Transport and Services Working Group
Internet-Draft
Intended status: Informational
Expires: 22 October 2026

M. Duke
Google

20 April 2026

Configuring UDP Sockets for ECN for Common Platforms
draft-ietf-tsvwg-udp-ecn-06

Abstract

Explicit Congestion Notification (ECN) applies to all transport protocols in principle. However, it had limited deployment for UDP until QUIC became widely adopted. As a result, documentation of UDP socket APIs for ECN on various platforms is sparse. This document records the results of experimenting with these APIs in order to get ECN working on UDP for Chromium on Apple, Linux, and Windows platforms.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://tsvwg.github.io/udp-ecn/draft-ietf-tsvwg-udp-ecn.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-tsvwg-udp-ecn/.

Discussion of this document takes place on the Transport and Services Working Group Working Group mailing list (mailto:tsvwg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tsvwg/. Subscribe at https://www.ietf.org/mailman/listinfo/tsvwg/.

Source for this draft and an issue tracker can be found at https://github.com/tsvwg/udp-ecn.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on 22 October 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 3
2. Conventions and Definitions 4
3. Receiving ECN codepoints 4
  3.1. Setting the socket to report incoming ECN codepoints . . 4
    3.1.1. Linux, Apple, and FreeBSD 4
    3.1.2. Windows 5
  3.2. Retrieving ECN codepoints on incoming packets 5
    3.2.1. Linux 5
    3.2.2. Apple and FreeBSD 6
    3.2.3. Windows 6
4. Sending ECN codepoints 6
  4.1. On a per-socket basis 7
    4.1.1. Apple, FreeBSD, and Linux 7
    4.1.2. Windows 7
  4.2. On a per-packet basis 7
    4.2.1. Apple, FreeBSD, and Linux 7
    4.2.2. Windows 8
5. Security Considerations 8
6. IANA Considerations 8
7. Informative References 8
Acknowledgments 10
Author's Address 10

1. Introduction

[RFC3168] defines a two-bit field in the IP header for Explicit Congestion Notification (ECN), which provides network feedback to endpoint congestion controllers. This has historically mostly been relevant to TCP ([RFC9293]), where any incoming ECN codepoints are internally consumed by the kernel, and therefore imply no application interface except enabling and disabling the capability.

The Stream Control Transport Protocol (SCTP) ([RFC9260]) has long supported ECN in its design. SCTP is sometimes carried over DTLS and UDP ([RFC8261]). In principle, user-space implementers might have leveraged UDP ECN APIs to deliver ECN codepoints between SCTP and the UDP socket. At the time of publication, the TSV Working Group is not aware of any such efforts.

[RFC6679] defines ECN over RTP over UDP. The Working Group is aware of a research implementation, but cannot confirm any commercial deployments.

However, QUIC [RFC9000] runs over UDP and has seen wider deployment than SCTP. The Low Latency, Low Loss, Scalable Throughput (L4S) experiment ([RFC9330]) and QUIC have combined to increase interest in ECN over UDP.

The Chromium Projects ([CHROMIUM]) provide a widely-deployed protocol library that includes QUIC. An effort to provide ECN support for QUIC on the many platforms on which Chromium is deployed revealed that many ECN-related UDP socket interfaces are poorly documented.

This informational document provides a record of that experience, to encourage further support for ECN in other QUIC implementations, and indeed any consumer of ECN codepoints that operates over UDP. It is not a standards-track document and does not bind platforms to any API, or suggest any such API.

Many socket APIs continue to reference the "ToS (Type of Service) byte", including the IP_TOS label, even though [RFC2474] obsoleted that in 1998. That 8-bit field now contains a 6-bit Differentiated Services Code Point (DSCP) and the 2-bit ECN field.

This document focuses on the APIs for the C and C++ languages. Other languages are likely to have different syntax and capabilities.

The "ecn-examples" code repository ([EXAMPLES]) is extremely compact code that can verify the information in this document.

This document addresses access to the ECN field in the IP header via socket APIs. It does not address UDP transport-layer options [RFC9868], which are a separate extension mechanism operating at a different layer.

2. Conventions and Definitions

This document is not a general tutorial on UDP socket programming, and assumes familiarity with basic socket concepts like binding, socket options, and common system error codes.

Throughout this document, "Apple" refers to both macOS and iOS.

3. Receiving ECN codepoints

Network devices can change the ECN codepoint in the IP header. Since this feedback is required at the packet sender, the packet receiver needs to extract this codepoint from the UDP socket in order to report to the sender.

There are two components to this: setting the socket to report incoming ECN marks, and retrieving the ECN codepoint for each incoming packet.

Note that Apple platforms additionally provide a framework for network connections that allows receiving ECN flags when using UDP without traditional socket option semantics. When sending or receiving UDP datagrams, IP protocol metadata carries ECN information in both directions. See [APPLE-NETWORK-FRAMEWORK].

3.1. Setting the socket to report incoming ECN codepoints

3.1.1. Linux, Apple, and FreeBSD

To receive ECN codepoints, applications set a socket option to true using a setsockopt() call.

On all platforms, IPv4 sockets require the IPPROTO_IP-level socket option with name IP_RECVTOS to be set.

On all platforms, IPv6 sockets require the IPPROTO_IPV6-level socket option with name IPV6_RECVTCLASS to be set. If the IPv6 socket is not IPv6 only, on Linux hosts it is required to also set the IPPROTO_IP-level socket option IP_RECVTOS to receive ECN codepoints for UDP/IPv4 packets.

At the time of writing, an example implementation can be found at [CHROMIUM-POSIX].

3.1.2. Windows

Windows documentation recommends using the function WSASetRecvIPEcn() to enable ECN codepoint reporting regardless of the IP version. This function dates to Windows 10 Build 20348, according to [WINDOWS-DOC].

However, this can also be accomplished by calling setsockopt() and using options of level IPPROTO_IP and name IP_RECVECN for IPv4, and IPPROTO_IPV6 and IPV6_RECVECN for IPv6. These options are documented at [WINDOWS-SOCKOPT].

For IPv6 sockets which are not IPv6 only, WSASetRecvIPEcn() will not enable ECN reporting for IPv4. This requires a separate setsockopt() call using the IP_RECVECN option.

If a socket is bound to a IPv4-mapped IPv6 address (i.e. it is of the format ::ffff:<IPv4 address>), calls to WSASetRecvIpEcn() return error EINVAL. These sockets should instead use an explicit setsockopt() call to set IP_RECVECN.

At the time of writing, an example implementation can be found at [CHROMIUM-WINDOWS].

3.2. Retrieving ECN codepoints on incoming packets

All platforms described in this document require the use of a recvmsg() call to read data from the socket to retrieve the ECN codepoint, because that information is provided as ancillary data. Those platforms all return zero or more "cmsg"s that contain requested information about the arriving packet.

Examples of the technique described below can be found at [CHROMIUM-POSIX] and [CHROMIUM-WINDOWS].

3.2.1. Linux

If a UDP/IPv4 message is received, Linux will include a cmsg of level IPPROTO_IP and type IP_TOS. The cmsg data contains an unsigned char. This applies to IPv4 sockets and IPv6 socket, which are not IPv6 only.

If a UDP/IPv6 message is received, Linux will include a cmsg of level IPPROTO_IPV6 and type IPV6_TCLASS. The cmsg data contains an int. This applies to IPv6 sockets.

The cmsg data contains the entire IP header byte, which includes the DSCP and the ECN codepoint. The ECN codepoint constitutes the two least-significant bits of this byte.

The same applies to the Linux-specific recvmmsg() call.

3.2.2. Apple and FreeBSD

If a UDP/IPv4 message is received on an IPv4 socket, the ancillary data will contain a cmsg of level IPPROTO_IP and type IP_RECVTOS. The cmsg data contains an unsigned char.

If a UDP/IPv6 or UDP/IPv4 message is received on an IPv6 socket, the ancillary data will contain a cmsg of level IPPROTO_IPV6 and type IPV6_TCLASS. The cmsg data contains an int.

The cmsg data contains the entire IP header byte, which includes the DSCP and the ECN codepoint. The ECN codepoint constitutes the two least-significant bits of this byte.

3.2.3. Windows

If the incoming packet is UDP/IPv4, the socket will include a cmsg of level IPPROTO_IP and type IP_ECN. The cmsg data contains an int.

If the incoming packet is UDP/IPv6, the socket will include a cmsg of level IPPROTO_IPV6 and type IPV6_ECN. The cmsg data contains an int.

The cmsg data solely consists of the ECN codepoint, and requires no further bitwise operations.

4. Sending ECN codepoints

Existing ECN specifications ([RFC3168], [RFC9330]} envision a particular connection consistently sending the same ECN codepoint. It might transition that marking after successfully completing a handshake, recognizing the path or the peer do not support ECN, or transitioning to a new path. Therefore, using a socket option to configure a consistent marking is generally more resource- efficient.

However, some server designs receive all incoming packets on a single socket. As the many connections that constitute this packet stream may have different support for ECN, it is suitable to provide the ECN codepoint on a per-packet basis.

Note that Apple platforms additionally provide a framework for network connections that allows sending ECN flags when using UDP without traditional socket option semantics. When sending or receiving UDP datagrams, IP protocol metadata carries ECN information in both directions. See [APPLE-NETWORK-FRAMEWORK].

4.1. On a per-socket basis

4.1.1. Apple, FreeBSD, and Linux

For sending UDP/IPv4 packets on an IPv4 socket, Apple, FreeBSD, and Linux platforms allow the outgoing ECN codepoint to be configured by using the IPPROTO_IP-level socket option with name IP_TOS. The value has the type int.

For sending UDP/IPv6 packets on an IPv6 socket, Apple, FreeBSD, and Linux platforms allow the outgoing ECN codepoint to be configured by using the IPPROTO_IPV6-level socket option with name IPV6_TCLASS. The value has the type int.

For sending UDP/IPv4 packets on an IPv6 socket, Linux platforms allow the the outgoing ECN codepoint to be configured by using the IPPROTO_IP-level socket option with name IP_TOS.

For sending UDP/IPv4 packets on an IPv6 socket, Apple and FreeBSD platforms allow the outgoing ECN codepoint to be configured by using the IPPROTO_IPV6-level socket option with name IPV6_TCLASS.

Except for Apple platforms, this setsockopt() call also sets the Differentiated Services Code Point (DSCP) that make up the rest of the header byte. Applications making this call ought to preserve any existing DSCP setting, which might require an additional getsockopt() call, to avoid overriding commands set by other code in the stack. If there are multiple threads making changes to this byte, further safeguards will be necessary.

An example of the technique described above can be found at [CHROMIUM-POSIX].

4.1.2. Windows

At the time of this writing, Windows does not provide a way to configure marking on a per-socket basis.

4.2. On a per-packet basis

Packets can be individually marked with ECN codepoints using the ancillary data that accompanies a sendmsg() call.

4.2.1. Apple, FreeBSD, and Linux

For sending UDP/IPv4 packets on an IPv4 socket, Apple, FreeBSD, and Linux use a cmsg with level IPPROTO_IP and type IP_TOS. On Apple and Linux the type of data is int and for FreeBSD it is unsigned char.

For sending UDP/IPv6 packets on an IPv6 socket, Apple, FreeBSD, and Linux use a cmsg with level IPPROTO_IPV6 and type IPV6_TCLASS. The type of the data is int.

For sending UDP/IPv4 packets on an IPv6 socket, Linux requires a cmsg with level IPPROTO_IP and type IP_TOS. Apple and FreeBSD accept a cmsg with level IPPROTO_IPV6 and type IPV6_TCLASS.

The same applies to the Linux-specific sendmmsg() call.

4.2.2. Windows

Windows uses a cmsg with level IPPROTO_IP and type IP_ECN for IPv4 packets.

Windows uses a cmsg with level IPPROTO_IPV6 and type IPV6_ECN for IPv6 packets.

An example of the technique described above can be found at [CHROMIUM-WINDOWS].

5. Security Considerations

The security implications of ECN are documented in [RFC3168] and [RFC9330]. This document is a guide to enabling these capabilities, which incurs no additional security considerations.

Note that implementing ECN capabilities on some platforms, but not others, can help peers identify the operating system in use by a host, which can have privacy implications. This document aims to mitigate that possibility.

6. IANA Considerations

This document has no IANA actions.

7. Informative References

[APPLE-NETWORK-FRAMEWORK]
"NWProtocolIP.Metadata", n.d.,
<https://developer.apple.com/documentation/network/nwprotocolip/metadata>.

[CHROMIUM] "The Chromium Projects", n.d.,
<https://www.chromium.org/chromium-projects/>.

[CHROMIUM-POSIX]
"udp_socket_posix.cc", n.d.,
<https://source.chromium.org/chromium/chromium/
src/+/main:net/socket/udp_socket_posix.cc>.

[CHROMIUM-WINDOWS]
"udp_socket_win.cc", n.d.,
<https://source.chromium.org/chromium/chromium/
src/+/main:net/socket/udp_socket_win.cc>.

[EXAMPLES] "ecn-examples", n.d.,
<https://github.com/nplab/ecn-examples>.

[WINDOWS-DOC]
"WSASetRecvIPEcn function (ws2tcpip.h)", n.d.,
<https://learn.microsoft.com/en-
us/windows/win32/api/ws2tcpip/nf-ws2tcpip-
wsasetrecvipecn>.

[WINDOWS-SOCKOPT]
"MSDN - IPPROTO_IP socket options", n.d.,
<https://learn.microsoft.com/en-us/windows/win32/winsock/
ipproto-ip-socket-options>.

[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,
<https://www.rfc-editor.org/info/rfc3168>.

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)",
STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022,
<https://www.rfc-editor.org/info/rfc9293>.

[RFC9260] Stewart, R., T端xen, M., and K. Nielsen, "Stream Control
Transmission Protocol", RFC 9260, DOI 10.17487/RFC9260,
June 2022, <https://www.rfc-editor.org/info/rfc9260>.

[RFC8261] Tuexen, M., Stewart, R., Jesup, R., and S. Loreto,
"Datagram Transport Layer Security (DTLS) Encapsulation of
SCTP Packets", RFC 8261, DOI 10.17487/RFC8261, November
2017, <https://www.rfc-editor.org/info/rfc8261>.

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

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9330] Briscoe, B., Ed., De Schepper, K., Bagnulo, M., and G. White, "Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture", RFC 9330, DOI 10.17487/RFC9330, January 2023, <https://www.rfc-editor.org/info/rfc9330>.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <https://www.rfc-editor.org/info/rfc2474>.

[RFC9868] Touch, J. and C. Heard, Ed., "Transport Options for UDP", RFC 9868, DOI 10.17487/RFC9868, October 2025, <https://www.rfc-editor.org/info/rfc9868>.

Acknowledgments

The author would like to thank Ryan Hamilton, who provided constant advice through this effort. Randall Meyer from Apple and Nick Grifka from Microsoft provided useful hints about the behavior of their respective operating systems. However, the author takes full responsibility for any errors above.

Michael Tuexen wrote the "ecn-examples" code that was very helpful in verifying the conclusions in this document. He also made multiple editorial contributions.

Neal Cardwell, Gorry Fairhurst, Max Franke, Rodney Grimes, Will Hawkins, Guillaume H迪tier, Max Inden, Jonathan Lennox, Colin Perkins, Marten Seemann, and Greg White made improvements to this draft.

Author’s Address

Martin Duke
Google
Email: martin.h.duke@gmail.com