TCP-CCC: single-path TCP congestion control coupling

draft-welzl-tcp-ccc-00

Michael Welzl, Safiqul Islam, Kristian Hiorth, You Jianjie

ICCRG
97th IETF Meeting
Seoul, South Korean
Nov 15 2017
Motivation

• **Parallel TCP connections between two hosts:** Combining congestions controllers can be beneficial
  
  – Very beneficial: short flows can immediately use an existing large cwnd, skip slow start; also avoids competition
  
  – Can divide available bandwidth between flows based on application needs

• Previous methods were hard to implement + hard to turn on/off (Congestion Manager)

• General problem with this: do parallel TCP connections follow the same path all the way?
  
  – Not necessarily, because of ECMP, etc.
Ensuring a common bottleneck

- Via configuration, e.g., app hint
  - Bottleneck is known, e.g., common wireless uplink
- Measurements can infer whether (long) flows traverse the same bottleneck [draft-ietf-rmcat-sbd]
- Encapsulation
  - VPNs, Generic UDP Encapsulation, TCP-in-UDP (TiU) ...
Motivation (from IETF 95) (ns-2 using TCP-Linux, kernel 3.17.4)

- 4 Reno flows, 10 Mb bottleneck, RTT 100ms; qlen = BDP = 83 Pkts (DropTail)
- TMIX traffic from 60-minute trace of campus traffic at Univ. North Carolina (available from the TCP evaluation suite); RTT of background TCP flows: 80~100 ms

- Link utilization: 68%
- Loss: 0.78%
- Average qlen: 58 pkts

- Link utilization: 66%
- Loss: 0.13%
- Average qlen: 37 pkts
Requirements

• Simple to implement
  – minimal changes to TCP code, avoid bursts
  – Correctly share TCP states
Requirements

• Simple to implement
• Correctly share TCP states
Design

• Basic idea similar to FSE in draft-ietf-rmcat-coupled-cc
  – *To emulate one flow’s behavior (... but easy to tune)*
  – Keep a table of all current connections *c* with their priorities *P(c)*; calculate each connection’s share as *P(c) / Σ(P) * Σ(cwnd)*; react when a connection updates its *cwnd* and use (*cwnd(c) – previous *cwnd(c))* to update Σ(cwnd)
Basic TCP changes

• The required changes to TCP:
  – This function call, to be executed at the beginning of a TCP connection ‘c’:
    
    ```
    register(c, P, cwnd, sshtresh);
    returns: cwnd, ssthresh, state
    ```
  – This function call, to be executed whenever TCP connection ‘c’ newly calculates cwnd:
    
    ```
    update(c, cwnd, sssthresh, state);
    returns: cwnd, ssthresh, state
    ```
  – This function call, to be executed whenever a TCP connection ‘c’ ends:
    
    ```
    leave(c)
    ```
ACK-clocking to avoid bursts

• A flow joining with a large share from the aggregate can create bursts in the network
  – If not paced
• Our approach:
  – Maintain the ack-clock of TCP
  – Using the ACKs of conn 1 to clock packet transmissions of connection 2 over the course of the first RTT when connection 2 joins
  – Similarly, we make use of the ACKs of connections 1 and 2 to clock packet transmissions of conn 3
  – Requires slightly more changes to the TCP code

Requirements

• Simple to implement

• Correctly share TCP states
TCP states

• Once in CA, Slow-Start(SS) shouldn’t happen as long as ACKs arrive on any flow ➔ only SS when all flows are in SS

• Avoid multiple congestion reactions to one loss event: draft-ietf-rmcat-coupled-cc uses a timer
  – TCP already has Fast Recovery (FR), use that instead
More results (FreeBSD implementation)

• Evaluations were repeated 10 times with randomly picked flow start times over the first second
• We generated internet traffic bursts using D-ITG to occupy 50% of the bottleneck capacity on average
More results (FreeBSD implementation)

**Avg. RTT**

- **Mean RTT (ms)**
  - Number of flows: 2, 3, 4, 5, 6, 7, 8, 9, 10
  - Graph showing the mean RTT for different numbers of flows.

**Loss ratio**

- **Loss ratio**
  - Number of flows: 2, 3, 4, 5, 6, 7, 8, 9, 10
  - Graph showing the loss ratio for different numbers of flows.

**Avg. goodput**

- **Mean goodput (Mbps)**
  - Number of flows: 2, 3, 4, 5, 6, 7, 8, 9, 10
  - Graph showing the mean goodput for different numbers of flows.

**Prioritization**

- **Throughput ratio**
  - Priority ratio: 1:1, 1:2, 1:3, 1:4, 1:5, 1:6, 1:7, 1:8, 1:9, 1:10
  - Graph showing the throughput ratio for different priority ratios.
More results (simulation – FCT of a short flow competing with a long flow)
More results – **Flow Completion Time (FCT)** (FreeBSD implementation)
Questions?