Robust ECN Signaling with Nonces

Neil Spring, David Wetherall, David Ely
University of Washington
IEEE CCW
October, 2001
ECN gives receivers power

Old congestion signals:
- Receivers claim lost packets for reliable delivery.
- Senders slow down when retransmitting.

ECN congestion signals:
- Good receivers return Congestion Experienced as ECN Congestion Echo.
- Senders slow down when ECE seen.
- Evil receivers have no reason to set ECE.

Evil is: greed, malice, or software bug.
Good vs. Evil

Evil receivers

- Get better performance than Good receivers
- Reduce performance of competing flows (Victims)
The Challenge

Can we have both:

- The benefits of ECN, and
- Protection against evil receivers?

Can we *detect* evil?
Can we *discourage* evil?
Outline

- ECN-nonce basics
- Header bits: IP and TCP
- Walkthroughs: Good and Evil receivers
- Endpoint memory requirements
- Resynchronization after loss/mark
- Policy: giving the ECN-nonce teeth
- Strange implementation cases
- Further reading
Nonces revoke receiver power

Sender attaches random “nonce” using ECT field
Receivers return the sum (parity) of nonces
Correct nonce sum depends on unmarked packets.
Receiver’s sum only incorrect when:
  - ECT cleared by CE.
  - Retransmission lacks ECT

Sender can detect an Evil receiver
## ECN bits to code-points

<table>
<thead>
<tr>
<th>bits</th>
<th>RFC 2481</th>
<th>RFC 3168</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>not ECN capable</td>
<td>same</td>
</tr>
<tr>
<td>0 1</td>
<td>unused</td>
<td>ECT (1)</td>
</tr>
<tr>
<td>1 0</td>
<td>ECN-Capable Transport (ECT)</td>
<td>ECT (0)</td>
</tr>
<tr>
<td>1 1</td>
<td>Congestion Experienced</td>
<td>same</td>
</tr>
</tbody>
</table>

Transition to CE removes original ECT(0) or ECT(1).
## Nonce Sum TCP header bit

<table>
<thead>
<tr>
<th>4 bit header length</th>
<th>reserved (3 bits)</th>
<th>NS</th>
<th>C</th>
<th>W</th>
<th>E</th>
<th>U</th>
<th>R</th>
<th>E</th>
<th>G</th>
<th>A</th>
<th>C</th>
<th>P</th>
<th>R</th>
<th>S</th>
<th>Y</th>
<th>F</th>
<th>I</th>
<th>N</th>
</tr>
</thead>
</table>

- Not yet standardized
- Defined whenever ACK is set
- Initial sum is 1.
ECN-Nonces with good receivers

ECT(0)

ECT(1) NS=0

ECT(1) ECE

ECT(1) CWR

NS=0

NS=1

ECN properly echoed

Nonce sum (NS) ignored

Synch. NS after CWR
ECN-Nonces with an evil receiver

CE improperly hidden.

Guessed **NS** is wrong.

Sender disables ECN.
Memory requirements

Senders store:

- Expected nonce sum for ack of each packet in retransmission buffer.
- The sequence number of the last CWR sent.
- A bit set when the expected nonce sum is wrong.

(next slide)

 Receivers store:

- Nonce Sum sent in ACKs.
- Nonces of unacknowledged packets.
Resynchronization

After ACK-of-CWR, NS may be “wrong.”
If so, set a flag: NS will continue to be “wrong” until another congestion signal
Keeps protocol complexity at sender.
Making detection sufficient (policy)

- Encourage nonce support
  - Preferred treatment
  - Non-ECN Optimizations

- Discourage misbehavior: On an incorrect nonce:
  - Stop sending ECT
  - Reduce cwnd, ssthresh to 1.
  - Alternatives: RST, limit send window, blacklist
Why $cwnd = ssthresh = 1$?

Tested long-term behavior: sender continues to set ECT. $cwnd /= 2$ left an advantage that $cwnd = 1$ removed. Stop sending ECT to be sure.
Strange Implementation Cases

Path MTU discovery:
retransmission without a congestion signal.

Fragmentation:
ECN-nonce doesn’t detect faulty reassembly.

Expected Nonce Sum storage:
Send buffer can change size: no static allocation.

Fast paths:
Linux ECN left fast path on ECE or CWR.
ECN-nonce verification is an every-packet thing.
Further Reading

Linux implementation (incomplete)
ICNP 2001 Paper
draft-ietf-tsvwg-tcp-nonce-01

nspring@cs.washington.edu
http://www.cs.washington.edu/homes/nspring/