The Open Network Lab
(Part 3)

Ken Wong
Applied Research Laboratory
Computer Science and Engineering Department

http://www.arl.wustl.edu/~kenw

kenw@arl.wustl.edu

http://www.onl.wustl.edu (ONL)

National Science Foundation ANI-023826,
CNS-0551651, REL-0632580
Lab 2 Setup

- **xrcv**
  - In-order accept policy
  - ACK in-order pkt
  - Silently drop all others

- **xsnd**
  - Send pkts over UDP
  - Internal hdr
    - Pkt sequence number: 0, 1, 2, ...
    - Timestamp
  - Tripling slow-start
    - Initial: \( \text{win} = 1, \text{nxtsn} = 0 \)
    - Rcv ACK: \( \text{win}’ = 3 \times \text{win} \)
      - Send next \( \text{win}’ \) pkts
  - Timeout
    - \( \text{win}’ = 1 \)
      - Send next pkt
Simple Features of an Approximate Solution

- Transmission Rate
  - \( R = 12 \text{ Mbps} = 1.5 \text{ Kpps} \) since
  - \( \Rightarrow \) Transmission Delay = 2/3 msec

- Bandwidth Delay Products
  - ACK BDP = \( 1.5 \text{ Kpps} \times 50 \text{ msec} = 75 \text{ pkts} \)
  - Data BDP = \( 4.5 \text{ Kpps} \times 100 \text{ msec} = 225 \text{ pkts} \)

- Queue Capacity
  - \( Q = 1,200,000 \text{ bytes} = 1,200 \text{ pkts} \)

- Max Queueing Time
  - Max Qtime = \( 1,200 \text{ pkts} / 1.5 \text{ Kpps} = 800 \text{ msec} \)

- Max RTT
  - Max RTT = Prop. Delay + Queuing Delay = 900 msec
TCP Flow and Congestion Control

- **Flow Control: Advertised Window Size**
  - Number of bytes receiver is willing to receive
  - Max Transmit = Min \{cwnd, Rcvr's advertised window\}

- **Implicit Feedback**
  - RTO ➔ Packet dropped due to congestion (not packet corruption)

- **Sliding-Window Protocol with Cumulative ACK**
  - ACK (with timers) for error control
  - Sliding window for efficiency

- **Dynamic Window**
  - *Slow-Start* then *Congestion Avoidance*
    - Increase cwnd exponentially until ssthresh (ACK: cwnd’ += 1 seg)
    - Then increase linearly for each ACK (cwnd’ += 1/cwnd seg)
  - **cwnd**: Congestion Window (Start with cwnd=1 segment)
  - **ssthresh**: Slow-start threshold
Fast Retransmit

- Generate an immediate duplicate ACK when an out-of-order segment is received
  - Notifies sender of out-of-order segment
  - But doesn't mean segment has been dropped

- Fast Retransmit Algorithm
  - 3 or more duplicate ACKs indicates that a packet was lost
  - Immediately retransmit only the lost packet
  - Don't wait for a retransmission timer to expire

- Fast Retransmit Rationale
  - Sender must wait 1 RTO before it discovers a dropped packet
  - 1 RTO can be much larger than 1 RTT → Duplicate ACKs provide earlier feedback
  - 1 duplicate ACK may only indicate packet reordering → 3 duplicate ACKs indicate packet drop
Fast Recovery

Fast Recovery Algorithm

» When third duplicate ACK arrives
  • $ssthresh' = \text{Min} \{ \text{Int}(cwnd/2), \text{Advertised Rcv Window} \} \text{ segments}$
  • Retransmit missing segment
  • $cwnd' = ssthresh' + 3 \text{ segments}$

» When new duplicate ACK for the same segment arrives
  • Increment $cwnd$
  • Transmit a new segment if possible

» When new ACK arrives
  • $cwnd' = ssthresh$
  • and stay in congestion avoidance algorithm

Fast Recovery Algorithm Rationale

» $cwnd = ssthresh + 3 \text{ segments}$
  • Inflates window by the cached segments

» Increment $cwnd$
  • Segments are still flowing; so inflate window

» Congestion Avoidance: Avoid slow start
One Flow
32,000-Byte Queue
Experiment

```
python
ssh n2p1
Last login: Thu Nov 13 23:41 2008 from onlusc.arl.wustl.edu
n2p1:~$ iperf -s -w 16
--- Server listening on TCP port 5001 ---
TCP window size: 32.0 MByte (WARNING: requested 16.0 MByte)

[ 4] local 192.168.2.32 port 5001 connected with 192.168.1.32 port 47401
  [ 4] 0.0-10.0 sec 7.63 MBytes 5.90 Mbits/sec

n2p1:~$ iperf -c n2p1 -w 16m -n 8000000 -1 1050
--- Client connecting to n2p1, TCP port 5001 ---
TCP window size: 32.0 MByte (WARNING: requested 16.0 MByte)

[ 3] local 192.168.1.32 port 47401 connected with 192.168.2.32 port 5001
  [ 3] 0.0- 0.0 sec 7.63 MBytes 1.50 Gbits/sec
```

--- n2p1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 100.096/143.086/315.018/85.966 ms

--- bandwidth ---

8 - Ken Wong – Nov 2008
tcptrace Displays

- **tcprof**
  - Runs as root “tcprof -i eth0 -w /tmp/onl.tcpdump”
  - Run at sender to produce /tmp/onl.tcpdump
  - Save /tmp/onl.tcpdump as ~/tcpdump-XXX

- Run tcptrace off-line to produce xplot files
  - Produces summary (trace.txt) and xplot files
  - “tcptrace ... tcpdump-32k > trace.txt”

- Run xplot to view TCP segment activity
  - “xplot ... a2b_tsg.xpl”
  - a2b_tsg.xpl shows TCP segment activity
Lots of Retransmissions
3 Duplicate ACKs → Retransmit

Begin Fast Retransmit/Fast Recovery

Recv Window

ACK line
Slow-Start
1488-byte segments
Fast Retransmit/Fast Recovery
One Flow
1,200,000-Byte Queue
Experiment

Duplicate ACKs exhausted
Retransmissions Dominate
Fast Retransmit/Fast Recovery

New Reno:
- cwnd increases
- Stay in fast recovery until ACK newest pkt
Fast Retransmit/Fast Recovery

~6 Mbps
1 ACK/2 msec

~24 Mbps
1 ACK/1 msec

Lot’s of dupe ACKs ➔ ssthresh getting smaller

K, Wong – Nov 2008
Duplicate ACKs Exhausted
Retransmissions

Retransmit
every other pkt

in rcvr
buffer
Leaving Fast Recovery

ACK of latest pkt arrives
Slow-Start After Fast Recovery

\[ \text{ssthresh} = 1 \text{ segment} \]
2 Pkts For Every ACK
Three Flows
32,000-Byte Queue
Shell Scripts

```bash
#!/bin/sh
#
# Usage: myping
#
source /users/cnl/.topology

ssh $n1p1 ping -c 3 n2p1
ssh $n1p2 ping -c 3 n2p2
ssh $n1p3 ping -c 3 n2p3

# Purpose: Start 3 TCP receivers on n1p[123]
# Usage: tsndrs-2mpr
#
source /users/cnl/.topology  # get defns of external interfaces

# start tcp receivers
for host in $n2p1 $n2p2 $n2p3; do
    ssh $host /usr/local/bin/iperc -c -w 16w -t 20 &
done
```

26 - Ken Wong - Nov 2008
Experiment

Server listening on TCP port 5001
TCP window size: 32.0 MByte (WARNING: requested 16.0 MByte)

- local 192.168.2.32 port 5001 connected with 192.168.1.32 port 47539
- local 192.168.2.48 port 5001 connected with 192.168.1.48 port 34702
- local 192.160.2.64 port 5001 connected with 192.165.1.64 port 40610

- 0.0-51.4 sec 23.6 MBytes 3.65 Mbits/sec
- 0.0-70.6 sec 30.9 MBytes 3.88 Mbits/sec
- 0.0-75.6 sec 31.4 MBytes 3.49 Mbits/sec
Three Flows
32,000-Byte Queue
For Each Flow
Fair Share Packet Scheduling

Separate Queues:

One Queue: