

Transport Area Working Group
Internet-Draft
Intended status: Best Current
Practice
Expires: May 22, 2008

L. Eggert
Nokia
G. Fairhurst
University of Aberdeen
November 19, 2007

UDP Usage Guidelines for Application Designers
draft-ietf-tsvwg-udp-guidelines-04

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on May 22, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2007).

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 upper-layer protocols that choose to use UDP as an Internet transport must employ mechanisms to prevent congestion collapse and establish some degree of fairness with concurrent traffic. This document provides guidelines on the use of

UDP for the designers of such applications and upper-layer protocols. Congestion control guidelines are a primary focus, but the document also provides guidance on other topics, including message sizes, reliability, checksums and middlebox traversal.

Table of Contents

1.	Introduction	3
2.	Terminology	4
3.	UDP Usage Guidelines	4
3.1.	Congestion Control Guidelines	5
3.2.	Message Size Guidelines	10
3.3.	Reliability Guidelines	10
3.4.	Checksum Guidelines	11
3.5.	Middlebox Traversal Guidelines	13
3.6.	Programming Guidelines	14
3.7.	ICMP Guidelines	16
4.	Security Considerations	16
5.	Summary	17
6.	IANA Considerations	18
7.	Acknowledgments	18
8.	References	19
8.1.	Normative References	19
8.2.	Informative References	20
	Authors' Addresses	23
	Intellectual Property and Copyright Statements	25

1. Introduction

The User Datagram Protocol (UDP) [[RFC0768](#)] provides a minimal, unreliable, best-effort, message-passing transport to applications and upper-layer protocols (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 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 messages 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. For example, an application that generates five 1500-byte UDP messages 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 message is carried in a single IP packet and is hence limited to a maximum payload of 65,507 bytes for IPv4 and 65,487 bytes for IPv6. The transmission of large IP packets usually requires IP

fragmentation. At least for IPv4, fragmentation decreases communication reliability and efficiency and should be avoided. One reason for this decrease in reliability is that many NATs and firewalls do not forward IPv4 fragments; other reasons are documented in [\[RFC4963\]](#). Some of the guidelines in [Section 3](#) describe how applications should determine appropriate message sizes.

This document provides guidelines to designers of applications that use UDP for unicast transmission. A special class of applications uses UDP for IP multicast transmissions. Congestion control, flow control or reliability for multicast transmissions is more difficult to establish than for unicast transmissions, because a single sender may transmit to multiple receivers across potentially very heterogeneous paths at the same time. Designing multicast applications requires expertise that goes beyond the simple guidelines given in this document. The IETF has defined a reliable multicast framework [\[RFC3048\]](#) and several building blocks to aid the designers of multicast applications, such as [\[RFC3738\]](#) or [\[RFC4654\]](#).

[2.](#) Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14](#), [RFC 2119](#) [\[RFC2119\]](#).

[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 Internet path 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](#)], SCTP [[RFC4960](#)] and SCTP-PR [[RFC3758](#)],

or DCCP [[RFC4340](#)] with its different congestion control types [[RFC4341](#)] [[RFC4342](#)] [[I-D.ietf-dccp-ccid4](#)].

If used correctly, these more fully-featured transport protocols are not as "heavyweight" as often claimed. 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 to interfere 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, for transfers that exchange more than a few messages.

3.1. Congestion Control Guidelines

If an application or upper-layer protocol chooses not to use a congestion-controlled transport protocol, it SHOULD control the rate at which it sends UDP messages 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 messages SHOULD perform congestion control over the aggregate traffic.

The remainder of this section discusses several approaches for this purpose. 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 messages 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. Finally, [Section 3.1.3](#) discusses congestion

control considerations when UDP is used as a tunneling protocol.

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.

Finally, in the past, the IETF has investigated integrated congestion control mechanisms that act on the traffic aggregate between two hosts, i.e., across all communication sessions active at a given time independent of specific transport protocols, such as the Congestion Manager [[RFC3124](#)]. Such mechanisms have failed to see deployment, but would otherwise also fulfill the congestion control requirements

in [[RFC2914](#)], because they provide congestion control for UDP sessions.

[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 small number of messages per RTT, SHOULD implement TCP-Friendly Rate Control (TFRC) [[RFC3448](#)], 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. TFRC is currently being updated [[I-D.ietf-dccp-rfc3448bis](#)], and application designers SHOULD always evaluate whether the latest published specification fits their needs. If an application implements TFRC, it need not follow the remaining guidelines in [Section 3.1](#), 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. [[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 chose 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 in the Internet. When the UDP traffic of such applications leaks out on 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 exchange only a small number of messages with a destination at any time 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 exchange only a small number of messages with a destination at any time SHOULD still control their transmission behavior by not sending more than one UDP message 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 [\[RFC2988\]](#) to compute a smoothed RTT (SRTT) estimate. A lost response from the peer SHOULD be treated as a very large RTT sample, instead of being ignored, in order to cause a sufficiently large (exponential) back-off. 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 [\[RFC2988\]](#), which is also RECOMMENDED as an initial value for UDP applications. SIP [\[RFC3261\]](#) and GIST [\[I-D.ietf-nsis-ntlp\]](#) 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 [\[RFC2988\]](#) 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 applications that exchange too few messages with a peer to establish a statistically accurate RTT estimate. Such applications MAY use a pre-determined transmission interval that is exponentially backed-off when packets are lost. TCP uses an initial value of 3 seconds [\[RFC2988\]](#), which is also RECOMMENDED as an initial value for UDP applications. SIP [\[RFC3261\]](#)

and GIST [\[I-D.ietf-nsis-ntlp\]](#) 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 message 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 [[RFC2988](#)], and shorter values are likely problematic in many cases. Note that the initial timeout interval 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 events, 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 independently detect and respond to congestion along both directions.

[3.1.3.](#) 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 messages and transmits them to another tunnel endpoint, which decapsulates the UDP messages 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 UDP transmissions.

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 tunneling scheme generates UDP traffic at a volume that corresponds to the volume of payload traffic carried within the tunnel. Second, whether the payload traffic is IP-based. This results in these possible cases:

1. Tunnel generates UDP traffic at a volume that corresponds to the volume of payload traffic, and the payload traffic is IP-based.

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. 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 is the case, for example, when link-layer protocols are encapsulated within UDP. 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. 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 not.

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 constrained 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.

than needed for many other UDP applications, because tunnels virtualize lower-layer components of the Internet, and the virtualized components need to correctly interact with the infrastructure at that layer. 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

Because IPv4 fragmentation lowers the efficiency and reliability of Internet communication [[RFC4963](#)], an application SHOULD NOT send UDP messages that result in IPv4 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 choose to not adapt their transmit message size SHOULD NOT send UDP messages that would result in IP datagrams that exceed the effective PMTU. In the absence of actual knowledge of the minimum MTU along the path, the effective PMTU depends upon the IP version used for transmission. It is the smaller of 576 bytes and the first-hop MTU for IPv4 [[RFC1122](#)] and 1280 bytes for IPv6 [[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 messages 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 MMS_S, 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 fragmentation even when

applications send messages that do not exceed the effective PMTU, so implementing PMTU discovery will still be beneficial in some cases.

[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 message duplication, i.e., an application may receive multiple copies of the same message. Application designers **SHOULD** verify that their application handles message duplication gracefully, and may consequently need to implement mechanisms to detect duplicates. Even if message reception triggers idempotent operations, applications may want to suppress duplicate messages to reduce load.

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

Finally, it is important to note that delay spikes can be very large. This can cause reordered packets to arrive many seconds after they were sent. The Internet protocol suite defines the Maximum Segment Lifetime (MSL) for TCP segments as 2 minutes [[RFC0793](#)]; this value applies to all IP datagrams, and hence also to UDP. The MSL is the maximum delay a packet should experience. Applications **SHOULD** be robust to the reception of delayed or duplicate packets that are received within this two-minute interval.

Applications that require reliable and ordered message delivery **SHOULD** choose an IETF standard transport protocol that provides these features. If this is not possible, it will need to implement a set of appropriate mechanisms itself.

[3.4.](#) Checksum Guidelines

The UDP header includes an optional, 16-bit ones-complement checksum that provides an integrity check. This results in a relatively weak protection from a coding point of view [[RFC3819](#)] and 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 UDP service.

The UDP checksum provides assurance 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.

Applications SHOULD enable UDP checksums, although [[RFC0768](#)] permits the option to disable their use. Applications that choose to disable UDP checksums when transmitting over IPv4 therefore MUST NOT make assumptions regarding the correctness of received data and MUST behave correctly when a message is received that was originally sent to a different destination or is otherwise corrupted. The use of the UDP checksum is MANDATORY when applications transmit UDP over IPv6 [[RFC2460](#)].

[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 MUST 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". 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 message 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.

The sending application SHOULD select the minimum checksum coverage to include all sensitive protocol headers. 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).

The receiver MUST set a minimum coverage threshold for incoming packets that is not smaller than the smallest coverage used by the sender. 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 may still experience packet loss, rather than corruption, 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 messages being treated the same as UDP messages, i.e., result in packet loss. Use of IP fragmentation can also prevent special treatment for UDP-Lite messages, and is another reason why applications SHOULD avoid IP fragmentation [Section 3.2](#).

[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 traffic 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 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 arriving after the per-flow state has timed out is dropped, as is other traffic arriving from the exterior.

Many applications that use UDP for communication operate across middleboxes without needing to employ additional mechanisms. One example is the 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 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. For NATs, [\[RFC4787\]](#) requires a state timeout of 2 minutes or longer. However, empirical evidence suggests that a significant fraction of the deployed middleboxes unfortunately uses shorter timeouts. The timeout of 15 seconds originates with the Interactive Connectivity Establishment (ICE) protocol [\[I-D.ietf-mmusic-ice\]](#). Applications that operate in more controlled network environments SHOULD investigate whether the environment they operate in allows them to use longer intervals, or whether it offers mechanisms to explicitly control middlebox state timeout durations, for example, using MIDCOM [\[RFC3303\]](#), NSIS [\[I-D.ietf-nsis-nslp-natfw\]](#), STUN [\[I-D.wing-behave-nat-control-stun-usage\]](#) or UPnP [\[UPNP\]](#).

It is important to note that sending keep-alives is not a substitute for implementing robust connection handling. Like all UDP messages, 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.

[3.6.](#) Programming Guidelines

The de facto standard application programming interface (API) for TCP/IP applications is the "sockets" interface [\[POSIX\]](#). Although this API was developed for UNIX in the early 1980s, a wide variety of non-UNIX operating systems also implements 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, the remainder of this section discusses these differences. [\[STEVENS\]](#) provides usage examples of the UDP sockets API.

UDP messages 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. When applications need to ensure that they receive packets from a particular source address, they **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 **MUST** ensure that it sends any UDP responses in reply to arriving UDP datagrams with an IP source address that matches the IP destination address of the datagram that carried the request.

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 IP source routing. 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. 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 data when the application issues them too frequently (i.e., when no new data has arrived) or not frequently enough (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.

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 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.

[3.7.](#) ICMP Guidelines

Applications can utilize ICMP error messages that the UDP layer passes up for a variety of purposes [[RFC1122](#)]. Applications SHOULD validate the information in the ICMP message payload, e.g., that the reported error corresponds to a UDP datagram that the application actually sent.

Any application response to ICMP error messages SHOULD be robust to temporary routing failures, i.e., transient ICMP "unreachable" messages should not normally cause a communication abort. Applications are RECOMMENDED to appropriately respond to ICMP messages generated in response to transmitted traffic. A correct response often requires context, such as local state about communication instances to each destination, that although readily available in connection-orientated transport protocols is not always maintained by UDP-based applications.

[4.](#) 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.

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) [[RFC4306](#)] 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. In order to be able to use IPsec, an application must

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) [[RFC4347](#)]. 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 of authenticating or encrypting UDP payloads exist, including other IETF standards, such as S/MIME [[RFC3851](#)] or PGP [[RFC4880](#)], security frameworks such as GSS-API [[RFC1964](#)], SASL [[RFC4422](#)] and EAP [[RFC3748](#)], as well as many non-IETF protocols. Out of these, S/MIME and PGP are likely to better suit less immediate and less ephemeral communications than typically the case for UDP applications, because they generally require public-key operations for each message.

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.

In terms of congestion control, [[RFC2309](#)] and [[RFC2914](#)] discuss the dangers of congestion-unresponsive flows to the Internet. This document provides guidelines to designers of UDP-based applications to congestion-control their transmissions, and does not raise any additional security concerns.

[5.](#) Summary

This section summarizes the guidelines made in [Section 3](#) and [Section 4](#) in a tabular format in Table 1 for easy referencing.

Recommendation	Section
MUST accommodate wide range of Internet path conditions SHOULD use a full-featured transport (TCP, SCTP, DCCP)	3
SHOULD control rate of transmission SHOULD perform congestion control over all traffic	3.1
for bulk transfers, SHOULD consider implementing TFRC else, SHOULD otherwise use bandwidth similar to TCP	3.1.1
for non-bulk transfers, SHOULD measure RTT and transmit 1 message/RTT else, SHOULD send at most 1 message every 3 seconds	3.1.2
SHOULD NOT send messages that exceed the PMTU, i.e., SHOULD discover PMTU or send messages < minimum PMTU	3.2
SHOULD handle message loss, duplication, reordering	3.3
SHOULD enable UDP checksum else, MAY use UDP-Lite with suitable checksum coverage	3.4 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	3.6
SHOULD use standard IETF security protocols when needed	4

Table 1: Summary of recommendations.

[6.](#) IANA Considerations

This document raises no IANA considerations.

[7.](#) Acknowledgments

Thanks to Paul Aitken, Mark Allman, Francois Audet, Stewart Bryant, Remi Denis-Courmont, Wesley Eddy, Sally Floyd, Jeffrey Hutzelman, Tero Kivinen, Philip Matthews, Joerg Ott, Colin Perkins, Carlos Pignataro, Pasi Sarolahti, Joe Touch and Magnus Westerlund for their comments on this document.

Eggert & Fairhurst

Expires May 22, 2008

[Page 18]

Internet-Draft

UDP Usage Guidelines

November 2007

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 Keegel.

[8.](#) References

[8.1.](#) Normative References

- [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.
- [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.
- [RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", [RFC 2988](#), November 2000.
- [RFC3448] Handley, M., Floyd, S., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", [RFC 3448](#), January 2003.
- [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.

[8.2](#). 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-dccp-ccid4]

Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion ID 4: TCP-Friendly Rate Control for Small Packets (TFRC-SP)", [draft-ietf-dccp-ccid4-00](#) (work in progress), October 2007.

[I-D.ietf-dccp-rfc3448bis]

Handley, M., "TCP Friendly Rate Control (TFRC): Protocol Specification", [draft-ietf-dccp-rfc3448bis-02](#) (work in progress), July 2007.

[I-D.ietf-mmusic-ice]

Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", [draft-ietf-mmusic-ice-19](#) (work in progress), October 2007.

[I-D.ietf-nsis-nslp-natfw]

Stiernerling, M., "NAT/Firewall NSIS Signaling Layer Protocol (NSLP)", [draft-ietf-nsis-nslp-natfw-15](#) (work in progress), July 2007.

[I-D.ietf-nsis-ntlp]

Schulzrinne, H. and R. Hancock, "GIST: General Internet Signalling Transport", [draft-ietf-nsis-ntlp-14](#) (work in progress), July 2007.

[I-D.wing-behave-nat-control-stun-usage]

Wing, D., Rosenberg, J., and H. Tschofenig, "Discovering, Querying, and Controlling Firewalls and NATs",

[draft-wing-behave-nat-control-stun-usage-05](#) (work in progress), October 2007.

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

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

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", [RFC 1964](#), June 1996.

- [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.
- [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.

- [RFC3738] Luby, M. and V. Goyal, "Wave and Equation Based Rate Control (WEBRC) Building Block", [RFC 3738](#), April 2004.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)",

[RFC 3748](#), June 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.
- [RFC3851] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", [RFC 3851](#), 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.
- [RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", [RFC 4306](#), 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.
- [RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", [RFC 4347](#), April 2006.
- [RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and

Security Layer (SASL)", [RFC 4422](#), June 2006.

- [RFC4654] Widmer, J. and M. Handley, "TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification", [RFC 4654](#), August 2006.
- [RFC4880] Callas, J., Donnerhackle, 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.
- [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.

Authors' Addresses

Lars Eggert
Nokia Research Center
P.O. Box 407
Nokia Group 00045
Finland

Phone: +358 50 48 24461
Email: lars.eggert@nokia.com
URI: http://research.nokia.com/people/lars_eggert/

Internet-Draft

UDP Usage Guidelines

November 2007

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/>

Internet-Draft

UDP Usage Guidelines

November 2007

Full Copyright Statement

Copyright (C) The IETF Trust (2007).

This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).