Transport Layer (Layer 4)

- **Sits between the application and network layers.**
- **Network layer (e.g., IP) provides basic addressing and routing service.**
  - Best effort.
  - No guarantee of delivery, integrity of datagram, ordering or duplicate removal.
- **Transport layer (e.g., UDP and TCP) provides other basic services needed by application**
  - Delivery to process rather than host (application multiplexing/demultiplexing) using port numbers.
  - Error checking. Error detection fields (checksum) in headers.
  - UDP does little more than this. TCP does a lot more.

Key Features of TCP

- **Connection oriented**
  - Establishes an end-to-end connection using a 3-way handshaking protocol before transfer.
- **Point to Point – two end points**
- **Reliable transfer**
  - Uses end-to-end acknowledgments and retransmissions
  - Ordering. Duplicate removal.
- **Full duplex**
  - One connection establishment necessary for two-way data transfer.
- **Stream oriented**
  - Continuous sequence of octets
  - User has no control over packetization
  - MSS: Maximum segment size. Implementation dependent.

Key Features of TCP

- **Flow control**
  - Sender will not overwhelm receiver.
- **Congestion control**
  - Sender will not overwhelm network.
- **Reliable connection of startup.**
  - Data on old connection does not confuse new connection.
- **Graceful connection shutdown**
  - Data sent before closing a connection is not lost.
- ***Assumption: you should be somewhat familiar with all the above features.***

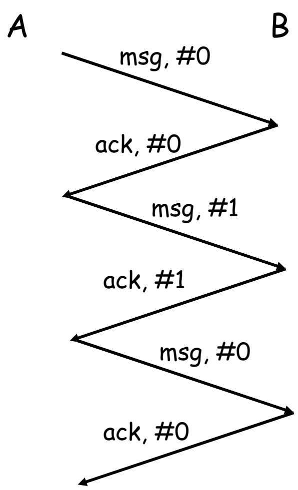
Automatic Repeat/reQuest (ARQ)

- **Motivation: congestion/flow control intertwined with reliable transport**
- **Basis for most reliable transport schemes**
- **Relies on acknowledgments (ACK) and timeouts**
- **Source sends packet**
- **Receiver ACKs each packet**
- **If data *or* ACK lost, timeout triggers and source re-transmits**
- **Simplest version: Stop-and-Wait**

What if ACK is Dropped?

- Receiver will receive duplicate packets, but unaware of this problem.
- Use sequence numbers.
- How many bits in sequence numbers? (Size of sequence number space?)

Alternating Bit Protocol

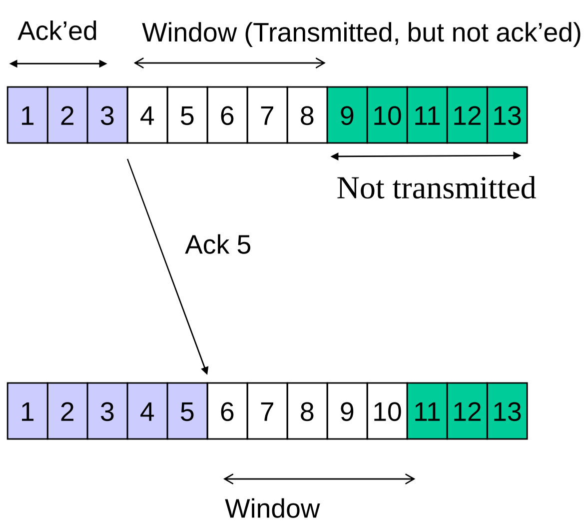


- 1-bit sequence number. Stop-and-wait protocol.
- Why is this inefficient?
- Consider 1Mb/s link, 100ms path delay, 1000 byte packets
- Send rate is only 40kb/s << 1Mb/s!

Sliding Window

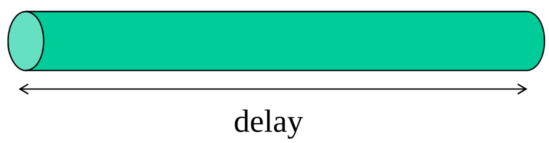
- **Pipelining: transmit next packet before current one ACK'd**
- **Window limits the amount of outstanding data**
- **How large should this window be?**
  - Another question: How many bits in sequence numbers?
- **Window size = Twice bandwidth-delay product**
  - Keep the "pipe" full

Window Based Congestion Control



Ideal Window Size

- **Ideal window size = Twice (delay * bandwidth) product of the network.**
- **View the network as a "pipe" with two parameters**
  - End to end delay for each bit (latency)
  - Bandwidth (rate at which data can be pumped in)
- **How much data can be pumped into the pipe while the first bit is still in flight?**
  - = delay*bandwidth
  - This is the max amount that can be in flight.



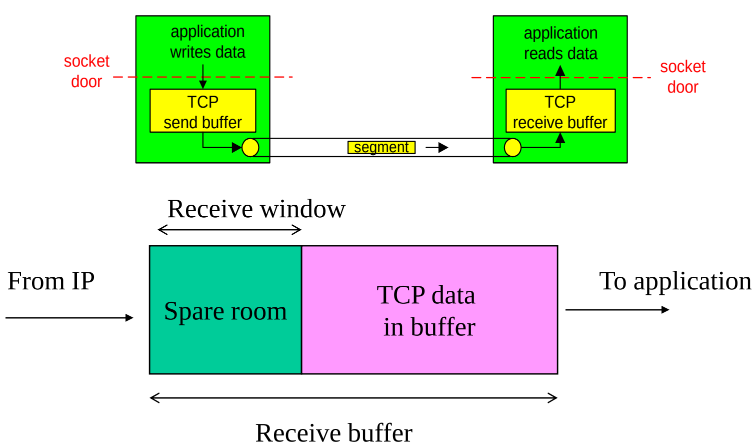
Window Based Congestion Control

- **Window bounds the amount of data that can be in flight.**
- **Throughput <= W / RTT**
  - Where, W = window size and RTT = round-trip time.
  - Sends no more than W before the first ack comes back, which will take RTT time.

Ideal Window Size

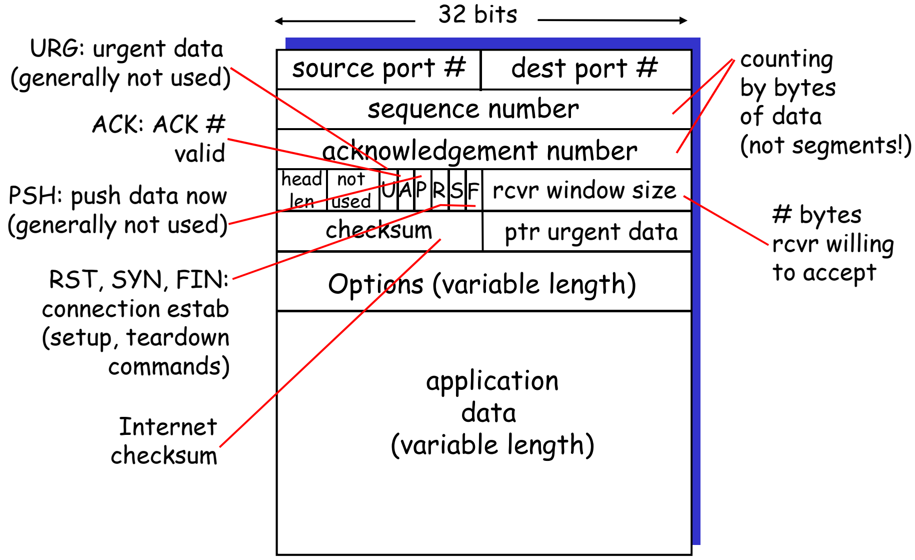
- **What if window size > 2 * delay * bandwidth?**
  – More data in flight than network can support.
  – Increased queuing at routers. Increased RTT should reduce amount of data in flight.
  – Will eventually lead to packet drops. Downsize window.
- **What if window size < 2 * delay*bandwidth**
  – Network can support more data in flight.
  – Inefficient (wasted bandwidth).
  – Step up window size as long as one window worth of data can be acknowledged without problem.

TCP Send & Receive Buffers, Receive Window



- **Note that each side maintains independent send and receive buffers.**

TCP Header Format



TCP Sequence Numbers and Acknowledgments

- **TCP uses byte sequence numbers**
  - Number is for the first byte in the TCP segment.
  - We may use packet sequence no.s to simplify examples.
- **Acknowledgment is cumulative**
  - ACK seq no. *n* means all bytes up to and including byte *n-1* have been received; and receiver is expecting the next segment with seq no. *n*.
  - We may use ACK *i* to mean than packet *i* has been received to simplify examples.

TCP Acknowledgments

- **Delayed acknowledgment**
  - Typically only alternate segments are acknowledged.
- **Exceptions:**
  - ACK Timer expiry (typically 200ms).
  - Receipt of out-of-order segments.
- **Duplicate acknowledgments for these exceptions.**
  - Note ACK is always cumulative.

TCP Sender Side

- **Sender sets up a retransmission timer for each segment transmitted.**
  - Timer is based on round-trip time estimate. Dynamically calculated.
- **If no ACK before timer expiry, sender retransmits segment.**
- **Sender also retransmits, if it receives 3 consecutive dupacks (fast retransmit)**

Calculation of the Retransmission Timeout (RTO)

- **RTO = estimated RTT + 4 * estimated deviation.**
  - Deviation = average of |sample – mean|
  - Note, std. deviation is sq root of average of (sample – mean)^2.
  - Deviation is easier to calculate than std. deviation.
- **Estimated RTT = weighted average of sample RTTs**
  - Estimated RTT = (1-x) * estimated RTT + x * sample RTT.
  - Similarly, estimated deviation = (1-x) estimated destination + x * |sample RTT – estimated RTT|
- **RTO is measured in discrete, large grain clock ticks (typically, 500ms, but tends to be finer in some recent stacks).**
- **RTO is doubled after timeout (exponential backoff).**

Fast Retransmit

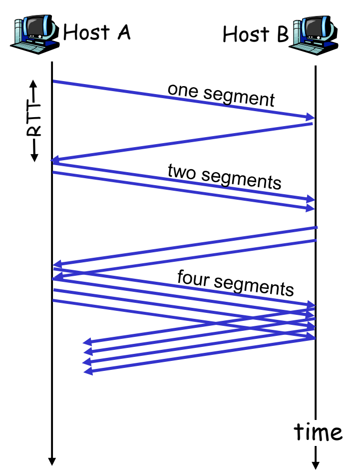
- **Timeouts may take too long. How to generate retransmissions quickly?**
- **Dupcaks indicate missed reception of a segment. Could be packet loss or packet delay.**
- **TCP sender retransmits after 3 dupacks.**
- **Note may not indicate packet loss. Makes sense only when the lower layer delivers packets "almost ordered."**

Window Based Flow and Congestion Control

- **Sliding window protocol. Window determines amount of unacknowledged data.**
  - Size controlled dynamically.
- **Window is minimum of congestion window and receiver window (advertised by receiver on Acks).**
  - Receiver window – how much more receiver can take.
  - Congestion window – how much more network can take.

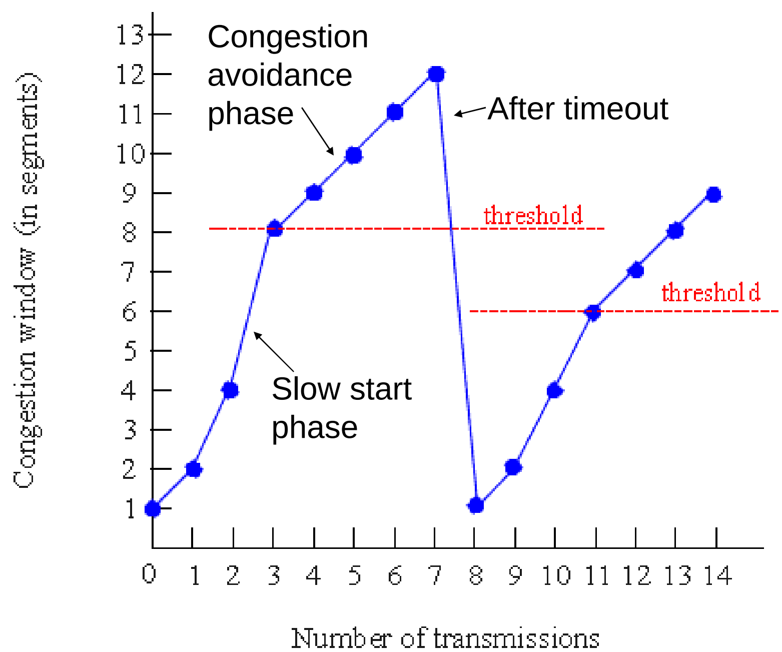
TCP Slow Start

- Initially, congestion window cwnd = 1 (in MSS unit).
- cwnd = cwnd + 1 after each ack.
- Until, cwnd > slow start threshold (ssthresh) or packet loss.
- Exponential increase in cwnd per RTT (not so slow!)
  - Factor of 2 if every segment is ack'ed.
  - Factor of 1.5 if alternate segments are ack'ed.



TCP Congestion Avoidance

- Slow start is over when cwnd > ssthresh.
- cwnd = cwnd + 1/cwnd after each ack.
  - That is, after a whole window worth of segments are acked cwnd is incremented by 1.
  - Linear increase per RTT -- by 1 if every segment is acked, by ½ if alternate segments are acked.
- After timeout, ssthresh = half of window size before packet loss.
  - More precisely, ssthresh = min(rcvwnd,cwnd)/2. Min. must be 2.
- Initiate slow start with cwnd = 1.
- Note slow start will continue until the half-way point where congestion occurred last time.



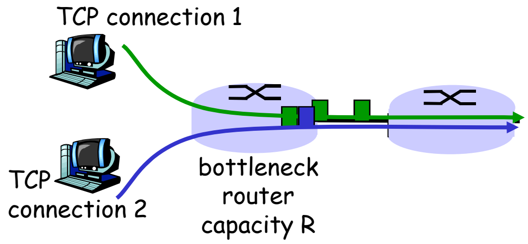
Various Flavors of TCP

- **TCP has a long history of refinement.**
- **TCP Tahoe is original TCP.**
  - Does slow start and congestion avoidance.
  - Sender only retransmits after timeout.
- **TCP Reno adds fast retransmit and fast recovery.**
  - Fast retransmit = Also retransmit after three consecutive dupacks regardless of timer.
  - Fast recovery = No slow start after fast retransmit.

Congestion Control in TCP Reno

- **Fast Retransmit**
  - Several consecutive dupacks may indicate a “low level” of congestion. Some segments are getting through, but some are missing.
  - Should retransmit the missing segment, but no need to scale back cwnd too much.
- **Fast Recovery**
  - Follows after fast retransmit.
  - ssthresh is set as before.
  - cwnd = ssthresh + #dupacks.
  - Enter congestion avoidance phase directly.
  - Note cwnd size is cut in half.

AIMD (Additive Increase, Multiplicative Decrease)

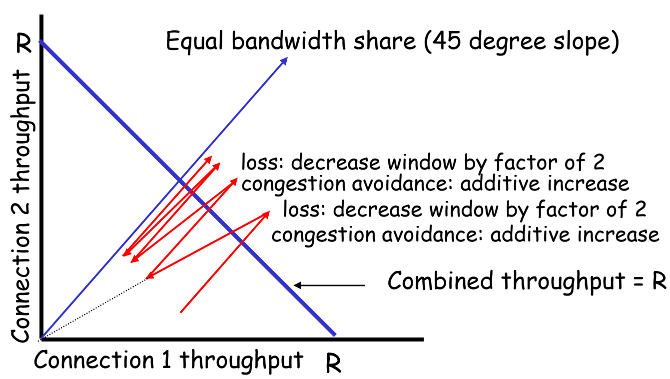


- TCP Reno – AIMD style control of congestion window.
- AIMD can be shown to converge to a "fair" state.
- "Fairness" goal: *If N TCP sessions share a bottleneck router, each should get 1/N of the router capacity*
  - assuming that each session has at least that much demand.

Back to TCP: AIMD Ensures "fairness"

Two competing sessions:

- **Additive increase gives slope of 1, as throughout increases**
- **multiplicative decrease decreases throughput proportionally**



Some Analysis

- **TCP throughput <= W / RTT. Actually, on average 0.75W/RTT.**
- **Assume, packet loss probability is p**
- **For each transmitted segment**
  - Segment delivered with prob. (1-p), window increases by 1 / W.
  - Segment lost with prob. p, window decreases by ½ W.
- **At steady state, (1-p)*(1/W) = p* ½ *W**
- **For small p, upper bound of throughput is inversely proportional to √p and RTT**
- **Weakness of TCP: Throughput very sensitive to delay and loss**
  - Satellite link has long delay
  - Wireless links may have intermittent losses, unrelated to congestion

Approaches for Congestion Control

- **TCP's approach is implicit.**
  - Probe a "black box" (the network).
  - Infer congestion from end system observed loss or delay.
- **Explicit congestion control**
  - Use "feedback" from the network elements.
  - Tell sender the max. sending rate to avoid congestion.
  - Need "intelligent" network elements (e.g., routers).

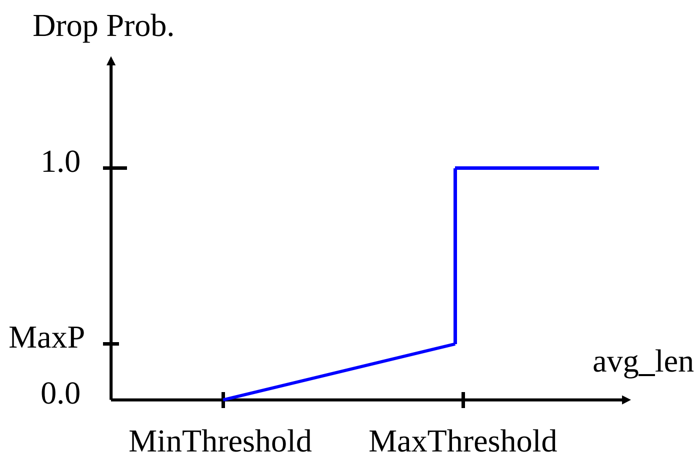
Heuristics for Congestion Avoidance

- **Still implicit, but does not wait for packet loss. Use cwnd and RTT stats to infer congestion.**
- **Example: TCP Vegas**
  - Expected throughput = Wnow / RTTmin
  - Actual throughput = Wnow / RTTnow
  - Expected – Actual > β indicates congestion. Backoff.
  - Expected – Actual < α indicates possible additional capacity. Probe.
  - Condition: $\alpha < \beta$. Also, if Expected < Actual, refine RTTmin estimate.

Explicit Congestion Control: RED

- **Random early detection (RED) technique built into the router.**
- **Idea: Drop packets randomly with small probability if router queue is close to full.**
- **Advantage: Helps source to fold back congestion window earlier. Less packet losses than drop-tail queues.**
- **Implementation**
  - Maintain a weighted running average of queue length (avg_len).
  - Drop an arriving packet with prob. p depending on the relationship of avg_len with two preset thresholds.

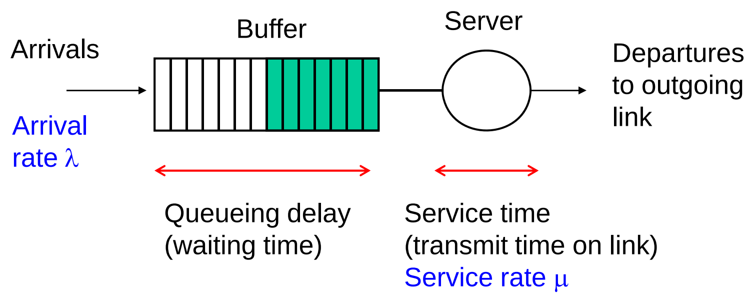
RED Details



Explicit Congestion Notification (ECN)

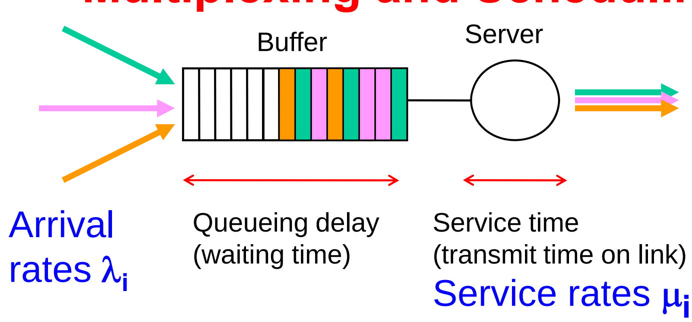
- **Idea: Don't drop packets. Notify sender directly about incipient congestion.**
  - Similar idea used in non-IP networks. Examples: DECbit scheme in DNA, Resource management (RM) cells in ATM.
- **Use ECN field in IP header. Router marks the ECN field in packets when average queue size is too high.**
  - Marking only a fraction of packets using a probabilistic mechanism like RED is sufficient.
- **Receiver echoes back the ECN marks on Ack packets.**
- **Source cuts back both ssthresh and cwnd by half (?) on receiving ECN marked Ack.**
  - Source enters congestion avoidance.
  - Source does not respond to ECN until all outstanding packets are ack'ed.

Queuing in a Router or Switch



- **Assume, arrival rate (λ) < service rate (μ).**
  - Needed for stability.
- **Link utilization (ρ) = λ/μ**
  - Mean service time = $1/\mu$.
  - In unit time, λ packets are transmitted, each taking $1/\mu$ time on average.

Multiplexing and Scheduling



- **A queue can multiplex many "flows" or "connections."**
- **How the server should "schedule" packets from different flows?**
  - Scheduling = mechanism to choose the next packet for transmission. Specifies how the resource (link) should be shared.
  - First-In-First-Out ??

Best Effort and Guaranteed Service

- **Best effort service = No guarantees. Used by adaptive applications.**
  - Example: email, file transfer.
- **Guaranteed service = specific service guarantees. Needed**
  - For example, bandwidth, delay, delay jitter, or drop or a combination.
- **Multimedia applications typically will not perform meaningfully if no guarantee**
  - Example: Interactive voice needs about 64 kbps BW and 150 ms delay bound.