Accurate ECN Linux Implementation

Experiences and Challenges

Ilpo Järvinen

29th Apr 2020
tcpm interim

Accurate ECN Linux Implementation

- Full Accurate ECN implementation[1]
  - Built on top of earlier work (by Mirja Kühlewind and Olivier Tilmans)
  - Initially based on -09
  - First(?) implementation with AccECN Option
  - Some technical challenges discovered, none insurmountable
  - Feedback from the implementation incorporated into -10
- In addition, created a packetdrill unit test suite[2]

[1] https://github.com/ij1/linux-accecn/tree/test-series3
[2] https://github.com/ij1/packetdrill-accecn/commits/accecn

Accurate ECN Handshake Reflector Background

What is the "handshake reflector"?

- Feeds back ECN codepoint during 3-way handshake to allow validating against path mangling
  - Overloads the same header bits (ECE, CWR, and AE) as AccECN ACE field later on
  - SYN's ECN codepoint encoded into SYNACK
  - In -09, SYNACK's ECN codepoint encoded into 3rd ACK of 3-way handshake & first data seg
    - These segment have SYN=0
    - Reliable channel provided by the first data seg
    - To avoid ACE field ambiguities, all similar segments must use the same encoding
    - Important note: changes made into this in -10

AccECN Handshake Challenges

- Challenges related to retransmissions
  - Receiving different ECN codepoint for original and rexmit
  - Unsure which packets arrive to the other end
    - s.cep & r.cep initialization and behavior on CE must consider all scenarios
- Challenges with handshake reflector using SYN=0 segments
  - SYN=0 reflector tx & rx require additional “state” (in -09)
  - Handshake reflection masks ACE in ACKs (in -09)
    - Disables AccECN for a half-connection with unidirectional flows
  - Segmentation offloading & reflector in 1st data seg (in -09)
  - TFO might skip the ordinary 3rd ACK & "first data seg" (data already in SYN)
    - Seqno assumptions cannot be synchronized
  - Delayed arrival of the reflected value

Handshake Reflector Solution Space (SYN=0 Case)

- More complex rules based on both sequence numbers
  - Offer only limited help, still problems with DupACKs
- Use option for SYN=0 reflector
  - Would take a step backwards, option should not be required
- Only send SYN=0 reflector in 3rd ACK
  - Signalling is unreliable but relatively simple
    - Can leverage existing state transitions & triggers
    - Occasional loss of ECN field mangling detection is not catastrophic
  - ACE interpretation is ambiguous in a few cases
    - But no catastrophic consequences from misinterpretation
  - Adopted in -10

AccECN and TCP Segmentation Offloading

- CWR flag behavior differs from RFC3168
  - RFC3168 aware tx clears CWR after 1st segment, corrupts ACE field
  - Changes in ACE field should not be masked by rx offloading
- Software-based offloading (GSO/GRO)
  - Requires changing a few lines
  - CWR flags was used on rx path to flush pending segs
    - Removed as AccECN may have a long run of segments with the same ACE field (and thus same CWR)
- HW offloading (not tested)
  - Added device/skb flag to indicate AccECN processing is supported/required
  - NIC not supporting CWR clearing could add the support flag immediately
  - Unknown if CWR clearing can be disabled in NICs supporting RFC3168 (not investigated, might depend on NIC model)
  - If any flags change on rx triggers flush, OK
    - Masking changes in ECE/CWR/AE bits during rx offloading corrupts ACE field

AccECN Option Background

- AccECN Option carries 24-bit LSB parts of 32-bit ECN byte counters
  - Sums of payload bytes with each ECN codepoint (ECT0/1, CE)
- AccECN Option is not always sent by the receiver
  - Draft gives rules when to send (at minimum, mostly with SHOULDs)
- Implementations are expected to estimate the ECN byte counters between AccECN Options
  - Requires byte counter delta based heuristic to decide which counter to increase next

AccECN Option and Change-Triggered ACKs

```
In -09:       (xx+ = counter incr)

              ...
              <------------- ACK+Opt
              DATA10 ---------ECT1->
              DATA11 ---------ECT1->
              DATA12 -----------CE->
              <-- ACK+Opt(e1b+,ceb+)
              DATA13 -----------CE->
e1b+          <----------------- ACK
or ceb+?
              DATA14 ---------ECT1->
e1b+          <-- ACK+Opt(e1b+,ceb+)
or ceb+?

Solution in -10:
              ...
              DATA12 -----------CE->
              <-- ACK+Opt(e1b+,ceb+)
              DATA13 -----------CE->
 ceb+,         <--------ACK+Opt(ceb+)
 e1b+
```

- To detect ECN codepoint changes, receiver SHOULD send change-triggered ACKs
- However, change-triggered ACKs in -09 are not enough to construct the received ECN pattern
- E.g., ECT1 $\rightarrow$ CE edge, the change-triggered ACK increases e1b & ceb
- Solution in -10: Send option in the change-triggered ACK and in the next ACK
    - Increases only ceb, unambiguous

Byte Counter Select Proposal for Simpler Estimation?

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Kind = TBD1   |  Length = 11  | Reserved   |Cnt|   EE0B field  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EE0B (cont'd)                 |           ECEB field          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| ECEB (cont'd) |                 EE1B field                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Place which counter to increase into AccECN Option
  - 2-bits required (Cnt), (same encoding as in IP ECN field)
- Last value of Cnt selects byte counter to increase when ACK w/o AccECN option (or w/o byte counters) arrives
  - Very simple, sender does not need to guess using heuristic
- 1 byte flag octet could provide necessary space (not included into -11)
  - 6-bits remain available for other/future use
  - Any comments on this from the working group?

AccECN Counter Updates and Estimation Errors

```
          ...
e1b+      <--------- ACK+Opt(e1b+)
          DATA10 -----------ECT1->
          DATA11 -----------ECT1->
          DATA12 -----------ECT0->
lost       X--- ACK+Opt(e1b+,e0b+)
          DATA13 -----------ECT0->
lost       X-------- ACK+Opt(e0b+)
          DATA14 -----------ECT0->
e1b+?     <------------------- ACK
          DATA15 -----------ECT1->
e1b-      <---- ACK+Opt(e0b+,e1b-)
```

- ACK loss may hide counter switch
  - Sender estimates into wrong unsigned 32-bit counter
  - Need to correct the counter downwards
  - Unsigned mod 2^{24} delta yields incorrect results
- Solutions
  - Either duplicate the counter variables
  - Or do update as 24-bit signed values (to allow decrease)
- Also, must "beacon" every 2^{22} bytes received to avoid counter overflow (large TCP windows)

AccECN Option CEP and CEB Deltas

- If option is not present in the previous ACK, CEB delta (d.ceb) is not available (or is just based on an estimate)
- Algorithm in Appendix A.2.2. depends on d.ceb for confirming ACE overflow
- Therefore, when ACK is sent and ACE/CEP has increased, include AccECN option
  - To synchronize s.ceb and calculate d.ceb
  - Not strictly necessary if the estimation works correctly
    - Still a good defense in depth approach from robustness point of view

Backup Slides

Handshake Reflector Masks ACE on ACKs (in -09)

```
Client seqno used:

        SYN ----------->
        <-------- SYNACK
        3rdACK+Refl --->
        <--------- DATA1
        <--------- DATA2
masked  ACK+Refl ------>
        <--------- DATA3
        <--------- DATA4
masked  ACK+Refl ------>

Both seqnos used:

        SYN ----------->
        <-------- SYNACK
        3rdACK+Refl --->
lost       X------ DATA1
        <--------- DATA2
masked  DupACK+Refl --->
        <--------- DATA3
masked  DupACK+Refl --->
```

- Reflector masks ACE/CEP for normal ACKs/DupACKs
- If both seqnos are used, impact is limited due to the dupACK thresh and response to (what is likely) a loss of data seg
  - SACKs could be used to differentiate further
- Even if reflector would not be put into DupACKs, ACE field interpretation is ambiguous

TFO and SYN=0 Handshake Reflector

```
   SYN+TFO+DATA1 ----->
   <-- DATA1+TFO+SYNACK
   DATA2+REFL? ------->

VS

   SYN+TFO+DATA1 ----->
    X- DATA1+TFO+SYNACK
   SYN --------------->
   <------------ SYNACK
   ACK+REFL ---------->
   DATA1+REFL -------->
```

- Bidirection data with TFO advances both sequence numbers
- Definition of "3rd" ACK or 1st data seg is not agreed by the end hosts
- The sending end cannot know what the receiving end got
  - Seqno assumptions cannot be synchronized

Handshake Reflector Prevents TSO for 1st Seg (in -09)

```
      SYN ----------->
      <-------- SYNACK
      3rdACK+REFL --->
No    DATA1+REFL ---->
TSO!  DATA2+ACE ----->
```

- Reflector prevents using TCP segmentation offloading (TSO) for 1st segment
- This may have some implications on processing requirements
  - Most flows are short
  - But no impact for 1 MSS flows

Final ACK/1st Segment not 1st Arriving Seg

```
              SYN ----------->
              <-------- SYNACK
lost          3rdACK+Refl -X
lost          DATA1+Refl --X
1st recv      DATA2+ACE ----->
```

- 3rd ACK/1st data seg is not always 1st segment the peer sees
  - Losses, reordering, unnecessary SYN/ACK rexmits
- ECN flags bleaching checks can still be applied to the 1st arriving segment though