TCP Revisited

CSci551: Computer Networks
SP2006 Thursday Section
John Heidemann

Agenda

- connection setup and teardown
- flow control
- congestion control theory
- congestion control practice (in TCP)
- loss recovery
- **security**
- **performance**

Agenda

- **security**
  - **TCP hijacking**
  - **DDoS mitigation**
- performance
  - delayed ACKs
  - TCP at high bitrates and over long fat pipes

Connection Hijacking

- Problem: connection hijacking
  - some systems authenticate based on TCP connections and source IP addresses
  - => if you can *steal* a running TCP connection, you're in
  - it *is* possible, but not easy

TCP Distributed Denial of Service

- Problem: lots of people have too much time on their hands
  - and lots of people don't have secure computers
  - ⇒ bad people take over computers (*zombies*) and have them all ask you at once
- mitigation: SYN cookies
  - rather than make a new TCB for a new (probably bogus) connection, encode the info in the ISN on the SYN-ACK
  - when you get the ACK, recreate the missing state
- but, sadly, there are other forms of DDoS…

Agenda

- security
  - TCP hijacking
  - DDoS mitigation
- **performance**
  - **delayed ACKs**
  - **TCP at high bitrates and over long fat pipes**

Bad Optimization: Just Send One Ack Per Flight of Packets

- idea: don't send ACKs frequently, just send one after you get a whole bunch of packet
  - save bandwidth in reverse path (fewer acks)
- Problems
  - if you lose the ACK, out of luck and have to wait RTO and retx a packet to get a new ACK saying it all really got there
  - can't do RTT estimation if you don't get many acks
  - destroy the steady pace of ACKs (the ACK clock) and makes TCP very bursty

Delayed ACKS

- Problem: it's a lot of work to ACK every TCP segment
  - especially if it's just a few bytes
  - and the ACKs are tiny
- Approach: *delay* sending ACKs
  - send if you get two full segments
  - or after at most 500ms
  - idea: preserve *most* of the ACK clocking, but reduce the number of return ACKs
- Side-effect:
  - slow-start grows *per ACK*, not per ACKed segment, so it's 1, 2, 3, 3, 5…, not 1, 2, 4, 8…
  - can make fast retransmit less likely (so if the receiver notices loss it turns off delayed acks temporarily to make fast retx more likely)

What about NACKs?

- just NACKs, or NACKs and ACKs
  - actually NACKs + ACKs is like SACK (select ACK)
- pro:
  - much lower reverse path traffic
- con:
  - no self-clocking
  - can't easily estimate RTT changes

Problem: High BW Connections

- How many packets to keep in flight?
  - must be > bw*delay product
  - 10Mb/s * 100ms rtt = 1Mb ~ 100kB
  - 1Gb/s * 100ms rtt = 100Mb ~ 10MB!
- Sequence number wraparound time vs. Link speed:
  - 1.5Mbps: 6.4 hours
  - 10Mbps: 57 minutes
  - 45Mbps: 13 minutes
  - 100Mbps: 6 minutes
  - 622Mbps: 55 seconds
  - 1.2Gbps: 28 seconds

TCP Extensions for "Long, Fat Pipes"

- timestamp option + PAWS (Protection Against Wrapped Sequences)
  - endpoints swap timestamps on each pkt
  - allows better RTT estimation
  - provides effectively larger sequence space (reject packets with old timestamps)
- window scaling
  - multiplicative factor on wnd
  - to keep the pipe full

High-bandwidth TCP

- How fast can TCP go? Need new protocol?
  - demonstrated at 4Gb/s (FAST TCP at Caltech)
  - the spec doesn't specify a speed
- *but requires some care*
  - sequence number issues on prior slide
  - slow start would be a problem if you do it a lot (ex. if you have short connection)
  - TCP segment size (depends on IP packet size) (want more than 1500B packets at Gb/s rates)
  - loss is really bad if you go to slow start
    - and it's difficult to recover from multiple losses per RTT, even with New Reno and SACK

Other comments?

- alternative to TCP? XCP
- xxx

