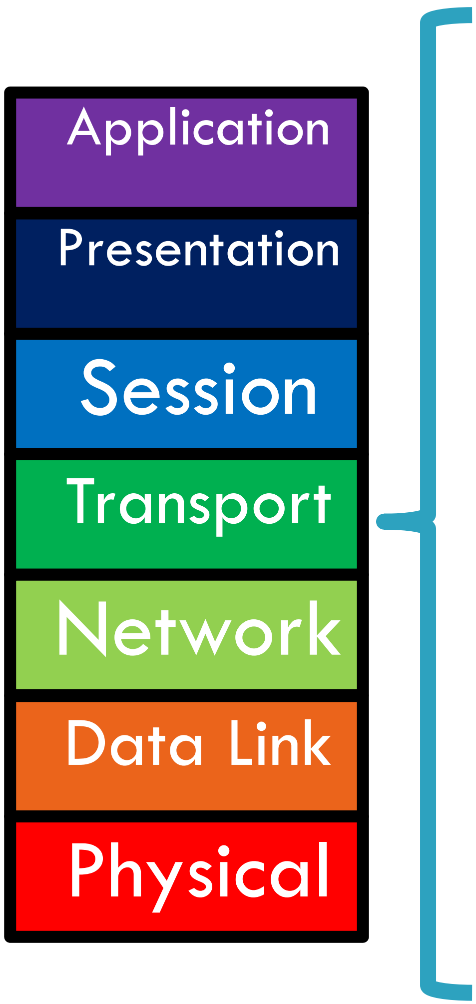
TCP: CONNECTION MANAGEMENT AND CONGESTION CONTROL

This slide is based on PPT from Northeastern University, USA

Introduction to Computer Networks
Faculty of Information Technology, KMITL
2018

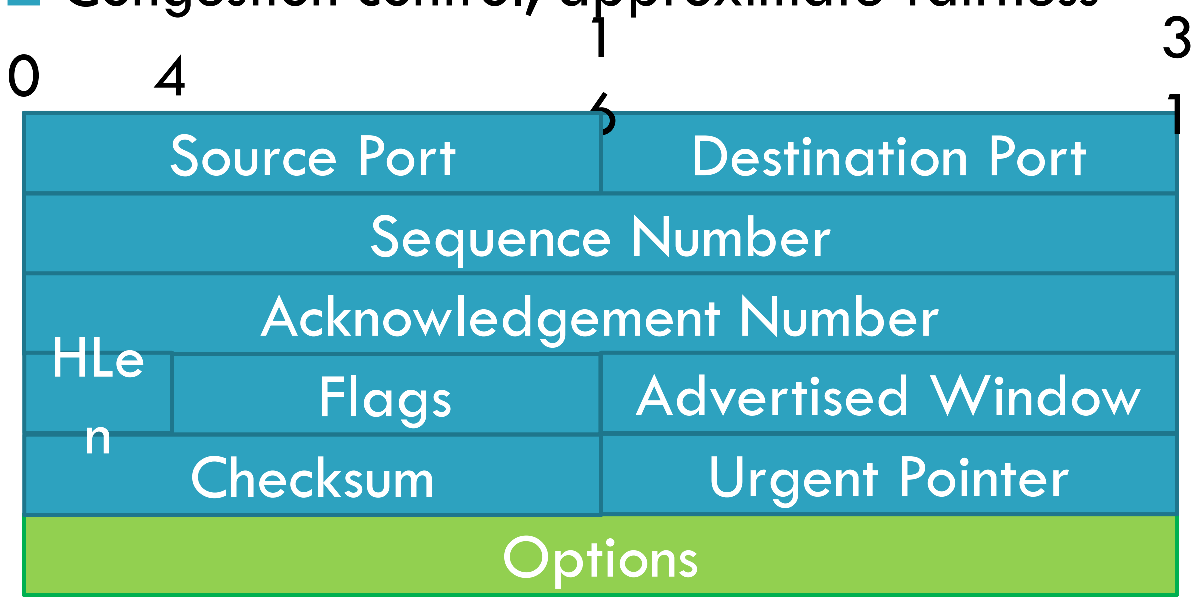
Transport Layer



- Function:
  - Demultiplexing of data streams
- Optional functions:
  - Creating long lived connections
  - Reliable, in-order packet delivery
  - Error detection
  - Flow and congestion control
- Key challenges:
  - Detecting and responding to congestion
  - Balancing fairness against high utilization

Transmission Control Protocol

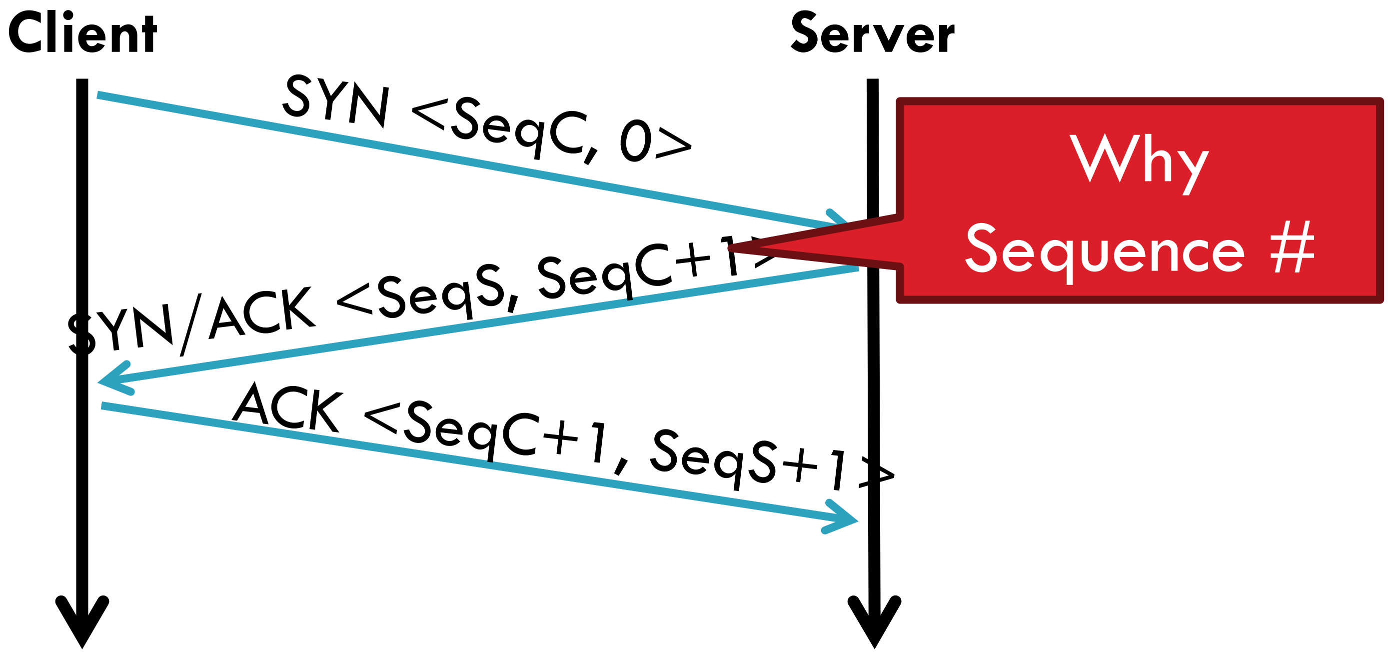
- Reliable, in-order, bi-directional byte streams
  - Port numbers for demultiplexing
  - Virtual circuits (connections)
  - Flow control
  - Congestion control, approximate fairness



Connection Setup

- Why do we need connection setup?
  - To establish state on both hosts
  - Most important state: sequence numbers
    - Count the number of bytes that have been sent
    - Initial value chosen at random
    - Why?
- Important TCP flags (1 bit each)
  - SYN – synchronization, used for connection setup
  - ACK – acknowledge received data
  - FIN – finish, used to tear down connection

Three Way Handshake



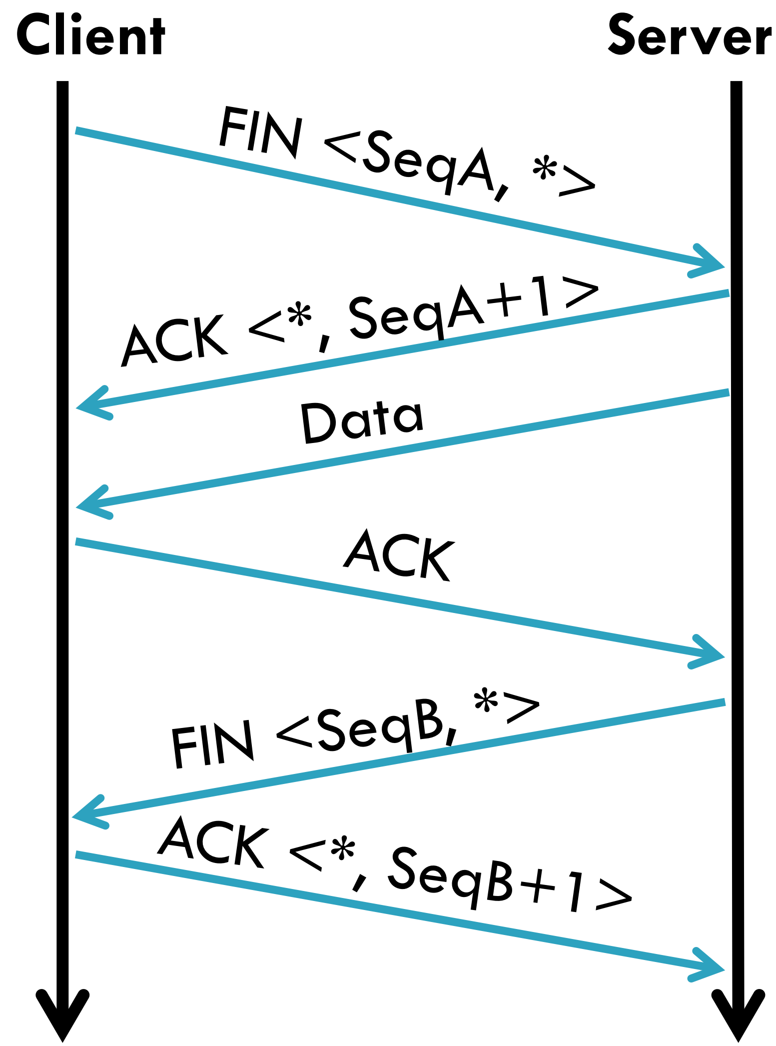
- Each side:
  - Notifies the other of starting sequence number
  - ACKs the other side's starting sequence number (+1)

Connection Setup Issues

- Connection confusion
  - How to disambiguate connections from the same host?
  - Random sequence numbers
- Packet injection
  - Need good random number generators!
- Connection state management
  - Each SYN allocates state on the server
  - SYN flood is a common denial of service attack

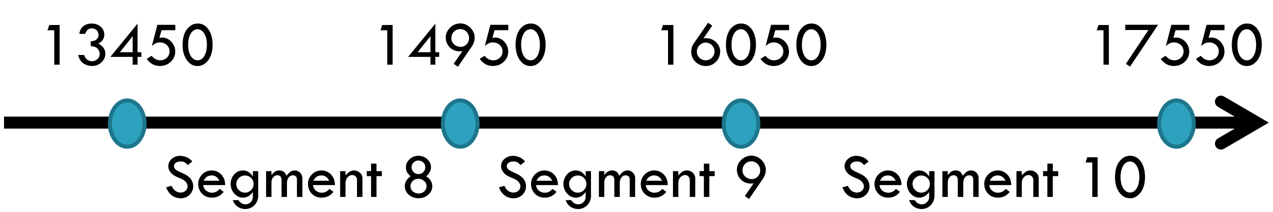
Connection Tear Down

- Either side can initiate tear down
- Other side may continue sending data
  - Half open connection
  - *shutdown()*
- Acknowledge the last FIN
  - Sequence number + 1

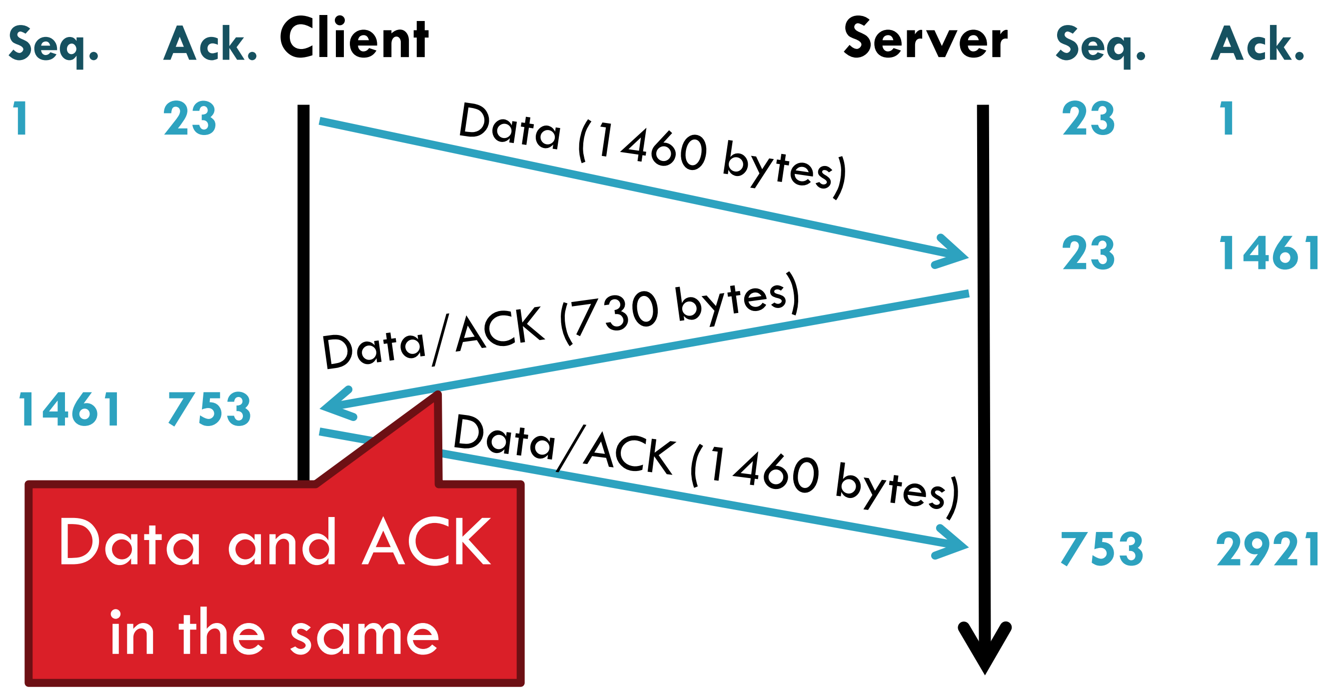


Sequence Number Space

- TCP uses a byte stream abstraction
  - Initial, random values selected during setup
  - Each byte in each stream is numbered
  - 32-bit value, wraps around
- Byte stream broken down into segments Size limited by the Maximum Segment Size (MSS, typically 1460 bytes)
  - Set to limit fragmentation
- Each segment has a sequence number



Bidirectional Communication

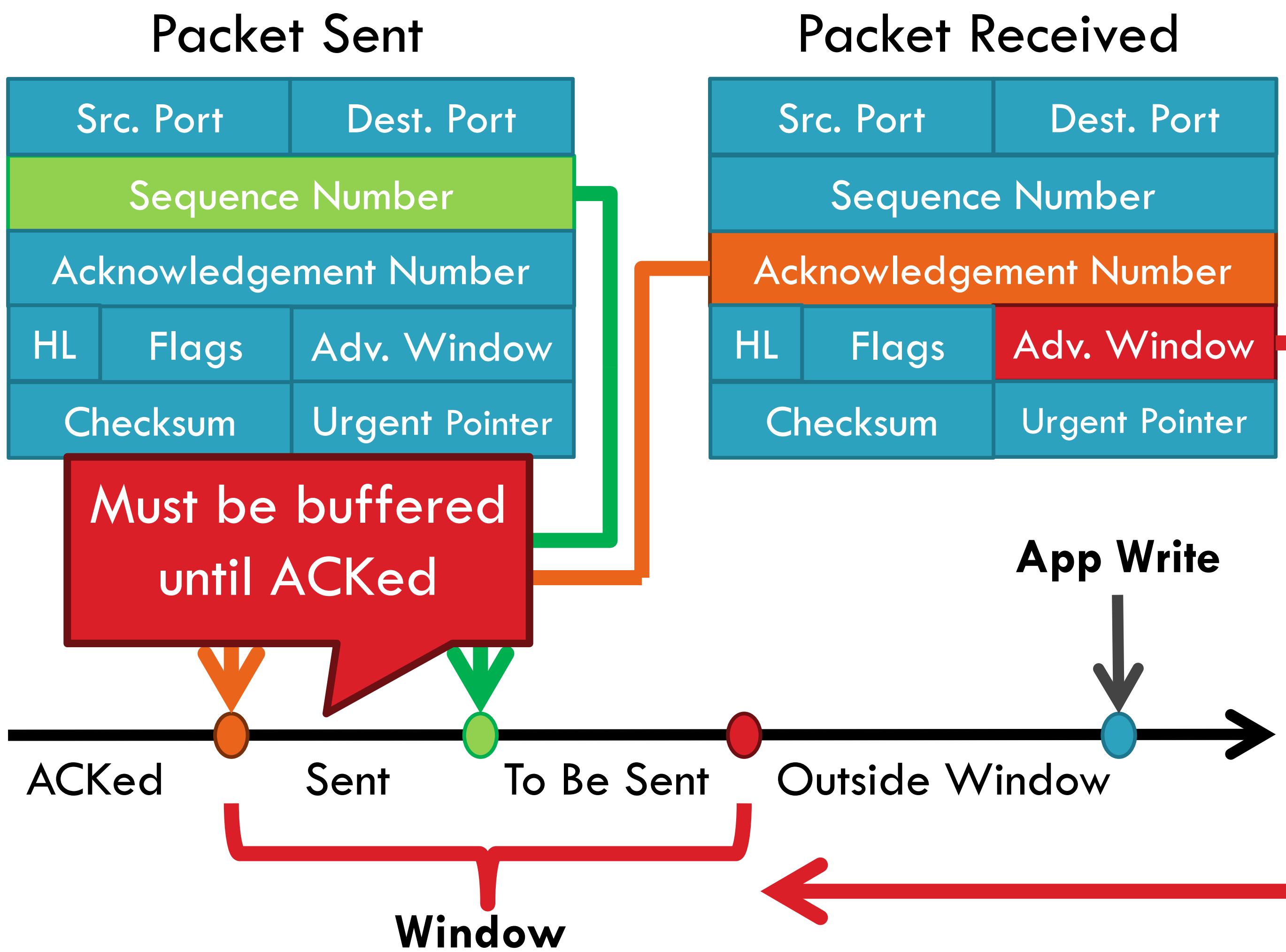


- Each side of the connection can send and receive
  - Different sequence numbers for each direction

Flow Control

- Problem: how many packets should a sender transmit?
  - Too many packets may overwhelm the receiver
  - Size of the receivers buffers may change over time
- Solution: sliding window
  - Receiver tells the sender how big their buffer is
  - Called the advertised window
  - For window size *n*, sender may transmit *n* bytes without receiving an ACK
  - After each ACK, the window slides forward
- Advertised window may go to zero!

Flow Control: Sender Side



What Should the Receiver ACK?

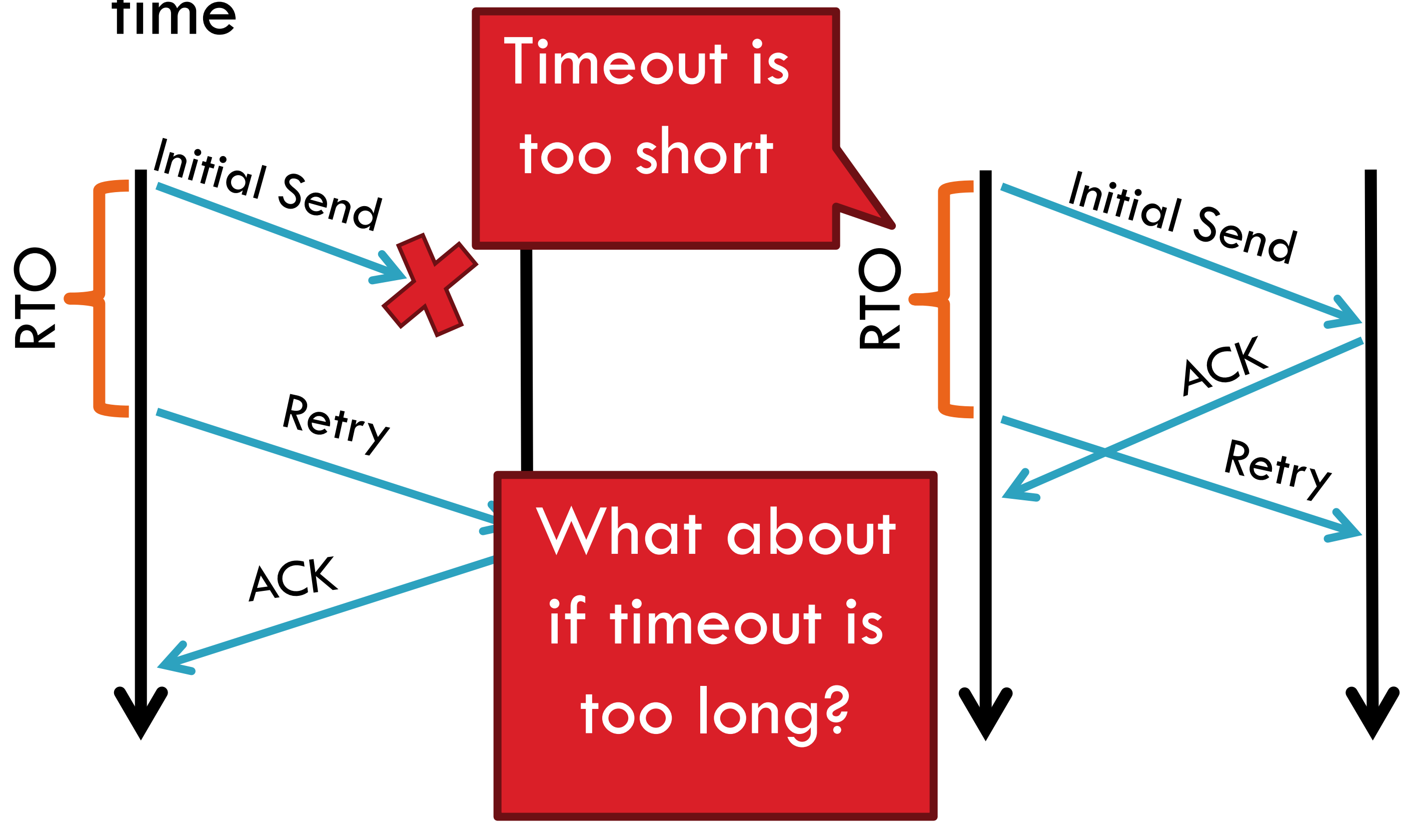
1. ACK every packet
2. Use *cumulative ACK*, where an ACK for sequence n implies ACKS for all $k < n$
3. Use *negative ACKs* (NACKs), indicating which packet did not arrive
4. Use *selective ACKs* (SACKs), indicating those that did arrive, even if not in order
   - SACK is an actual TCP extension

Error Detection

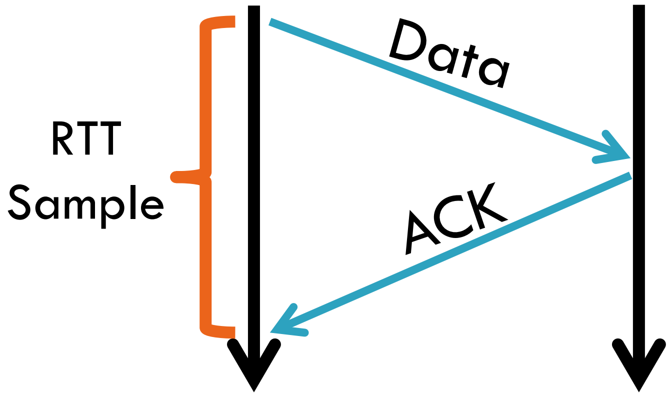
- Checksum detects (some) packet corruption
  - Computed over IP header, TCP header, and data
- Sequence numbers catch sequence problems
  - Duplicates are ignored
  - Out-of-order packets are reordered or dropped
  - Missing sequence numbers indicate lost packets
- Lost segments detected by sender
  - Use timeout to detect missing ACKs
  - Need to estimate RTT to calibrate the timeout
  - Sender must keep copies of all data until ACK

Retransmission Time Outs (RTO)

- Problem: time-out is linked to round trip time

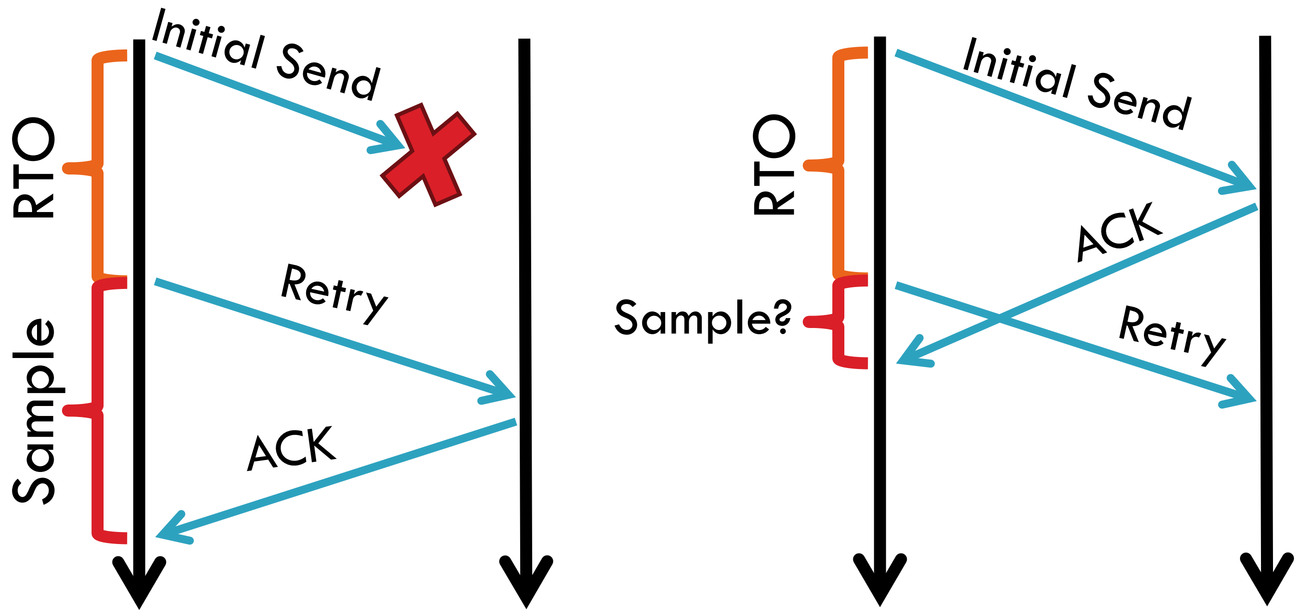


Round Trip Time Estimation



- Original TCP round-trip estimator
  - RTT estimated as a moving average
  - new_rtt = α (old_rtt) + (1 – α)(new_sample)
  - Recommended α: 0.8-0.9 (0.875 for most TCPs)
- RTO = 2 * new_rtt (i.e. TCP is conservative)

RTT Sample Ambiguity



- Karn's algorithm: ignore samples for retransmitted segments

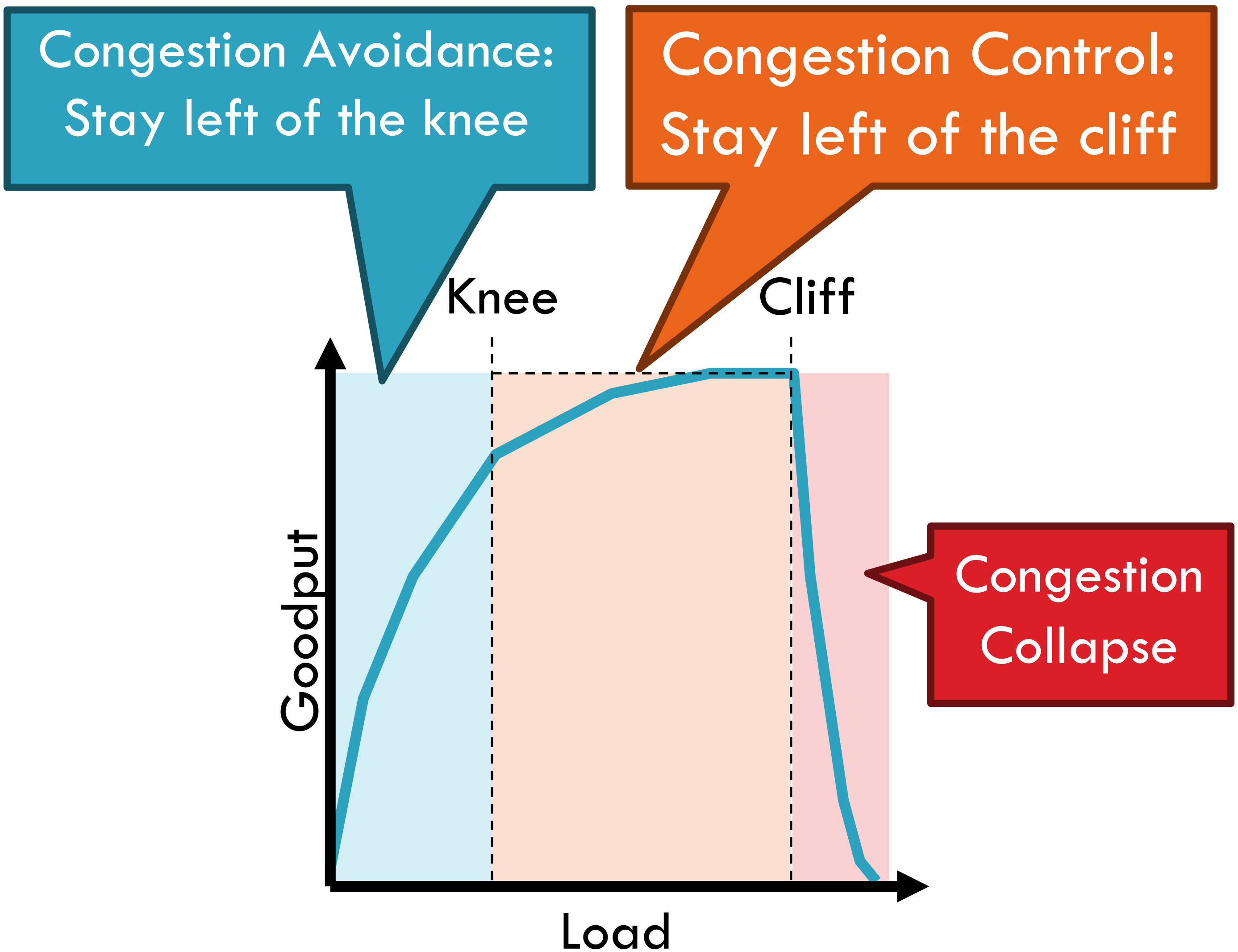
What is Congestion?

- Load on the network is higher than capacity
  - Capacity is not uniform across networks
    - Modem vs. Cellular vs. Cable vs. Fiber Optics
  - There are multiple flows competing for bandwidth
    - Residential cable modem vs. corporate datacenter
  - Load is not uniform over time
    - 10pm, Sunday night = BitTorrent Game of Thrones

Why is Congestion Bad?

- Results in packet loss
  - Routers have finite buffers, packets must be dropped
- Practical consequences
  - Router queues build up, delay increases
  - Wasted bandwidth from retransmissions
  - Low network goodput

Cong. Control vs. Cong. Avoidance



Advertised Window, Revisited

- Does TCP's advertised window solve congestion?

NO

- The advertised window only protects the receiver
- A sufficiently fast receiver can max the advertised window (i.e. 2^{16} bytes)
  - What if the network is slower than the receiver?
  - What if there are other concurrent flows?
- Key points
  - Window size determines send rate
  - Window must be adjusted to prevent congestion collapse

Goals of Congestion Control

1. Adjusting to the bottleneck bandwidth
2. Adjusting to variations in bandwidth
3. Sharing bandwidth between flows
4. Maximizing throughput

General Approaches

- Do nothing, send packets indiscriminately
  - Many packets will drop, totally unpredictable performance
  - May lead to congestion collapse
- Reservations
  - Pre-arrange bandwidth allocations for flows
  - Requires negotiation before sending packets
  - Must be supported by the network
- Dynamic adjustment
  - Use probes to estimate level of congestion
  - Speed up when congestion is low
  - Slow down when congestion increases
  - Messy dynamics, requires distributed coordination

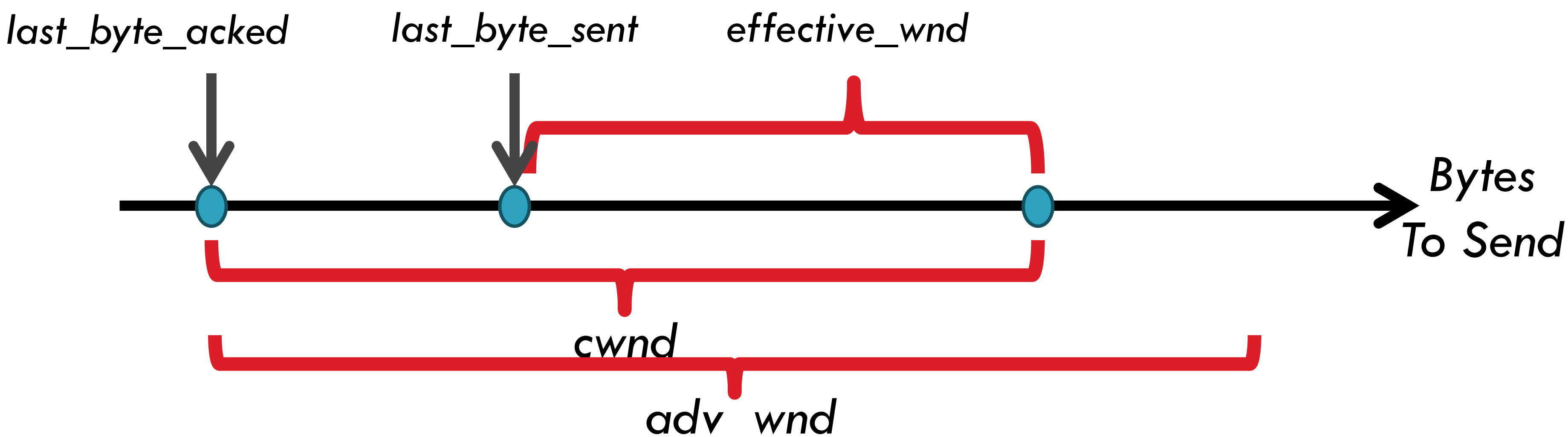
TCP Congestion Control

- Introduce a congestion window at the sender
  - Congestion control is sender-side problem
  - Controls the number of unACKed packets
  - Separate variable, but bounded by the receiver's advertised window
- Sending rate is ~ congestion window/RTT
  - Why is the send rate proportional to RTT?
  - Recall that TCP is ACK-clocked
- Idea: vary the window size to control the send rate

Congestion Window (*cwnd*)

- Limits how much data is in transit, denominated in bytes

1. *wnd* = min(*cwnd*, *adv_wnd*);
2. *effective_wnd* = *wnd* – (*last_byte_sent* – *last_byte_acked*);



Two Basic Components

1. Detect congestion
   - Packet dropping is most reliably signal
     - Delay-based methods are hard and risky
   - How do you detect packet drops? ACKs
     - Timeout after not receiving an ACK
     - Several duplicate ACKs in a row (ignore for now)
2. Rate adjustment algorithm
   - Modify *cwnd*
   - Probe for bandwidth
   - Responding to congestion

Except on wireless networks

Rate Adjustment

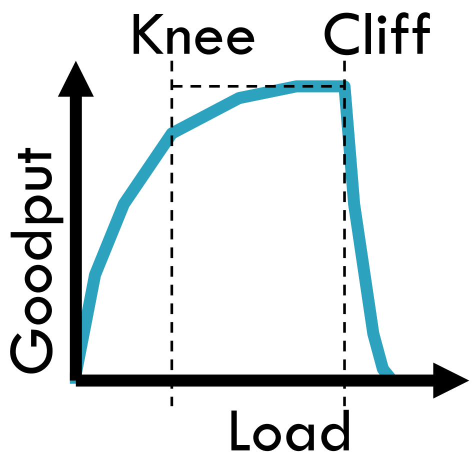
- Recall: TCP is ACK clocked
  - Congestion = delay = long wait between ACKs
  - No congestion = low delay = ACKs arrive quickly
- Basic algorithm
  - Upon receipt of ACK: increase cwnd
    - Data was delivered, perhaps we can send faster
    - *cwnd* growth is proportional to RTT
  - On loss: decrease cwnd
    - Data is being lost, there must be congestion
- Question: what increase/decrease functions to use?

Implementing Congestion Control

- Maintains three variables:
  - *cwnd*: congestion window
  - *adv_wnd*: receiver advertised window
  - *ssthresh*: threshold size (used to update *cwnd*)
- For sending, use: *wnd = min(cwnd, adv_wnd)*
- Two phases of congestion control
  1. Slow start (*cwnd* < *ssthresh*)
     - Probe for bottleneck bandwidth
  2. Congestion avoidance (*cwnd* >= *ssthresh*)
     - AIMD

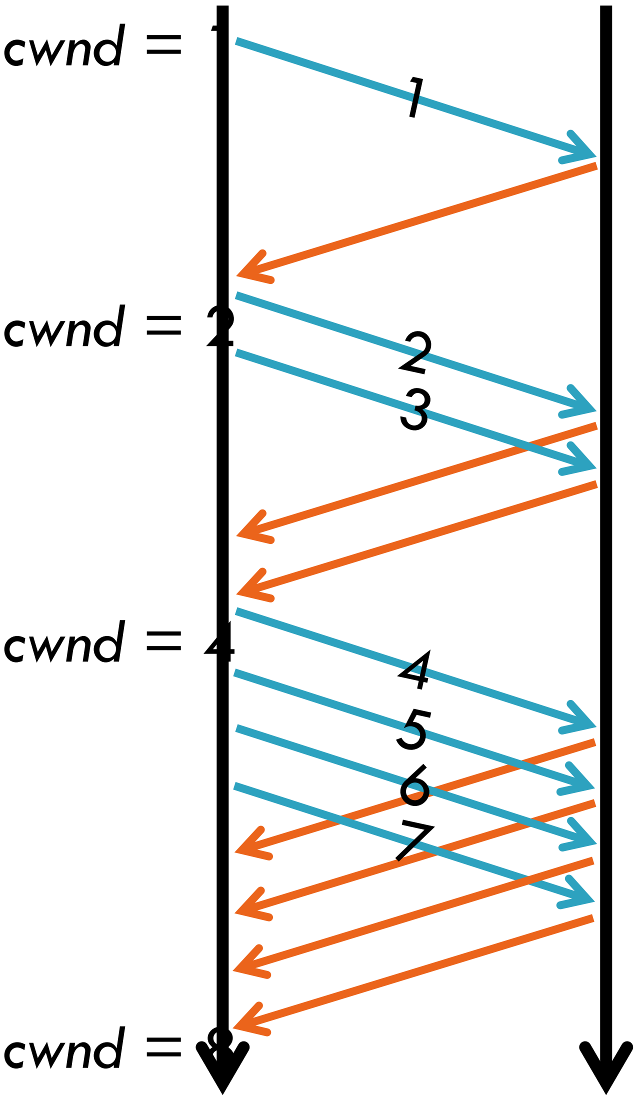
Slow Start

- Goal: reach knee quickly
- Upon starting (or restarting) a connection
  - *cwnd* =1
  - *ssthresh* = *adv_wnd*
  - Each time a segment is ACKed, *cwnd*++
- Continues until…
  - *ssthresh* is reached
  - Or a packet is lost
- Slow Start is not actually slow
  - *cwnd* increases exponentially



Slow Start Example

- *cwnd* grows rapidly
- Slows down when…
  - *cwnd* >= *ssthresh*
  - Or a packet drops



Congestion Avoidance

- AIMD mode
- *ssthresh* is lower-bound guess about location of the knee
- **If** *cwnd* >= *ssthresh* **then**
  each time a segment is ACKed *cwnd* += 1/*cwnd*
- So *cwnd* is increased by one only if all segments have been acknowledged

Congestion Avoidance Example