CSE 461 – Connection establishment and flow control

David Wetherall/Aruna Balasubramanian
Transmission Control Protocol (TCP)

- Reliable bi-directional byte stream
- In order delivery of packets
- Flow control (Today)
- Congestion control (Next class)
TCP Delivery
TCP Header Format

- Ports plus IP addresses identify a connection
TCP connection

**Server**

1. Socket()
2. Bind()
3. Listen()
4. Accept()

   **Connection Establishment.**

   **Client**

1. Socket()
2. Connect()
3. Send()
4.Recv()

Data (request)

Data (reply)

Block until connect

Process request

Connection Establishment.
Connection Establishment in TCP

• Both sender and receiver must be ready before we start to transfer the data
  – Sender and receiver need to agree on a set of parameters

• This is signaling
  – It sets up state at the endpoints
  – Compare to “dialing” in the telephone network
Problems with Connection Establishment

Key problem is to ensure reliability even though packets may be lost, corrupted, delayed, and duplicated — Don’t treat an old or duplicate packet as new

What did we do in data link layer to avoid duplicates?
Ways to avoid duplicates

Use sequence numbers to keep track of duplicates

Use a maximum segment lifetime (MSL)
  - Wait until MSL to repeat sequence numbers (120 seconds in the Internet)
  - How long should the sequence number be?

This may still not be enough for connection establishment
  - Because we do not want to maintain state across connections
Three-Way Handshake

- Opens both directions for transfer
TCP Connections – simultaneous connect

Normal case

Simultaneous connect

SYN (SEQ = x)

SYN (SEQ = y, ACK = x + 1)

SEQ = x + 1, ACK = y + 1

SYN (SEQ = y)

SYN (SEQ = y, ACK = x + 1)

SYN (SEQ = x, ACK = y + 1)
Recap

Why do TCP packets have a lifetime?

What is the use of a receive buffer?

How protocol is used for connection establishment?
Connection Teardown

• Cleans up state in sender and receiver

• TCP provides a “symmetric” close
  – both sides shutdown independently
  – Why?
Connection Release problem

Key problem is to ensure reliability while releasing

Asymmetric release (when one side breaks connection) is abrupt and may lose data
But symmetric release is difficult

Symmetric release (both sides agree to release) can’t be handled solely by the transport layer

CN5E by Tanenbaum & Wetherall, © Pearson Education-Prentice Hall and D. Wetherall, 2011
Connection Release

Normal release sequence, initiated by transport user on Host 1
- DR=Disconnect Request
- Both DRs are ACKed by the other side

What happens if ack is lost?
Error handling in connection release

Host 1
- Send DR + start timer
- Release connection
- Send ACK

Host 2
- Send DR + start timer
- (Timeout) release connection

Connections between Host 1 and Host 2, with error handling and timeout mechanisms.
TCP State Transitions

- Wow!
TCP Connection Teardown with states

Web server

FIN_WAIT_1

FIN_WAIT_2

TIME_WAIT

CLOSED

Web browser

FIN

ACK

FIN

ACK

CLOSE_WAIT

CLOSED
Recap

• Why does TCP need to explicitly teardown a connection?
• How does TCP handle lost disconnection requests?
• What is symmetric close?
• How long does the TCP wait, after the close call, to actually close the connection?
Error Control

How is error control done in data link layer?

TCP uses similar ideas
An ARQ solution: Stop-and-Wait Protocol

- Sender doesn’t send next packet until he’s sure receiver has last packet
- What are the problems with Stop and Wait?

Here’s what it looks like when things are going well (no transmission errors).
Performance problem with stop and wait

• Problem: “keeping the pipe full”
  – If the bandwidth-delay product is much larger than a packet size, the sender will be unable to keep the link busy

• What is the solution?
Sliding window
But there is still the problem of flow control

• A sender may choose a sliding window size that can overwhelm the receiver

• Why does the sender overwhelm the receiver?

• What can we do about it? Answer: Flow control
Two key ideas in flow control

• Receiver to control the senders sliding window size

• Acknowledgement and window size are decoupled
TCP flow control

The diagram illustrates the flow control process in TCP. It shows the interaction between the sender and the receiver, including the sequence of events, acknowledgment (ACK), window size (WIN), and buffer status. The diagram starts with an application sending a 2K write to the sender, followed by the sender sending a segment with a sequence number (SEQ) of 0 and a window size (WIN) of 2048. The receiver acknowledges the segment and sends an ACK with the ACK number being the sequence number plus the window size minus one. This process continues with the sender sending more data segments, and the receiver acknowledging them. The diagram also shows how the receiver's buffer fills up and the sender is blocked until the receiver reads the data. The buffer statuses are indicated as empty, full, and various data segments.
TCP Header Format

- Advertised window is used for flow control
TCP Receiver

• Sends cumulative acks - send ACK for largest frame such that all frames less than this have been received

• Maintain three state variables
  – receive window size (RWS)
  – largest frame acceptable (LFA)
  – last frame received (LFR)
TCP Receiver

• Maintain invariant: \( \text{LFA} - \text{LFR} \leq \text{RWS} \)

• Frame \text{SeqNum} \text{arrives:}
  
  – if \( \text{LFR} < \text{SeqNum} \leq \text{LFA} \) \( \Rightarrow \) accept + send ACK
  
  – if \( \text{SeqNum} \leq \text{LFR} \) or \( \text{SeqNum} > \text{LFA} \) \( \Rightarrow \) discard
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 Summary

• Sliding window is a well-known algorithm in networking
• First role is to enable reliable delivery of packets
  – Timeouts and acknowledgements
• Second role is to enable in order delivery of packets
  – Receiver doesn’t pass data up to app until it has packets in order
• Third role is to enable flow control
  – Prevents server from overflowing receiver’s buffer