A Test for IP-ECN Propagation by a Remote Tunnel Endpoint

Bob Briscoe*

28 Nov 2023

Abstract

This memo defines a brief set of tests to determine the decapsulation behaviour of an unknown remote tunnel endpoint with respect to the Explicit Congestion Notification (ECN) field in the Internet Protocol (IP) header. The tests could be automated to be used by a tunnel ingress to determine whether the egress that it is paired with will propagate ECN correctly.

1 Introduction

This memo defines a brief set of tests to determine the decapsulation behaviour of an unknown remote tunnel endpoint, with respect to the ECN field in the IP header. It provides a table that says whether each possible detected behaviour will propagate ECN correctly.

Test prerequisites are given in § 2.1, the main hurdle being the ability to overwrite the ECN field in the outer header at some point along the span of a tunnel. This makes it hard to test 'bump in the wire' tunnels. To overcome this hurdle, a convenient arrangement would be to set up the ingress of the tunnel under test on a host under the control of the tester.

The tests could be automated to be used by a tunnel ingress to determine whether the egress it is paired with will propagate ECN correctly. Without such a test, a tunnel ingress is required to zero the outer ECN field if it does not know whether the egress it is paired with will propagate ECN correctly [Bri23].

In scenarios where there is no control protocol for a tunnel ingress to discover the ECN capability of the egress, such a test could widen ECN coverage to tunnelled paths where it is currently absent.

1.1 Terminology

Encap: the encapsulation function at the ingress tunnel endpoint;

Decap: The decapsulation function at the egress tunnel endpoint;

The following terms will be used for the IP header at different locations on the path relative to the tunnel, considering only the direction from application client to application server:

Initial: the header arriving at the tunnel ingress;

Inner: the header that is encapsulated between the tunnel ingress and egress;

Outer: the header that encapsulates the inner between the tunnel ingress and egress;

Onward: the header leaving the tunnel egress.

2 The Tests

2.1 Test Prerequisites

- A working tunnel, e.g. a VPN;
- Access to one of the devices along the path of the tunnel, where the ECN field of the outer IP header can be altered[1];
- A remote application server, e.g. a web server (preferably a variety of different servers) that supports Accurate ECN feedback over either TCP [BKS23] or QUIC [IT21].
- A local application client (e.g. a web browser), optionally with the ability to configure whether it sends ECN-capable packets (prior to tunnelling), and if so whether it sets ECT(0) or ECT(1).

2.2 Test Setup

Set up the tunnel as normal (procedure will depend on which type of tunnel).

*research@bobbriscoe.net,

[1] Ideally so it can be altered arbitrarily, but just being able to set congestion experiences (CE, i.e. 0b11) would support all the tests except one, which is a less important one anyway.

If using TCP configure the client TCP stack to use Accurate ECN (AccECN) feedback:

Linux:
```
$ sysctl -w net.ipv4.tcp_ecn=3
```
MacOS:
```
$ sysctl -w net.inet.tcp.accurate_ecn=1
```

If using QUIC make sure your QUIC implementation supports accurate ECN feedback (at the time of writing, some still don't comply with the spec [IT21]).

Make sure your application traffic is being routed via the tunnel.

2.3 Control Test

The aim of this control test is to send packets with each of the four ECN codepoints from the application client, then check that feedback from the application server reflects the same codepoint.

Also it will be necessary to check that the tunnel ingress is copying each ECN codepoint to the outer. If it's not, in order to test the remote tunnel endpoint, it will be necessary to overwrite the outer with a copy of the Initial ECN codepoint (using a similar approach to that for the main tests in § 2.4).

Details: The Initial IP-ECN field can either be controlled by configuring the client stack, or by overwriting the field in the packet before it enters the tunnel. Given all codepoints cannot be set by configuration on all packets, only the overwrite approach will be described here. One example technique is to use the `tc` (traffic control) command to add a filter that applies an action to packets matching the filter. The Linux `tc` command is used for the example here, but `tc` is also available for MacOS.

```
$ tc filter add \
     dev DEV ingress flower MATCH_LIST \
     action pedit ex munge \
     ip dsfield set N retain 0x3
```

where `N` would be respectively 0 to 3 to set the ECN field to Not-ECT, ECT(1), ECT(0) or CE. `DEV` might be `eth0` for example. And an example of a `MATCH_LIST` might be `ip_proto tcp dst_port 80` (see the tc-flower manual page for details).

To check the Outer (outgoing) ECN, and the server's (incoming) feedback of the Onward ECN, Wireshark is recommended (version 4.0 onward supports AccECN in TCP). For this control test, check that the Initial is the same as the feedback of the Onward ECN, and that they are also the same as the Outer and the Inner.

The most specific feedback for testing purposes is given by TCP AccECN feedback in the SYN-ACK from the server in response to the initial TCP SYN packet from the client. The feedback is written with the 'handshake encoding' into the three ECN flags (AE, CWR, ECE) in the main TCP header as in the following table (from Table 3 of [BKS23], which uses the TCP flags as newly defined in Figure 2 of the same draft):

IP-ECN (outward)	TCP-ECN (inward)	Wireshark
Not-ECT	0b010	`.C.`
ECT(1)	0b011	`.CE`
ECT(0)	0b100	`A..`
CE	0b110	`AC.`

If your client sends data packets to the server once the TCP connection has been established, their feedback can be checked in AccECN TCP options that that server sends to the client. These give a count of how many bytes of each codepoint has been received by the server during the connection (counting from 1, not zero). However, they are not sent in response to every data packet (and they are optional). So further explanation will not be given, but if the reader wants to interpret this feedback, the definition of these TCP Options is in § 3.2.3 of [BKS23].

QUIC feedback can also be checked, but it has to be decrypted first. Apple gives instructions for how to allow Wireshark to decrypt QUIC for Cloudflare's quiche stack in order to check the ECN feedback[2], so that will not be repeated here. Then, any packet containing an ACK_ECN frame can be viewed in Wireshark to read a count of the number of packets received by the server with each ECN codepoint: ECT(0), ECT(1) and ECN-CE.

Test robustness: The test ought to be repeated a few times, and preferably conducted with a few different application servers (but over the same tunnel). This should help eliminate the possibility that:

- Active Queue Management (AQM) within the span of the tunnel is intermittently (and legitimately) setting the congestion experienced (CE) codepoint on the outer of some packets;
- A remote application server might have been chosen that provides incorrect ECN feedback due to an implementation bug.

Initial	Outer	RFC6040 (unified)	RFC4301 (IPsec)	RFC3168 (original)	RFC2003 (simple)	mangled
Not-ECT	CE	dropped	Not-ECT	dropped	Not-ECT	other
ECT(1)	CE	CE	CE	CE	ECT(1)	
ECT(0)	CE	CE	CE	CE	ECT(0)	
ECT(0)	ECT(1)	ECT(1)	ECT(0)	ECT(0)	ECT(0)	

Table 1: Main Test: Possible Results and their Interpretation

It should not be necessary to test both IPv4 & IPv6 (and both combinations of the two), because the definition of ECN is the same in both, so ECN processing code should be common to both. However, a full test could include all four combinations of IPv4 & IPv6.

2.4 Main Test

To test for correct operation of the remote tunnel egress, it is only necessary to test the combinations in the first two (grey) columns of Table 1 (in addition to the control test above).

For this test, the filter action will need to be applied after tunnel encapsulation. Then the outer will need to be overwritten with CE, for instance using the `tc` command as already outlined in § 2.3 with `N = 3` (decimal), and in the case of the last row, with `N = 1`.

2.5 Interpretation of Results

If the results conform with any of the green columns in Table 1, the tunnel egress correctly propagates ECN-marking, because it either complies with the latest ECN tunnelling spec (RFC 6040 [Bri10]) or with an earlier compatible spec updated by RFC 6040 (IPsec [KS05] or the original ECN spec [RFB01]).

If, on the other hand, the results conform to one of the red columns, the tunnel egress does not propagate ECN correctly. For instance, the first red column shows the outcome of a 'simple' tunnel, which just strips the outer on decapsulation (as used before ECN tunnelling was first specified in 2001). The final column 'mangled' captures all other possible outcomes.

References

[BKS23] Bob Briscoe, Mirja Kühlewind, and Richard Scheffenegger. More Accurate ECN Feedback in TCP. Internet Draft draft-ietf-tcpm-accurate-ecn-28, IETF, November 2023. (Work in Progress).

[Bri10] Bob Briscoe. Tunnelling of Explicit Congestion Notification. RFC 6040, RFC Editor, November 2010.

[Bri23] Bob Briscoe. Propagating Explicit Congestion Notification Across IP Tunnel Headers Separated by a Shim. Internet Draft draft-ietf-tsvwg-rfc6040update-shim-21, IETF, November 2023. (Work in Progress).

[IT21] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC rfc9000, RFC Editor, May 2021.

[KS05] Stephen Kent and Karen Seo. Security Architecture for the Internet Protocol. RFC 4301, RFC Editor, December 2005.

[RFB01] K. K. Ramakrishnan, Sally Floyd, and David Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, RFC Editor, September 2001.

[2] Testing and Debugging L4S in Your App

A History of ECN propagation by tunnels

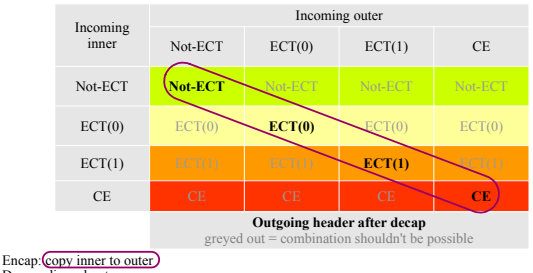

Figure 1: Simple (pre-ECN) tunnel, e.g. RFC2003

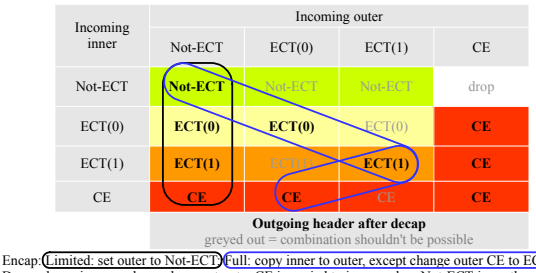

Figure 2: Original RFC 3168 ECN-tunnel [RFB01]

Figures 1–4 illustrate the evolution of ECN tunnelling, starting from pre-ECN days in Figure 1.

The table in each figure visualizes the outcome as each spec slightly altered the decapsulation rules. The rows represent the Inner and the columns represent the Outer header arriving at the tunnel egress. The text in each cell (and the associated background colour) gives the Onward (outgoing) header.

The loops group together the combinations of Inner and Outer that would be expected, given the behaviour of an encapsulator that complies with the same spec as the decapsulator. Where the spec allows two encap options, different coloured loops are shown for each.

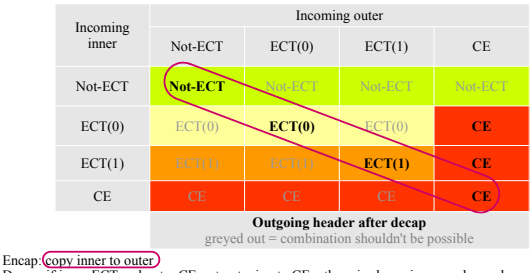

Figure 3: IPsec v2 (RFC 4301) [KS05]

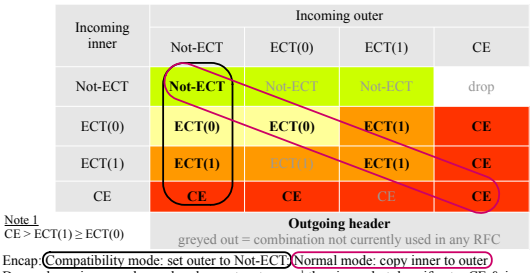

Figure 4: Universal ECN tunnel (RFC 6040) [Bri10]

The text in cells outside the loops is greyed out to illustrate that this combination would not be expected. Nonetheless, some other combinations of Inner and Outer can occur when an encap complying with one spec is paired with a decap complying with another. Figure 5 overlays the three behaviours that correctly propagate ECN to show how the three specs interact with each other. It shows the union of all three possible encap behaviours as not greyed out text, and two colours are used for the cell background where there are two possible decap behaviours.

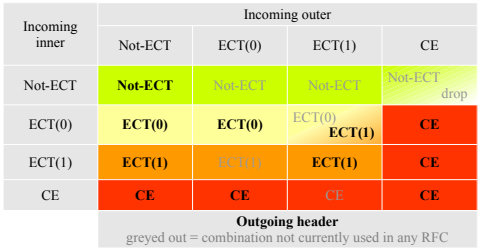

Figure 5: Black box view of all three combinations of ECN-tunnel specs (Figures 2–4)

Document history

Version	Date	Author	Details of change
00A	28 Nov 2023	Bob Briscoe	First Draft
01	28 Nov 2023	Bob Briscoe	First Issue