UDP Usage Guidelines

Abstract

   The User Datagram Protocol (UDP) provides a minimal message-passing
   transport that has no inherent congestion control mechanisms. This
   document provides guidelines on the use of UDP for the designers of
   applications, tunnels, and other protocols that use UDP. Congestion
   control guidelines are a primary focus, but the document also
   provides guidance on other topics, including message sizes,
   reliability, checksums, middlebox traversal, the use of Explicit
   Congestion Notification (ECN), Differentiated Services Code Points
   (DSCPs), and ports.

   Because congestion control is critical to the stable operation of the
   Internet, applications and other protocols that choose to use UDP as
   an Internet transport must employ mechanisms to prevent congestion
   collapse and to establish some degree of fairness with concurrent
   traffic. They may also need to implement additional mechanisms,
   depending on how they use UDP.

   Some guidance is also applicable to the design of other protocols
   (e.g., protocols layered directly on IP or via IP-based tunnels),
   especially when these protocols do not themselves provide congestion
   control.

   This document obsoletes RFC 5405 and adds guidelines for multicast
   UDP usage.

Status of This Memo

   This memo documents an Internet Best Current Practice.

   This document is a product of the Internet Engineering Task Force
   (IETF). It represents the consensus of the IETF community. It has
   received public review and has been approved for publication by the
   Internet Engineering Steering Group (IESG). Further information on
   BCPs is available in Section 2 of RFC 7841.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc8085.

Copyright Notice

Table of Contents

[1](#). Introduction

   The User Datagram Protocol (UDP) [[RFC768](#)] provides a minimal,
   unreliable, best-effort, message-passing transport to applications
   and other protocols (such as tunnels) that wish to operate over IP.
   Both are simply called "applications" in the remainder of this
   document.

   Compared to other transport protocols, UDP and its UDP-Lite variant
   [[RFC3828](#)] are unique in that they do not establish end-to-end
   connections between communicating end systems. UDP communication
   consequently does not incur connection establishment and teardown
   overheads, and there is minimal associated end-system state. Because
   of these characteristics, UDP can offer a very efficient
   communication transport to some applications.

   A second unique characteristic of UDP is that it provides no inherent
   congestion control mechanisms. On many platforms, applications can
   send UDP datagrams at the line rate of the platform's link interface,
   which is often much greater than the available end-to-end path
   capacity, and doing so contributes to congestion along the path.
   [[RFC2914](#)] describes the best current practice for congestion control

in the Internet. It identifies two major reasons why congestion
control mechanisms are critical for the stable operation of the
Internet:

1. The prevention of congestion collapse, i.e., a state where an
    increase in network load results in a decrease in useful work
    done by the network.

2. The establishment of a degree of fairness, i.e., allowing
    multiple flows to share the capacity of a path reasonably
    equitably.

Because UDP itself provides no congestion control mechanisms, it is
up to the applications that use UDP for Internet communication to
employ suitable mechanisms to prevent congestion collapse and
establish a degree of fairness. [RFC2309] discusses the dangers of
congestion-unresponsive flows and states that "all UDP-based
streaming applications should incorporate effective congestion
avoidance mechanisms." [RFC7567] reaffirms this statement. This is
an important requirement, even for applications that do not use UDP
for streaming. In addition, congestion-controlled transmission is of
benefit to an application itself, because it can reduce self-induced
packet loss, minimize retransmissions, and hence reduce delays.
Congestion control is essential even at relatively slow transmission
rates. For example, an application that generates five 1500-byte UDP
datagrams in one second can already exceed the capacity of a 56 Kb/s
path. For applications that can operate at higher, potentially
unbounded data rates, congestion control becomes vital to prevent
congestion collapse and establish some degree of fairness. Section 3
describes a number of simple guidelines for the designers of such
applications.

A UDP datagram is carried in a single IP packet and is hence limited
to a maximum payload of 65,507 bytes for IPv4 and 65,527 bytes for
IPv6. The transmission of large IP packets usually requires IP
fragmentation. Fragmentation decreases communication reliability and
efficiency and should be avoided. IPv6 allows the option of
transmitting large packets ("jumbograms") without fragmentation when
all link layers along the path support this [RFC2675]. Some of the
guidelines in Section 3 describe how applications should determine
appropriate message sizes. Other sections of this document provide
guidance on reliability, checksums, middlebox traversal and use of
multicast.

This document provides guidelines and recommendations. Although most
UDP applications are expected to follow these guidelines, there do
exist valid reasons why a specific application may decide not to
follow a given guideline. In such cases, it is RECOMMENDED that

application designers cite the respective section(s) of this document
in the technical specification of their application or protocol and
explain their rationale for their design choice.

[RFC5405] was scoped to provide guidelines for unicast applications
only, whereas this document also provides guidelines for UDP flows
that use IP anycast, multicast, broadcast, and applications that use
UDP tunnels to support IP flows.

Finally, although this document specifically refers to usage of UDP,
the spirit of some of its guidelines also applies to other message-
passing applications and protocols (specifically on the topics of
congestion control, message sizes, and reliability). Examples
include signaling, tunnel or control applications that choose to run
directly over IP by registering their own IP protocol number with
IANA. This document is expected to provide useful background reading
to the designers of such applications and protocols.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and
"OPTIONAL" in this document are to be interpreted as described in
[RFC2119].

3. UDP Usage Guidelines

Internet paths can have widely varying characteristics, including
transmission delays, available bandwidths, congestion levels,
reordering probabilities, supported message sizes, or loss rates.
Furthermore, the same Internet path can have very different
conditions over time. Consequently, applications that may be used on
the Internet MUST NOT make assumptions about specific path
characteristics. They MUST instead use mechanisms that let them
operate safely under very different path conditions. Typically, this
requires conservatively probing the current conditions of the
Internet path they communicate over to establish a transmission
behavior that it can sustain and that is reasonably fair to other
traffic sharing the path.

These mechanisms are difficult to implement correctly. For most
applications, the use of one of the existing IETF transport protocols
is the simplest method of acquiring the required mechanisms. Doing
so also avoids issues that protocols using a new IP protocol number
face when being deployed over the Internet, where middleboxes that
only support TCP and UDP are sometimes present. Consequently, the
RECOMMENDED alternative to the UDP usage described in the remainder
of this section is the use of an IETF transport protocol such as TCP

[RFC793], Stream Control Transmission Protocol (SCTP) [RFC4960], and
SCTP Partial Reliability Extension (SCTP-PR) [RFC3758], or Datagram
Congestion Control Protocol (DCCP) [RFC4340] with its different
congestion control types [RFC4341][RFC4342][RFC5622], or transport
protocols specified by the IETF in the future. (UDP-encapsulated
SCTP [RFC6951] and DCCP [RFC6773] can offer support for traversing
firewalls and other middleboxes where the native protocols are not
supported.)

If used correctly, these more fully featured transport protocols are
not as "heavyweight" as often claimed. For example, the TCP
algorithms have been continuously improved over decades, and they
have reached a level of efficiency and correctness that custom
application-layer mechanisms will struggle to easily duplicate. In
addition, many TCP implementations allow connections to be tuned by
an application to its purposes. For example, TCP's "Nagle" algorithm
[RFC1122] can be disabled, improving communication latency at the
expense of more frequent -- but still congestion controlled -- packet
transmissions. Another example is the TCP SYN cookie mechanism
[RFC4987], which is available on many platforms. TCP with SYN
cookies does not require a server to maintain per-connection state
until the connection is established. TCP also requires the end that
closes a connection to maintain the TIME-WAIT state that prevents
delayed segments from one connection instance from interfering with a
later one. Applications that are aware of and designed for this
behavior can shift maintenance of the TIME-WAIT state to conserve
resources by controlling which end closes a TCP connection [FABER].
Finally, TCP's built-in capacity-probing and awareness of the maximum
transmission unit supported by the path (PMTU) results in efficient
data transmission that quickly compensates for the initial connection
setup delay, in the case of transfers that exchange more than a few
segments.

3.1. Congestion Control Guidelines

If an application or protocol chooses not to use a congestion-
controlled transport protocol, it SHOULD control the rate at which it
sends UDP datagrams to a destination host, in order to fulfill the
requirements of [RFC2914]. It is important to stress that an
application SHOULD perform congestion control over all UDP traffic it
sends to a destination, independently from how it generates this
traffic. For example, an application that forks multiple worker
processes or otherwise uses multiple sockets to generate UDP
datagrams SHOULD perform congestion control over the aggregate
traffic.

Several approaches to perform congestion control are discussed in the
remainder of this section. This section describes generic topics
with an intended emphasis on unicast and anycast [RFC1546] usage.
Not all approaches discussed below are appropriate for all UDP-
transmitting applications. Section 3.1.2 discusses congestion
control options for applications that perform bulk transfers over
UDP. Such applications can employ schemes that sample the path over
several subsequent round-trips during which data is exchanged to
determine a sending rate that the path at its current load can
support. Other applications only exchange a few UDP datagrams with a
destination. Section 3.1.3 discusses congestion control options for
such "low data-volume" applications. Because they typically do not
transmit enough data to iteratively sample the path to determine a
safe sending rate, they need to employ different kinds of congestion
control mechanisms. Section 3.1.11 discusses congestion control
considerations when UDP is used as a tunneling protocol. Section 4
provides additional recommendations for broadcast and multicast
usage.

It is important to note that congestion control should not be viewed
as an add-on to a finished application. Many of the mechanisms
discussed in the guidelines below require application support to
operate correctly. Application designers need to consider congestion
control throughout the design of their application, similar to how
they consider security aspects throughout the design process.

In the past, the IETF has also investigated integrated congestion
control mechanisms that act on the traffic aggregate between two
hosts, i.e., a framework such as the Congestion Manager [RFC3124],
where active sessions may share current congestion information in a
way that is independent of the transport protocol. Such mechanisms
have currently failed to see deployment, but would otherwise simplify
the design of congestion control mechanisms for UDP sessions, so that
they fulfill the requirements in [RFC2914].

3.1.1. Protocol Timer Guidelines

Understanding the latency between communicating endpoints is usually
a crucial part of effective congestion control implementations for
protocols and applications. Latency estimation can be used in a
number of protocol functions, such as calculating a congestion-
controlled transmission rate, triggering retransmission, and
detecting packet loss. Additional protocol functions, for example,
determining an interval for probing a path, determining an interval
between keep-alive messages, determining an interval for measuring
the quality of experience, or determining if a remote endpoint has

responded to a request to perform an action, typically operate over
longer timescales than congestion control and therefore are not
covered in this section.

The general recommendation in this document is that applications
SHOULD leverage existing congestion control techniques and the
latency estimators specified therein (see next subsection). The
following guidelines are provided for applications that need to
design their own latency estimation mechanisms.

The guidelines are framed in terms of "latency" and not "round-trip
time" because some situations require characterizing only the
network-based latency (e.g., TCP-Friendly Rate Control (TFRC)
[RFC5348]), while other cases necessitate inclusion of the time
required by the remote endpoint to provide feedback (e.g., developing
an understanding of when to retransmit a message).

The latency between endpoints is generally a dynamic property.
Therefore, estimates SHOULD represent some sort of averaging of
multiple recent measurement samples to account for variance.
Leveraging an Exponentially Weighted Moving Average (EWMA) has proven
useful for this purpose (e.g., in TCP [RFC6298] and TFRC [RFC5348]).

Independent latency estimates SHOULD be maintained for each
destination with which an endpoint communicates.

Latency samples MUST NOT be derived from ambiguous transactions. The
canonical example is in a protocol that retransmits data, but
subsequently cannot determine which copy is being acknowledged. This
ambiguity makes correct computation of the latency problematic. See
the discussion of Karn's algorithm in [RFC6298]. This requirement
ensures a sender establishes a sound estimate of the latency without
relying on misleading measurements.

When a latency estimate is used to arm a timer that provides loss
detection -- with or without retransmission -- expiry of the timer
MUST be interpreted as an indication of congestion in the network,
causing the sending rate to be adapted to a safe conservative rate
(e.g., TCP collapses the congestion window to one segment [RFC5681]).

Some applications require an initial latency estimate before the
latency between endpoints can be empirically sampled. For instance,
when arming a retransmission timer, an initial value is needed to
protect the messages sent before the endpoints sample the latency.
This initial latency estimate SHOULD generally be as conservative
(large) as possible for the given application. For instance, in the
absence of any knowledge about the latency of a path, TCP requires
the initial Retransmission Timeout (RTO) to be set to no less than 1

second [RFC6298]. UDP applications SHOULD similarly use an initial
latency estimate of 1 second. Values shorter than 1 second can be
problematic (see the data analysis in the appendix of [RFC6298]).

3.1.2. Bulk-Transfer Applications

Applications that perform bulk transmission of data to a peer over
UDP, i.e., applications that exchange more than a few UDP datagrams
per RTT, SHOULD implement TFRC [RFC5348], window-based TCP-like
congestion control, or otherwise ensure that the application complies
with the congestion control principles.

TFRC has been designed to provide both congestion control and
fairness in a way that is compatible with the IETF's other transport
protocols. If an application implements TFRC, it need not follow the
remaining guidelines in Section 3.1.2, because TFRC already addresses
them, but it SHOULD still follow the remaining guidelines in the
subsequent subsections of Section 3.

Bulk-transfer applications that choose not to implement TFRC or TCP-
like windowing SHOULD implement a congestion control scheme that
results in bandwidth (capacity) use that competes fairly with TCP
within an order of magnitude.

Section 2 of [RFC3551] suggests that applications SHOULD monitor the
packet-loss rate to ensure that it is within acceptable parameters.
Packet loss is considered acceptable if a TCP flow across the same
network path under the same network conditions would achieve an
average throughput, measured on a reasonable timescale, that is not
less than that of the UDP flow. The comparison to TCP cannot be
specified exactly, but is intended as an "order-of-magnitude"
comparison in timescale and throughput. The recommendations for
managing timers specified in Section 3.1.1 also apply.

Finally, some bulk-transfer applications may choose not to implement
any congestion control mechanism and instead rely on transmitting
across reserved path capacity (see Section 3.1.9). This might be an
acceptable choice for a subset of restricted networking environments,
but is by no means a safe practice for operation over the wider
Internet. When the UDP traffic of such applications leaks out into
unprovisioned Internet paths, it can significantly degrade the
performance of other traffic sharing the path and even result in
congestion collapse. Applications that support an uncontrolled or
unadaptive transmission behavior SHOULD NOT do so by default and
SHOULD instead require users to explicitly enable this mode of
operation, and they SHOULD verify that sufficient path capacity has
been reserved for them.

3.1.3. Low Data-Volume Applications

   When applications that at any time exchange only a few UDP datagrams
   with a destination implement TFRC or one of the other congestion
   control schemes in Section 3.1.2, the network sees little benefit,
   because those mechanisms perform congestion control in a way that is
   only effective for longer transmissions.

   Applications that at any time exchange only a few UDP datagrams with
   a destination SHOULD still control their transmission behavior by not
   sending on average more than one UDP datagram per RTT to a
   destination. Similar to the recommendation in [RFC1536], an
   application SHOULD maintain an estimate of the RTT for any
   destination with which it communicates using the methods specified in
   Section 3.1.1.

   Some applications cannot maintain a reliable RTT estimate for a
   destination. These applications do not need to or are unable to use
   protocol timers to measure the RTT (Section 3.1.1). Two cases can be
   identified:

   1. The first case is that of applications that exchange too few UDP
       datagrams with a peer to establish a statistically accurate RTT
       estimate but that can monitor the reliability of transmission
       (Section 3.3). Such applications MAY use a predetermined
       transmission interval that is exponentially backed off when
       packets are deemed lost. TCP specifies an initial value of 1
       second [RFC6298], which is also RECOMMENDED as an initial value
       for UDP applications. Some low data-volume applications, e.g.,
       SIP [RFC3261] and General Internet Signaling Transport (GIST)
       [RFC5971] use an interval of 500 ms, and shorter values are
       likely problematic in many cases. As in the previous case, note
       that the initial timeout is not the maximum possible timeout, see
       Section 3.1.1.

   2. A second case of applications cannot maintain an RTT estimate for
       a destination, because the destination does not send return
       traffic. Such applications SHOULD NOT send more than one UDP
       datagram every 3 seconds and SHOULD use an even less aggressive
       rate when possible. Shorter values are likely problematic in
       many cases. Note that the sending rate in this case must be more
       conservative than in the previous cases, because the lack of
       return traffic prevents the detection of packet loss, i.e.,
       congestion, and the application therefore cannot perform
       exponential back off to reduce load.

3.1.4. Applications Supporting Bidirectional Communications

   Applications that communicate bidirectionally SHOULD employ
   congestion control for both directions of the communication. For
   example, for a client-server, request-response-style application,
   clients SHOULD congestion-control their request transmission to a
   server, and the server SHOULD congestion-control its responses to the
   clients. Congestion in the forward and reverse directions is
   uncorrelated, and an application SHOULD either independently detect
   and respond to congestion along both directions or limit new and
   retransmitted requests based on acknowledged responses across the
   entire round-trip path.

3.1.5. Implications of RTT and Loss Measurements on Congestion Control

   Transports such as TCP, SCTP, and DCCP provide timely detection of
   congestion that results in an immediate reduction of their maximum
   sending rate when congestion is experienced. This reaction is
   typically completed 1-2 RTTs after loss/congestion is encountered.
   Applications using UDP SHOULD implement a congestion control scheme
   that provides a prompt reaction to signals indicating congestion
   (e.g., by reducing the rate within the next RTT following a
   congestion signal).

   The operation of a UDP congestion control algorithm can be very
   different from the way TCP operates. This includes congestion
   controls that respond on timescales that fit applications that cannot
   usefully work within the "change rate every RTT" model of TCP.
   Applications that experience a low or varying RTT are particularly
   vulnerable to sampling errors (e.g., due to measurement noise or
   timer accuracy). This suggests the need to average loss/congestion
   and RTT measurements over a longer interval; however, this also can
   contribute additional delay in detecting congestion. Some
   applications may not react by reducing their sending rate immediately
   for various reasons, including the following: RTT and loss
   measurements are only made periodically (e.g., using RTCP),
   additional time is required to filter information, or the application
   is only able to change its sending rate at predetermined interval
   (e.g., some video codecs).

   When designing a congestion control algorithm, the designer therefore
   needs to consider the total time taken to reduce the load following a
   lack of feedback or a congestion event. An application where the
   most recent RTT measurement is smaller than the actual RTT or the
   measured loss rate is smaller than the current rate, can result in
   over estimating the available capacity. Such over-estimation can

result in a sending rate that creates congestion to the application
or other flows sharing the path capacity, and can contribute to
congestion collapse -- both of these need to be avoided.

A congestion control designed for UDP SHOULD respond as quickly as
possible when it experiences congestion, and it SHOULD take into
account both the loss rate and the response time when choosing a new
rate. The implemented congestion control scheme SHOULD result in
bandwidth (capacity) use that is comparable to that of TCP within an
order of magnitude, so that it does not starve other flows sharing a
common bottleneck.

3.1.6. Burst Mitigation and Pacing

UDP applications SHOULD provide mechanisms to regulate the bursts of
transmission that the application may send to the network. Many TCP
and SCTP implementations provide mechanisms that prevent a sender
from generating long bursts at line-rate, since these are known to
induce early loss to applications sharing a common network
bottleneck. The use of pacing with TCP [ALLMAN] has also been shown
to improve the coexistence of TCP flows with other flows. The need
to avoid excessive transmission bursts is also noted in
specifications for applications (e.g., [RFC7143]).

Even low data-volume UDP flows may benefit from packet pacing, e.g.,
an application that sends three copies of a packet to improve
robustness to loss is RECOMMENDED to pace out those three packets
over several RTTs, to reduce the probability that all three packets
will be lost due to the same congestion event (or other event, such
as burst corruption).

3.1.7. Explicit Congestion Notification

Internet applications can use Explicit Congestion Notification (ECN)
[RFC3168] to gain benefits for the services they support [RFC8087].

Internet transports, such as TCP, provide a set of mechanisms that
are needed to utilize ECN. ECN operates by setting an ECN-capable
codepoint (ECT(0) or ECT(1)) in the IP header of packets that are
sent. This indicates to ECN-capable network devices (routers and
other devices) that they may mark (set the congestion experienced,
Congestion Experience (CE) codepoint) rather than drop the IP packet
as a signal of incipient congestion.

UDP applications can also benefit from enabling ECN, providing that
the API supports ECN and that they implement the required protocol
mechanisms to support ECN.

   The set of mechanisms required for an application to use ECN over UDP
   are:

   o A sender MUST provide a method to determine (e.g., negotiate) that
      the corresponding application is able to provide ECN feedback
      using a compatible ECN method.

   o A receiver that enables the use of ECN for a UDP port MUST check
      the ECN field at the receiver for each UDP datagram that it
      receives on this port.

   o The receiving application needs to provide feedback of congestion
      information to the sending application. This MUST report the
      presence of datagrams received with a CE-mark by providing a
      mechanism to feed this congestion information back to the sending
      application. The feedback MAY also report the presence of ECT(1)
      and ECT(0)/Not-ECT packets [RFC7560]. ([RFC3168] and [RFC7560]
      specify methods for TCP.)

   o An application sending ECN-capable datagrams MUST provide an
      appropriate congestion reaction when it receives feedback
      indicating that congestion has been experienced. This ought to
      result in reduction of the sending rate by the UDP congestion
      control method (see Section 3.1) that is not less than the
      reaction of TCP under equivalent conditions.

   o A sender SHOULD detect network paths that do not support the ECN
      field correctly. When detected, they need to either
      conservatively react to congestion or even fall back to not using
      ECN [RFC8087]. This method needs to be robust to changes within
      the network path that may occur over the lifetime of a session.

   o A sender is encouraged to provide a mechanism to detect and react
      appropriately to misbehaving receivers that fail to report
      CE-marked packets [RFC8087].

   [RFC6679] provides guidance and an example of this support, by
   describing a method to allow ECN to be used for UDP-based
   applications using the Real-Time Protocol (RTP). Applications that
   cannot provide this set of mechanisms, but wish to gain the benefits
   of using ECN, are encouraged to use a transport protocol that already
   supports ECN (such as TCP).

3.1.8. Differentiated Services Model

   An application using UDP can use the differentiated services
   (DiffServ) Quality of Service (QoS) framework. To enable
   differentiated services processing, a UDP sender sets the

Differentiated Services Code Point (DSCP) field [RFC2475] in packets
sent to the network. Normally, a UDP source/destination port pair
will set a single DSCP value for all packets belonging to a flow, but
multiple DSCPs can be used as described later in this section. A
DSCP may be chosen from a small set of fixed values (the class
selector code points), or from a set of recommended values defined in
the Per Hop Behavior (PHB) specifications, or from values that have
purely local meanings to a specific network that supports DiffServ.
In general, packets may be forwarded across multiple networks between
source and destination.

In setting a non-default DSCP value, an application must be aware
that DSCP markings may be changed or removed between the traffic
source and destination. This has implications on the design of
applications that use DSCPs. Specifically, applications SHOULD be
designed not to rely on implementation of a specific network
treatment; they need instead to implement congestion control methods
to determine if their current sending rate is inducing congestion in
the network.

[RFC7657] describes the implications of using DSCPs and provides
recommendations on using multiple DSCPs within a single network five-
tuple (source and destination addresses, source and destination
ports, and the transport protocol used, in this case, UDP or
UDP-Lite), and particularly the expected impact on transport protocol
interactions, with congestion control or reliability functionality
(e.g., retransmission, reordering). Use of multiple DSCPs can result
in reordering by increasing the set of network forwarding resources
used by a sender. It can also increase exposure to resource
depletion or failure.

3.1.9. QoS, Pre-Provisioned, or Reserved Capacity

   The IETF usually specifies protocols for use within the Best Effort
   General Internet. Sometimes it is relevant to specify protocols with
   a different applicability. An application using UDP can use the
   integrated services QoS framework. This framework is usually made
   available within controlled environments (e.g., within a single
   administrative domain or bilaterally agreed connection between
   domains). Applications intended for the Internet SHOULD NOT assume
   that QoS mechanisms are supported by the networks they use, and
   therefore need to provide congestion control, error recovery, etc.,
   in case the actual network path does not provide provisioned service.

   Some UDP applications are only expected to be deployed over network
   paths that use pre-provisioned capacity or capacity reserved using
   dynamic provisioning, e.g., through the Resource Reservation Protocol
   (RSVP). Multicast applications are also used with pre-provisioned

capacity (e.g., IPTV deployments within access networks). These
applications MAY choose not to implement any congestion control
mechanism and instead rely on transmitting only on paths where the
capacity is provisioned and reserved for this use. This might be an
acceptable choice for a subset of restricted networking environments,
but is by no means a safe practice for operation over the wider
Internet. Applications that choose this option SHOULD carefully and
in detail describe the provisioning and management procedures that
result in the desired containment.

Applications that support an uncontrolled or unadaptive transmission
behavior SHOULD NOT do so by default and SHOULD instead require users
to explicitly enable this mode of operation.

Applications designed for use within a controlled environment (see
Section 3.6) may be able to exploit network management functions to
detect whether they are causing congestion, and react accordingly.
If the traffic of such applications leaks out into unprovisioned
Internet paths, it can significantly degrade the performance of other
traffic sharing the path and even result in congestion collapse.
Protocols designed for such networks SHOULD provide mechanisms at the
network edge to prevent leakage of traffic into unprovisioned
Internet paths (e.g., [RFC7510]). To protect other applications
sharing the same path, applications SHOULD also deploy an appropriate
circuit breaker, as described in Section 3.1.10.

An IETF specification targeting a controlled environment is expected
to provide an applicability statement that restricts the application
to the controlled environment (see Section 3.6).

3.1.10. Circuit Breaker Mechanisms

A transport circuit breaker is an automatic mechanism that is used to
estimate the congestion caused by a flow, and to terminate (or
significantly reduce the rate of) the flow when excessive congestion
is detected [RFC8084]. This is a safety measure to prevent
congestion collapse (starvation of resources available to other
flows), essential for an Internet that is heterogeneous and for
traffic that is hard to predict in advance.

A circuit breaker is intended as a protection mechanism of last
resort. Under normal circumstances, a circuit breaker should not be
triggered; it is designed to protect things when there is severe
overload. The goal is usually to limit the maximum transmission rate
that reflects the available capacity of a network path. Circuit
breakers can operate on individual UDP flows or traffic aggregates,
e.g., traffic sent using a network tunnel.

[RFC8084] provides guidance and examples on the use of circuit
breakers. The use of a circuit breaker in RTP is specified in
[RFC8083].

Applications used in the general Internet SHOULD implement a
transport circuit breaker if they do not implement congestion control
or operate a low data-volume service (see Section 3.6). All
applications MAY implement a transport circuit breaker [RFC8084] and
are encouraged to consider implementing at least a slow-acting
transport circuit breaker to provide a protection of last resort for
their network traffic.

3.1.11. UDP Tunnels

One increasingly popular use of UDP is as a tunneling protocol
[INT-TUNNELS], where a tunnel endpoint encapsulates the packets of
another protocol inside UDP datagrams and transmits them to another
tunnel endpoint, which decapsulates the UDP datagrams and forwards
the original packets contained in the payload. One example of such a
protocol is Teredo [RFC4380]. Tunnels establish virtual links that
appear to directly connect locations that are distant in the physical
Internet topology and can be used to create virtual (private)
networks. Using UDP as a tunneling protocol is attractive when the
payload protocol is not supported by middleboxes that may exist along
the path, because many middleboxes support transmission using UDP.

Well-implemented tunnels are generally invisible to the endpoints
that happen to transmit over a path that includes tunneled links. On
the other hand, to the routers along the path of a UDP tunnel, i.e.,
the routers between the two tunnel endpoints, the traffic that a UDP
tunnel generates is a regular UDP flow, and the encapsulator and
decapsulator appear as regular UDP-sending and UDP-receiving
applications. Because other flows can share the path with one or
more UDP tunnels, congestion control needs to be considered.

Two factors determine whether a UDP tunnel needs to employ specific
congestion control mechanisms: first, whether the payload traffic is
IP-based; and second, whether the tunneling scheme generates UDP
traffic at a volume that corresponds to the volume of payload traffic
carried within the tunnel.

IP-based unicast traffic is generally assumed to be congestion
controlled, i.e., it is assumed that the transport protocols
generating IP-based unicast traffic at the sender already employ
mechanisms that are sufficient to address congestion on the path.
Consequently, a tunnel carrying IP-based unicast traffic should

already interact appropriately with other traffic sharing the path,
and specific congestion control mechanisms for the tunnel are not
necessary.

However, if the IP traffic in the tunnel is known not to be
congestion controlled, additional measures are RECOMMENDED to limit
the impact of the tunneled traffic on other traffic sharing the path.
For the specific case of a tunnel that carries IP multicast traffic,
see Section 4.1.

The following guidelines define these possible cases in more detail:

1. A tunnel generates UDP traffic at a volume that corresponds to
    the volume of payload traffic, and the payload traffic is IP
    based and congestion controlled.

    This is arguably the most common case for Internet tunnels. In
    this case, the UDP tunnel SHOULD NOT employ its own congestion
    control mechanism, because congestion losses of tunneled traffic
    will already trigger an appropriate congestion response at the
    original senders of the tunneled traffic. A circuit breaker
    mechanism may provide benefit by controlling the envelope of the
    aggregated traffic.

    Note that this guideline is built on the assumption that most
    IP-based communication is congestion controlled. If a UDP tunnel
    is used for IP-based traffic that is known to not be congestion
    controlled, the next set of guidelines applies.

2. A tunnel generates UDP traffic at a volume that corresponds to
    the volume of payload traffic, and the payload traffic is not
    known to be IP based, or is known to be IP based but not
    congestion controlled.

    This can be the case, for example, when some link-layer protocols
    are encapsulated within UDP (but not all link-layer protocols;
    some are congestion controlled). Because it is not known that
    congestion losses of tunneled non-IP traffic will trigger an
    appropriate congestion response at the senders, the UDP tunnel
    SHOULD employ an appropriate congestion control mechanism or
    circuit breaker mechanism designed for the traffic it carries.
    Because tunnels are usually bulk-transfer applications as far as
    the intermediate routers are concerned, the guidelines in
    Section 3.1.2 apply.

3. A tunnel generates UDP traffic at a volume that does not
    correspond to the volume of payload traffic, independent of
    whether the payload traffic is IP based or congestion controlled.

Examples of this class include UDP tunnels that send at a
constant rate, increase their transmission rates under loss, for
example, due to increasing redundancy when Forward Error
Correction is used, or are otherwise unconstrained in their
transmission behavior. These specialized uses of UDP for
tunneling go beyond the scope of the general guidelines given in
this document. The implementer of such specialized tunnels
SHOULD carefully consider congestion control in the design of
their tunneling mechanism and SHOULD consider use of a circuit
breaker mechanism.

The type of encapsulated payload might be identified by a UDP port;
identified by an Ethernet Type or IP protocol number. A tunnel
SHOULD provide mechanisms to restrict the types of flows that may be
carried by the tunnel. For instance, a UDP tunnel designed to carry
IP needs to filter out non-IP traffic at the ingress. This is
particularly important when a generic tunnel encapsulation is used
(e.g., one that encapsulates using an EtherType value). Such tunnels
SHOULD provide a mechanism to restrict the types of traffic that are
allowed to be encapsulated for a given deployment (see
[INT-TUNNELS]).

Designing a tunneling mechanism requires significantly more expertise
than needed for many other UDP applications, because tunnels are
usually intended to be transparent to the endpoints transmitting over
them, so they need to correctly emulate the behavior of an IP link
[INT-TUNNELS], for example:

o Requirements for tunnels that carry or encapsulate using ECN code
   points [RFC6040].

o Usage of the IP DSCP field by tunnel endpoints [RFC2983].

o Encapsulation considerations in the design of tunnels [ENCAP].

o Usage of ICMP messages [INT-TUNNELS].

o Handling of fragmentation and packet size for tunnels
   [INT-TUNNELS].

o Source port usage for tunnels designed to support equal cost
   multipath (ECMP) routing (see Section 5.1.1).

o Guidance on the need to protect headers [INT-TUNNELS] and the use
   of checksums for IPv6 tunnels (see Section 3.4.1).

o Support for operations and maintenance [INT-TUNNELS].

At the same time, the tunneled traffic is application traffic like
any other from the perspective of the networks the tunnel transmits
over. This document only touches upon the congestion control
considerations for implementing UDP tunnels; a discussion of other
required tunneling behavior is out of scope.

3.2. Message Size Guidelines

IP fragmentation lowers the efficiency and reliability of Internet
communication. The loss of a single fragment results in the loss of
an entire fragmented packet, because even if all other fragments are
received correctly, the original packet cannot be reassembled and
delivered. This fundamental issue with fragmentation exists for both
IPv4 and IPv6.

In addition, some network address translators (NATs) and firewalls
drop IP fragments. The network address translation performed by a
NAT only operates on complete IP packets, and some firewall policies
also require inspection of complete IP packets. Even with these
being the case, some NATs and firewalls simply do not implement the
necessary reassembly functionality; instead, they choose to drop all
fragments. Finally, [RFC4963] documents other issues specific to
IPv4 fragmentation.

Due to these issues, an application SHOULD NOT send UDP datagrams
that result in IP packets that exceed the Maximum Transmission Unit
(MTU) along the path to the destination. Consequently, an
application SHOULD either use the path MTU information provided by
the IP layer or implement Path MTU Discovery (PMTUD) itself [RFC1191]
[RFC1981] [RFC4821] to determine whether the path to a destination
will support its desired message size without fragmentation.

However, the ICMP messages that enable path MTU discovery are being
increasingly filtered by middleboxes (including Firewalls) [RFC4890].
When the path includes a tunnel, some devices acting as a tunnel
ingress discard ICMP messages that originate from network devices
over which the tunnel passes, preventing these from reaching the UDP
endpoint.

Packetization Layer Path MTU Discovery (PLPMTUD) [RFC4821] does not
rely upon network support for ICMP messages and is therefore
considered more robust than standard PMTUD. It is not susceptible to
"black holing" of ICMP messages. To operate, PLPMTUD requires
changes to the way the transport is used: both to transmit probe
packets and to account for the loss or success of these probes. This
not only updates the PMTU algorithm, it also impacts loss recovery,
congestion control, etc. These updated mechanisms can be implemented

within a connection-oriented transport (e.g., TCP, SCTP, DCCP), but
they are not a part of UDP; this type of feedback is not typically
present for unidirectional applications.

Therefore, PLPMTUD places additional design requirements on a UDP
application that wishes to use this method. This is especially true
for UDP tunnels, because the overhead of sending probe packets needs
to be accounted for and may require adding a congestion control
mechanism to the tunnel (see Section 3.1.11) as well as complicating
the data path at a tunnel decapsulator.

Applications that do not follow the recommendation to do PMTU/PLPMTUD
discovery SHOULD still avoid sending UDP datagrams that would result
in IP packets that exceed the path MTU. Because the actual path MTU
is unknown, such applications SHOULD fall back to sending messages
that are shorter than the default effective MTU for sending (EMTU_S
in [RFC1122]). For IPv4, EMTU_S is the smaller of 576 bytes and the
first-hop MTU [RFC1122]. For IPv6, EMTU_S is 1280 bytes [RFC2460].
The effective PMTU for a directly connected destination (with no
routers on the path) is the configured interface MTU, which could be
less than the maximum link payload size. Transmission of minimum-
sized UDP datagrams is inefficient over paths that support a larger
PMTU, which is a second reason to implement PMTU discovery.

To determine an appropriate UDP payload size, applications MUST
subtract the size of the IP header (which includes any IPv4 optional
headers or IPv6 extension headers) as well as the length of the UDP
header (8 bytes) from the PMTU size. This size, known as the Maximum
Segment Size (MSS), can be obtained from the TCP/IP stack [RFC1122].

Applications that do not send messages that exceed the effective PMTU
of IPv4 or IPv6 need not implement any of the above mechanisms. Note
that the presence of tunnels can cause an additional reduction of the
effective PMTU [INT-TUNNELS], so implementing PMTU discovery may be
beneficial.

Applications that fragment an application-layer message into multiple
UDP datagrams SHOULD perform this fragmentation so that each datagram
can be received independently, and be independently retransmitted in
the case where an application implements its own reliability
mechanisms.

3.3. Reliability Guidelines

   Application designers are generally aware that UDP does not provide
   any reliability, e.g., it does not retransmit any lost packets.
   Often, this is a main reason to consider UDP as a transport protocol.
   Applications that do require reliable message delivery MUST implement
   an appropriate mechanism themselves.

   UDP also does not protect against datagram duplication, i.e., an
   application may receive multiple copies of the same UDP datagram,
   with some duplicates arriving potentially much later than the first.
   Application designers SHOULD handle such datagram duplication
   gracefully, and they may consequently need to implement mechanisms to
   detect duplicates. Even if UDP datagram reception triggers only
   idempotent operations, applications may want to suppress duplicate
   datagrams to reduce load.

   Applications that require ordered delivery MUST reestablish datagram
   ordering themselves. The Internet can significantly delay some
   packets with respect to others, e.g., due to routing transients,
   intermittent connectivity, or mobility. This can cause reordering,
   where UDP datagrams arrive at the receiver in an order different from
   the transmission order.

   Applications that use multiple transport ports need to be robust to
   reordering between sessions. Load-balancing techniques within the
   network, such as Equal Cost Multipath (ECMP) forwarding can also
   result in a lack of ordering between different transport sessions,
   even between the same two network endpoints.

   It is important to note that the time by which packets are reordered
   or after which duplicates can still arrive can be very large. Even
   more importantly, there is no well-defined upper boundary here.
   [RFC793] defines the maximum delay a TCP segment should experience --
   the Maximum Segment Lifetime (MSL) -- as 2 minutes. No other RFC
   defines an MSL for other transport protocols or IP itself. The MSL
   value defined for TCP is conservative enough that it SHOULD be used
   by other protocols, including UDP. Therefore, applications SHOULD be
   robust to the reception of delayed or duplicate packets that are
   received within this 2-minute interval.

   Retransmission of lost packets or messages is a common reliability
   mechanism. Such retransmissions can increase network load in
   response to congestion, worsening that congestion. Any application
   that uses retransmission is responsible for congestion control of its
   retransmissions (as well as the application's original traffic);
   hence, it is subject to the Congestion Control guidelines in

   Section 3.1. Guidance on the appropriate measurement of RTT in
   Section 3.1.1 also applies for timers used for retransmission packet-
   loss detection.

   Instead of implementing these relatively complex reliability
   mechanisms by itself, an application that requires reliable and
   ordered message delivery SHOULD whenever possible choose an IETF
   standard transport protocol that provides these features.

3.4. Checksum Guidelines

   The UDP header includes an optional, 16-bit one's complement checksum
   that provides an integrity check. These checks are not strong from a
   coding or cryptographic perspective and are not designed to detect
   physical-layer errors or malicious modification of the datagram
   [RFC3819]. Application developers SHOULD implement additional checks
   where data integrity is important, e.g., through a Cyclic Redundancy
   Check (CRC) or keyed or non-keyed cryptographic hash included with
   the data to verify the integrity of an entire object/file sent over
   the UDP service.

   The UDP checksum provides a statistical guarantee that the payload
   was not corrupted in transit. It also allows the receiver to verify
   that it was the intended destination of the packet, because it covers
   the IP addresses, port numbers, and protocol number, and it verifies
   that the packet is not truncated or padded, because it covers the
   size field. Therefore, it protects an application against receiving
   corrupted payload data in place of, or in addition to, the data that
   was sent. More description of the set of checks performed using the
   checksum field is provided in Section 3.1 of [RFC6396].

   Applications SHOULD enable UDP checksums [RFC1122]. For IPv4,
   [RFC768] permits an option to disable their use, by setting a zero
   checksum value. An application is permitted to optionally discard
   UDP datagrams with a zero checksum [RFC1122].

   When UDP is used over IPv6, the UDP checksum is relied upon to
   protect both the IPv6 and UDP headers from corruption (because IPv6
   lacks a checksum) and MUST be used as specified in [RFC2460]. Under
   specific conditions, a UDP application is allowed to use a zero UDP
   zero-checksum mode with a tunnel protocol (see Section 3.4.1).

   Applications that choose to disable UDP checksums MUST NOT make
   assumptions regarding the correctness of received data and MUST
   behave correctly when a UDP datagram is received that was originally
   sent to a different destination or is otherwise corrupted.

3.4.1. IPv6 Zero UDP Checksum

   [RFC6935] defines a method that enables use of a zero UDP zero-
   checksum mode with a tunnel protocol, providing that the method
   satisfies the requirements in [RFC6936]. The application MUST
   implement mechanisms and/or usage restrictions when enabling this
   mode. This includes defining the scope for usage and measures to
   prevent leakage of traffic to other UDP applications (see Appendix A
   and Section 3.6). These additional design requirements for using a
   zero IPv6 UDP checksum are not present for IPv4, since the IPv4
   header validates information that is not protected in an IPv6 packet.
   Key requirements are:

   o Use of the UDP checksum with IPv6 MUST be the default
      configuration for all implementations [RFC6935]. The receiving
      endpoint MUST only allow the use of UDP zero-checksum mode for
      IPv6 on a UDP destination port that is specifically enabled.

   o An application that supports a checksum different than that in
      [RFC2460] MUST comply with all implementation requirements
      specified in Section 4 of [RFC6936] and with the usage
      requirements specified in Section 5 of [RFC6936].

   o A UDP application MUST check that the source and destination IPv6
      addresses are valid for any packets with a UDP zero-checksum and
      MUST discard any packet for which this check fails. To protect
      from misdelivery, new encapsulation designs SHOULD include an
      integrity check at the transport layer that includes at least the
      IPv6 header, the UDP header and the shim header for the
      encapsulation, if any [RFC6936].

   o One way to help satisfy the requirements of [RFC6936] may be to
      limit the usage of such tunnels, e.g., to constrain traffic to an
      operator network, as discussed in Section 3.6. The encapsulation
      defined for MPLS in UDP [RFC7510] chooses this approach.

   As in IPv4, IPv6 applications that choose to disable UDP checksums
   MUST NOT make assumptions regarding the correctness of received data
   and MUST behave correctly when a UDP datagram is received that was
   originally sent to a different destination or is otherwise corrupted.

   IPv6 datagrams with a zero UDP checksum will not be passed by any
   middlebox that validates the checksum based on [RFC2460] or that
   updates the UDP checksum field, such as NATs or firewalls. Changing
   this behavior would require such middleboxes to be updated to
   correctly handle datagrams with zero UDP checksums. To ensure end-
   to-end robustness, applications that may be deployed in the general
   Internet MUST provide a mechanism to safely fall back to using a

checksum when a path change occurs that redirects a zero UDP checksum
flow over a path that includes a middlebox that discards IPv6
datagrams with a zero UDP checksum.

3.4.2. UDP-Lite

A special class of applications can derive benefit from having
partially damaged payloads delivered, rather than discarded, when
using paths that include error-prone links. Such applications can
tolerate payload corruption and MAY choose to use the Lightweight
User Datagram Protocol (UDP-Lite) [RFC3828] variant of UDP instead of
basic UDP. Applications that choose to use UDP-Lite instead of UDP
should still follow the congestion control and other guidelines
described for use with UDP in Section 3.

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. The interface of UDP-Lite differs
from that of UDP by the addition of a single (socket) option that
communicates the checksum coverage length: at the sender, this
specifies the intended checksum coverage, with the remaining
unprotected part of the payload called the "error-insensitive part".
By default, the UDP-Lite checksum coverage extends across the entire
datagram. If required, an application may dynamically modify this
length value, e.g., to offer greater protection to some messages.
UDP-Lite always verifies that a packet was delivered to the intended
destination, i.e., always verifies the header fields. Errors in the
insensitive part will not cause a UDP datagram to be discarded by the
destination. Therefore, applications using UDP-Lite MUST NOT make
assumptions regarding the correctness of the data received in the
insensitive part of the UDP-Lite payload.

A UDP-Lite sender SHOULD select the minimum checksum coverage to
include all sensitive payload information. For example, applications
that use the Real-Time Protocol (RTP) [RFC3550] will likely want to
protect the RTP header against corruption. Applications, where
appropriate, MUST also introduce their own appropriate validity
checks for protocol information carried in the insensitive part of
the UDP-Lite payload (e.g., internal CRCs).

A UDP-Lite receiver MUST set a minimum coverage threshold for
incoming packets that is not smaller than the smallest coverage used
by the sender [RFC3828]. The receiver SHOULD select a threshold that
is sufficiently large to block packets with an inappropriately short
coverage field. This may be a fixed value, or it may be negotiated
by an application. UDP-Lite does not provide mechanisms to negotiate
the checksum coverage between the sender and receiver. Therefore,
this needs to be performed by the application.

   Applications can still experience packet loss when using UDP-Lite.
   The enhancements offered by UDP-Lite rely upon a link being able to
   intercept the UDP-Lite header to correctly identify the partial
   coverage required. When tunnels and/or encryption are used, this can
   result in UDP-Lite datagrams being treated the same as UDP datagrams,
   i.e., result in packet loss. Use of IP fragmentation can also
   prevent special treatment for UDP-Lite datagrams, and this is another
   reason why applications SHOULD avoid IP fragmentation (Section 3.2).

   UDP-Lite is supported in some endpoint protocol stacks. Current
   support for middlebox traversal using UDP-Lite is poor, because UDP-
   Lite uses a different IPv4 protocol number or IPv6 "next header"
   value than that used for UDP; therefore, few middleboxes are
   currently able to interpret UDP-Lite and take appropriate actions
   when forwarding the packet. This makes UDP-Lite less suited for
   applications needing general Internet support, until such time as
   UDP-Lite has achieved better support in middleboxes.

3.5. Middlebox Traversal Guidelines

   NATs and firewalls are examples of intermediary devices
   ("middleboxes") that can exist along an end-to-end path. A middlebox
   typically performs a function that requires it to maintain per-flow
   state. For connection-oriented protocols, such as TCP, middleboxes
   snoop and parse the connection-management information, and create and
   destroy per-flow state accordingly. For a connectionless protocol
   such as UDP, this approach is not possible. Consequently,
   middleboxes can create per-flow state when they see a packet that --
   according to some local criteria -- indicates a new flow, and destroy
   the state after some time during which no packets belonging to the
   same flow have arrived.

   Depending on the specific function that the middlebox performs, this
   behavior can introduce a time-dependency that restricts the kinds of
   UDP traffic exchanges that will be successful across the middlebox.
   For example, NATs and firewalls typically define the partial path on
   one side of them to be interior to the domain they serve, whereas the
   partial path on their other side is defined to be exterior to that
   domain. Per-flow state is typically created when the first packet
   crosses from the interior to the exterior, and while the state is
   present, NATs and firewalls will forward return traffic. Return
   traffic that arrives after the per-flow state has timed out is
   dropped, as is other traffic that arrives from the exterior.

Many applications that use UDP for communication operate across
middleboxes without needing to employ additional mechanisms. One
example is the Domain Name System (DNS), which has a strict request-
response communication pattern that typically completes within
seconds.

Other applications may experience communication failures when
middleboxes destroy the per-flow state associated with an application
session during periods when the application does not exchange any UDP
traffic. Applications SHOULD be able to gracefully handle such
communication failures and implement mechanisms to re-establish
application-layer sessions and state.

For some applications, such as media transmissions, this
re-synchronization is highly undesirable, because it can cause user-
perceivable playback artifacts. Such specialized applications MAY
send periodic keep-alive messages to attempt to refresh middlebox
state (e.g., [RFC7675]). It is important to note that keep-alive
messages are not recommended for general use -- they are unnecessary
for many applications and can consume significant amounts of system
and network resources.

An application that needs to employ keep-alive messages to deliver
useful service over UDP in the presence of middleboxes SHOULD NOT
transmit them more frequently than once every 15 seconds and SHOULD
use longer intervals when possible. No common timeout has been
specified for per-flow UDP state for arbitrary middleboxes. NATs
require a state timeout of 2 minutes or longer [RFC4787]. However,
empirical evidence suggests that a significant fraction of currently
deployed middleboxes unfortunately use shorter timeouts. The timeout
of 15 seconds originates with the Interactive Connectivity
Establishment (ICE) protocol [RFC5245]. When an application is
deployed in a controlled environment, the deployer SHOULD investigate
whether the target environment allows applications to use longer
intervals, or whether it offers mechanisms to explicitly control
middlebox state timeout durations, for example, using the Port
Control Protocol (PCP) [RFC6887], Middlebox Communications (MIDCOM)
[RFC3303], Next Steps in Signaling (NSIS) [RFC5973], or Universal
Plug and Play (UPnP) [UPnP]. It is RECOMMENDED that applications
apply slight random variations ("jitter") to the timing of keep-alive
transmissions, to reduce the potential for persistent synchronization
between keep-alive transmissions from different hosts [RFC7675].

Sending keep-alive messages is not a substitute for implementing a
mechanism to recover from broken sessions. Like all UDP datagrams,
keep-alive messages can be delayed or dropped, causing middlebox
state to time out. In addition, the congestion control guidelines in
Section 3.1 cover all UDP transmissions by an application, including
the transmission of middlebox keep-alive messages. Congestion
control may thus lead to delays or temporary suspension of keep-alive
transmission.

Keep-alive messages are NOT RECOMMENDED for general use. They are
unnecessary for many applications and may consume significant
resources. For example, on battery-powered devices, if an
application needs to maintain connectivity for long periods with
little traffic, the frequency at which keep-alive messages are sent
can become the determining factor that governs power consumption,
depending on the underlying network technology.

Because many middleboxes are designed to require keep-alive messages
for TCP connections at a frequency that is much lower than that
needed for UDP, this difference alone can often be sufficient to
prefer TCP over UDP for these deployments. On the other hand, there
is anecdotal evidence that suggests that direct communication through
middleboxes, e.g., by using ICE [RFC5245], does succeed less often
with TCP than with UDP. The trade-offs between different transport
protocols -- especially when it comes to middlebox traversal --
deserve careful analysis.

UDP applications that could be deployed in the Internet need to be
designed understanding that there are many variants of middlebox
behavior, and although UDP is connectionless, middleboxes often
maintain state for each UDP flow. Using multiple UDP flows can
consume available state space and also can lead to changes in the way
the middlebox handles subsequent packets (either to protect its
internal resources, or to prevent perceived misuse). The probability
of path failure can increase when applications use multiple UDP flows
in parallel (see Section 5.1.2 for recommendations on usage of
multiple ports).

3.6. Limited Applicability and Controlled Environments

Two different types of applicability have been identified for the
specification of IETF applications that utilize UDP:

General Internet. By default, IETF specifications target deployment
   on the general Internet. Experience has shown that successful
   protocols developed in one specific context or for a particular
   application tend to become used in a wider range of contexts. For
   example, a protocol with an initial deployment within a local area

      network may subsequently be used over a virtual network that
      traverses the Internet, or in the Internet in general.
      Applications designed for general Internet use may experience a
      range of network device behaviors and, in particular, should
      consider whether applications need to operate over paths that may
      include middleboxes.

   Controlled Environment. A protocol/encapsulation/tunnel could be
      designed to be used only within a controlled environment. For
      example, an application designed for use by a network operator
      might only be deployed within the network of that single network
      operator or on networks of an adjacent set of cooperating network
      operators. The application traffic may then be managed to avoid
      congestion, rather than relying on built-in mechanisms, which are
      required when operating over the general Internet. Applications
      that target a limited applicability use case may be able to take
      advantage of specific hardware (e.g., carrier-grade equipment) or
      underlying protocol features of the subnetwork over which they are
      used.

   Specifications addressing a limited applicability use case or a
   controlled environment SHOULD identify how, in their restricted
   deployment, a level of safety is provided that is equivalent to that
   of a protocol designed for operation over the general Internet (e.g.,
   a design based on extensive experience with deployments of particular
   methods that provide features that cannot be expected in general
   Internet equipment and the robustness of the design of MPLS to
   corruption of headers both helped justify use of an alternate UDP
   integrity check [RFC7510]).

   An IETF specification targeting a controlled environment is expected
   to provide an applicability statement that restricts the application
   traffic to the controlled environment, and it would be expected to
   describe how methods can be provided to discourage or prevent escape
   of corrupted packets from the environment (for example, Section 5 of
   [RFC7510]).

4. Multicast UDP Usage Guidelines

   This section complements Section 3 by providing additional guidelines
   that are applicable to multicast and broadcast usage of UDP.

   Multicast and broadcast transmission [RFC1112] usually employ the UDP
   transport protocol, although they may be used with other transport
   protocols (e.g., UDP-Lite).

There are currently two models of multicast delivery: the Any-Source
Multicast (ASM) model as defined in [RFC1112] and the Source-Specific
Multicast (SSM) model as defined in [RFC4607]. ASM group members
will receive all data sent to the group by any source, while SSM
constrains the distribution tree to only one single source.

Specialized classes of applications also use UDP for IP multicast or
broadcast [RFC919]. The design of such specialized applications
requires expertise that goes beyond simple, unicast-specific
guidelines, since these senders may transmit to potentially very many
receivers across potentially very heterogeneous paths at the same
time, which significantly complicates congestion control, flow
control, and reliability mechanisms.

This section provides guidance on multicast and broadcast UDP usage.
Use of broadcast by an application is normally constrained by routers
to the local subnetwork. However, use of tunneling techniques and
proxies can and does result in some broadcast traffic traversing
Internet paths. These guidelines therefore also apply to broadcast
traffic.

The IETF has defined a reliable multicast framework [RFC3048] and
several building blocks to aid the designers of multicast
applications, such as [RFC3738] or [RFC4654].

Senders to anycast destinations must be aware that successive
messages sent to the same anycast IP address may be delivered to
different anycast nodes, i.e., arrive at different locations in the
topology.

Most UDP tunnels that carry IP multicast traffic use a tunnel
encapsulation with a unicast destination address, such as Automatic
Multicast Tunneling [RFC7450]. These MUST follow the same
requirements as a tunnel carrying unicast data (see Section 3.1.11).
There are deployment cases and solutions where the outer header of a
UDP tunnel contains a multicast destination address, such as
[RFC6513]. These cases are primarily deployed in controlled
environments over reserved capacity, often operating within a single
administrative domain, or between two domains over a bilaterally
agreed upon path with reserved capacity, and so congestion control is
OPTIONAL, but circuit breaker techniques are still RECOMMENDED in
order to restore some degree of service should the offered load
exceed the reserved capacity (e.g., due to misconfiguration).

4.1. Multicast Congestion Control Guidelines

   Unicast congestion-controlled transport mechanisms are often not
   applicable to multicast distribution services, or simply do not scale
   to large multicast trees, since they require bidirectional
   communication and adapt the sending rate to accommodate the network
   conditions to a single receiver. In contrast, multicast distribution
   trees may fan out to massive numbers of receivers, which limits the
   scalability of an in-band return channel to control the sending rate,
   and the one-to-many nature of multicast distribution trees prevents
   adapting the rate to the requirements of an individual receiver. For
   this reason, generating TCP-compatible aggregate flow rates for
   Internet multicast data, either native or tunneled, is the
   responsibility of the application implementing the congestion
   control.

   Applications using multicast SHOULD provide appropriate congestion
   control. Multicast congestion control needs to be designed using
   mechanisms that are robust to the potential heterogeneity of both the
   multicast distribution tree and the receivers belonging to a group.
   Heterogeneity may manifest itself in some receivers experiencing more
   loss that others, higher delay, and/or less ability to respond to
   network conditions. Congestion control is particularly important for
   any multicast session where all or part of the multicast distribution
   tree spans an access network (e.g., a home gateway). Two styles of
   congestion control have been defined in the RFC Series:

   o Feedback-based congestion control, in which the sender receives
      multicast or unicast UDP messages from the receivers allowing it
      to assess the level of congestion and then adjust the sender
      rate(s) (e.g., [RFC5740],[RFC4654]). Multicast methods may
      operate on longer timescales than for unicast (e.g., due to the
      higher group RTT of a heterogeneous group). A control method
      could decide not to reduce the rate of the entire multicast group
      in response to a control message received from a single receiver
      (e.g., a sender could set a minimum rate and decide to request a
      congested receiver to leave the multicast group and could also
      decide to distribute content to these congested receivers at a
      lower rate using unicast congestion control).

   o Receiver-driven congestion control, which does not require a
      receiver to send explicit UDP control messages for congestion
      control (e.g., [RFC3738], [RFC5775]). Instead, the sender
      distributes the data across multiple IP multicast groups (e.g.,
      using a set of {S,G} channels). Each receiver determines its own
      level of congestion and controls its reception rate using only
      multicast join/leave messages sent in the network control plane.
      This method scales to arbitrary large groups of receivers.

   Any multicast-enabled receiver may attempt to join and receive
   traffic from any group. This may imply the need for rate limits on
   individual receivers or the aggregate multicast service. Note, at
   the transport layer, there is no way to prevent a join message
   propagating to the next-hop router.

   Some classes of multicast applications support applications that can
   monitor the user-level quality of the transfer at the receiver.
   Applications that can detect a significant reduction in user quality
   SHOULD regard this as a congestion signal (e.g., to leave a group
   using layered multicast encoding); if not, they SHOULD use this
   signal to provide a circuit breaker to terminate the flow by leaving
   the multicast group.

4.1.1. Bulk-Transfer Multicast Applications

   Applications that perform bulk transmission of data over a multicast
   distribution tree, i.e., applications that exchange more than a few
   UDP datagrams per RTT, SHOULD implement a method for congestion
   control. The currently RECOMMENDED IETF methods are as follows:
   Asynchronous Layered Coding (ALC) [RFC5775], TCP-Friendly Multicast
   Congestion Control (TFMCC) [RFC4654], Wave and Equation Based Rate
   Control (WEBRC) [RFC3738], NACK-Oriented Reliable Multicast (NORM)
   transport protocol [RFC5740], File Delivery over Unidirectional
   Transport (FLUTE) [RFC6726], Real Time Protocol/Control Protocol
   (RTP/RTCP) [RFC3550].

   An application can alternatively implement another congestion control
   scheme following the guidelines of [RFC2887] and utilizing the
   framework of [RFC3048]. Bulk-transfer applications that choose not
   to implement [RFC4654], [RFC5775], [RFC3738], [RFC5740], [RFC6726],
   or [RFC3550] SHOULD implement a congestion control scheme that
   results in bandwidth use that competes fairly with TCP within an
   order of magnitude.

   Section 2 of [RFC3551] states that multimedia applications SHOULD
   monitor the packet-loss rate to ensure that it is within acceptable
   parameters. Packet loss is considered acceptable if a TCP flow
   across the same network path under the same network conditions would
   achieve an average throughput, measured on a reasonable timescale,
   that is not less than that of the UDP flow. The comparison to TCP
   cannot be specified exactly, but is intended as an "order-of-
   magnitude" comparison in timescale and throughput.

4.1.2. Low Data-Volume Multicast Applications

   All the recommendations in Section 3.1.3 are also applicable to low
   data-volume multicast applications.

4.2. Message Size Guidelines for Multicast

   A multicast application SHOULD NOT send UDP datagrams that result in
   IP packets that exceed the effective MTU as described in Section 3 of
   [RFC6807]. Consequently, an application SHOULD either use the
   effective MTU information provided by the "Population Count
   Extensions to Protocol Independent Multicast (PIM)" [RFC6807] or
   implement path MTU discovery itself (see Section 3.2) to determine
   whether the path to each destination will support its desired message
   size without fragmentation.

5. Programming Guidelines

   The de facto standard application programming interface (API) for
   TCP/IP applications is the "sockets" interface [POSIX]. Some
   platforms also offer applications the ability to directly assemble
   and transmit IP packets through "raw sockets" or similar facilities.
   This is a second, more cumbersome method of using UDP. The
   guidelines in this document cover all such methods through which an
   application may use UDP. Because the sockets API is by far the most
   common method, the remainder of this section discusses it in more
   detail.

   Although the sockets API was developed for UNIX in the early 1980s, a
   wide variety of non-UNIX operating systems also implement it. The
   sockets API supports both IPv4 and IPv6 [RFC3493]. The UDP sockets
   API differs from that for TCP in several key ways. Because
   application programmers are typically more familiar with the TCP
   sockets API, this section discusses these differences. [STEVENS]
   provides usage examples of the UDP sockets API.

   UDP datagrams may be directly sent and received, without any
   connection setup. Using the sockets API, applications can receive
   packets from more than one IP source address on a single UDP socket.
   Some servers use this to exchange data with more than one remote host
   through a single UDP socket at the same time. Many applications need
   to ensure that they receive packets from a particular source address;
   these applications MUST implement corresponding checks at the
   application layer or explicitly request that the operating system
   filter the received packets.

   Many operating systems also allow a UDP socket to be connected, i.e.,
   to bind a UDP socket to a specific pair of addresses and ports. This
   is similar to the corresponding TCP sockets API functionality.
   However, 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. Binding a UDP socket does not
   establish a connection -- UDP does not notify the remote end when a

local UDP socket is bound. Binding a socket also allows configuring
options that affect the UDP or IP layers, for example, use of the UDP
checksum or the IP Timestamp option. On some stacks, a bound socket
also allows an application to be notified when ICMP error messages
are received for its transmissions [RFC1122].

If a client/server application executes on a host with more than one
IP interface, the application SHOULD send any UDP responses with an
IP source address that matches the IP destination address of the UDP
datagram that carried the request (see [RFC1122], Section 4.1.3.5).
Many middleboxes expect this transmission behavior and drop replies
that are sent from a different IP address, as explained in
Section 3.5.

A UDP receiver can receive a valid UDP datagram with a zero-length
payload. Note that this is different from a return value of zero
from a read() socket call, which for TCP indicates the end of the
connection.

UDP provides no flow-control, i.e., the sender at any given time does
not know whether the receiver is able to handle incoming
transmissions. This is another reason why UDP-based applications
need to be robust in the presence of packet loss. This loss can also
occur within the sending host, when an application sends data faster
than the line rate of the outbound network interface. It can also
occur at the destination, where receive calls fail to return all the
data that was sent when the application issues them too infrequently
(i.e., such that the receive buffer overflows). Robust flow control
mechanisms are difficult to implement, which is why applications that
need this functionality SHOULD consider using a full-featured
transport protocol such as TCP.

When an application closes a TCP, SCTP, or DCCP socket, the transport
protocol on the receiving host is required to maintain TIME-WAIT
state. This prevents delayed packets from the closed connection
instance from being mistakenly associated with a later connection
instance that happens to reuse the same IP address and port pairs.
The UDP protocol does not implement such a mechanism. Therefore,
UDP-based applications need to be robust to reordering and delay.
One application may close a socket or terminate, followed in time by
another application receiving on the same port. This later
application may then receive packets intended for the first
application that were delayed in the network.

5.1. Using UDP Ports

   The rules and procedures for the management of the "Service Name and
   Transport Protocol Port Number Registry" are specified in [RFC6335].
   Recommendations for use of UDP ports are provided in [RFC7605].

   A UDP sender SHOULD NOT use a source port value of zero. A source
   port number that cannot be easily determined from the address or
   payload type provides protection at the receiver from data injection
   attacks by off-path devices. A UDP receiver SHOULD NOT bind to port
   zero.

   Applications SHOULD implement receiver port and address checks at the
   application layer or explicitly request that the operating system
   filter the received packets to prevent receiving packets with an
   arbitrary port. This measure is designed to provide additional
   protection from data injection attacks from an off-path source (where
   the port values may not be known).

   Applications SHOULD provide a check that protects from off-path data
   injection, avoiding an application receiving packets that were
   created by an unauthorized third party. TCP stacks commonly use a
   randomized source port to provide this protection [RFC6056]; UDP
   applications should follow the same technique. Middleboxes and end
   systems often make assumptions about the system ports or user ports;
   hence, it is recommended to use randomized ports in the Dynamic and/
   or Private Port range. Setting a "randomized" source port also
   provides greater assurance that reported ICMP errors originate from
   network systems on the path used by a particular flow. Some UDP
   applications choose to use a predetermined value for the source port
   (including some multicast applications), these applications need to
   therefore employ a different technique. Protection from off-path
   data attacks can also be provided by randomizing the initial value of
   another protocol field within the datagram payload, and checking the
   validity of this field at the receiver (e.g., RTP has random initial
   sequence number and random media timestamp offsets [RFC3550]).

   When using multicast, IP routers perform a reverse-path forwarding
   (RPF) check for each multicast packet. This provides protection from
   off-path data injection, restricting opportunities to forge a
   packet's source address. When a receiver joins a multicast group and
   filters based on the source address the filter verifies the sender's
   IP address. This is always the case when using an SSM {S,G} channel.

5.1.1. Usage of UDP for Source Port Entropy and the IPv6 Flow Label

   Some applications use the UDP datagram header as a source of entropy
   for network devices that implement ECMP [RFC6438]. A UDP tunnel
   application targeting this usage encapsulates an inner packet using
   UDP, where the UDP source port value forms a part of the entropy that
   can be used to balance forwarding of network traffic by the devices
   that use ECMP. A sending tunnel endpoint selects a source port value
   in the UDP datagram header that is computed from the inner flow
   information (e.g., the encapsulated packet headers). To provide
   sufficient entropy, the sending tunnel endpoint maps the encapsulated
   traffic to one of a range of UDP source values. The value SHOULD be
   within the ephemeral port range, i.e., 49152 to 65535, where the high
   order two bits of the port are set to one. The available source port
   entropy of 14 bits (using the ephemeral port range) plus the outer IP
   addresses seems sufficient for entropy for most ECMP applications
   [ENCAP].

   To avoid reordering within an IP flow, the same UDP source port value
   SHOULD be used for all packets assigned to an encapsulated flow
   (e.g., using a hash of the relevant headers). The entropy mapping
   for a flow MAY change over the lifetime of the encapsulated flow
   [ENCAP]. For instance, this could be changed as a Denial of Service
   (DOS) mitigation, or as a means to effect routing through the ECMP
   network. However, the source port selected for a flow SHOULD NOT
   change more than once in every thirty seconds (e.g., as in
   [RFC8086]).

   The use of the source port field for entropy has several side effects
   that need to be considered, including:

   o It can increase the probability of misdelivery of corrupted
      packets, which increases the need for checksum computation or an
      equivalent mechanism to protect other UDP applications from
      misdelivery errors Section 3.4.

   o It is expected to reduce the probability of successful middlebox
      traversal Section 3.5. This use of the source port field will
      often not be suitable for applications targeting deployment in the
      general Internet.

   o It can prevent the field being usable to protect from off-path
      attacks (described in Section 5.1). Designers therefore need to
      consider other mechanisms to provide equivalent protection (e.g.,
      to restrict use to a controlled environment [RFC7510]
      Section 3.6).

The UDP source port number field has also been leveraged to produce
entropy with IPv6. However, in the case of IPv6, the "flow label"
[RFC6437] may also alternatively be used to provide entropy for load
balancing [RFC6438]. This use of the flow label for load balancing
is consistent with the definition of the field, although further
clarity was needed to ensure the field can be consistently used for
this purpose. Therefore, an updated IPv6 flow label [RFC6437] and
ECMP routing [RFC6438] usage was specified.

To ensure future opportunities to use the flow label, UDP
applications SHOULD set the flow label field, even when an entropy
value is also set in the source port field (e.g., An IPv6 tunnel
endpoint could copy the source port flow entropy value to the IPv6
flow label field [RFC8086]). Router vendors are encouraged to start
using the IPv6 flow label as a part of the flow hash, providing
support for IP-level ECMP without requiring use of UDP. The end-to-
end use of flow labels for load balancing is a long-term solution.
Even if the usage of the flow label has been clarified, there will be
a transition time before a significant proportion of endpoints start
to assign a good quality flow label to the flows that they originate.
The use of load balancing using the transport header fields will
likely continue until widespread deployment is finally achieved.

5.1.2. Applications Using Multiple UDP Ports

A single application may exchange several types of data. In some
cases, this may require multiple UDP flows (e.g., multiple sets of
flows, identified by different five-tuples). [RFC6335] recommends
application developers not to apply to IANA to be assigned multiple
well-known ports (user or system). It does not discuss the
implications of using multiple flows with the same well-known port or
pairs of dynamic ports (e.g., identified by a service name or
signaling protocol).

Use of multiple flows can affect the network in several ways:

o Starting a series of successive connections can increase the
   number of state bindings in middleboxes (e.g., NAPT or Firewall)
   along the network path. UDP-based middlebox traversal usually
   relies on timeouts to remove old state, since middleboxes are
   unaware when a particular flow ceases to be used by an
   application.

o Using several flows at the same time may result in seeing
   different network characteristics for each flow. It cannot be
   assumed both follow the same path (e.g., when ECMP is used,
   traffic is intentionally hashed onto different parallel paths
   based on the port numbers).

   o Using several flows can also increase the occupancy of a binding
      or lookup table in a middlebox (e.g., NAPT or Firewall), which may
      cause the device to change the way it manages the flow state.

   o Further, using excessive numbers of flows can degrade the ability
      of a unicast congestion control to react to congestion events,
      unless the congestion state is shared between all flows in a
      session. A receiver-driven multicast congestion control requires
      the sending application to distribute its data over a set of IP
      multicast groups, each receiver is therefore expected to receive
      data from a modest number of simultaneously active UDP ports.

   Therefore, applications MUST NOT assume consistent behavior of
   middleboxes when multiple UDP flows are used; many devices respond
   differently as the number of used ports increases. Using multiple
   flows with different QoS requirements requires applications to verify
   that the expected performance is achieved using each individual flow
   (five-tuple), see Section 3.1.9.

5.2. ICMP Guidelines

   Applications can utilize information about ICMP error messages that
   the UDP layer passes up for a variety of purposes [RFC1122].
   Applications SHOULD appropriately validate the payload of ICMP
   messages to ensure these are received in response to transmitted
   traffic (i.e., a reported error condition that corresponds to a UDP
   datagram actually sent by the application). This requires context,
   such as local state about communication instances to each
   destination, that although readily available in connection-oriented
   transport protocols is not always maintained by UDP-based
   applications. Note that not all platforms have the necessary APIs to
   support this validation, and some platforms already perform this
   validation internally before passing ICMP information to the
   application.

   Any application response to ICMP error messages SHOULD be robust to
   temporary routing failures (sometimes called "soft errors"), e.g.,
   transient ICMP "unreachable" messages ought to not normally cause a
   communication abort.

   ICMP messages are being increasingly filtered by middleboxes. A UDP
   application therefore SHOULD NOT rely on their delivery for correct
   and safe operation.

6. Security Considerations

   UDP does not provide communications security. Applications that need
   to protect their communications against eavesdropping, tampering, or
   message forgery SHOULD employ end-to-end security services provided
   by other IETF protocols.

   UDP applications SHOULD provide protection from off-path data
   injection attacks using a randomized source port or equivalent
   technique (see Section 5.1).

   Applications that respond to short requests with potentially large
   responses are a potential vector for amplification attacks, and
   SHOULD take steps to minimize their potential for being abused as
   part of a DoS attack. That could mean authenticating the sender
   before responding; noting that the source IP address of a request is
   not a useful authenticator, because it can easily be spoofed. Or it
   may mean otherwise limiting the cases where short unauthenticated
   requests produce large responses. Applications MAY also want to
   offer ways to limit the number of requests they respond to in a time
   interval, in order to cap the bandwidth they consume.

   One option for securing UDP communications is with IPsec [RFC4301],
   which can provide authentication for flows of IP packets through the
   Authentication Header (AH) [RFC4302] and encryption and/or
   authentication through the Encapsulating Security Payload (ESP)
   [RFC4303]. Applications use the Internet Key Exchange (IKE)
   [RFC7296] to configure IPsec for their sessions. Depending on how
   IPsec is configured for a flow, it can authenticate or encrypt the
   UDP headers as well as UDP payloads. If an application only requires
   authentication, ESP with no encryption but with authentication is
   often a better option than AH, because ESP can operate across
   middleboxes. An application that uses IPsec requires the support of
   an operating system that implements the IPsec protocol suite, and the
   network path must permit IKE and IPsec traffic. This may become more
   common with IPv6 deployments [RFC6092].

   Although it is possible to use IPsec to secure UDP communications,
   not all operating systems support IPsec or allow applications to
   easily configure it for their flows. A second option for securing
   UDP communications is through Datagram Transport Layer Security
   (DTLS) [RFC6347][RFC7525]. DTLS provides communication privacy by
   encrypting UDP payloads. It does not protect the UDP headers.
   Applications can implement DTLS without relying on support from the
   operating system.

Many other options for authenticating or encrypting UDP payloads
exist. For example, the GSS-API security framework [RFC2743] or
Cryptographic Message Syntax (CMS) [RFC5652] could be used to protect
UDP payloads. There exist a number of security options for RTP
[RFC3550] over UDP, especially to accomplish key-management, see
[RFC7201]. These options covers many usages, including point-to-
point, centralized group communication as well as multicast. In some
applications, a better solution is to protect larger stand-alone
objects, such as files or messages, instead of individual UDP
payloads. In these situations, CMS [RFC5652], S/MIME [RFC5751] or
OpenPGP [RFC4880] could be used. In addition, there are many
non-IETF protocols in this area.

Like congestion control mechanisms, security mechanisms are difficult
to design and implement correctly. It is hence RECOMMENDED that
applications employ well-known standard security mechanisms such as
DTLS or IPsec, rather than inventing their own.

The Generalized TTL Security Mechanism (GTSM) [RFC5082] may be used
with UDP applications when the intended endpoint is on the same link
as the sender. This lightweight mechanism allows a receiver to
filter unwanted packets.

In terms of congestion control, [RFC2309] and [RFC2914] discuss the
dangers of congestion-unresponsive flows to the Internet. [RFC8084]
describes methods that can be used to set a performance envelope that
can assist in preventing congestion collapse in the absence of
congestion control or when the congestion control fails to react to
congestion events. This document provides guidelines to designers of
UDP-based applications to congestion-control their transmissions, and
does not raise any additional security concerns.

Some network operators have experienced surges of UDP attack traffic
that are multiple orders of magnitude above the baseline traffic rate
for UDP. This can motivate operators to limit the data rate or
packet rate of UDP traffic. This may in turn limit the throughput
that an application can achieve using UDP and could also result in
higher packet loss for UDP traffic that would not be experienced if
other transport protocols had been used.

A UDP application with a long-lived association between the sender
and receiver, ought to be designed so that the sender periodically
checks that the receiver still wants ("consents") to receive traffic
and need to be designed to stop if there is no explicit confirmation
of this [RFC7675]. Applications that require communications in two
directions to implement protocol functions (such as reliability or

congestion control) will need to independently check both directions
of communication, and may have to exchange keep-alive messages to
traverse middleboxes (see Section 3.5).

7. Summary

   This section summarizes the key guidelines made in Sections 3 - 6 in
   a tabular format (Table 1) for easy referencing.

```
+----------------------------------------------------------+---------+
| Recommendation                                           | Section |
+----------------------------------------------------------+---------+
| MUST tolerate a wide range of Internet path conditions   | 3       |
| SHOULD use a full-featured transport (e.g., TCP)         |         |
|                                                          |         |
| SHOULD control rate of transmission                      | 3.1     |
| SHOULD perform congestion control over all traffic       |         |
|                                                          |         |
| for bulk transfers,                                      | 3.1.2   |
| SHOULD consider implementing TFRC                         |         |
| else, SHOULD in other ways use bandwidth similar to TCP  |         |
|                                                          |         |
| for non-bulk transfers,                                  | 3.1.3   |
| SHOULD measure RTT and transmit max. 1 datagram/RTT      | 3.1.1   |
| else, SHOULD send at most 1 datagram every 3 seconds     |         |
| SHOULD back-off retransmission timers following loss     |         |
|                                                          |         |
| SHOULD provide mechanisms to regulate the bursts of      | 3.1.6   |
| transmission                                             |         |
|                                                          |         |
| MAY implement ECN; a specific set of application         | 3.1.7   |
| mechanisms are REQUIRED if ECN is used.                  |         |
|                                                          |         |
| for DiffServ, SHOULD NOT rely on implementation of PHBs  | 3.1.8   |
|                                                          |         |
| for QoS-enabled paths, MAY choose not to use CC          | 3.1.9   |
|                                                          |         |
| SHOULD NOT rely solely on QoS for their capacity         | 3.1.10  |
| non-CC controlled flows SHOULD implement a transport     |         |
| circuit breaker                                          |         |
| MAY implement a circuit breaker for other applications   |         |
|                                                          |         |
| for tunnels carrying IP traffic,                         | 3.1.11  |
| SHOULD NOT perform congestion control                    |         |
| MUST correctly process the IP ECN field                  |         |
|                                                          |         |
```

```
| for non-IP tunnels or rate not determined by traffic, |        |
| SHOULD perform CC or use circuit breaker              | 3.1.11 |
| SHOULD restrict types of traffic transported by the  |        |
| tunnel                                               |        |
|                                                      |        |
| SHOULD NOT send datagrams that exceed the PMTU, i.e., | 3.2    |
| SHOULD discover PMTU or send datagrams < minimum PMTU;|        |
| Specific application mechanisms are REQUIRED if PLPMTUD|       |
| is used.                                             |        |
|                                                      |        |
| SHOULD handle datagram loss, duplication, reordering | 3.3    |
| SHOULD be robust to delivery delays up to 2 minutes  |        |
|                                                      |        |
| SHOULD enable IPv4 UDP checksum                       | 3.4    |
| SHOULD enable IPv6 UDP checksum; Specific application | 3.4.1  |
| mechanisms are REQUIRED if a zero IPv6 UDP checksum is |       |
| used.                                                |        |
|                                                      |        |
| SHOULD provide protection from off-path attacks      | 5.1    |
| else, MAY use UDP-Lite with suitable checksum coverage| 3.4.2  |
|                                                      |        |
| SHOULD NOT always send middlebox keep-alive messages | 3.5    |
| MAY use keep-alives when needed (min. interval 15 sec)|       |
|                                                      |        |
| Applications specified for use in limited use (or    | 3.6    |
| controlled environments) SHOULD identify equivalent  |        |
| mechanisms and describe their use case.              |        |
|                                                      |        |
| Bulk-multicast apps SHOULD implement congestion control| 4.1.1 |
|                                                      |        |
| Low volume multicast apps SHOULD implement congestion | 4.1.2  |
| control                                              |        |
|                                                      |        |
| Multicast apps SHOULD use a safe PMTU                 | 4.2    |
|                                                      |        |
| SHOULD avoid using multiple ports                    | 5.1.2  |
| MUST check received IP source address                |        |
|                                                      |        |
| SHOULD validate payload in ICMP messages             | 5.2    |
|                                                      |        |
| SHOULD use a randomized source port or equivalent    | 6      |
| technique, and, for client/server applications, SHOULD|       |
| send responses from source address matching request |        |
| 5.1                                                  |        |
| SHOULD use standard IETF security protocols when needed | 6    |
+-------------------------------------------------------+--------+
```

                   Table 1: Summary of Recommendations

8. References

8.1. Normative References

   [RFC768]   Postel, J., "User Datagram Protocol", STD 6, RFC 768,
              DOI 10.17487/RFC0768, August 1980,
              <http://www.rfc-editor.org/info/rfc768>.

   [RFC793]   Postel, J., "Transmission Control Protocol", STD 7,
              RFC 793, DOI 10.17487/RFC0793, September 1981,
              <http://www.rfc-editor.org/info/rfc793>.

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

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

   [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>.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119,
              DOI 10.17487/RFC2119, March 1997,
              <http://www.rfc-editor.org/info/rfc2119>.

   [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>.

   [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41,
              RFC 2914, DOI 10.17487/RFC2914, September 2000,
              <http://www.rfc-editor.org/info/rfc2914>.

   [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>.

   [RFC4787] Audet, F., Ed. and C. Jennings, "Network Address
              Translation (NAT) Behavioral Requirements for Unicast
              UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January
              2007, <http://www.rfc-editor.org/info/rfc4787>.

   [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>.

   [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>.

   [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines
              for Application Designers", BCP 145, RFC 5405,
              DOI 10.17487/RFC5405, November 2008,
              <http://www.rfc-editor.org/info/rfc5405>.

   [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion
              Notification", RFC 6040, DOI 10.17487/RFC6040, November
              2010, <http://www.rfc-editor.org/info/rfc6040>.

   [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent,
              "Computing TCP's Retransmission Timer", RFC 6298,
              DOI 10.17487/RFC6298, June 2011,
              <http://www.rfc-editor.org/info/rfc6298>.

   [RFC8084] Fairhurst, G., "Network Transport Circuit Breakers",
              BCP 208, RFC 8084, DOI 10.17487/RFC8084, March 2017,
              <http://www.rfc-editor.org/info/rfc8084>.

8.2. Informative References

   [ALLMAN]   Allman, M. and E. Blanton, "Notes on burst mitigation for
              transport protocols", March 2005.

   [BEHAVE-APP]
              Ford, B., "Application Design Guidelines for Traversal
              through Network Address Translators", Work in Progress,
              draft-ford-behave-app-05, March 2007.

   [ENCAP]   Nordmark, E., Ed., Tian, A., Gross, J., Hudson, J.,
              Kreeger, L., Garg, P., Thaler, P., and T. Herbert,
              "Encapsulation Considerations", Work in Progress,
              draft-ietf-rtgwg-dt-encap-02, October 2016.

   [FABER]   Faber, T., Touch, J., and W. Yue, "The TIME-WAIT State in
              TCP and Its Effect on Busy Servers", Proc. IEEE Infocom,
              March 1999.

   [INT-TUNNELS]
              Touch, J. and W. Townsley, "IP Tunnels in the Internet
              Architecture", Work in Progress,
              draft-ietf-intarea-tunnels-03, July 2016.

   [POSIX]   IEEE Std. 1003.1-2001, , "Standard for Information
              Technology - Portable Operating System Interface (POSIX)",
              Open Group Technical Standard: Base Specifications Issue
              6, ISO/IEC 9945:2002, December 2001.

   [RFC919]   Mogul, J., "Broadcasting Internet Datagrams", STD 5,
              RFC 919, DOI 10.17487/RFC0919, October 1984,
              <http://www.rfc-editor.org/info/rfc919>.

   [RFC1112] Deering, S., "Host extensions for IP multicasting", STD 5,
              RFC 1112, DOI 10.17487/RFC1112, August 1989,
              <http://www.rfc-editor.org/info/rfc1112>.

   [RFC1536] Kumar, A., Postel, J., Neuman, C., Danzig, P., and S.
              Miller, "Common DNS Implementation Errors and Suggested
              Fixes", RFC 1536, DOI 10.17487/RFC1536, October 1993,
              <http://www.rfc-editor.org/info/rfc1536>.

   [RFC1546] Partridge, C., Mendez, T., and W. Milliken, "Host
              Anycasting Service", RFC 1546, DOI 10.17487/RFC1546,
              November 1993, <http://www.rfc-editor.org/info/rfc1546>.

   [RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering,
              S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G.,
              Partridge, C., Peterson, L., Ramakrishnan, K., Shenker,
              S., Wroclawski, J., and L. Zhang, "Recommendations on
              Queue Management and Congestion Avoidance in the
              Internet", RFC 2309, DOI 10.17487/RFC2309, April 1998,
              <http://www.rfc-editor.org/info/rfc2309>.

   [RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z.,
              and W. Weiss, "An Architecture for Differentiated
              Services", RFC 2475, DOI 10.17487/RFC2475, December 1998,
              <http://www.rfc-editor.org/info/rfc2475>.

   [RFC2675] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms",
              RFC 2675, DOI 10.17487/RFC2675, August 1999,
              <http://www.rfc-editor.org/info/rfc2675>.

   [RFC2743] Linn, J., "Generic Security Service Application Program
              Interface Version 2, Update 1", RFC 2743,
              DOI 10.17487/RFC2743, January 2000,
              <http://www.rfc-editor.org/info/rfc2743>.

   [RFC2887] Handley, M., Floyd, S., Whetten, B., Kermode, R.,
              Vicisano, L., and M. Luby, "The Reliable Multicast Design
              Space for Bulk Data Transfer", RFC 2887,
              DOI 10.17487/RFC2887, August 2000,
              <http://www.rfc-editor.org/info/rfc2887>.

   [RFC2983] Black, D., "Differentiated Services and Tunnels",
              RFC 2983, DOI 10.17487/RFC2983, October 2000,
              <http://www.rfc-editor.org/info/rfc2983>.

   [RFC3048] Whetten, B., Vicisano, L., Kermode, R., Handley, M.,
              Floyd, S., and M. Luby, "Reliable Multicast Transport
              Building Blocks for One-to-Many Bulk-Data Transfer",
              RFC 3048, DOI 10.17487/RFC3048, January 2001,
              <http://www.rfc-editor.org/info/rfc3048>.

   [RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager",
              RFC 3124, DOI 10.17487/RFC3124, June 2001,
              <http://www.rfc-editor.org/info/rfc3124>.

   [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>.

   [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston,
              A., Peterson, J., Sparks, R., Handley, M., and E.
              Schooler, "SIP: Session Initiation Protocol", RFC 3261,
              DOI 10.17487/RFC3261, June 2002,
              <http://www.rfc-editor.org/info/rfc3261>.

   [RFC3303] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A., and
              A. Rayhan, "Middlebox communication architecture and
              framework", RFC 3303, DOI 10.17487/RFC3303, August 2002,
              <http://www.rfc-editor.org/info/rfc3303>.

   [RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W.
              Stevens, "Basic Socket Interface Extensions for IPv6",
              RFC 3493, DOI 10.17487/RFC3493, February 2003,
              <http://www.rfc-editor.org/info/rfc3493>.

   [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>.

   [RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and
              Video Conferences with Minimal Control", STD 65, RFC 3551,
              DOI 10.17487/RFC3551, July 2003,
              <http://www.rfc-editor.org/info/rfc3551>.

   [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>.

   [RFC3819] Karn, P., Ed., Bormann, C., Fairhurst, G., Grossman, D.,
              Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L.
              Wood, "Advice for Internet Subnetwork Designers", BCP 89,
              RFC 3819, DOI 10.17487/RFC3819, July 2004,
              <http://www.rfc-editor.org/info/rfc3819>.

   [RFC4301] Kent, S. and K. Seo, "Security Architecture for the
              Internet Protocol", RFC 4301, DOI 10.17487/RFC4301,
              December 2005, <http://www.rfc-editor.org/info/rfc4301>.

   [RFC4302] Kent, S., "IP Authentication Header", RFC 4302,
              DOI 10.17487/RFC4302, December 2005,
              <http://www.rfc-editor.org/info/rfc4302>.

   [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)",
              RFC 4303, DOI 10.17487/RFC4303, December 2005,
              <http://www.rfc-editor.org/info/rfc4303>.

   [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>.

   [RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through
              Network Address Translations (NATs)", RFC 4380,
              DOI 10.17487/RFC4380, February 2006,
              <http://www.rfc-editor.org/info/rfc4380>.

   [RFC4607] Holbrook, H. and B. Cain, "Source-Specific Multicast for
              IP", RFC 4607, DOI 10.17487/RFC4607, August 2006,
              <http://www.rfc-editor.org/info/rfc4607>.

   [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>.

   [RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R.
              Thayer, "OpenPGP Message Format", RFC 4880,
              DOI 10.17487/RFC4880, November 2007,
              <http://www.rfc-editor.org/info/rfc4880>.

   [RFC4890] Davies, E. and J. Mohacsi, "Recommendations for Filtering
              ICMPv6 Messages in Firewalls", RFC 4890,
              DOI 10.17487/RFC4890, May 2007,
              <http://www.rfc-editor.org/info/rfc4890>.

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

   [RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly
              Errors at High Data Rates", RFC 4963,
              DOI 10.17487/RFC4963, July 2007,
              <http://www.rfc-editor.org/info/rfc4963>.

   [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common
              Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007,
              <http://www.rfc-editor.org/info/rfc4987>.

   [RFC5082] Gill, V., Heasley, J., Meyer, D., Savola, P., Ed., and C.
              Pignataro, "The Generalized TTL Security Mechanism
              (GTSM)", RFC 5082, DOI 10.17487/RFC5082, October 2007,
              <http://www.rfc-editor.org/info/rfc5082>.

   [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment
              (ICE): A Protocol for Network Address Translator (NAT)
              Traversal for Offer/Answer Protocols", RFC 5245,
              DOI 10.17487/RFC5245, April 2010,
              <http://www.rfc-editor.org/info/rfc5245>.

   [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>.

   [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70,
              RFC 5652, DOI 10.17487/RFC5652, September 2009,
              <http://www.rfc-editor.org/info/rfc5652>.

   [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>.

   [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>.

   [RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet
              Mail Extensions (S/MIME) Version 3.2 Message
              Specification", RFC 5751, DOI 10.17487/RFC5751, January
              2010, <http://www.rfc-editor.org/info/rfc5751>.

   [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>.

   [RFC5971] Schulzrinne, H. and R. Hancock, "GIST: General Internet
              Signalling Transport", RFC 5971, DOI 10.17487/RFC5971,
              October 2010, <http://www.rfc-editor.org/info/rfc5971>.

   [RFC5973] Stiemerling, M., Tschofenig, H., Aoun, C., and E. Davies,
              "NAT/Firewall NSIS Signaling Layer Protocol (NSLP)",
              RFC 5973, DOI 10.17487/RFC5973, October 2010,
              <http://www.rfc-editor.org/info/rfc5973>.

   [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>.

   [RFC6092] Woodyatt, J., Ed., "Recommended Simple Security
              Capabilities in Customer Premises Equipment (CPE) for
              Providing Residential IPv6 Internet Service", RFC 6092,
              DOI 10.17487/RFC6092, January 2011,
              <http://www.rfc-editor.org/info/rfc6092>.

   [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S.
              Cheshire, "Internet Assigned Numbers Authority (IANA)
              Procedures for the Management of the Service Name and
              Transport Protocol Port Number Registry", BCP 165,
              RFC 6335, DOI 10.17487/RFC6335, August 2011,
              <http://www.rfc-editor.org/info/rfc6335>.

   [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>.

   [RFC6396] Blunk, L., Karir, M., and C. Labovitz, "Multi-Threaded
              Routing Toolkit (MRT) Routing Information Export Format",
              RFC 6396, DOI 10.17487/RFC6396, October 2011,
              <http://www.rfc-editor.org/info/rfc6396>.

   [RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme,
              "IPv6 Flow Label Specification", RFC 6437,
              DOI 10.17487/RFC6437, November 2011,
              <http://www.rfc-editor.org/info/rfc6437>.

   [RFC6438] Carpenter, B. and S. Amante, "Using the IPv6 Flow Label
              for Equal Cost Multipath Routing and Link Aggregation in
              Tunnels", RFC 6438, DOI 10.17487/RFC6438, November 2011,
              <http://www.rfc-editor.org/info/rfc6438>.

   [RFC6513] Rosen, E., Ed. and R. Aggarwal, Ed., "Multicast in MPLS/
              BGP IP VPNs", RFC 6513, DOI 10.17487/RFC6513, February
              2012, <http://www.rfc-editor.org/info/rfc6513>.

   [RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P.,
              and K. Carlberg, "Explicit Congestion Notification (ECN)
              for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August
              2012, <http://www.rfc-editor.org/info/rfc6679>.

   [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>.

   [RFC6807] Farinacci, D., Shepherd, G., Venaas, S., and Y. Cai,
              "Population Count Extensions to Protocol Independent
              Multicast (PIM)", RFC 6807, DOI 10.17487/RFC6807, December
              2012, <http://www.rfc-editor.org/info/rfc6807>.

   [RFC6887] Wing, D., Ed., Cheshire, S., Boucadair, M., Penno, R., and
              P. Selkirk, "Port Control Protocol (PCP)", RFC 6887,
              DOI 10.17487/RFC6887, April 2013,
              <http://www.rfc-editor.org/info/rfc6887>.

   [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>.

   [RFC7143] Chadalapaka, M., Satran, J., Meth, K., and D. Black,
              "Internet Small Computer System Interface (iSCSI) Protocol
              (Consolidated)", RFC 7143, DOI 10.17487/RFC7143, April
              2014, <http://www.rfc-editor.org/info/rfc7143>.

   [RFC7201] Westerlund, M. and C. Perkins, "Options for Securing RTP
              Sessions", RFC 7201, DOI 10.17487/RFC7201, April 2014,
              <http://www.rfc-editor.org/info/rfc7201>.

   [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T.
              Kivinen, "Internet Key Exchange Protocol Version 2
              (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October
              2014, <http://www.rfc-editor.org/info/rfc7296>.

   [RFC7450] Bumgardner, G., "Automatic Multicast Tunneling", RFC 7450,
              DOI 10.17487/RFC7450, February 2015,
              <http://www.rfc-editor.org/info/rfc7450>.

   [RFC7510] Xu, X., Sheth, N., Yong, L., Callon, R., and D. Black,
              "Encapsulating MPLS in UDP", RFC 7510,
              DOI 10.17487/RFC7510, April 2015,
              <http://www.rfc-editor.org/info/rfc7510>.

   [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>.

   [RFC7560] Kuehlewind, M., Ed., Scheffenegger, R., and B. Briscoe,
              "Problem Statement and Requirements for Increased Accuracy
              in Explicit Congestion Notification (ECN) Feedback",
              RFC 7560, DOI 10.17487/RFC7560, August 2015,
              <http://www.rfc-editor.org/info/rfc7560>.

   [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF
              Recommendations Regarding Active Queue Management",
              BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015,
              <http://www.rfc-editor.org/info/rfc7567>.

   [RFC7605] Touch, J., "Recommendations on Using Assigned Transport
              Port Numbers", BCP 165, RFC 7605, DOI 10.17487/RFC7605,
              August 2015, <http://www.rfc-editor.org/info/rfc7605>.

   [RFC7657] Black, D., Ed. and P. Jones, "Differentiated Services
              (Diffserv) and Real-Time Communication", RFC 7657,
              DOI 10.17487/RFC7657, November 2015,
              <http://www.rfc-editor.org/info/rfc7657>.

   [RFC7675] Perumal, M., Wing, D., Ravindranath, R., Reddy, T., and M.
              Thomson, "Session Traversal Utilities for NAT (STUN) Usage
              for Consent Freshness", RFC 7675, DOI 10.17487/RFC7675,
              October 2015, <http://www.rfc-editor.org/info/rfc7675>.

   [RFC8083] Perkins, C. and V. Singh, "Multimedia Congestion Control:
              Circuit Breakers for Unicast RTP Sessions", RFC 8083,
              DOI 10.17487/RFC8083, March 2017,
              <http://www.rfc-editor.org/info/rfc8083>.

   [RFC8086] Yong, L., Ed., Crabbe, E., Xu, X., and T. Herbert, "GRE-
              in-UDP Encapsulation", RFC 8086, DOI 10.17487/RFC8086,
              March 2017, <http://www.rfc-editor.org/info/rfc8086>.

   [RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using
              Explicit Congestion Notification (ECN)", RFC 8087,
              DOI 10.17487/RFC8087, March 2017,
              <http://www.rfc-editor.org/info/rfc8087>.

   [STEVENS] Stevens, W., Fenner, B., and A. Rudoff, "UNIX Network
              Programming, The sockets Networking API", Addison-Wesley,
              2004.

   [UPnP]   UPnP Forum, , "Internet Gateway Device (IGD) Standardized
              Device Control Protocol V 1.0", November 2001.

Appendix A. Case Study of the Use of IPv6 UDP Zero-Checksum Mode

   This appendix provides a brief review of MPLS-in-UDP as an example of
   a UDP Tunnel Encapsulation that defines a UDP encapsulation. The
   purpose of the appendix is to provide a concrete example of which
   mechanisms were required in order to safely use UDP zero-checksum
   mode for MPLS-in-UDP tunnels over IPv6. By default, UDP requires a
   checksum for use with IPv6. An option has been specified that
   permits a zero IPv6 UDP checksum when used in specific environments,
   specified in [RFC7510], and defines a set of operational constraints
   for use of this mode. These are summarized below:

   A UDP tunnel or encapsulation using a zero-checksum mode with IPv6
   must only be deployed within a single network (with a single network
   operator) or networks of an adjacent set of cooperating network
   operators where traffic is managed to avoid congestion, rather than
   over the Internet where congestion control is required. MPLS-in-UDP
   has been specified for networks under single administrative control
   (such as within a single operator's network) where it is known
   (perhaps through knowledge of equipment types and lower-layer checks)
   that packet corruption is exceptionally unlikely and where the
   operator is willing to take the risk of undetected packet corruption.

   The tunnel encapsulator SHOULD use different IPv6 addresses for each
   UDP tunnel that uses the UDP zero-checksum mode, regardless of the
   decapsulator, to strengthen the decapsulator's check of the IPv6
   source address (i.e., the same IPv6 source address SHOULD NOT be used
   with more than one IPv6 destination address, independent of whether
   that destination address is a unicast or multicast address). Use of
   MPLS-in-UDP may be extended to networks within a set of closely
   cooperating network administrations (such as network operators who
   have agreed to work together to jointly provide specific services)
   [RFC7510].

   The requirement for MPLS-in-UDP endpoints to check the source IPv6
   address in addition to the destination IPv6 address, plus the strong
   recommendation against reuse of source IPv6 addresses among MPLS-in-
   UDP tunnels collectively provide some mitigation for the absence of
   UDP checksum coverage of the IPv6 header. In addition, the MPLS data
   plane only forwards packets with valid labels (i.e., labels that have
   been distributed by the tunnel egress Label Switched Router, LSR),
   providing some additional opportunity to detect MPLS-in-UDP packet
   misdelivery when the misdelivered packet contains a label that is not
   valid for forwarding at the receiving LSR. The expected result for
   IPv6 UDP zero-checksum mode for MPLS-in-UDP is that corruption of the
   destination IPv6 address will usually cause packet discard, as
   offsetting corruptions to the source IPv6 and/or MPLS top label are
   unlikely.

Additional assurance is provided by the restrictions in the above
exceptions that limit usage of IPv6 UDP zero-checksum mode to well-
managed networks for which MPLS packet corruption has not been a
problem in practice. Hence, MPLS-in-UDP is suitable for transmission
over lower layers in well-managed networks that are allowed by the
exceptions stated above and the rate of corruption of the inner IP
packet on such networks is not expected to increase by comparison to
MPLS traffic that is not encapsulated in UDP. For these reasons,
MPLS-in-UDP does not provide an additional integrity check when UDP
zero-checksum mode is used with IPv6, and this design is in
accordance with requirements 2, 3, and 5 specified in Section 5 of
[RFC6936].

The MPLS-in-UDP encapsulation does not provide a mechanism to safely
fall back to using a checksum when a path change occurs that
redirects a tunnel over a path that includes a middlebox that
discards IPv6 datagrams with a zero UDP checksum. In this case, the
MPLS-in-UDP tunnel will be black-holed by that middlebox.
Recommended changes to allow firewalls, NATs and other middleboxes to
support use of an IPv6 zero UDP checksum are described in Section 5
of [RFC6936]. MPLS does not accumulate incorrect state as a
consequence of label-stack corruption. A corrupt MPLS label results
in either packet discard or forwarding (and forgetting) of the packet
without accumulation of MPLS protocol state. Active monitoring of
MPLS-in-UDP traffic for errors is REQUIRED because the occurrence of
errors will result in some accumulation of error information outside
the MPLS protocol for operational and management purposes. This
design is in accordance with requirement 4 specified in Section 5 of
[RFC6936]. In addition, IPv6 traffic with a zero UDP checksum MUST
be actively monitored for errors by the network operator.

Operators SHOULD also deploy packet filters to prevent IPv6 packets
with a zero UDP checksum from escaping from the network due to
misconfiguration or packet errors. In addition, IPv6 traffic with a
zero UDP checksum MUST be actively monitored for errors by the
network operator.

Acknowledgments

Authors' Addresses

   Lars Eggert
   NetApp
   Sonnenallee 1
   Kirchheim 85551
   Germany

   Phone: +49 151 120 55791
   Email: lars@netapp.com
   URI: https://eggert.org/

   Godred Fairhurst
   University of Aberdeen
   Department of Engineering
   Fraser Noble Building
   Aberdeen AB24 3UE
   Scotland

   Email: gorry@erg.abdn.ac.uk
   URI: http://www.erg.abdn.ac.uk/

   Greg Shepherd
   Cisco Systems
   Tasman Drive
   San Jose
   United States of America

   Email: gjshep@gmail.com