TCP-in-UDP

draft-welzl-irtf-iccrg-tcp-in-udp-00.txt

M. Welzl, S. Islam, K. Hiorth, J. You
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 in the network, and can support priorities  
    (similar to some of the benefits of multi-streaming in e.g. SCTP)

• Previous methods were hard to implement + hard to turn on/off (Congestion Manager)
  – Can be made easier (minimize changes to TCP code)

• General problem with this: do parallel TCP connections follow the same path all the way?
  – Not necessarily, because of ECMP 
    (or: any form of per-flow load balancing in the net)
Encapsulation

• This draft makes one concrete proposal (to be explained later)

• Other possibilities mentioned on the list (thanks!!)
  – Joe Touch: Not necessary
  – Tom Herbert:
    • IPv6 flow label
    • GUE

• Our conclusion: don’t prescribe one method
  – Mention the possibilities

Enables one-sided deployment?!?! I love this!!!
Does it really really work?
Coupled congestion control for TCP

• Basic idea similar to FSE in draft-ietf-rmcat-coupled-cc
  – Keep a table of all current connections $c$ with their priorities $P(c)$; calculate each connection’s share as $P(c) / \Sigma(P) \ast \Sigma(cwnd)$; react when a connection updates its $cwnd$ and use $(cwnd(c) – \text{previous} \ cwnd(c))$ to update $\Sigma(cwnd)$

• Some TCP-specific differences
  – 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 FR, use that instead
  – Also, generally a slightly more conservative CC behavior than the algorithm in draft-ietf-rmcat-coupled-cc
First simulation results
(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 bg TCP flows: 80~100 ms

Not coupled
- Link utilization: 68%
- Loss: 0.78%
- Average qlen: 58 pkts

Coupled
- Link utilization: 66%
- Loss: 0.13%
- Average qlen: 37 pkts
First simulation results - prioritization

- 2 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 bg TCP flows: 80~100 ms
Encapsulation: TCP-in-UDP (TiU)

• Avoid Packet size overhead
  – Avoid MTU problems

• Some ideas on TCP-over-UDP encapsulation shown in draft-denis-udp-transport-00 and draft-cheshire-tcp-over-udp-00
  – Suppress TCP checksum and TCP urgent pointer field and set 0 for URG flag: we do that
  – Suppress TCP src and dst ports (rely on UDP ports only): we do that too, but... want to multiplex!
  ➔ still need ports in some form
Encapsulation: TiU (Contd.)

With Flow id (5 bits) we can multiplex \(2^5 = 32\) parallel connections.

We use TiU SYN/SYN-ACK options to map ports to FID.

Offset change: related to STUN [draft-cheshire-tcp-over-udp-00]
Set up

• Happy eyeball for TiU
  – Put port-FID-mapping options in TiU-SYN and SYN/ACK

• Client
  1. Send UDP/TiU-SYN packet on TiU port
  2. Send TCP SYN

• Server (we write both)
  – Process UDP/TiU-SYN before processing TCP SYN

• UDP en-/de-capsulation added to TCP header processing
  – Just before sending, first when receiving
  – Small code change; normal TCP otherwise!
What this encapsulation (but also GUE) can give us

• A TCP that can easily evolve 😊
  – Maybe good as an intermediate experiment platform?

• Some benefits related to STUN
  [draft-cheshire-tcp-over-udp-00]

• Possible to support other transport protocols too
  [draft-cheshire-tcp-over-udp-00]

• In-line SPUD support without MTU problems: when the sender inserts SPUD, take SPUD header size into account for MSS calculation
Disadvantages

• Blocking / rate limiting of UDP
  – QUIC is going to help here, but only for ports 80 and 443 😞

• Prevents ECMP, but ECMP can be a good thing
  – It’s a socket option, maybe only use it when you expect to have many short flows or when priorities are important?
Current state

• Encapsulation
  – Finished for FreeBSD kernel

• Coupled-cc
  – Under development (simulations)
  – Rudimentary code being developed for FreeBSD, so should be easy to incorporate algorithm updates
Questions?