic can also cause the loss of other UDP
traffic in the same or other flows for the same reasons.

Messages with bit errors are ordinarily detected by an invalid end-
to-end checksum and are discarded before being delivered to an
application. There are some exceptions to this general rule,
however. UDP-Lite (see [RFC3828], and below) provides the ability
for portions of the message contents to be exempt from checksum
coverage. It is also possible to create UDP datagrams with no
checksum, and while this is generally discouraged [RFC1122]
[RFC5405], certain special cases permit its use [RFC6935]. The
checksum support considerations for omitting the checksum are defined
in [RFC6936]. Note that due to the relatively weak form of checksum
used by UDP, applications that require end to end integrity of data

are recommended to include a stronger integrity check of their
payload data.

On transmission, UDP encapsulates each datagram into an IP packet,
which may in turn be fragmented by IP. Applications concerned with
fragmentation or that have other requirements such as receiver flow
control, congestion control, PathMTU discovery/PLPMTUD, support for
ECN, etc need to be provided by protocols other than UDP [RFC5405].

3.4.2. Interface Description

[RFC0768] describes basic requirements for an API for UDP. Guidance
on use of common APIs is provided in [RFC5405].

A UDP endpoint consists of a tuple of (IP address, port number).
Demultiplexing using multiple abstract endpoints (sockets) on the
same IP address are supported. The same socket may be used by a
single server to interact with multiple clients (note: this behavior
differs from TCP, which uses a pair of tuples to identify a
connection). Multiple server instances (processes) binding 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
[RFC5405].

3.4.3. Transport Protocol Components

The transport protocol components provided by UDP are:

o unidirectional

o port multiplexing

o 2-tuple endpoints

o IPv4 broadcast, multicast and anycast

o IPv6 multicast and anycast

o IPv6 jumbograms

o message-oriented delivery

o error detection (checksum)

o checksum optional

3.5. Lightweight User Datagram Protocol (UDP-Lite)

   The Lightweight User Datagram Protocol (UDP-Lite) [RFC3828] is an
   IETF standards track transport protocol. UDP-Lite provides a
   bidirectional set of logical unicast or multicast message streams
   over a datagram protocol. IETF guidance on the use of UDP-Lite is
   provided in [RFC5405].

   [EDITOR'S NOTE: Gorry Fairhurst signed up as a contributor for this
   section.]

3.5.1. Protocol Description

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

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

   UDP-Lite fragments packets into IP packets, constrained by the
   maximum size of IP packet.

   UDP-Lite changes the semantics of the UDP "payload length" field to
   that of a "checksum coverage length" field. Otherwise, UDP-Lite is
   semantically identical to UDP. Applications using UDP-Lite therefore
   can not make assumptions regarding the correctness of the data
   received in the insensitive part of the UDP-Lite payload.

   As for UDP, mechanisms for receiver flow control, congestion control,
   PMTU or PLPMTU discovery, support for ECN, etc need to be provided by
   upper layer protocols [RFC5405].

   Examples of use include a class of applications that can derive
   benefit from having partially-damaged payloads delivered, rather than
   discarded. One use is to support error tolerate payload corruption
   when used over paths that include error-prone links, another
   application is when header integrity checks are required, but payload
   integrity is provided by some other mechanism (e.g. [RFC6936].

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

3.5.2. Interface Description

There is no current API specified in the RFC Series, but guidance on
use of common APIs is provided in [RFC5405].

The interface of UDP-Lite differs from that of UDP by the addition of
a single (socket) option that communicates a checksum coverage length
value: at the sender, this specifies the intended checksum coverage,
with the remaining unprotected part of the payload called the "error-
insensitive part". The checksum coverage may also be made visible to
the application via the UDP-Lite MIB module [RFC5097].

3.5.3. Transport Protocol Components

The transport protocol components provided by UDP-Lite are:

o unicast

o IPv4 broadcast, multicast and anycast

o port multiplexing

o non-reliable, non-ordered delivery

o message-oriented delivery

o partial integrity protection

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 unreliable messages.

[EDITOR'S NOTE: Gorry Fairhurst signed up as a contributor for this
section.]

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

It offers low overhead, and many characteristics common to UDP, but
can avoid "Re-inventing the wheel" each time a new multimedia
application emerges. Specifically it includes core functions

(feature negotiation, path state management, RTT calculation, PMTUD,
etc): This allows applications to use a compatible method defining
how they send packets and where suitable to choose common algorithms
to manage their functions. Examples of suitable applications include
interactive applications, streaming media or on-line games [RFC4336].

3.6.1. Protocol Description

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

It provides multiplexing to multiple sockets on each host using port
numbers. An active DCCP session is identified by its four-tuple of
local and remote IP addresses and local port and remote port numbers.
At connection setup, DCCP also exchanges the the service code
[RFC5595] mechanism to allow transport instantiations to indicate the
service treatment that is expected from the network.

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

Each message is identified by a sequence number. The sequence number
is used to identify segments in acknowledgments, to detect
unacknowledged segments, to measure RTT, etc. The protocol may
support ordered or unordered delivery of data, and does not itself
provide retransmission. There is a Data Checksum option, which
contains a strong CRC, lets endpoints detect application data
corruption. It also supports reduced checksum coverage, a partial
integrity mechanisms similar to UDP-lIte.

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

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

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

DCCP use a Connect packet to start a session, and permits half-connections that allow each client to choose features it wishes to support. Simultaneous open [RFC5596], as in TCP, can enable interoperability in the presence of middleboxes. The Connect packet includes a Service Code field [RFC5595] designed to allow middle boxes and endpoints to identify the characteristics required by a session. A lightweight UDP-based encapsulation (DCCP-UDP) has been defined [RFC6773] that permits DCCP to be used over paths where it is not natively supported. Support in NAPT/NATs is defined in [RFC4340] and [RFC5595].

Upper layer protocols specified on top of DCCP include: DTLS [RFC5595], RTP [RFC5672], ICE/SDP [RFC6773].

A DCCP service is unicast.

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

3.6.2. Interface Description

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

There is no current API specified in the RFC Series.

3.6.3. Transport Protocol Components

The transport protocol components provided by DCCP are:

o unicast

o connection setup with feature negotiation and application-to-port mapping

o Service Codes

o port multiplexing

o non-reliable, ordered delivery

o flow control (slow receiver function)

o drop notification

o timestamps

o message-oriented delivery

o partial integrity protection

3.7. Realtime Transport Protocol (RTP)

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

[EDITOR'S NOTE: Varun Singh signed up as contributor for this
section.]

3.8. NACK-Oriented Reliable Multicast (NORM)

NORM is an IETF standards track protocol specified in [RFC5740]. The
protocol was designed to support reliable bulk data dissemination to
receiver groups using IP Multicast but also provides for point-to-
point unicast operation. Its support for bulk data dissemination
includes discrete file or computer memory-based "objects" as well as
byte- and message-streaming. NORM is designed to incorporate packet
erasure coding as an inherent part of its selective ARQ in response
to receiver negative acknowledgements. The packet erasure coding can
also be proactively applied for forward protection from packet loss.
NORM transmissions are governed by TCP-friendly congestion control.
NORM's reliability, congestion control, and flow control mechanism
are distinct components and can be separately controlled to meet
different application needs.

3.8.1. Protocol Description

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

The NORM transport model supports bulk "objects" such as file or in-
memory content but also can treat a stream of data as a logical bulk
object for purposes of packet erasure coding. In the case of stream
transport, NORM can support either byte streams or message streams
where application-defined message boundary information is carried in

the NORM protocol messages. This allows the receiver(s) to join/re-
join and recover message boundaries mid-stream as needed.
Application content is carried and identified by the NORM protocol
with encoding symbol identifiers depending upon the Forward Error
Correction (FEC) Scheme [RFC3452] configured. NORM uses NACK-based
selective ARQ to reliably deliver the application content to the
receiver(s). NORM proactively measures round-trip timing information
to scale ARQ timers appropriately and to support congestion control.
For multicast operation, timer-based feedback suppression is uses to
achieve group size scaling with low feedback traffic levels. The
feedback suppression is not applied for unicast operation.

NORM uses rate-based congestion control based upon the TCP-Friendly
Rate Control (TFRC) [RFC4324] principles that are also used in DCCP
[RFC4340]. NORM uses control messages to measure RTT and collect
congestion event (e..g, loss event, ECN event, etc) information from
the receiver(s) to support dynamic rate control adjustment. The TCP-
Friendly Multicast Congestion Conrol (TFMCC) [RFC4654] used provides
some extra features to support multicast but is functionally
equivalent to TFRC in the unicast case. Because NORM's reliability
mechanism is decoupled from congestion control, alternative forms of
transport service can be invoked. For example, fixed-rate reliable
delivery can be supported or unreliable (but optionally "better than
best effort" via packet erasure coding) delivery with rate-control
per TFRC can be achieved. Additionally, alternative congestion
control techniques may be applied. For example, TFRC rate control
with congestion event detection based on ECN for links with high
packet loss (e.g., wireless) has been implemented and demonstrated
with NORM.

While NORM is NACK-based for reliability transfer, it also supports a
postitive acknowledgment (ACK) mechanim that can be used for receiver
flow control. Again, since this mechanism is decoupled from the
reliability and congestion control, applications that have different
needs in this aspect can use the protocol differently. One example
is use of NORM for quasi-reliable delivery where timely delivery of
newer content may be favored over completely reliable deliver of
older contenet within buffering and RTT constraints.

3.8.2. 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.

3.8.3. Transport Protocol Components

   The transport protocol components provided by NORM are:

   o unicast

   o multicast

   o port multiplexing

   o reliable delivery

   o ordered delivery for each byte or mesage stream

   o unordered delivery of in-memory data or file bulk content objects

   o error detection (UDP checksum)

   o segmentation

   o stream-oriented delivery in a single stream

   o object-oriented delivery of discrete data or file items

   o data bundling (Nagle's algorithm)

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

   o congestion control

   o packet erasure coding both proactively and as part of ARQ

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

   pseudo transport

   [NOTE: A few words on TLS [RFC5246] and DTLS [RFC6347] here, and how
   they get used by other protocols to meet security goals as an add-on
   interlayer above transport.]

3.9.1. Protocol Description

3.9.2. Interface Description

3.9.3. Transport Protocol Components

3.10. Hypertext Transport Protocol (HTTP) as a pseudotransport

   Hypertext Transfer Protocol (HTTP) is an application-level protocol
   widely use on the Internet. Version 1.1 of the protocol is specified
   in [RFC7230] [RFC7231] [RFC7232] [RFC7233] [RFC7234] [RFC7235], and
   version 2 in [RFC7540]. Furthermore, HTTP is used as a substrate for
   other application-layer protocols. There are various reasons for
   this practice listed in [RFC3205]; these include being a well-known
   and well-understood protocol, reusability of existing servers and
   client libraries, easy use of existing security mechanisms such as
   HTTP digest authentication [RFC2617] and TLS [RFC5246], the ability
   of HTTP to traverse firewalls which makes it work with a lot of
   infrastructure, and cases where a application server often needs to
   support HTTP anyway.

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

3.10.1. Protocol Description

   Hypertext Transfer Protocol (HTTP) is a request/response protocol. A
   client sends a request containing a request method, URI and protocol
   version followed by a MIME-like message (see [RFC7231] for the
   differences between an HTTP object and a MIME message), containing
   information about the client and request modifiers. The message can
   contain a message body carrying application data as well. The server
   responds with a status or error code followed by a MIME-like message
   containing information about the server and information about carried
   data and it can include a message body. It is possible to specify a
   data format for the message body using MIME media types [RFC2045].
   Furthermore, the protocol has numerous additional features; features
   relevant to pseudotransport are described below.

   Content negotiation, specified in [RFC7231], is a mechanism provided
   by HTTP for selecting a representation on a requested resource. The
   client and server negotiate acceptable data formats, charsets, data
   encoding (e.g. data can be transferred compressed, gzip), etc. HTTP
   can accommodate exchange of messages as well as data streaming (using
   chunked transfer encoding [RFC7230]). It is also possible to request
   a part of a resource using range requests specified in [RFC7233].

The protocol provides powerful cache control signalling defined in
[RFC7234].

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

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

Application layer protocols using HTTP as substrate may use existing
method and data formats, or specify new methods and data formats.
Furthermore some protocols may not fit a request/response paradigm
and instead rely on HTTP to send messages (e.g. [RFC6546]). Because
HTTP is working in many restricted infrastructures, it is also used
to tunnel other application-layer protocols.

3.10.2. Interface Description

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

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

Representational State Transfer (REST) [REST] is another example how
applications can use HTTP as transport protocol. REST is an

architecture style for building application on the Internet. It uses
HTTP as a communication protocol.

3.10.3. Transport Protocol Components

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

o unicast

o reliable delivery

o ordered delivery

o message and stream-oriented

o object range request

o message content type negotiation

o congestion control

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

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

o confidentiality

o integrity protection

3.11. WebSockets

[RFC6455]

[EDITOR'S NOTE: No identified contributor for this section yet.]

3.11.1. Protocol Description

3.11.2. Interface Description

3.11.3. Transport Protocol Components

4. Transport Service Features

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

o Destination selection features

   * unicast

   * broadcast (IPv4 only)

   * multicast

   * anycast

   * transport layer multihoming for resilience

   * transport layer mobility

   * port multiplexing

   * service codes

o Connection setup features

   * connection setup with feature negotiation and application-to-
      port mapping

o Delivery features

   * reliable delivery

   * partially reliable delivery

   * unreliable delivery

   * packet erasure coding

   * ordered delivery

   * unordered delivery

   * stream-oriented delivery

   * message-oriented delivery

   * message fragmentation

   * object-oriented delivery of discrete data or file items

   * unordered delivery of in-memory data or file bulk content
      objects

   *   object range request

   *   object content type negotiation

   *   single streaming

   *   multiple streaming

   *   stream scheduling prioritization

   *   segmentation

   *   data bundling (Nagle's algorithm)

   *   message bundling

   o Transmission control features

   *   timer-based rate control

   *   ack-based flow control

   *   drop notification

   *   packet erasure coding

   *   congestion control

   o Integrity protection features

   *   checksum for error detection

   *   partial checksum protection

   *   checksum optional

   *   cryptographic integrity protection

   o Security features

   *   authentication of one end of a connection

   *   authentication of both ends of a connection

   *   confidentiality

   The next revision of this document will define transport service
   features based upon this list.

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

4.1. Complete Protocol Feature Matrix

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

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

The table below summarises protocol mechanisms that have been standardised. It does not make an assessment on whether specific implementations are fully compliant to these specifications.

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

Note (1): this feature requires support in an upper layer protocol.

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

5. IANA Considerations

   This document has no considerations for IANA.

6. Security Considerations

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

7. Contributors

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

   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)

   o Section 3.8 on NORM was contributed by Brian Adamson
      (brian.adamson@nrl.navy.mil)

   o Section 3.10 on HTTP was contributed by Dragana Damjanovic
      (ddamjanovic@mozilla.com)

8. Acknowledgments

   This work is partially supported by the European Commission under
   grant agreement FP7-ICT-318627 mPlane; support does not imply
   endorsement.

9. References

9.1. Normative References

   [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September
              1981.

9.2. Informative References

   [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768,
              August 1980.

   [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC
              793, September 1981.

   [RFC0896] Nagle, J., "Congestion control in IP/TCP internetworks",
              RFC 896, January 1984.

   [RFC1122] Braden, R., "Requirements for Internet Hosts -
              Communication Layers", STD 3, RFC 1122, October 1989.

   [RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191,
              November 1990.

   [RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery
              for IP version 6", RFC 1981, August 1996.

   [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP
              Selective Acknowledgment Options", RFC 2018, October 1996.

   [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail
              Extensions (MIME) Part One: Format of Internet Message
              Bodies", RFC 2045, November 1996.

   [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6
              (IPv6) Specification", RFC 2460, December 1998.

   [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, June 1999.

   [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition
              of Explicit Congestion Notification (ECN) to IP", RFC
              3168, September 2001.

   [RFC3205] Moore, K., "On the use of HTTP as a Substrate", BCP 56,
              RFC 3205, February 2002.

   [RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's
              Initial Window", RFC 3390, October 2002.

   [RFC3436] Jungmaier, A., Rescorla, E., and M. Tuexen, "Transport
              Layer Security over Stream Control Transmission Protocol",
              RFC 3436, December 2002.

   [RFC3452] Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley,
              M., and J. Crowcroft, "Forward Error Correction (FEC)
              Building Block", RFC 3452, December 2002.

   [RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P.
              Conrad, "Stream Control Transmission Protocol (SCTP)
              Partial Reliability Extension", RFC 3758, May 2004.

   [RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., and
              G. Fairhurst, "The Lightweight User Datagram Protocol
              (UDP-Lite)", RFC 3828, July 2004.

   [RFC4324] Royer, D., Babics, G., and S. Mansour, "Calendar Access
              Protocol (CAP)", RFC 4324, December 2005.

   [RFC4336] Floyd, S., Handley, M., and E. Kohler, "Problem Statement
              for the Datagram Congestion Control Protocol (DCCP)", RFC
              4336, March 2006.

   [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram
              Congestion Control Protocol (DCCP)", RFC 4340, March 2006.

   [RFC4341] Floyd, S. and E. Kohler, "Profile for Datagram Congestion
              Control Protocol (DCCP) Congestion Control ID 2: TCP-like
              Congestion Control", RFC 4341, March 2006.

   [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,
              March 2006.

   [RFC4614] Duke, M., Braden, R., Eddy, W., and E. Blanton, "A Roadmap
              for Transmission Control Protocol (TCP) Specification
              Documents", RFC 4614, September 2006.

   [RFC4654] Widmer, J. and M. Handley, "TCP-Friendly Multicast
              Congestion Control (TFMCC): Protocol Specification", RFC
              4654, August 2006.

   [RFC4820] Tuexen, M., Stewart, R., and P. Lei, "Padding Chunk and
              Parameter for the Stream Control Transmission Protocol
              (SCTP)", RFC 4820, March 2007.

   [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU
              Discovery", RFC 4821, March 2007.

[RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla,
           "Authenticated Chunks for the Stream Control Transmission
           Protocol (SCTP)", RFC 4895, August 2007.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC
           4960, September 2007.

[RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M.
           Kozuka, "Stream Control Transmission Protocol (SCTP)
           Dynamic Address Reconfiguration", RFC 5061, September
           2007.

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
           (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP
           Friendly Rate Control (TFRC): Protocol Specification", RFC
           5348, September 2008.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines
           for Application Designers", BCP 145, RFC 5405, November
           2008.

[RFC5595] Fairhurst, G., "The Datagram Congestion Control Protocol
           (DCCP) Service Codes", RFC 5595, September 2009.

[RFC5596] Fairhurst, G., "Datagram Congestion Control Protocol
           (DCCP) Simultaneous-Open Technique to Facilitate NAT/
           Middlebox Traversal", RFC 5596, September 2009.

[RFC5662] Shepler, S., Eisler, M., and D. Noveck, "Network File
           System (NFS) Version 4 Minor Version 1 External Data
           Representation Standard (XDR) Description", RFC 5662,
           January 2010.

[RFC5672] Crocker, D., "RFC 4871 DomainKeys Identified Mail (DKIM)
           Signatures -- Update", RFC 5672, August 2009.

[RFC5740] Adamson, B., Bormann, C., Handley, M., and J. Macker,
           "NACK-Oriented Reliable Multicast (NORM) Transport
           Protocol", RFC 5740, November 2009.

[RFC6773] Phelan, T., Fairhurst, G., and C. Perkins, "DCCP-UDP: A
           Datagram Congestion Control Protocol UDP Encapsulation for
           NAT Traversal", RFC 6773, November 2012.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP
           Authentication Option", RFC 5925, June 2010.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion
           Control", RFC 5681, September 2009.

[RFC6083] Tuexen, M., Seggelmann, R., and E. Rescorla, "Datagram
           Transport Layer Security (DTLS) for Stream Control
           Transmission Protocol (SCTP)", RFC 6083, January 2011.

[RFC6093] Gont, F. and A. Yourtchenko, "On the Implementation of the
           TCP Urgent Mechanism", RFC 6093, January 2011.

[RFC6525] Stewart, R., Tuexen, M., and P. Lei, "Stream Control
           Transmission Protocol (SCTP) Stream Reconfiguration", RFC
           6525, February 2012.

[RFC6546] Trammell, B., "Transport of Real-time Inter-network
           Defense (RID) Messages over HTTP/TLS", RFC 6546, April
           2012.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent,
           "Computing TCP's Retransmission Timer", RFC 6298, June
           2011.

[RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and
           UDP Checksums for Tunneled Packets", RFC 6935, April 2013.

[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement
           for the Use of IPv6 UDP Datagrams with Zero Checksums",
           RFC 6936, April 2013.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC
           6455, December 2011.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer
           Security Version 1.2", RFC 6347, January 2012.

[RFC6458] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V.
           Yasevich, "Sockets API Extensions for the Stream Control
           Transmission Protocol (SCTP)", RFC 6458, December 2011.

[RFC6691] Borman, D., "TCP Options and Maximum Segment Size (MSS)",
           RFC 6691, July 2012.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure,
           "TCP Extensions for Multipath Operation with Multiple
           Addresses", RFC 6824, January 2013.

[RFC6951]   Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream
            Control Transmission Protocol (SCTP) Packets for End-Host
            to End-Host Communication", RFC 6951, May 2013.

[RFC7053]   Tuexen, M., Ruengeler, I., and R. Stewart, "SACK-
            IMMEDIATELY Extension for the Stream Control Transmission
            Protocol", RFC 7053, November 2013.

[RFC7230]   Fielding, R. and J. Reschke, "Hypertext Transfer Protocol
            (HTTP/1.1): Message Syntax and Routing", RFC 7230, June
            2014.

[RFC7231]   Fielding, R. and J. Reschke, "Hypertext Transfer Protocol
            (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC7232]   Fielding, R. and J. Reschke, "Hypertext Transfer Protocol
            (HTTP/1.1): Conditional Requests", RFC 7232, June 2014.

[RFC7233]   Fielding, R., Lafon, Y., and J. Reschke, "Hypertext
            Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233,
            June 2014.

[RFC7234]   Fielding, R., Nottingham, M., and J. Reschke, "Hypertext
            Transfer Protocol (HTTP/1.1): Caching", RFC 7234, June
            2014.

[RFC7235]   Fielding, R. and J. Reschke, "Hypertext Transfer Protocol
            (HTTP/1.1): Authentication", RFC 7235, June 2014.

[RFC7301]   Friedl, S., Popov, A., Langley, A., and E. Stephan,
            "Transport Layer Security (TLS) Application-Layer Protocol
            Negotiation Extension", RFC 7301, July 2014.

[RFC7323]   Borman, D., Braden, B., Jacobson, V., and R.
            Scheffenegger, "TCP Extensions for High Performance", RFC
            7323, September 2014.

[RFC7540]   Belshe, M., Peon, R., and M. Thomson, "Hypertext Transfer
            Protocol Version 2 (HTTP/2)", RFC 7540, May 2015.

[I-D.ietf-aqm-ecn-benefits]
            Welzl, M. and G. Fairhurst, "The Benefits and Pitfalls of
            using Explicit Congestion Notification (ECN)", draft-ietf-
            aqm-ecn-benefits-00 (work in progress), October 2014.

[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-prpolicies]
          Tuexen, M., Seggelmann, R., Stewart, R., and S. Loreto,
          "Additional Policies for the Partial Reliability Extension
          of the Stream Control Transmission Protocol", draft-ietf-
          tsvwg-sctp-prpolicies-07 (work in progress), February
          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-03 (work in progress), March 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-07 (work in progress),
          February 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 Irvune),
          Chapter 5: Representational State Transfer", 2000.

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