CSE561 – Reliable Transport

David Wetherall
djw@cs.washington.edu
Reliable Transport

- Focus:
  - Reliably delivering content across the network

- Connections
- Retransmission (ARQ)
- Sliding windows
- Flow control
Where the pieces fit

app stuff
write(), sendto(), send()

OS stuff
Socket file descriptor
socket

Protocol stuff
port
Network Layer
Link Layer
Link
Router
Local Network
Internet
Router
Local Network
TCP Connection Setup

- Three-way handshake opens both directions for transfer
Some Comments

- We could abbreviate this setup, but it was chosen to be robust, especially against delayed duplicates
  - Three-way handshake from Tomlinson 1975

- Incrementing initial sequence numbers (ISNs) minimizes the chance of hosts that crash getting confused by a previous incarnation of a connection

- Random ISNs proves two hosts can communicate
  - Weak form of authentication
Diversion: TCP SYN cookies

- Goal is for server to keep no unnecessary state to be as robust as possible

- SYN cookie solution:
  - Instead, make client store state in response to SYN
  - Server picks return seq $y = \oplus$ that encrypts $x$
  - Gets $\oplus + 1$ from sender; unpacks to yield $x$
TCP Connection Teardown

Web server

FIN_WAIT_1

FIN

FIN_WAIT_2

ACK

TIME_WAIT

FIN

CLOSED

Web browser

CLOSE_WAIT

LAST_ACK

ACK

CLOSED
Kinds of Teardown

- FIN
  - TIME_WAIT for 2MSL (two times the maximum segment lifetime of 60 seconds) before completing the close
  - This is in case the ACK was lost and FIN will be resent

- RST
  - Not an orderly connection close
  - Server reliably sends data, then RST (unreliable), and moves on
  - Client deals with it
Automatic Repeat Request (ARQ)

- Packets can be corrupted or lost. How do we add reliability?
- Acknowledgments (ACKs) and retransmissions after a timeout
- ARQ is generic name for protocols based on this strategy
The Need for Sequence Numbers

• In the case of ACK loss (or poor choice of timeout) the receiver can’t distinguish this message from the next
  – Need to understand how many packets can be outstanding and number the packets; here, a single bit will do
Stop-and-Wait

- Only one outstanding packet at a time
- Also called alternating bit protocol
Limitation of Stop-and-Wait

- Lousy performance if wire time $\ll$ prop. delay
  - How bad? You do the math
- Want to utilize all available bandwidth
  - Need to keep more data “in flight”
  - How much? Remember the bandwidth-delay product?
- Leads to Sliding Window Protocol
Sliding Window Protocol

- There is some maximum number of un-ACK’ed frames the sender is allowed to have in flight
  - We call this “the window size”
  - Example: window size = 2

Once the window is full, each ACK’ed frame allows the sender to send one more frame.
Sliding Window: Sender

- Assign sequence number to each frame ($\text{SeqNum}$)
- Maintain three state variables:
  - send window size ($\text{SWS}$)
  - last acknowledgment received ($\text{LAR}$)
  - last frame sent ($\text{LFS}$)
- Maintain invariant: $\text{LFS} - \text{LAR} \leq \text{SWS}$
- Advance $\text{LAR}$ when ACK arrives
- Buffer up to $\text{SWS}$ frames
Sliding Window: Receiver

- Maintain three state variables
  - receive window size (RWS)
  - largest frame acceptable (LFA)
  - last frame received (LFR)
- Maintain invariant: \( \text{LFA} - \text{LFR} \leq \text{RWS} \)

Frame \( \text{SeqNum} \) arrives:
- if \( \text{LFR} < \text{SeqNum} \leq \text{LFA} \) ⇒ accept + send ACK
- if \( \text{SeqNum} \leq \text{LFR} \) or \( \text{SeqNum} > \text{LFA} \) ⇒ discard

Send cumulative ACKs – send ACK for largest frame such that all frames less than this have been received
Flow Control

• Sender must transmit data no faster than it can be consumed by the receiver
  – Receiver might be a slow machine
  – App might consume data slowly

• Implement by adjusting the size of the sliding window used at the sender based on receiver feedback about available buffer space
Example – Exchange of Packets

Receiver has buffer of size 4 and application doesn’t read

Stall due to flow control here

T=1
SEQ=1
ACK=2; WIN=3

T=2
SEQ=2
ACK=3; WIN=2

T=3
SEQ=3

T=4
SEQ=4

T=5
ACK=4; WIN=1

T=6
ACK=5; WIN=0

dwj // CSE 561, Spring 2010, L10
Example – Buffer at Sender

\[\begin{array}{c|cccccccc}
T=1 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
T=2 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
T=3 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
T=4 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
T=5 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
T=6 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}\]

- Green =acked
- Orange =sent
- Yellow =advertised
TCP Header Format

- Sequence, Ack numbers used for the sliding window
- AdvertisedWindow used for flow control
Digital Fountain discussion

• What is the content distribution goal?
• What is the scaling problem with using retransmissions?
• What is the tradeoff between Tornado and RS codes?
• How much does interleaving help?
• What is layered multicast?