Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: May 17, 2017

G. Fairhurst
T. Jones
University of Aberdeen

November 13, 2016

Features of the User Datagram Protocol (UDP) and Lightweight UDP (UDP-Lite) Transport Protocols
draft-ietf-taps-transports-usage-udp-00

Abstract

This document describes how the User Datagram Protocol (UDP) and the Lightweight User Datagram Protocol (UDP-Lite) transport protocols expose services to applications and how an application can configure and use the features offered by the transport service. The document is intended as a contribution to the Transport Services (TAPS) working group to assist in analysis of the UDP and UDP-Lite transport interface.

Table of Contents

1. Terminology 2
2. Introduction 2
3. UDP and UDP-Lite Primitives 3
3.1. Primitives Provided by UDP 3
3.1.1. Excluded Primitives 8
3.2. Primitives Provided by UDP-Lite 9
4. Acknowledgements 10
5. IANA Considerations 10
6. Security Considerations 10
7. References 10
7.1. Normative References 10
7.2. Informative References 11
Appendix A. Revision Notes 13
Appendix B. Notes Based on Typical Usage 14
Appendix C. UDP Multicast 15
C.1. Multicast Primitives 15
Authors' Addresses 18

1. Terminology

This document uses common terminology defined in [I-D.ietf-taps-transports-usage]. This document also refers to the terminology of [RFC2119], but does not itself define new terms using this terminology.

2. Introduction

This document presents defined interactions between transport protocols and applications in the form of 'primitives' (function calls). Primitives can be invoked by an application or a transport protocol; the latter type is called an "event". The list of transport service features and primitives in this document is strictly based on the parts of protocol specifications that relate to what the protocol provides to an application using it and how the application interacts with it. It does not cover parts of a protocol that are explicitly stated as optional to implement.

This follows the methodology defined in [I-D.ietf-taps-transports-usage], specifically it provides the first pass of this process. It discusses the relevant RFC text describing primitives for each protocol. This also provides documentation that may help users of UDP and UDP-Lite.

3. UDP and UDP-Lite Primitives

This summarizes the relevant text parts of the RFCs describing the UDP and UDP-Lite protocols, focusing on what the transport protocols provide to the application and how the transport is used (based on abstract API descriptions, where they are available).

3.1. Primitives Provided by UDP

The User Datagram Protocol (UDP) [RFC0768] States: "This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks." It "provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism (..)".

The User Interface section of [RFC0768] specifies that the user interface to an application should be able to create receive ports, source and destination ports and addresses, and provide operations to receive data based on ports with an indication of source port and address. Operations should be provided that allows datagrams be sent specifying the source and destination ports and addresses to be sent.

UDP for IPv6 is defined by [RFC2460], and API extensions to support this in [RFC3493]. [RFC6935] and [RFC6936] defines an update to the UDP transport specified in RFC 2460. This enables use of a zero UDP checksum mode with a tunnel protocol, providing that the method satisfies the requirements in [RFC6936].

UDP offers only a basic transport interface. UDP datagrams may be directly sent and received, without exchanging messages between the endpoints to setup a connection (i.e., there is no handshake prior to communication). Using the sockets API, applications can receive packets from more than one IP source address on a single UDP socket. Common support allows specification of the local IP address, destination IP address, local port and destination port values. Any or all of these can be indicated, with defaults supplied by the local system when these are not specified. The local endpoint is set using the BIND call and set on the remote endpoint using the CONNECT call. The CLOSE function has local significance only. This does not impact the status of the remote endpoint.

UDP and UDP-Lite do not provide congestion control, retransmission, nor support to optimise fragmentation etc. This means that applications using UDP need to provide additional functions on top of the UDP transport API. This requires parameters to be passed through the API to control the network layer (IPv4 or IPv6). These additional primitives could be considered a part of the network layer

(e.g., control of the setting of the Don't Fragment flag on a transmitted datagram), but are nonetheless essential to allow a user of the UDP API to implement functions that are normally associated with the transport layer (such as probing for Path maximum transmission size). Although this adds complexity to the analysis of the API, this document includes such primitives.

[I-D.ietf-tsvwg-rfc5405bis] also states "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]."

The [POSIX] API offers mechanisms for an application to receive asynchronous data events at the socket layer. Calls such as poll, select or queue allow an application to be notified when data has arrived at a socket or a socket has flushed its buffers. It is possible to structure a callback-driven API to the network interface on top of these calls. There are protocols that allow a macro interface to network primitives, [RFC6458] describes implicit association setup for sending datagram messages using SCTP. Implicit connection setup allows an application to delegate connection life management to the transport API. The transport API uses protocol primitives to offer the automated service to the application via the socket API. By combining UDP primitives (CONNECT.UDP, SEND.UDP), a higher level API could offer a similar service.

Guidance on the use of services provided by UDP is provided in [I-D.ietf-tsvwg-rfc5405bis].

The following primitives are specified:

CONNECT: The CONNECT primitive allows the association of source and port sets to a socket to enable creation of a 'connection' for UDP traffic. This UDP connection allows an application to be notified of errors received from the network stack and provides a shorthand access to the send and receive primitives. Since UDP is itself connectionless, no datagrams are sent because this primitive is executed. A further connect call can be used to change the association to a source/port pair.

Two forms of usage may be identified for the CONNECT primitive:

1. bind(): A bind operation sets the local port, either implicitly, triggered by a send to operation on an unbound, unconnected socket using an ephemeral port. Or by an explicit bind to makes use of a configured or well-known port.

2. bind(); connect(): A bind operation followed by a CONNECT primitive. The bind operation establishes the use of a known local port for datagrams, rather than using an ephemeral port. The connect operation specifies a known address port combination to be used by default for future datagrams. This form is used either after receiving a datagram from an endpoint causing the creation of a connection or can be triggered by third party configuration or a protocol trigger (such as reception of a UDP Service Description Protocol, SDP [RFC4566], record).

LISTEN: The roles of a client and a server are often not appropriate for UDP, where connections can be peer-to-peer. The listening functions are performed using one of the forms of CONNECT primitive described above.

SEND: The SEND primitive hands over a provided number of bytes that UDP should send to the other side of a UDP connection in a UDP datagram. The primitive can be used by an application to directly send datagrams to an endpoint defined by an address/port pair. If a connection has been created, then the address/port pair is inferred from the current connection for the socket. A connection created on the socket will allow network errors to be returned to the application as a notification on the send primitive. Messages passed to the send primitive that cannot be sent atomically in a datagram will not be sent by the network layer, generating an error.

RECEIVE: The RECEIVE primitive allocates a receiving buffer to accommodate a received datagram. The primitive returns the number of bytes provided from a received UDP datagram. Section 4.1.3.5 of [RFC1122] states "When a UDP datagram is received, its specific-destination address MUST be passed up to the application layer."

DISABLE_CHECKSUM: The CHECKSUM function controls whether a sender disables the UDP checksum when sending datagrams. [RFC0768] and IPv6 [RFC6935] [RFC6936] [I-D.ietf-tsvwg-rfc5405bis]. When set it overrides the default UDP behaviour disabling the checksum on

sending. Section 4.1.3.4 of [RFC1122] states "An application MAY optionally be able to control whether a UDP checksum will be generated, but it MUST default to checksumming on."

REQUIRE_CHECKSUM: The REQUIRE_CHECKSUM function determines whether UDP datagrams received with a zero checksum are permitted or discarded. Section 4.1.3.4 of [RFC1122] states "An application MAY optionally be able to control whether UDP datagrams without checksums should be discarded or passed to the application." Section 3.1 of [RFC3828] requires that the checksum field is non-zero, and hence UDP-Lite need to discard all datagrams received with a zero checksum.

SET_IP_OPTIONS: The SET_IP_OPTIONS function enables a datagram to be sent with the specified IP options. Section 4.1.3.2 of[RFC1122] states that an "application MUST be able to specify IP options to be sent in its UDP datagrams, and UDP MUST pass these options to the IP layer."

GET_IP_OPTIONS: The GET_IP_OPTIONS function is a network-layer function that enables a receiver to read the IP options of a received datagram. Section 4.1.3.2 of[RFC1122] states that a UDP receiver "MUST pass any IP option that it receives from the IP layer transparently to the application layer".

SET_DF: The SET_DF function is a network-layer function that sets the Don't Fragment (DF) flag to be used in the field of an IP header of a packet that carries a UDP datagram. A UDP application should implement a method that avoids IP fragmentation (section 4 of [I-D.ietf-tsvwg-rfc5405bis]). It can use Packetization-Layer-Path MTU Discovery (PLPMTUD) [RFC4821] or Path MTU Discovery [RFC1191]. NOTE: In many other IETF transports (e.g. TCP) the transport provides the support needed to use DF, when using UDP, the application is responsible for the techniques needed to discover the path MTU, coordinating with the network layer.

GET_INTERFACE_MTU: The GET_INTERFACE_MTU function a network-layer function that indicates the largest unfragmented IP packet that may be sent. A UDP endpoint can subtract the size of all network and transport headers to determine the maximum size of unfragmented UDP payload. UDP applications should use this value as part of a method to avoid sending UDP datagrams that would result in IP packets that exceed the effective path maximum transmission unit (PMTU) allowed on the network path. The effective PMTU specified in Section 1 of [RFC1191] is equivalent to the "effective MTU for sending" specified in [RFC1122]. [RFC4821] states: "If PLPMTUD updates the MTU for a particular path, all Packetization Layer sessions that share the path

representation (as described in Section 5.2) SHOULD be notified to make use of the new MTU and make the required congestion control adjustments."

SET_TTL: The SET_TTL function a network-layer function that sets the hop limit (TTL field) to be used in the field of an IPv4 header of a packet that carries an UDP datagram. This is used to limit the scope of unicast datagrams. Section 3.2.2.4 of [RFC1122] states an "incoming Time Exceeded message MUST be passed to the transport layer".

GET_TTL: The GET_TTL function is a network-layer function that reads the value of the TTL field from the IPv4 header of a received UDP datagram. Section 3.2.2.4 of [RFC1122] states that a UDP receiver "MAY pass the received TOS up to the application layer" When used for applications such as the Generalized TTL Security Mechanism (GTSM) [RFC5082], this needs the UDP receiver API to pass the received value of this field to the application.

SET_IPV6_UNICAST_HOPS: The SET_IPV6_UNICAST_HOPS function is a network-layer function that sets the hop limit field to be used in the field of an IPv6 header of a packet that carries a UDP datagram. For IPv6 unicast datagrams, this is functionally equivalent to the SET_TTL IPv4 function.

GET_IPV6_UNICAST_HOPS: The GET_IPV6_UNICAST_HOPS function is a network-layer function that reads the value from the hop count field in the IPv6 header from the IP header information of a received UDP datagram. For IPv6 unicast datagrams, this is functionally equivalent to the GET_TTL IPv4 function.

SET_DSCP: The SET_DSCP function is a network-layer function that sets the DSCP (or legacy TOS) value to be used in the field of an IP header of a packet that carries a UDP Datagram. Section 2.4 of [RFC1122] states that "Applications MUST select appropriate TOS values when they invoke transport layer services, and these values MUST be configurable.". The application should be able to change the TOS during the connection lifetime, and the TOS value should be passed to the IP layer unchanged. Section 4.1.4 of [RFC1122] also states that on reception the "UDP MAY pass the received TOS value up to the application layer". [RFC2475] [RFC3260] replaces this field in the IP Header assigning the six most significant bits to carry the Differentiated Services Code Point (DSCP) field. Preserving the intention of [RFC1122] to allow the application to specify the "Type of Service", this should be interpreted to mean that an API should allow the application to set the DSCP. Section 3.1.6 of [I-D.ietf-tsvwg-rfc5405bis] describes the way UDP applications should use this field. Normally a UDP socket will

assign a single DSCP value to all Datagrams in a flow, but it is allowed to use different DSCP values for datagrams within the same flow in some cases, as described in [I-D.ietf-tsvwg-rfc5405bis]. Guidelines for WebRTC that illustrate this use are provided in [RFC7657].

SET_ECN: The SET_ECN function is a network-layer function that sets the ECN field in the IP Header of a UDP Datagram. When use of the TOS field was redefined [RFC3260], 2 bits of the field were assigned to support Explicit Congestion Notification (ECN) [RFC3168]. Section 3.1.5 [I-D.ietf-tsvwg-rfc5405bis] describes the way UDP applications should use this field. NOTE: In many other IETF transports (e.g. TCP) the transport provides the support needed to use ECN, when using UDP, the application itself is responsible for the techniques needed to use ECN.

GET_ECN: The GET_ECN function is a network-layer function that returns the value of the ECN field in the IP Header of a received UDP Datagram. Section 3.1.5 [I-D.ietf-tsvwg-rfc5405bis] states that a UDP receiver "MUST check the ECN field at the receiver for each UDP datagram that it receives on this port", requiring the UDP receiver API to pass to pass the received ECN field up to the application layer to enable appropriate congestion feedback.

ERROR_REPORT The ERROR_REPORT event informs an application of "soft errors", including the arrival of an ICMP or ICMPv6 error message. Section 4.1.4 of [RFC1122] states "UDP MUST pass to the application layer all ICMP error messages that it receives from the IP layer." For example, this event is required to implement ICMP-based Path MTU Discovery [RFC1191] [RFC1981].

CLOSE: The close primitive closes a connection. No further datagrams may be sent/received. Since UDP is itself connectionless, no datagrams are sent because this command is executed.

3.1.1. Excluded Primitives

Section 3.4 of [RFC1122] also describes "GET_MAXSIZES: - replaced, GET_SRCADDR (Section 3.3.4.3) and ADVISE_DELIVPROB:". These mechanisms are no longer used. It also specifies use of the Source Quench ICMP message, which has since been deprecated [RFC6633]. The IPV6_V6ONLY function defined in Section 5.3 of [RFC3493] restricts the use of information from the name resolver to only allow communication of AF_INET6 sockets to use IPv6 only. This is not considered part of the transport service.

3.2. Primitives Provided by UDP-Lite

The Lightweight User Datagram Protocol (UDP-Lite) [RFC3828] provides similar services to UDP. It changed the semantics of the UDP "payload length" field to that of a "checksum coverage length" field. UDP-Lite requires the pseudo-header checksum to be computed at the sender and checked at a receiver. Apart from the length and coverage changes, UDP-Lite is semantically identical to UDP.

The sending interface of UDP-Lite differs from that of UDP by the addition of a single (socket) option that communicates the checksum coverage length. This specifies the intended checksum coverage, with the remaining unprotected part of the payload called the "error-insensitive part".

The receiving interface of UDP-Lite differs from that of UDP by the addition of a single (socket) option that specifies the minimum acceptable checksum coverage.

The UDP-Lite Management Information Base (MIB) further defines the checksum coverage method [RFC5097]. Guidance on the use of services provided by UDP-Lite is provided in [I-D.ietf-tsvwg-rfc5405bis].

UDP-Lite requires use of the UDP or UDP-Lite checksum, and hence it is not permitted to use the "DISABLE_CHECKSUM:" function to disable use of a checksum, nor is it possible to disable receiver checksum processing using the "REQUIRE_CHECKSUM:" function . All other primitives and functions for UDP are permitted.

In addition, the following are defined:

SET_CHECKSUM_COVERAGE: The SET_CHECKSUM_COVERAGE function sets the coverage area for a sent datagram. UDP-Lite traffic uses this primitive to set the coverage length provided by the UDP checksum. Section 3.3 of [RFC5097] states that "Applications that wish to define the payload as partially insensitive to bit errors ... Should do this by an explicit system call on the sender side." The default is to provide the same coverage as for UDP.

SET_MIN_COVERAGE The SET_MIN_COVERAGE function sets the minimum a acceptable coverage protection for received datagrams. UDP-Lite traffic uses this primitive to set the coverage length that is checked on receive (section 1.1 of [RFC5097] describes the corresponding MIB entry as udpliteEndpointMinCoverage). Section 3.3 of [RFC3828] states that "applications that wish to receive payloads that were only partially covered by a checksum should inform the receiving system by an explicit system call".

The default is to require only minimal coverage of the datagram payload.

4. Acknowledgements

This work was partially funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT). The views expressed are solely those of the author(s). Thanks to all who have commented or contributed, including Joe Touch, Ted Hardie, Aaron Falk.

5. IANA Considerations

This memo includes no request to IANA.

If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

6. Security Considerations

Security considerations for the use of UDP and UDP-Lite are provided in the referenced RFCs. Security guidance for application usage is provide in the UDP-Guidelines [I-D.ietf-tsvwg-rfc5405bis].

7. References

7.1. Normative References

[I-D.ietf-taps-transports-usage]
Welzl, M., Tuexen, M., and N. Khademi, "On the Usage of Transport Service Features Provided by IETF Transport Protocols", draft-ietf-taps-transports-usage-01 (work in progress), July 2016.

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

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

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

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

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <http://www.rfc-editor.org/info/rfc2460>.

[RFC2553] Gilligan, R., Thomson, S., Bound, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 2553, DOI 10.17487/RFC2553, March 1999, <http://www.rfc-editor.org/info/rfc2553>.

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

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

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

[RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", RFC 6935, DOI 10.17487/RFC6935, April 2013, <http://www.rfc-editor.org/info/rfc6935>.

7.2. Informative 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.

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

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, DOI 10.17487/RFC1981, August 1996, <http://www.rfc-editor.org/info/rfc1981>.

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

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

[RFC3678] Thaler, D., Fenner, B., and B. Quinn, "Socket Interface Extensions for Multicast Source Filters", RFC 3678, DOI 10.17487/RFC3678, January 2004, <http://www.rfc-editor.org/info/rfc3678>.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <http://www.rfc-editor.org/info/rfc4566>.

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

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

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

[RFC5790] Liu, H., Cao, W., and H. Asaeda, "Lightweight Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Version 2 (MLDv2) Protocols", RFC 5790, DOI 10.17487/RFC5790, February 2010, <http://www.rfc-editor.org/info/rfc5790>.

[RFC6458] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)", RFC 6458, DOI 10.17487/RFC6458, December 2011, <http://www.rfc-editor.org/info/rfc6458>.

[RFC6633] Gont, F., "Deprecation of ICMP Source Quench Messages", RFC 6633, DOI 10.17487/RFC6633, May 2012, <http://www.rfc-editor.org/info/rfc6633>.

[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, DOI 10.17487/RFC6936, April 2013, <http://www.rfc-editor.org/info/rfc6936>.

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

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

Appendix A. Revision Notes

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

Individual draft -00:

- This is the first version. Comments and corrections are welcome directly to the authors or via the IETF TAPS working group mailing list.

Individual draft -01:

- Includes ability of a UDP receiver to disallow zero checksum datagrams.
- Fixes to references and some connect on UDP usage.

Individual draft -02:

- Fixes to address issues noted by WG.
- Completed Multicast section to specify modern APIs.
- Noted comments on API usage for UDP.
- Feedback from various reviewers.

Individual draft -03:

- Removes pass 2 and 3 of the TAPS analysis from this revision. These are expected to be incorporated into a combined draft of the TAPS WG.

- Fixed Typos.

-

TAPS WG draft -00:

- Expected to progfress with draft-ietf-taps-transports-usage of the TAPS WG.

-

Appendix B. Notes Based on Typical Usage

This appendix contains notes to assist in a later revision.

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 use of this API is discussed in the RFC series in [I-D.ietf-tsvwg-rfc5405bis].

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.

This section provides notes on some topics relating to implemented UDP APIs.

A UDP application can use the recv() and send() POSIX functions as well as the recvfrom() and sendto() and recvmsg and sendmsg() functions.

SO_REUSEADDR specifies that the rules used in validating addresses supplied to bind() should allow reuse of local addresses.

SO_REUSEPORT specifies that the rules used in validating ports supplied to bind() should allow reuse of a local port

Accessing TTL From applications: If the IP_RECVTTL option is enabled on a SOCK_DGRAM socket, the recvmsg(2) call will return the IP TTL (time to live) field for a UDP datagram. The msg_control field in the msghdr structure points to a buffer that contains a cmsghdr structure followed by the TTL.

Appendix C. UDP Multicast

UDP and UDP-Lite Multicast may be considered in later versions of this document. This appendix contains notes to assist in this later revision.

A host must request the ability to broadcast before it can send/receive ipv4 broadcast traffic. A host must become a member of a multicast group at the network layer before it can receive datagrams sent to the group.

C.1. Multicast Primitives

UDP and UDP-Lite support IPv4 broadcast and IPv4/IPv6 Multicast. Use of multicast requires additional functions at the transport API that must be called to coordinate operation of the IPv4 and IPv6 network layer protocols.

Guidance on the use of UDP and UDP-Lite for multicast services is provided in [I-D.ietf-tsvwg-rfc5405bis].

The following are defined:

JoinLocalGroup: 1 of [RFC3493] provides a function that allows joining of a local IPv4 multicast group.

IPV6_MULTICAST_IF: Section 5.2 of [RFC2553] states that this sets the interface to use for outgoing multicast packets.

IP_MULTICAST_TTL: This sets the hop limit to use for outgoing multicast packets. This is used to limit scope of multicast datagrams. When used for applications such as GTSM, this needs the UDP receiver API to pass the received value of this field to the application. (This is equivalent to IPV6_MULTICAST_HOPS for IPv6 multicast and TTL/IPV6_UNICAST_HOPS for unicast datagrams).

IPV6_MULTICAST_HOPS: Section 5.2 of [RFC2553] states that this sets the hop limit to use for outgoing multicast packets. When used for applications such as GTSM, this needs the UDP receiver API to pass the received value of this field to the application. (This is equivalent to IP_MULTICAST_TTL for IPv4 multicast and TTL/IPV6_UNICAST_HOPS for unicast datagrams).

IPV6_MULTICAST_LOOP: Section 5.2 of [RFC2553] states that this sets whether a copy of a datagram is looped back by the IP layer for local delivery when the datagram is sent to a group to which the sending host itself belongs).

IPV6_JOIN_GROUP: Section 5.2 of [RFC2553] provides a function that allows joining of an IPv6 multicast group.

SIOCGIPMSFILTER: Section 8.1 of [RFC3678] provides a function that allows reading the multicast source filters.

SIOCSIPMSFILTER: Section 8.1 of [RFC3678] provides a function that allows setting/modifying the multicast source filters.

IPV6_LEAVE_GROUP: Section 5.2 of [RFC2553] provides a function that allows leaving of a multicast group.

LeaveHostGroup: Section 7.1 of [RFC3493] provides a function that allows joining of an IPv4 multicast group.

LeaveLocalGroup: Section 7.1 of [RFC3493] provides a function that allows joining of a local IPv4 multicast group.

Section 4.1.1 of [RFC3678] updates the interface to add support for Multicast Source Filters (MSF) to IGMPv3 for Any Source Multicast (ASM):

This identifies three sets of API functionality:

1. IPv4 Basic (Delta-based) API. "Each function call specifies a single source address which should be added to or removed from the existing filter for a given multicast group address on which to listen."

2. IPv4 Advanced (Full-state) API. "This API allows an application to define a complete source-filter comprised of zero or more source addresses, and replace the previous filter with a new one."

3. Protocol-Independent Basic MSF (Delta-based) API

4. Protocol-Independent Advanced MSF (Full-state) API

It specifies the following primitives:

IP_ADD_MEMBERSHIP: This is used to join an ASM group.

IP_BLOCK_SOURCE: This is a MSF that can be used to block data from a given multicast source to a given group for ASM or SSM.

IP_UNBLOCK_SOURCE: This updates an MSF to undo a previous call to IP_UNBLOCK_SOURCE for ASM or SSM.

IP_DROP_MEMBERSHIP: This is used to leave an ASM or SSM group. (In SSM this drops all sources that have been joined for a particular group and interface. The operations are the same as if the socket had been closed.)

Section 4.1.2 of [RFC3678] updates the interface to add Multicast Source Filter (MSF) support for IGMPv3 with Any Source Multicast (ASM) using IPv4:

IP_ADD_SOURCE_MEMBERSHIP: This is used to join an SSM group.

IP_DROP_SOURCE_MEMBERSHIP: This is used to leave an SSM group.

Section 4.1.2 of [RFC3678] defines the Advanced (Full-state) API:

setipv4sourcefilter This is used to join an IPv4 multicast group, or to enable multicast from a specified source.

getipv4sourcefilter: This is used to leave an IPv4 multicast group, or to filter multicast from a specified source.

Section 5.1 of [RFC3678] specifies Protocol-Independent Multicast API functions:

MCAST_JOIN_GROUP This is used to join an ASM group.

MCAST_JOIN_SOURCE_GROUP This is used to join an SSM group.

MCAST_BLOCK_SOURCE: This is used to block a source in an ASM group.

MCAST_UNBLOCK_SOURCE: This removes a previous MSF set by MCAST_BLOCK_SOURCE:

MCAST_LEAVE_GROUP: This leaves a SSM group.

MCAST_LEAVE_GROUP: This leaves a ASM or SSM group.

Section 5.2 of [RFC3678] specifies the Protocol-Independent Advanced MSF (Full-state) API applicable for both IPv4 and IPv6 multicast:

setsourcefilter This is used to join an IPv4 or IPv6 multicast group, or to enable multicast from a specified source.

getsourcefilter: This is used to leave an IPv4 or IPv6 multicast group, or to filter multicast from a specified source.

Section 7.2 of [RFC5790] updates the interface to specify support for Lightweight IGMPv3 (LW_IGMPv3) and MLDv2.

According to the MSF API definition [RFC3678], "an LW-IGMPv3 host should implement either the IPv4 Basic MSF API or the Protocol-Independent Basic MSF API, and an LW-MLDv2 host should implement the Protocol- Independent Basic MSF API. Other APIs, IPv4 Advanced MSF API and Protocol-Independent Advanced MSF API, are optional to implement in an LW-IGMPv3/LW-MLDv2 host."

Authors' Addresses

Godred Fairhurst
University of Aberdeen
School of Engineering
Fraser Noble Building
Fraser Noble Building Aberdeen AB24 3UE
UK

Email: gorry@erg.abdn.ac.uk

Tom Jones
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen AB24 3UE
UK

Email: tom@erg.abdn.ac.uk