Make TCP more Robust to Long Connectivity Disruptions

Alexander Zimmermann and Arnd Hannemann

Department of Computer Science, Informatik 4
RWTH Aachen University, Ahornstr. 55, 52074 Aachen, Germany

75th IETF TCPM Meeting - Stockholm, Sweden
July 26, 2009

Changes from previous draft version

draft-zimmermann-tcp-lcd-00

- Miscellaneous editorial changes in Section 1, 2 and 3
- Section 2: Updated motivation for the algorithm
  ⇒ Congestion versus Non-Congestion Events/Loss
  ⇒ In-line with RFC "Improving the Robustness of TCP to Non-Congestion Events" [RFC4653]
- Section 4.1: Add basic idea of the algorithm
- Section 4.2: Update algorithm (suggestions Tim Shepard)
  - Special case of the first received ICMP destination unreachable after an RTO could be removed
  - "Backoff_cnt" variable was introduced so it is no longer possible to perform more reverts than backoff
- Section 4.3: Expanded according to the algorithm changes

Problem of Long Connectivity Disruptions (1/2)

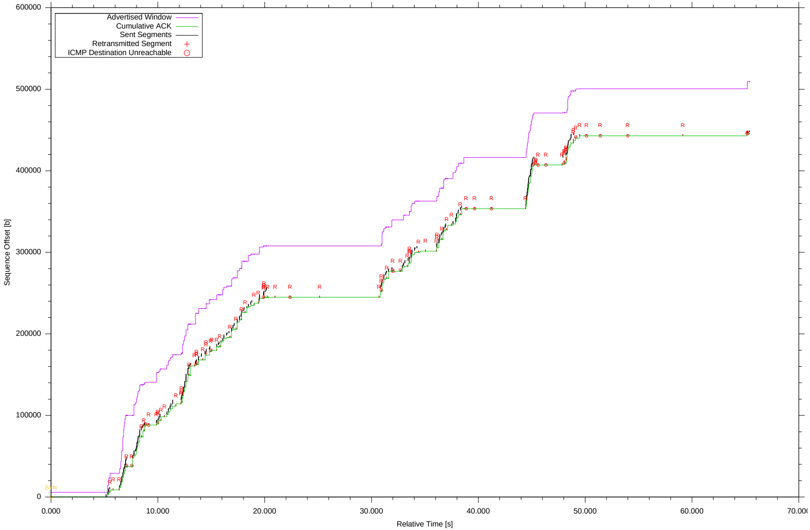
Observation

- Disruptions in e2e path connectivity which last longer than one RTO cause suboptimal TCP performance

Problem statement

- TCP interprets segment loss as a sign of congestion
  $\Rightarrow$ Means to detect loss: DUPACKs and RTO
- RTO case: (repeated) backoff(s) of the retransmission timer
- Deferred detection of connection re-establishment since TCP has to wait until next RTO before retransmit again

Problem of Long Connectivity Disruptions (2/2)



Solution for Long Connectivity Disruptions

Disruption Indication

- Disambiguate true congestion loss from non-congestion loss caused by long connectivity disruptions
- Exploit standard ICMP destination unreachable messages during timeout-based loss recovery

Disruption Reaction

- *Connectivity disruption loss:* undoing one RTO backoff if an ICMP unreachable message reports on a lost retransmission
  $\Rightarrow$ Enables prompt detection when connectivity is restored
- *Congestion loss:* Retaining std. timeout-based loss recovery

$\Rightarrow$ Sender-only modification

Connectivity Disruption Indication

Queue of the router experiencing the link outage is

- *Deep enough*: buffers all incoming packets
  $\Rightarrow$ Cause only variation in delay
  $\Rightarrow$ Eifel [RFC3522], F-RTO [RFC4138]
- *Not deep enough*: drops packets; discards according route
  $\Rightarrow$ TCP sender is notified about the dropped packets via ICMP destination unreachable messages [RFC1812]

Idea

- Interpret ICMP unreachable messages of code 0 (net unreachable) or code 1 (host unreachable) as long connectivity disruption indication

ICMP messages as Connectivity Disruptions Indication

Issues

- Do not ignore congestion indication from the network
- ICMP messages do not necessarily operate on the same timescale as the packets eliciting them [RFC1812]
- ICMP messages are subject to rate limiting [RFC1812]

Useful

- ICMP unreachable messages contain the IP header of the datagram eliciting the ICMP messages plus the first $64\,\mathrm{bit}$ of the payload [RFC0791]
  $\Rightarrow$ Allows to identify which segment of the respective connection triggered the ICMP unreachable message

Connectivity Disruption Reaction

Goal

- Prompt detection of the end of the connectivity disruption
- Retaining appropriate behavior in case of congestion

Basic Idea

- Increase the TCP's retransmission frequency by undoing one RTO backoff if ICMP message reports on a presumably lost retransmission
- If either the (re-)transmission itself, or the corresponding ICMP message is dropped the backoff is performed

The Algorithm (1/2)

State: retransmission timer is expired

1. Initialize backoff counter:
   - Backoff_cnt := 0
2. Placeholder for standard TCP timeout-based loss recovery
   - In particular RFC 2988 steps (5.4) – (5.6) go here
3. If RTO was backed off in step 2, then:
   - Backoff_cnt := Backoff_cnt + 1
4. Wait either for
   - RTO, then Goto 2
   - ACK, then Goto 9
   - ICMP unreachable, then Goto 5
5. If Backoff_cnt $\geq$ 0, i.e., if an undoing of the last RTO backoff is allowed, then Goto 6, else Goto4

The Algorithm (2/2)

6. Extract TCP segment included in `ICMP unreachable`:
   - $\texttt{SEG} := \texttt{Extract(ICMP\_DU)}$
7. If $\texttt{SEG.SEQ} == \texttt{SND.UNA}$, i.e., `ICMP_DU` reports on the oldest outstanding segment, undo last `RTO` backoff:
   - $\texttt{RTO} := \texttt{RTO}/2$
   - $\texttt{Backoff\_cnt} := \texttt{Backoff\_cnt} - 1$
8. If the `RTO` expires due to undoing in step 7, then `Goto 2`, else `Goto 4`
9. Placeholder for standard TCP behavior when an `ACK` has arrived; no further processing

Methodology

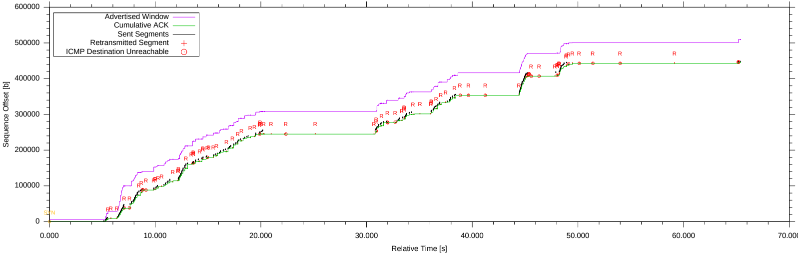
Code

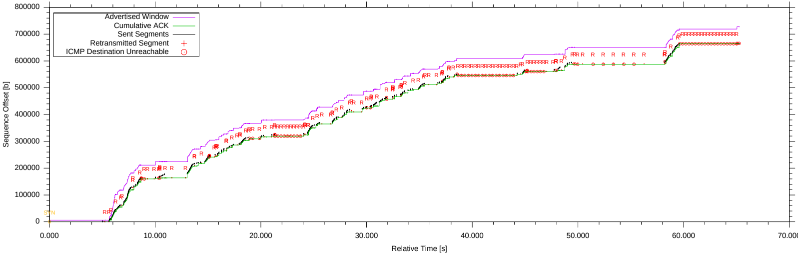
- Algorithm is implemented in Linux 2.6.28.7
- Publicly available: `http://www.umic-mesh.net/downloads`

Setup

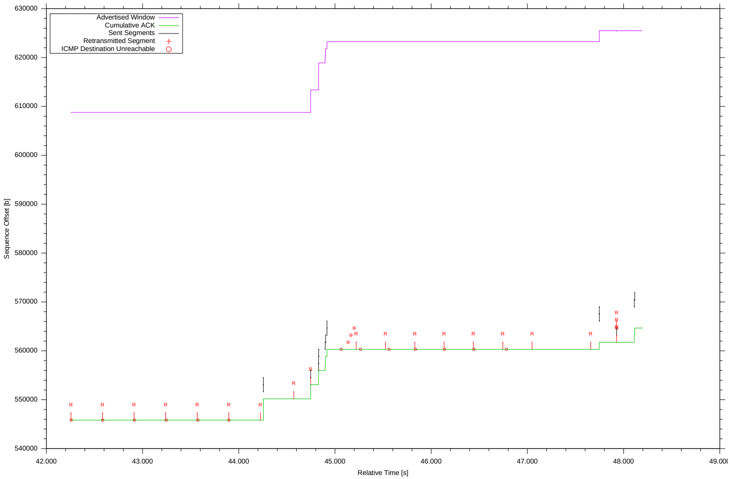
- Wireless mesh network with 51 nodes
- Routing protocol: OLSR [RFC3626]; standard parameter
- Path length: 2 to 4 hops
- Two parallel flows: standard and patched
- $60\,\mathrm{s}$ bulk TCP transfer; 500 measurements

Evaluation (1/3)

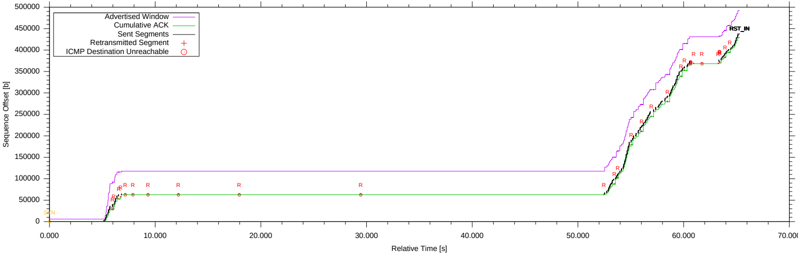


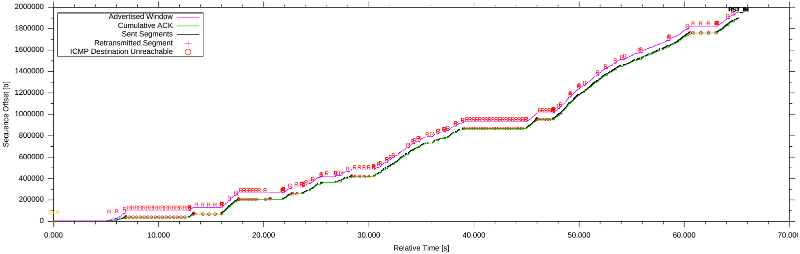


Evaluation (2/3)



Evaluation (3/3)





Features

We ...

- React only on ICMP unreachable messages during timeout-based loss recovery that reporting on SND.UNA
- Fall back to usual backoff in case there is congestion along the path after connectivity is restored
- Modify only the sender ⇒ Easy to implement

We do not ...

- Alter TCP's behavior in case of
  - slow-start, steady-state or fast recovery
  - timeout-based loss recovery with CWND > 1
  - no receiving ICMP unreachable messages
- Probe for route repair faster than slowest TCP can send

Special cases

Retransmission ambiguity problem

- No problem since the assumption that the ICMP message provides evidence that one link loss was wrongly considered as congestion loss is still true

Wrapped sequence numbers

- Late ICMP unreachable message reporting on an old error may coincidentally fit as input
- Possibility is minuscule, since ICMP message must contain the exact sequence number of `SND.UNA`, while at the same TCP is in timeout-based loss recovery

Next steps

Any interest from the WG?