Transport Area Working Group
Internet-Draft
Obsoletes: 5405 (if approved)
Intended status: Best Current Practice
Expires: June 18, 2015

L. Eggert
NetApp
G. Fairhurst
University of Aberdeen
G. Shepherd
Cisco Systems
December 15, 2014

UDP Usage Guidelines
draft-ietf-tsvwg-rfc5405bis-00

Abstract

The User Datagram Protocol (UDP) provides a minimal message-passing transport that has no inherent congestion control mechanisms. 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.

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, and middlebox traversal.

If published as an RFC, this document will obsolete RFC5405.

Status of This Memo

Table of Contents

1. Introduction

The User Datagram Protocol (UDP) [RFC0768] provides a minimal, unreliable, best-effort, message-passing transport to applications and other protocols (such as tunnels) that desire to operate over UDP (both 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 tear-down 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 link interface, which is often much greater than the available 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". 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, and middlebox traversal.

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 and broadcast, and applications that use UDP tunnels to support IP flows.

Finally, although this document specifically refers to applications that use 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 or control applications that choose to run directly over IP by registering their own IP protocol number with IANA. This document may 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. 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 [RFC0793], 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].

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 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 [RFC0896] 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. The 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 3.1.1 discusses congestion control options for applications that perform bulk transfers over UDP. Such applications can employ schemes that sample the path over several subsequent RTTs during which data is exchanged, in order 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 3.1.2 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 3.1.6 discusses congestion control considerations when UDP is used as a tunneling protocol. Section 4 provides additional recommendations for broadcast and multicast usage.

UDP applications may take advantage of Explicit Congestion Notification (ECN), providing that the application programming interface can support ECN and the congestion control can appropriately react to ECN-marked packets. [RFC6679] provides guidance on how to use ECN for UDP-based applications using the Real-Time Protocol (RTP).

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. 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 TCP-Friendly Rate Control (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 3.1.1, because TFRC already addresses them, but 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 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.

Finally, some bulk transfer applications may choose not to implement any congestion control mechanism and instead rely on transmitting across reserved path capacity. 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.

3.1.2. 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 3.1.1, 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 round-trip time (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. Applications SHOULD implement the algorithm specified in [RFC6298] to compute a smoothed RTT (SRTT) estimate. They SHOULD also detect packet loss and exponentially back their retransmission timer off when a loss event

occurs. When implementing this scheme, applications need to choose a sensible initial value for the RTT. This value SHOULD generally be as conservative as possible for the given application. TCP uses an initial value of 3 seconds [RFC6298], which is also RECOMMENDED as an initial value for UDP applications. SIP [RFC3261] and GIST [RFC5971] use an initial value of 500 ms, and initial timeouts that are shorter than this are likely problematic in many cases. It is also important to note that the initial timeout is not the maximum possible timeout -- the RECOMMENDED algorithm in [RFC6298] yields timeout values after a series of losses that are much longer than the initial value.

Some applications cannot maintain a reliable RTT estimate for a destination. The first case is that of applications that exchange too few UDP datagrams with a peer to establish a statistically accurate RTT estimate. Such applications MAY use a predetermined transmission interval that is exponentially backed-off when packets are lost. TCP uses an initial value of 3 seconds [RFC6298], which is also RECOMMENDED as an initial value for UDP applications. SIP [RFC3261] and 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.

A second class 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. The 3-second interval was chosen based on TCP's retransmission timeout when the RTT is unknown [RFC6298], and shorter values are likely problematic in many cases. Note that the sending rate in this case must be more conservative than in the two 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.

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 direction 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.

3.1.3. 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 has also been shown to improve the coexistence of TCP flows with other flows.

Even low data-volume UDP flows may benefit from rate control, 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.

3.1.4. QoS, Pre-Provisioned or Reserved Capacity

An application using UDP can use the differentiated services and integrated services QoS frameworks. These are usually 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.

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. For this reason, and to protect other applications sharing the same path, applications SHOULD deploy an appropriate circuit breaker, as described in Section 3.1.5. 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 used in networks within a controlled environment may be able to exploit network management functions to detect whether they are causing congestion, and react accordingly.

3.1.5. 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 [I-D.ietf-tsvwg-circuit-breaker]. 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. Later sections provide examples of cases where circuit breakers may or may not be desirable.

[I-D.ietf-tsvwg-circuit-breaker] provides guidance on the use of circuit breakers and examples of usage. The use of a circuit breaker in RTP is specified in [I-D.ietf-avtcore-rtp-circuit-breakers].

3.1.6. UDP Tunnels

One increasingly popular use of UDP is as a tunneling protocol, 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. 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 -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; 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 traffic is generally assumed to be congestion-controlled, i.e., it is assumed that the transport protocols generating IP-based traffic at the sender already employ mechanisms that are sufficient to address congestion on the path. Consequently, a tunnel carrying IP-based 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 to not be congestion-controlled, additional measures are RECOMMENDED in order to limit the impact of the tunneled traffic on other traffic sharing the path.

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.

   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. Because tunnels are usually bulk-transfer applications as far as

the intermediate routers are concerned, the guidelines in Section 3.1.1 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.

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, e.g., handling fragmentation, generating and responding to ICMP messages, etc. 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, and instead 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 MTU of 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 itself [RFC1191][RFC1981][RFC4821] to determine whether the path to a destination will support its desired message size without fragmentation.

Applications that do not follow this recommendation to do PMTU 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 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, 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.

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. 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 updates not only 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 are not a part of UDP. PLPMTUD therefore places additional design requirements on a UDP application that wishes to use this method.

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. 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 verify that their application handles such datagram duplication gracefully, and 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.

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. [RFC0793] 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.

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) 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. It therefore 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 are provided in Section 3.1 of [RFC6396].

Applications SHOULD enable UDP checksums. For IPv4, [RFC0768] permits the option to disable their use. The use of the UDP checksum was required when applications transmit UDP over IPv6 [RFC2460]. This requirement was updated in [RFC6395], but only for specific protocols and applications, and the implementation of the set of functions defined in [RFC6396] is then REQUIRED. These additional design requirements for using a zero IPv6 UDP checksum [RFC6396] are not present for IPv4, since the network-layer header validates information that is not protected for an IPv6 packet.

Applications that choose to disable UDP checksums when transmitting over IPv4 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. 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 a checksum coverage length value: at the sender, this specifies the intended checksum coverage, with the remaining unprotected part of the payload called the "error-insensitive part". 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. Applications using UDP-Lite therefore 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 may be negotiated by an application. UDP-Lite does not provide mechanisms to negotiate the checksum coverage between the sender and receiver.

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).

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 and endpoints.

3.5. Middlebox Traversal Guidelines

Network address translators (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 may 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 period of 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. 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-alives 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 network 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 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.

Sending keep-alives is not a substitute for implementing a mechanism to recover from broken sessions. Like all UDP datagrams, keep-alives 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-alives. 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-alives 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-alives 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 need to be designed understanding that there are many variants of middlebox behavior, and although UDP is connection-less, middleboxes often maintain state for each UDP flow. Using multiple 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). This has implications on applications that use multiple UDP flows in parallel, even on multiple ports Section 5.1.1.

4. Multicast UDP Usage Guidelines

This section complements Section 3 by providing additional guidelines that are applicable to multicast and broacast 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 [RFC0919]. 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 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]. Anycast senders 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. These MUST follow the same requirements as a tunnel carrying unicast data (see Section 3.1.6). 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 bi-laterally agreed upon path with reserved bandwidth, 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 mechanism are often not applicable to multicast distribution services, or simply do not scale to large multicast trees, since they require bi-directional 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.

Congestion control mechanisms for multicast may operate on longer timescales than for unicast (e.g., due to the higher group RTT of a heterogeneous group); appropriate methods are particularly for any multicast session were all or part of the multicast distribution tree spans an access network (e.g., a home gateway).

Multicast congestion control needs to consider 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. 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 there is no way at the transport layer to prevent a join message propagating to the next-hop router. A multicast congestion control method MAY therefore decide not to reduce the rate of the entire multicast group in response to a report received by a single receiver; instead it can decide to expel each congested receiver from the multicast group and to then distribute content to these congested receivers at a lower-rate using unicast congestion-control. Care needs to be taken when this action results in many flows being simultaneously transitioned, so that this does not result in excessive traffic exasperating congestion and potentially contributing to congestion collapse.

Some classes of multicast applications support real-time transmissions in which the quality of the transfer may be monitored at the receiver. Applications that detect a significant reduction in user quality SHOULD regard this as a congestion signal (e.g., to leave a group using layered multicast encoding).

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: 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 schemes 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 3.1.2 are also applicable to such 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 [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.

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.

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].

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 on 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 in this case. 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 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 [I-D.ietf-tsvwg-port-use].

A UDP sender SHOULD NOT use a zero source port value, and a UDP receiver should not bind to port zero. Applications SHOULD implement corresponding receiver 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). Although the source port value is often not directly used in

multicast applications, this should still be set to a random or pre-determined value.

The UDP port number fields have been used as a basis to design load-balancing solutions for IPv4. This approach has also been leveraged for IPv6 [RFC6438], but the IPv6 "flow label" [RFC6437]may also be used as a basis for entropy for load balancing. This use of the flow label for load balancing is consistent with the intended use, 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 were specified. Router vendors are encouraged to start using the 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.

5.1.1. 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 5-tuples). [RFC6335] recommends applications developers not to apply to IANA to be assigned multiple well-known ports (user or system). This 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 impact the network in several ways:

- 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.

- Using several flows at the same time may result in seeing different network characteristics for each flow. It can not 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 congestion control to react to congestion events, unless the congestion state is shared between all flows in a session.

Therefore, applications MUST NOT assume consistent behavior of middleboxes when multiple UDP flows are used; many devices respond differently as the number of ports used 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 3.1.4.

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, e.g., transient ICMP "unreachable" messages should not normally cause a communication abort.

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. Applications that respond to short requests with potentially large responses are vulnerable to amplification attacks, and SHOULD authenticate the sender before responding. The source IP address of a request is not a useful authenticator, because it can easily be spoofed.

One option of 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.

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 of securing UDP communications is through Datagram Transport Layer Security (DTLS) [RFC6347]. 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. The IETF standard for securing RTP [RFC3550] communication sessions over UDP is the Secure Real-time Transport Protocol (SRTP) [RFC3711]. 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 (especially when the intended endpoint is on the same link as the sender). This is a lightweight mechanism that 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. [I-D.ietf-tsvwg-circuit-breaker] 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.

7. Summary

This section summarizes the guidelines made in Sections 3 and 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 (TCP, SCTP, DCCP)	
SHOULD control rate of transmission	3.1
SHOULD perform congestion control over all traffic	
for bulk transfers,	3.1.1
SHOULD consider implementing TFRC	
else, SHOULD in other ways use bandwidth similar to TCP	
for non-bulk transfers,	3.1.2
SHOULD measure RTT and transmit max. 1 datagram/RTT	
else, SHOULD send at most 1 datagram every 3 seconds	
SHOULD back-off retransmission timers following loss	
for tunnels carrying IP Traffic,	3.1.6
SHOULD NOT perform congestion control	
for non-IP tunnels or rate not determined by traffic,	3.1.6

SHOULD perform congestion control	
SHOULD NOT send datagrams that exceed the PMTU, i.e., SHOULD discover PMTU or send datagrams < minimum PMTU; Specific application mechanisms are REQUIRED if PLPMTUD is used.	3.2
SHOULD handle datagram loss, duplication, reordering SHOULD be robust to delivery delays up to 2 minutes	3.3
SHOULD enable IPv4 UDP checksum SHOULD enable IPv6 UDP checksum; Specific application mechanisms are REQUIRED if a zero IPv6 UDP checksum is used.	3.4
else, MAY use UDP-Lite with suitable checksum coverage	3.4.1
SHOULD NOT always send middlebox keep-alives MAY use keep-alives when needed (min. interval 15 sec)	3.5
MUST check IP source address and, for client/server applications SHOULD send responses from src address matching request	5
SHOULD use standard IETF security protocols when needed	6

Table 1: Summary of recommendations

8. IANA Considerations

Note to RFC-Editor: please remove this entire section prior to publication.

This document raises no IANA considerations.

9. Acknowledgments

The middlebox traversal guidelines in Section 3.5 incorporate ideas from Section 5 of [I-D.ford-behave-app] by Bryan Ford, Pyda Srisuresh, and Dan Kegel.

10. References

10.1. Normative References

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

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

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

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

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.

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

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

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

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

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

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

10.2. Informative References

[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.

[I-D.ford-behave-app] Ford, B., "Application Design Guidelines for Traversal through Network Address Translators", draft-ford-behave-app-05 (work in progress), March 2007.

[I-D.ietf-avtcore-rtp-circuit-breakers] Perkins, C. and V. Singh, "Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions", draft-ietf-avtcore-rtp-circuit-breakers-08 (work in progress), December 2014.

[I-D.ietf-tsvwg-circuit-breaker] Fairhurst, G., "Network Transport Circuit Breakers", draft-ietf-tsvwg-circuit-breaker-00 (work in progress), September 2014.

[I-D.ietf-tsvwg-port-use] Touch, J., "Recommendations for Transport Port Number Uses", draft-ietf-tsvwg-port-use-06 (work in progress), November 2014.

[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.

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

[RFC0919] Mogul, J., "Broadcasting Internet Datagrams", STD 5, RFC 919, October 1984.

[RFC1112] Deering, S., "Host extensions for IP multicasting", STD 5, RFC 1112, August 1989.

[RFC1536] Kumar, A., Postel, J., Neuman, C., Danzig, P., and S. Miller, "Common DNS Implementation Errors and Suggested Fixes", RFC 1536, October 1993.

[RFC1546] Partridge, C., Mendez, T., and W. Milliken, "Host Anycasting Service", RFC 1546, November 1993.

[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, April 1998.

[RFC2675] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms", RFC 2675, August 1999.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[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, August 2000.

[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, January 2001.

[RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC 3124, June 2001.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3303] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A., and A. Rayhan, "Middlebox communication architecture and framework", RFC 3303, August 2002.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC3738] Luby, M. and V. Goyal, "Wave and Equation Based Rate Control (WEBRC) Building Block", RFC 3738, April 2004.

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

[RFC3819] Karn, P., 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, July 2004.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC4302] Kent, S., "IP Authentication Header", RFC 4302, December 2005.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

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

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

[RFC4342] Floyd, S., Kohler, E., and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)", RFC 4342, March 2006.

[RFC4607] Holbrook, H. and B. Cain, "Source-Specific Multicast for IP", RFC 4607, August 2006.

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

[RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, November 2007.

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

[RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, July 2007.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.

[RFC5082] Gill, V., Heasley, J., Meyer, D., Savola, P., and C. Pignataro, "The Generalized TTL Security Mechanism (GTSM)", RFC 5082, October 2007.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[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, August 2009.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

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

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[RFC5775] Luby, M., Watson, M., and L. Vicisano, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 5775, April 2010.

[RFC5971] Schulzrinne, H. and R. Hancock, "GIST: General Internet Signalling Transport", RFC 5971, October 2010.

[RFC5973] Stiemerling, M., Tschofenig, H., Aoun, C., and E. Davies, "NAT/Firewall NSIS Signaling Layer Protocol (NSLP)", RFC 5973, October 2010.

[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, August 2011.

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

[RFC6395] Gulrajani, S. and S. Venaas, "An Interface Identifier (ID) Hello Option for PIM", RFC 6395, October 2011.

[RFC6396] Blunk, L., Karir, M., and C. Labovitz, "Multi-Threaded Routing Toolkit (MRT) Routing Information Export Format", RFC 6396, October 2011.

[RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, November 2011.

[RFC6438] Carpenter, B. and S. Amante, "Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels", RFC 6438, November 2011.

[RFC6513] Rosen, E. and R. Aggarwal, "Multicast in MPLS/BGP IP VPNs", RFC 6513, February 2012.

[RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, August 2012.

[RFC6726] Paila, T., Walsh, R., Luby, M., Roca, V., and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport", RFC 6726, November 2012.

[RFC6807] Farinacci, D., Shepherd, G., Venaas, S., and Y. Cai, "Population Count Extensions to Protocol Independent Multicast (PIM)", RFC 6807, December 2012.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, October 2014.

[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. Revision Notes

Note to RFC-Editor: please remove this entire section prior to publication.

Changes in draft-eggert-tsvwg-rfc5405bis-01:

- o Added Greg Shepherd as a co-author, based on the multicast guidelines that originated with him.

Changes in draft-eggert-tsvwg-rfc5405bis-00 (relative to RFC5405):

- o The words "application designers" were removed from the draft title and the wording of the abstract was clarified abstract.

- o New text to clarify various issues and set new recommendations not previously included in RFC 5405. These include new recommendations for multicast, the use of checksums with IPv6, ECMP, recommendations on port usage, use of ECN, use of DiffServ, circuit breakers (initial text), etc.

draft-ietf-tsvwg-rfc5405bis-00 was adopted by the TSVWG (based on the above)

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
USA

EMail: gjshep@gmail.com