State of ECN and improving congestion feedback with AccECN in Linux

Mirja Kühlewind <mirja.kuehlewind@tik.ee.ethz.ch>

Nov 10, 2016

NetDev 2.2 Conference, Seoul, Korea
Outline

- Overview of the ECN mechanism
- DCTCP in a nutshell
- Overview of the AccECN mechanism & reference implementation
- ECN support on webservers
- Network support for ECN
- ECN deployment by in iOS/macOS
- Next steps
Explicit Congestion Notification (ECN)

- TCP/IP extension for explicit congestion marking
  - Specified RFC3168 (2001)
  - Network nodes can mark packets instead of dropping them in cause of congestion
  - Endpoints can react early to avoid buffer overflows
- Implemented in most OSes
  - By default in server mode: negotiate the use of ECN (in SYN/ACK) if requested but do not request ECN (in SYN)
  - Early problems hindered wide-spread deployment
ECN Marking in IP header

- Endpoint can negotiate ECN support in TCP handshake

- If both endpoints support ECN, sender can mark packets as ECN Capable Transport (ECT) in the ECN field in the IP header

- Network nodes can mark ECT packets as Congestion Experience (CE) instead of dropping them before buffer overflows (requires use of Active Queue Management (AQM))
ECN/AccECN bits in TCP and IP header

- **2 flags in TCP header**: Congestion Window Reduced (CWR), ECN-Echo (ECE)
- **4 codepoints in IP header**: Not-ECT, ECT(0), ECT(1), CE
ECN Feedback in TCP header

- Receiver observes these markings and sends feedback to the original data sender using the ECN-Echo flags in the TCP header, until
- sender confirm reception of congestion feedback by setting the Congestion Window Reduced (CWR) flag in the TCP header

Sender receives only one congestion notification per RTT and does not know how many CE markings have been received in this time period
Data Center TCP (DCTCP)

- DCTCP adaptively calculates the window reduction factor $\alpha$ based on the current level of congestion

  - Multiplicative Decrease (e.g. $\alpha=0.5$ for Reno, 0.7 for Cubic):
    \[
    \text{cwnd} \leftarrow \alpha \times \text{cwnd}
    \]

- DCTCP’s moving average of congestion level with $M$ (= fraction of CE-marked bytes in observation window) and gain $g$:
  \[
  \alpha = \alpha \times (1 - g) + g \times M
  \]

  DCTCP implements own feedback scheme but no negotiation
More accurate ECN (AccECN) in Handshake

- Backward compatible negotiation in TCP handshake with the use of the AccECN (AE flag)
- Former NS flag but the ECN nonce experiment has recently been declared historic as it has never been deployed

<table>
<thead>
<tr>
<th>Header Length</th>
<th>Re-served</th>
<th>A</th>
<th>E</th>
<th>C</th>
<th>W</th>
<th>E</th>
<th>C</th>
<th>U</th>
<th>R</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>
<tbody>
<tr>
<td>Byte 13 und 14 of TCP header</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>IP Header Length</th>
<th>Differentiated Services Codepoint</th>
<th>ECN Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 1 und 2 of the IP header</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More accurate ECN (AccECN)

- Replaces feedback mechanisms of classic ECN to provide a more accurate feedback about the number of marking observed
- Use of 3 ECN flags as **ACE field** to signal number of observed CE marks

**Overview of headers**

<table>
<thead>
<tr>
<th>Header Length</th>
<th>Reserved</th>
<th>ACE Field</th>
<th>U R G</th>
<th>A C K</th>
<th>P S H</th>
<th>R S T</th>
<th>S Y N</th>
<th>F I N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Byte 13 und 14 of TCP header

<table>
<thead>
<tr>
<th>Version</th>
<th>IP Header Length</th>
<th>Differentiated Services Codepoint</th>
<th>ECN Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Byte 1 und 2 of the IP header
## AccECN TCP Option

<table>
<thead>
<tr>
<th>Kind (TBD)</th>
<th>Length</th>
<th>EE0B field</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE0B cont'd</td>
<td></td>
<td>ECEB field</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EE1B field</td>
</tr>
</tbody>
</table>

- **Kind**: TDB on publication but use of experimental TCP option with Kind=254 possible for experiments, with magic number 0xACCE
- **Length**: 11 (incl. EE0B, ECEB, and EE1B), 7 (EEOB and ECEB), 5 (only EE0B)
- **EE0B**: 24 least significant bits of byte counter of ECT(0) marks
- **ECEB**: 24 least significant bits of byte counter of CE marks
- **EE1B**: 24 least significant bits of byte counter of ECT(1) marks
Implementation of AccECN in Linux

- Reference implementation available on GitHub
  - Use of net.ipv4.tcp_ecn=4 to enable AccECN
    - Or just make AccECN default? In server mode or not?
  - Does not implement all fallback detection mechanisms incl. recently added IP codepoint feedback in handshake
  - No GSO/GRO support currently implemented
- Interface needed to configure use of AccECN option (never, always, on change as specified in draft)?
- For future use of AccECN information as input for congestion control an even clearer separation of ECN and loss handling/reaction to these signals would be beneficial
- Further: Integration of AccECN with DCTCP as next step
ECN support on webservers (Alexa 1Mio)

IPv4

IPv6

- 2000: 8%
- 2002: 1%
- 2004: 1%
- 2006: 1%
- 2008: 1%
- 2010: 1%
- 2012: 1%
- 2014: 1%
- 2016: 95.23%

- 2000: 0%
- 2002: 1%
- 2004: 1%
- 2006: 1%
- 2008: 1%
- 2010: 1%
- 2012: 1%
- 2014: 1%
- 2016: 73.87%
Network support for ECN

- Bleaching of the IP ECN codepoints (8-bit ToS field)
  - 2011: 25.2% to 28.5% (see Bauer et al. [1])
- Bleaching can be problematic if congestion signal (CE) gets lost
  - 2012: 8.2% of ECN-enabled webservers did not feed back CE
- Less problematic: small number of drops of CE-marked packets observed
- Below 1% connectivity failures when ECN is requested in SYN
  - SYN fallback as specified in RFC3168 address this case

ECN deployment by Apple as client default (see maprg presentation at IETF-98)

- Probabilistically enabled on
  - 5% of randomly selected connections over WiFi and Ethernet in iOS9 and macOS El Capitan
  - 50% of randomly selected connections over WiFi, Ethernet, and a few cellular carriers in iOS10 and macOS Sierra
  - Initial problems with reordering on one carrier (reported at tcpm/IETF-93, July 2015)

- Apple reported increasing adoption rates for network support:
  - United States (0.2%), China (1%), Mexico (3.2%), France (6%), Argentine Republic (30%)

- Heuristics to fallback to non-ECN when problems detected like
  - high reordering, an unexpected high number of CE marks, full connection loss, SYN loss, or RST on first data packet
Next steps for Linux?

- Is it time to enable (Acc)ECN as default on client side?
- Which heuristics are needed for fallback?
- Should this be deployed together with AccECN or separately?