Reliability

G54ACC

Lecture 11

richard.mortier@nottingham.ac.uk

Recap

- Congestion control cf. flow control
- Adapt transmission window based on timers
- TCP reacts to loss by backing off
  – Loss is assumed to be caused by network congestion
- TCP reacts to successful delivery by probing for more bandwidth
  – In the simple case, generates a “saw tooth” usage

Contents

- Performance and reliability
- TCP congestion control
- Multimedia

Contents

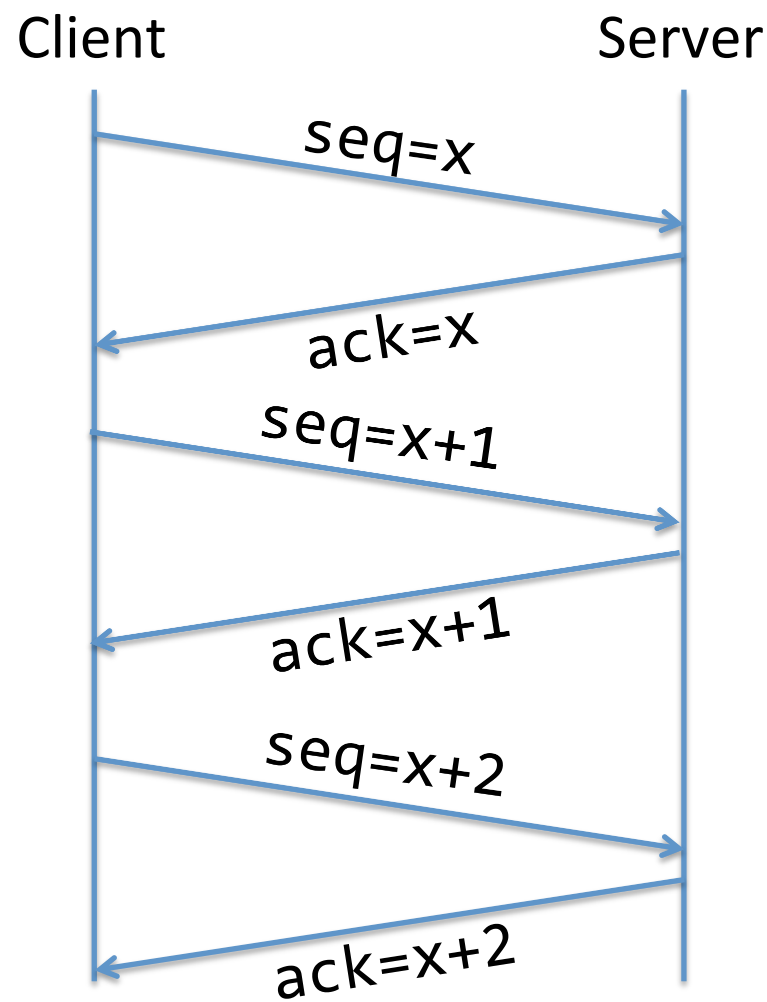
- Performance and reliability
  - Simple schemes
  - Congestion collapse
- TCP congestion control
- Multimedia

Achieving Reliability

- Reconstruct lost data
  - E.g., Forward Error Correction
  - Cf. coding and block codes
- Retransmit lost data
  - ...but how to detect loss?
- Rx explicitly acknowledges: ARQ
  - <u>A</u>utomatic <u>R</u>epeat Re<u>Q</u>uest
  - Alternative: negative acknowledgements
  - Cf. OSPF/ISIS

Stop ‘n’ Wait

- Simplest possible
  - Tx `seq(x)`
  - Tx wait for `ack(x)`
  - Tx `seq(x+1)`
- Really poor performance in high-latency high-bandwidth networks
  - Why?

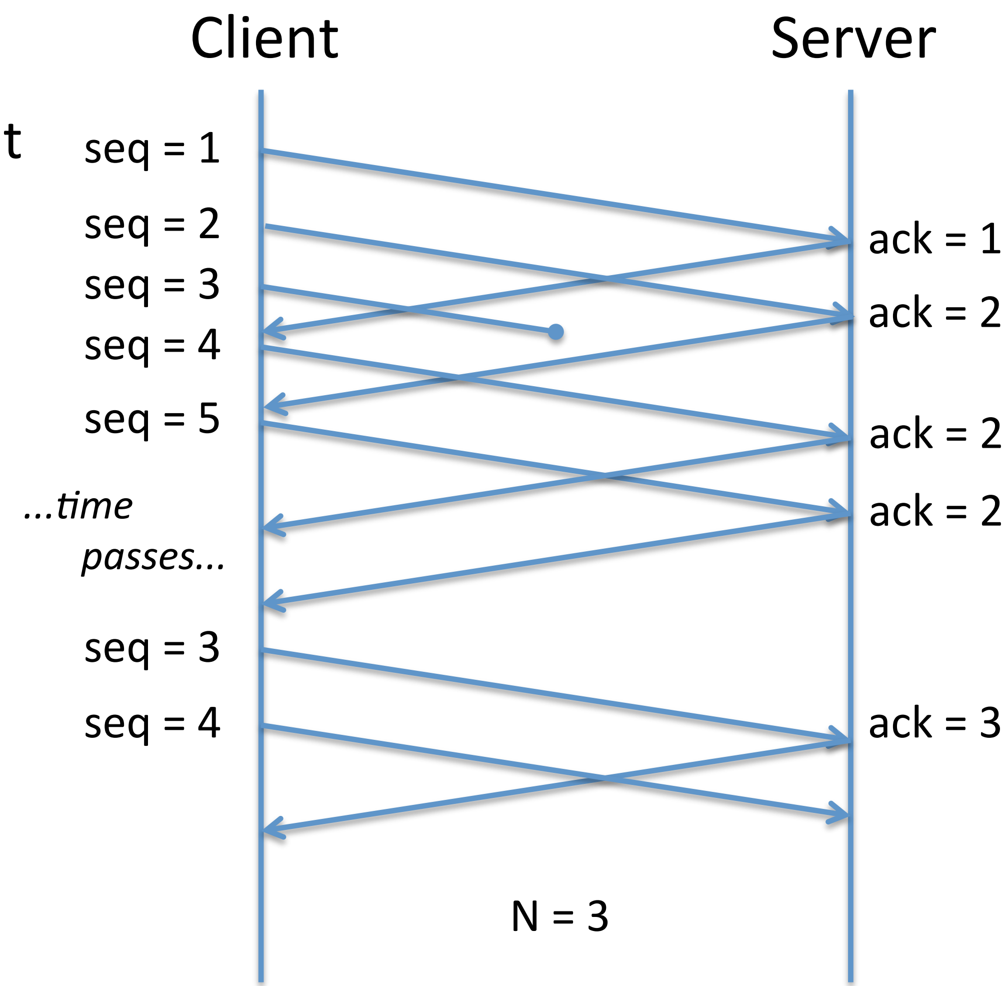


Reliability and Performance

- *Rate control*: never send too fast for network
  - How to estimate appropriate rate?
- *Sliding window*: allow unacknowledged data in flight
  - How to determine correct window size?
- *<u>R</u>etransmission <u>T</u>ime<u>O</u>ut*
  - How long to wait to decide a segment is lost?
- Require estimates of dynamic quantities
  - Actually, proxies for the network's current capacity

First Attempt (pre-1988)

- Go-Back-N ARQ
  - Permit N segments in flight
  - Timeout implies loss
  - Retransmit from lost packet onward
- Self-clocking behaviour
  - Tx next segment after previous segment acknowledged
- Loss wastes throughput
  - *Goodput* vs. *throughput*



Congestion Collapse

- If network load gets too high, experience *congestion collapse*
  - Initially observed on NSFNet Oct/1986
  - LBNL—UCB (400yds/3 hops) dropped 32kb/s to 40b/s
- Why?
  - Router buffers fill, traffic is discarded, hosts retransmit
  - But early TCP (4.2BSD) was naive! Retransmit **more** since data was lost...
  - Boom!
- Solution: Van Jacobson's congestion control algorithm (SIGCOMM'88): Slow-start and Congestion avoidance

Contents

- Performance and reliability
- TCP congestion control
  - Avoidance and recovery
  - Tahoe, Reno, NewReno, SACK, Vegas
  - ECN marking
- Multimedia

Congestion Control

- Need to determine current parameters
  - Window size for most TCPs
  - Either estimate, or *actively probe*
- Additive-Increase, Multiplicative-Decrease
  - Maintain a *congestion window*, `cwnd`
  - `awnd` gives Rx window flow control
  - `wnd = min(awnd, cwnd)`
- First introduced as TCP Tahoe in 4.3BSD (1988)

TCP Tahoe

- Congestion Avoidance: given `wnd = N` segments
  - Segment acknowledged: `cwnd += 1/cwnd`
  - RTO: `cwnd := 1; ssthresh /= 2`
  - But how to initialise `cwnd`?
- Slow-Start (actually, *exponential growth*!)
  - Original behaviour was constant initial `cwnd`
  - Every segment acknowledged, `cwnd += 1`
  - Stops when loss occurs or `cwnd == ssthresh`
  - Adjust `ssthresh` up once we reach avoidance

Fast Retransmit

- Detecting just a single loss takes a full RTO
  - 500ms or more, at 500ms granularity
- Rx can signal a "hole" in the stream
- Send *duplicate ACKs (dup-acks)*, typically 3
- Tx retransmits the relevant segment
- `ssthresh := max(wnd/2, 2);`
  `cwnd := 1`
- Return to slow-start

Evolution: Reno, NewReno, SACK

- Reno (4.4BSD, 1990) adds *fast recovery*
  - Dup ACKs trigger retransmit
  - But now set `cwnd /= 2`
  - Then inflate `wnd` by number of dup ACKs as self-clocking means segments have left the pipe
- NewReno (4.4BSD, 1993) modifies fast recovery
  - Reno has poor performance for multiple losses
  - ACK for part of window no longer exits recovery
- <u>S</u>elective <u>ACK</u>nowledgement
  - Maintain more state to retransmit *only* lost segments

Revolution: Vegas

- Alternative approach to congestion avoidance
  - Source uses RTT to estimate #packets in pipe
  - End-to-end queuing delay as congestion measure
- `for every RTT:`

  $RTT_{min} = \min(RTT, RTT_{min})$

  if $wnd/RTT_{min} - wnd/RTT < \alpha$ then wnd++

  if $wnd/RTT_{min} - wnd/RTT > \beta$ then wnd--
- Never widely deployed

Marking

- Many *many* other variants
  - FACK, VENO, BIC, CUBIC, HYBLA, Compound, Westwood, Fusion, ...
  - Cf. doi:10.1109/SURV.2010.042710.00114
- Basically: loss signals congestion, then react
  - But waiting for loss takes a full RTO
- <u>E</u>xplicit <u>C</u>ongestion <u>N</u>otification, RFC3168
  - Signal congestion *early* via flags
  - Requires router support: marking, queuing

Contents

- Performance and reliability
- TCP congestion control
- Multimedia
  - RT*P
  - TCP friendliness
  - SST

Multimedia

- TCP is not always useful
  - Messages/frames preferably to bytestream
  - Reliability can cause untimely delivery
  - Canonical example: Real-time media
- Audio/video codecs usually produce frames
  - Not a continuous bytestream
- Losing a frame is better than delaying all subsequent data
  - We can skip or interpolate missing frames

Real Time Protocol

- Encapsulates media over UDP
  - Sequencing, timestamping, delivery monitoring
  - No reliability, no QoS
- Adds a control channel (RTCP)
  - Backchannel to report statistics, participants, &c
- Transport only
  - Leaves encodings, floor control, &c to application
- Extensible

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      defined by profile       |           length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        header extension                     ...
```

```
        0                   1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
header |V=2|P|    RC   |   PT=RR=201   |             length            |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                     SSRC of packet sender                     |
       +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
report |                 SSRC_1 (SSRC of first source)                 |
block  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  1    | fraction lost |       cumulative number of packets lost       |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |           extended highest sequence number received           |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                      interarrival jitter                      |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                         last SR (LSR)                         |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                   delay since last SR (DLSR)                  |
       +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
report |                 SSRC_2 (SSRC of second source)                |
block  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  2    :                               ...                             :
       +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
       |                  profile-specific extensions                  |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TCP Friendliness

- Real-time media being sent at a given rate
  - May cause TCP to experience loss and backoff
  - Media will eventually win
- Need to adapt real-time media
  - Equation based congestion control
  - `T = avg(wnd)/RTT`
  - `p(drop) = 2/(3*w`^{2}`) => w = sqrt(2/3p)`
  - Thus `T(p) ~= (1/RTT)*sqrt(3/2p)`
- Send drop rate to Tx and it uses T(p) directly

Structured Stream Transport

- Presented in SIGCOMM'07
- Single TCP stream imposes total order
  - Cf. HTTP/1.1 later
- Alternative: enable "substreams" to be created
  - More flexible scheduling by both ends
  - Treat datagram transmission as ephemeral substream
  - Multiplex substreams onto congestion controlled channels

Contents

- Performance and reliability
  - Simple schemes
  - Congestion collapse
- TCP congestion control
  - Avoidance and recovery
  - Tahoe, Reno, NewReno, SACK, Vegas
  - ECN marking
- Multimedia
  - RT*P
  - TCP friendliness
  - SST

Summary

- Achieving both reliability and performance is *hard*!
- There're many subtle interactions particularly if your scheme is to be general
  - Even trickier if you mix in non-responsive traffic
- But try not to let TCP become a strait-jacket...

Quiz

1. What are the different tradeoffs made by FEC and ARQ in providing reliability?
2. How does each behave on a very reliable network?
3. How does each behave on a very unreliable network?
4. Why does stop'n'wait perform so badly on a high-bandwidth, high-latency network?
5. Sketch the evolution of the window size for TCP Tahoe and TCP Reno, assuming no loss until **after** exiting slow-start.
6. Why does TCP Tahoe suffer in the face of a burst of losses? Why was this particularly bad?
7. Why might you **not** want TCP's features when transporting real-time multimedia data?