TCP recap

Three phases

1. Connection setup
2. Data transfer
   • Flow control – don’t overwhelm the receiver
     • ARQ – one outstanding packet
     • Go-back-N, selective repeat -- sliding window of W packets
     • Tuning flow control (ack clocking, RTT estimation)

   • Congestion control
3. Connection release
ACK Clocking
Sliding Window ACK Clock

• Typically, the sender does not know B or D
• Each new ACK advances the sliding window and lets a new segment enter the network
  • ACKs “clock” data segments
Benefit of ACK Clocking

• Consider what happens when sender injects a burst of segments into the network
Benefit of ACK Clocking (2)

- Segments are buffered and spread out on slow link
Benefit of ACK Clocking (3)

- **ACKs** maintain the spread back to the original sender

![Diagram showing slow link and ACKs maintaining spread](image-url)
Benefit of ACK Clocking (4)

• Sender clocks new segments with the spread
  • Now sending at the bottleneck link without queuing!
Benefit of ACK Clocking (4)

• Helps run with low levels of loss and delay!
• The network smooths out the burst of data segments
• **ACK** clock transfers this smooth timing back to sender
• Subsequent data segments are not sent in bursts so do not queue up in the network
TCP Uses ACK Clocking

• TCP uses a sliding window because of the value of ACK clocking

• Sliding window controls how many segments are inside the network

• TCP only sends small bursts of segments to let the network keep the traffic smooth
Problem

• Sliding window has pipelining to keep network busy
  • What if the receiver is overloaded?
Receiver Sliding Window

• Consider receiver with W buffers
  • LAS = LAST ACK SENT
  • app pulls in-order data from buffer with recv() call
Receiver Sliding Window (2)

• Suppose the next two segments arrive but app does not call recv()
Suppose the next two segments arrive but app does not call recv()

LAS rises, but we can’t slide window!

W=5

.. Finished  Acked  .. Too high  ..

↑

LAS  seq. number
Receiver Sliding Window (4)

- Further segments arrive (in order) we fill buffer
  - Must drop segments until app recvs!

![Diagram showing sequence numbers and conditions for sliding window protocol]

W=5

Finished

Acked

Too high

Nothing
Acceptable!

.. LAS seq. number
Receiver Sliding Window (5)

• App recv() takes two segments
  • Window slides (phew)
Flow Control

• Avoid loss at receiver by telling sender the available buffer space
  • $\text{WIN} =$ #Acceptable, not W (from LAS)
Flow Control (2)

- Sender uses lower of the sliding window and flow control window \((WIN)\) as the effective window size.
Flow Control (3)

- TCP-style example
  - SEQ/ACK sliding window
  - Flow control with WIN
  - SEQ + length < ACK+WIN
  - 4KB buffer at receiver
  - Circular buffer of bytes
Topic

• How to set the timeout for sending a retransmission
  • Adapting to the network path
Retransmissions

- With sliding window, detecting loss with **timeout**
  - Set timer when a segment is sent
  - Cancel timer when ack is received
  - If timer fires, **retransmit** data as lost
Timeout Problem

• Timeout should be “just right”
  • Too long $\rightarrow$ inefficient network capacity use
  • Too short $\rightarrow$ spurious resends waste network capacity

• But what is “just right”? 
  • Easy to set on a LAN (Link)
    • Short, fixed, predictable RTT
  • Hard on the Internet (Transport)
    • Wide range, variable RTT
Example of RTTs

Round Trip Time (ms)

Seconds
Example of RTTs (2)

BCN $\rightarrow$ SEA $\rightarrow$ BCN

Variation due to queuing at routers, changes in network paths, etc.

Propagation (+transmission) delay $\approx 2D$
Example of RTTs (3)

Round Trip Time (ms)

Timer too high!

Need to adapt to the network conditions

Timer too low!
Adaptive Timeout

- Smoothed estimates of the RTT (1) and variance in RTT (2)
  - Update estimates with a moving average
    1. $\text{SRTT}_{N+1} = 0.9 \times \text{SRTT}_N + 0.1 \times \text{RTT}_{N+1}$
    2. $\text{Svar}_{N+1} = 0.9 \times \text{Svar}_N + 0.1 \times |RTT_{N+1} - \text{SRTT}_{N+1}|$
- Set timeout to a multiple of estimates
  - To estimate the upper RTT in practice
  - TCP Timeout$_N = \text{SRTT}_N + 4 \times \text{Svar}_N$
Example of Adaptive Timeout

RTT (ms)

SRTT

Svar

Seconds
Example of Adaptive Timeout (2)

Early timeout

Timeout (SRTT + 4 * Svar)
Adaptive Timeout (2)

- Simple to compute, does a good job of tracking actual RTT
  - Little “headroom” to lower
  - Yet very few early timeouts

- Turns out to be important for good performance and robustness
Congestion
TCP to date:

• We can set up and tear connections
  • Connection establishment and release handshakes
• Keep the sending and receiving buffers from overflowing (flow control)

What’s missing?
Network Congestion

• A “traffic jam” in the network
  • Later we will learn how to control it
Congestion Collapse in the 1980s

• Early TCP used fixed size window (e.g., 8 packets)
  • Initially fine for reliability

• But something happened as the network grew
  • Links stayed busy but transfer rates fell by orders of magnitude!
Nature of Congestion

- Routers/switches have internal buffering
Nature of Congestion (2)

- Simplified view of per port output queues
  - Typically FIFO (First In First Out), discard when full
Nature of Congestion (3)

• Queues help by absorbing bursts when input > output rate
• But if input > output rate persistently, queue will overflow
  • This is congestion
• Congestion is a function of the traffic patterns – can occur even if every link has the same capacity
Effects of Congestion

• What happens to performance as we increase load?
Effects of Congestion (2)

• What happens to performance as we increase load?
Effects of Congestion (3)

• As offered load rises, congestion occurs as queues begin to fill:
  • Delay and loss rise sharply with load
  • Throughput < load (due to loss)
  • Goodput << throughput (due to spurious retransmissions)

• None of the above is good!
  • Want network performance just before congestion
TCP Tahoe/Reno

• TCP extensions and features we will study:
  • AIMD
  • Fair Queuing
  • Slow-start
  • Fast Retransmission
  • Fast Recovery
TCP Timeline

3-way handshake (Tomlinson, ’75)
Origins of “TCP” (Cerf & Kahn, ’74)

TCP and IP (RFC 791/793, ’81)
TCP/IP “flag day” (BSD Unix 4.2, ‘83)
TCP Tahoe (Jacobson, ’88)
Congestion collapse Observed, ‘86

TCP Reno (Jacobson, ’90)

Pre-history Congestion control
TCP Timeline (2)

- Classic congestion control
- Diversification

**1990**
- TCP Reno (Jacobson, '90)

**1995**
- TCP New Reno (Hoe, '95)
- ECN (Floyd, '94)
- TCP Vegas (Brakmo, '93)

**2000**
- TCP with SACK (Floyd, '96)
- FAST TCP (Low et al., '04)

**2005**
- TCP BIC (Linux, '04)

**2010**
- TCP CUBIC (Linux, '06)
- TCP LEDEBAT (IETF '08)
- Compound TCP (Windows, '07)

- Delay based
- Router support

**Background**

**...**
Bandwidth Allocation

• Important task for network is to allocate its capacity to senders
  • Good allocation is both efficient and fair

• Efficient: most capacity is used but there is no congestion

• Fair: every sender gets a reasonable share of the network
Efficiency vs. Fairness

• Cannot always have both!
  • Example network with traffic:
    • $A \rightarrow B$, $B \rightarrow C$ and $A \rightarrow C$
  • How much traffic can we carry?
Efficiency vs. Fairness (2)

• If we care about fairness:
  • Give equal bandwidth to each flow
  • $A \rightarrow B$: ½ unit, $B \rightarrow C$: ½, and $A \rightarrow C$, ½
  • Total traffic carried is 1 ½ units
Efficiency vs. Fairness (3)

• If we care about efficiency:
  • Maximize total traffic in network
  • A → B: 1 unit, B → C: 1, and A → C, 0
  • Total traffic rises to 2 units!
Fairness

• What’s a “fair” bandwidth allocation?
  • The max-min fair allocation
The Slippery Notion of Fairness

• Why is “equal per flow” fair anyway?
  • A→C uses more network resources than A→B or B→C
  • Host A sends two flows, B sends one

• Not productive to seek exact fairness
  • More important to avoid starvation
    • A node that cannot use any bandwidth
  • “Equal per flow” is good enough
Generalizing “Equal per Flow”

- **Bottleneck** for a flow of traffic is the link that limits its bandwidth
  - Where congestion occurs for the flow
  - For $A \rightarrow C$, link $A$–$B$ is the bottleneck
Generalizing “Equal per Flow” (2)

• Flows may have different bottlenecks
  • For A→C, link A–B is the bottleneck
  • For B→C, link B–C is the bottleneck
  • Can no longer divide links equally ...

![Diagram showing flows and bottlenecks](image-url)
Max-Min Fairness

• Intuitively, flows bottlenecked on a link get an equal share of that link

• **Max-min fair allocation** is one that:
  • Increasing the rate of one flow will decrease the rate of a smaller flow
  • This “maximizes the minimum” flow
Max-Min Fairness (2)

• To find it given a network, imagine “pouring water into the network”
  1. Start with all flows at rate 0
  2. Increase the flows until there is a new bottleneck in the network
  3. Hold fixed the rate of the flows that are bottlenecked
  4. Go to step 2 for any remaining flows
Max-Min Example

• Example: network with 4 flows, link bandwidth = 1
  • What is the max-min fair allocation?
Max-Min Example (2)

• When rate=1/3, flows B, C, and D bottleneck R4—R5
  • Fix B, C, and D, continue to increase A
Max-Min Example (3)

• When rate=2/3, flow A bottlenecks R2—R3. Done.
Max-Min Example (4)

- End with $A=2/3$, $B, C, D=1/3$, and $R2\rightarrow R3, R4\rightarrow R5$ full
Adapting over Time

• Allocation changes as flows start and stop
Adapting over Time (2)

Flow 1 slows when Flow 2 starts
Flow 1 speeds up when Flow 2 stops
Flow 3 limit is elsewhere