

Transmission Control Protocol

ITS 413 – Internet Technologies and Applications

Contents

- Overview of TCP (Review)
- TCP and Congestion Control
 - The Causes of Congestion
 - Approaches to Congestion Control
 - TCP Congestion Control
- TCP Performance Issues
 - Window Size
 - RTT Estimation
 - Fairness

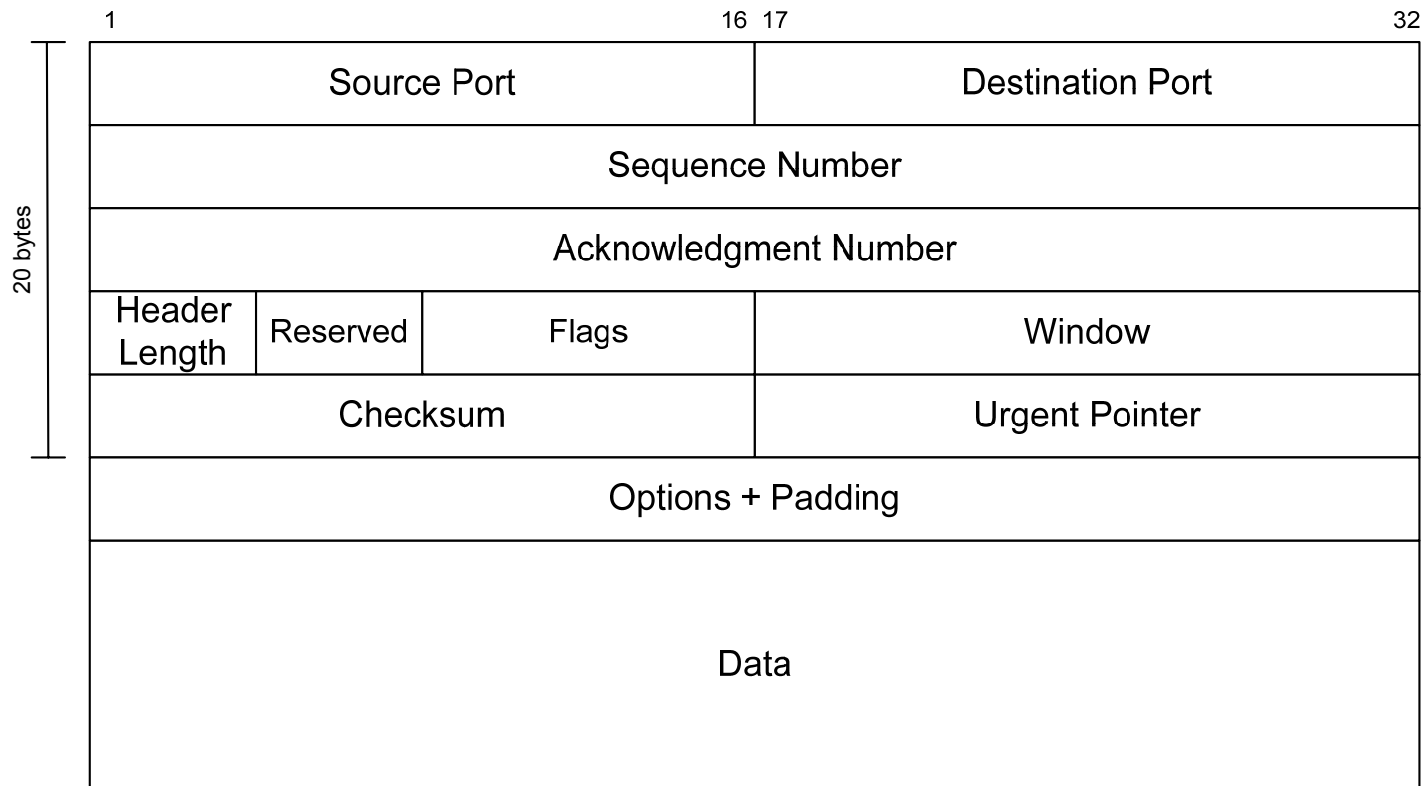
Overview of TCP

Transmission Control Protocol

- The most commonly used transport protocol today
 - Almost all Internet applications that require reliability use TCP
 - Web browsing, email, file sharing, instant messaging, file transfer, database access, proprietary business applications, some multimedia applications (at least for control purposes), ...
- TCP provides a reliable, stream-oriented transport service:
 - Stream of bits (or bytes) flow between end-points
 - Stream is unstructured
 - Connection-oriented data transfer
 - Set up a connection before sending data
 - Buffered transfer
 - Applications generate any sized messages
 - TCP may buffer messages until large datagram is formed
 - Option to force (push) the transmission
 - Full duplex connection
 - Once the connection is setup, data can be sent in both directions
 - Reliability
 - Positive acknowledgement with retransmission

TCP Segment

- Header contains 20 bytes, plus optional fields
 - Optional fields must be padded out to multiple of 4 bytes



TCP Segment Fields

- **Source/Destination port:** 16 bit port number of the source/destination
- **Sequence number** of the first data byte in this segment
 - Unless the SYN flag is set, in which case the sequence number is the Initial Sequence Number (ISN)
- **Acknowledgement number:** sequence number of the next data byte TCP expects to receive
- **Header Length:** Size of header (measured in 4 bytes)
- **Reserved** for future use
- **Flags** see next slide
- **Window** contains the number of bytes the receiver is willing to accept (for flow control)
- **Checksum** for detecting errors in the TCP segment
- **Urgent pointer** points to the sequence number of the last byte of urgent data in the segment
- **Options:** such as maximum segment size, window scaling, selective acknowledgement, ...

TCP Segment Flags

- Flags (1 bit each, if 1 the flag is true or on):
 - CWR: Congestion Window Reduced
 - ECE: Explicit Congestion Notification Echo
 - CWR and ECE are used on a special congestion control mechanism – we do not cover this in ITS 323
 - URG: segment carries urgent data, use the urgent pointer field; receiver should notify application program of urgent data as soon as possible
 - ACK: segment carries ACK, use the ACK field
 - PSH: push function
 - RST: reset the connection
 - SYN: synchronise the sequence numbers
 - FIN: no more data from sender
- Note
 - There is only one type of TCP packet
 - However the purpose of that packet may differ depending on the flags set
 - If SYN flag is set, we may call it a “SYN packet or TCP SYN”
 - If the ACK flag is set, we may call it a “ACK packet”
 - If the packet carries data, we may call it a “DATA packet”
 - If the packet carries data and the ACK flag is set, it is both a DATA and ACK packet

Main TCP Features

- Connection Management
 - Aim: Initialise parameters for data transfer
 - Setup a connection before sending data
 - Teardown a connection when finished
- Reliability
 - Aim: ensure all data is delivered intact, in-order to receiver
 - Sequence numbers
 - Re-transmission schemes
 - Basic (retransmit after timeout), Fast Retransmit (retransmit after receiving 3 duplicate ACKs)
- Flow Control
 - Aim: ensure the sender does not overflow the receiver
 - Receiver indicates free space in buffer in Advertised Window of ACK
 - Sender cannot send more than the Advertised Window
- Congestion Control
 - Aim: ensure the sender does not overflow the network (routers)

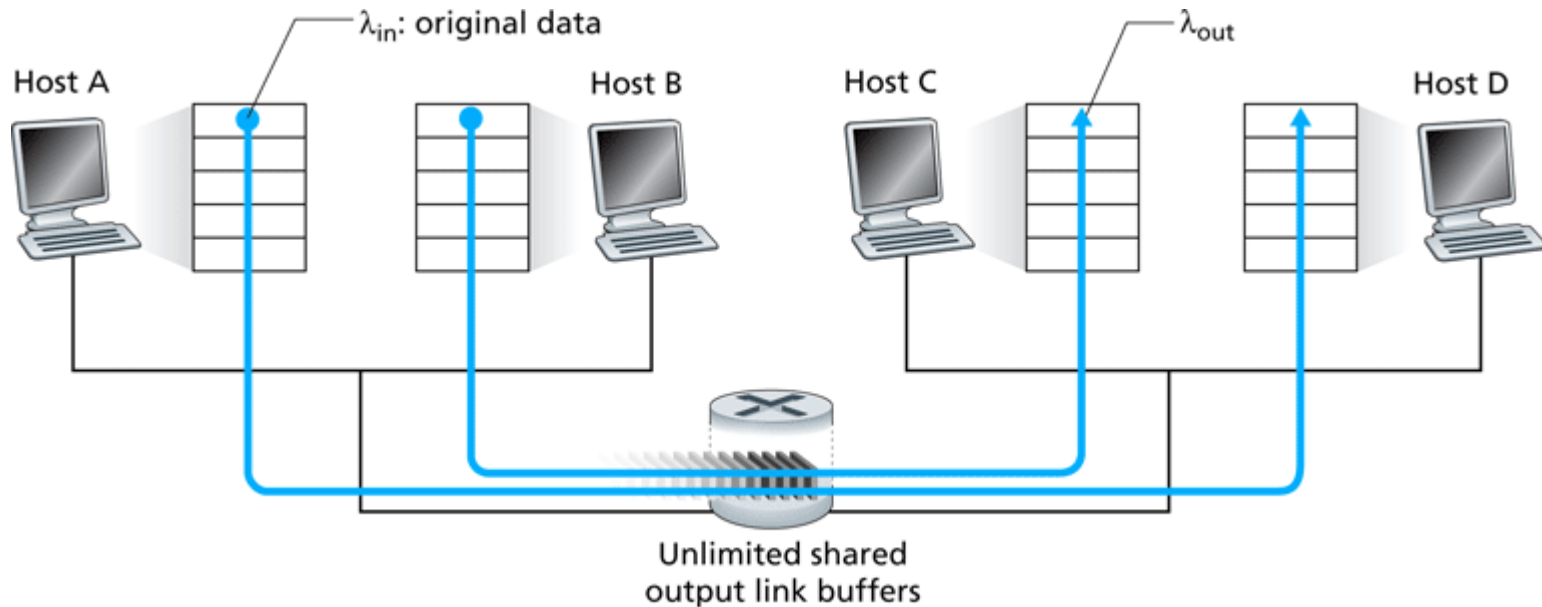
The Causes of Congestion

What is Congestion?

- Congestion occurs when the number of packets being transmitted through the network approaches the packet handling capacity of the network
 - What is the packet handling capacity of a network?
 - What happens when capacity is approached?
- Congestion control aims to keep number of packets below a level at which performance falls off dramatically
 - How to keep number of packets below level?
- Congestion is caused by too many sources trying to send data at too high a rate
 - In IP networks, this typically results in routers dropping packets
 - For TCP, lost packets (and larger delay) result in retransmissions
 - Retransmission cause more congestion, and more packet losses, and more retransmissions, ...
- Congestion control aims to reduce the rate at which sources send

Congestion Scenario 1

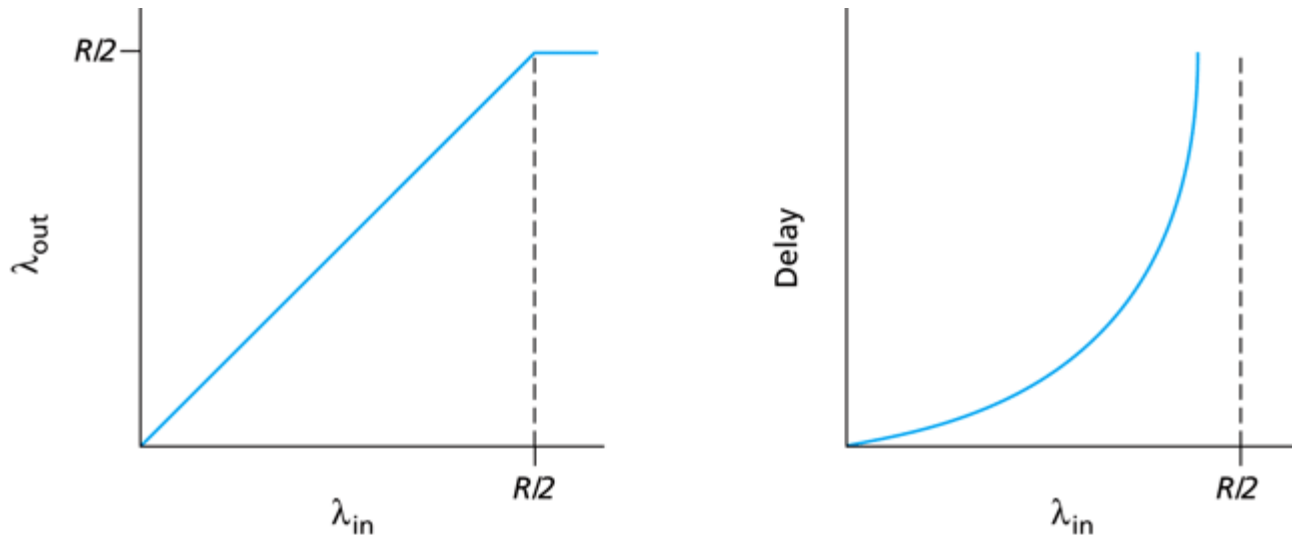
- Two senders (and receivers); a router with infinite buffers



- Router outgoing link has capacity R
- Host A sends packets to router at rate λ_{in} bytes per sec; so does Host B
- Router has infinite buffer space to store packets when input rate exceeds output rate
- λ_{out} is throughput for a connection

Congestion Scenario 1

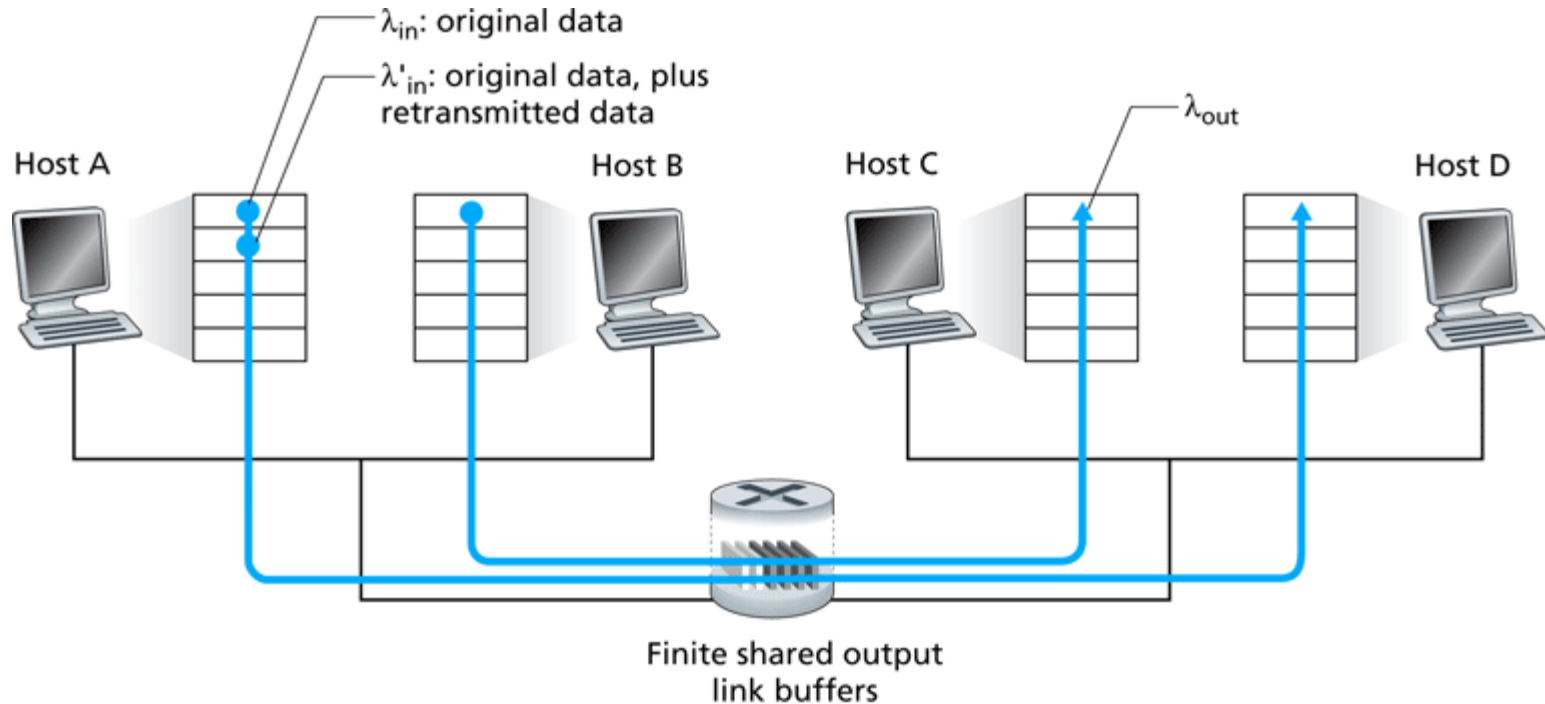
- Plot of throughput (λ_{out}) and delay for each connection



- While sending rate is less than output capacity at router, each connection achieves full sending rate in throughput
- When sending rate is greater than output capacity at router, each connection is limited to half of router output capacity
- However, as sending rate approaches output capacity, delay rises significantly (packets must wait in router buffer)

Congestion Scenario 2

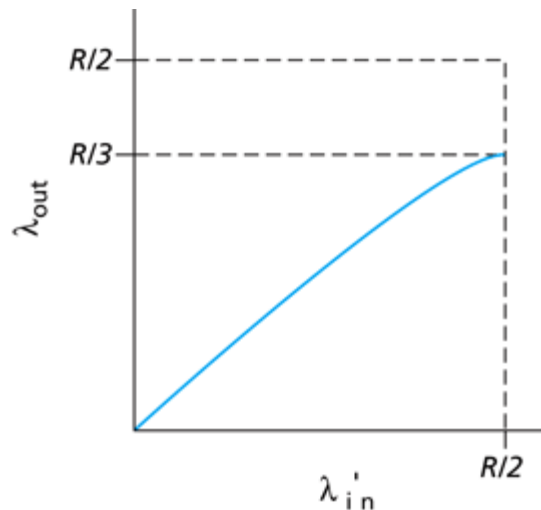
- Two senders (and receivers); a router with finite buffers



- If buffer is full and new packets arrive, packets will be dropped/lost
- Lost packets leads to retransmissions by the source hosts
- λ'_{in} is the *offered load*: original sending rate + retransmission rate

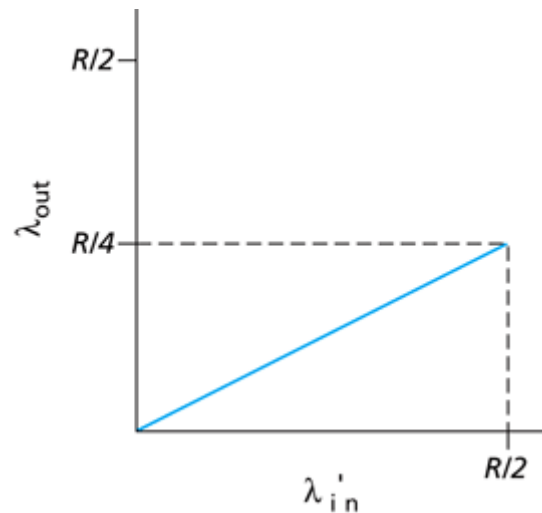
Congestion Scenario 2

- Lets assume source retransmits only when a packet is known to be lost
 - Offered load (λ'_{in}): original sent + retransmissions
 - If every second packet is lost (due to buffer at router being full):
 - For every 2 original packets, 1 retransmitted packet;
 - Offered load is 3 packets
 - 2 packets successfully received at destination

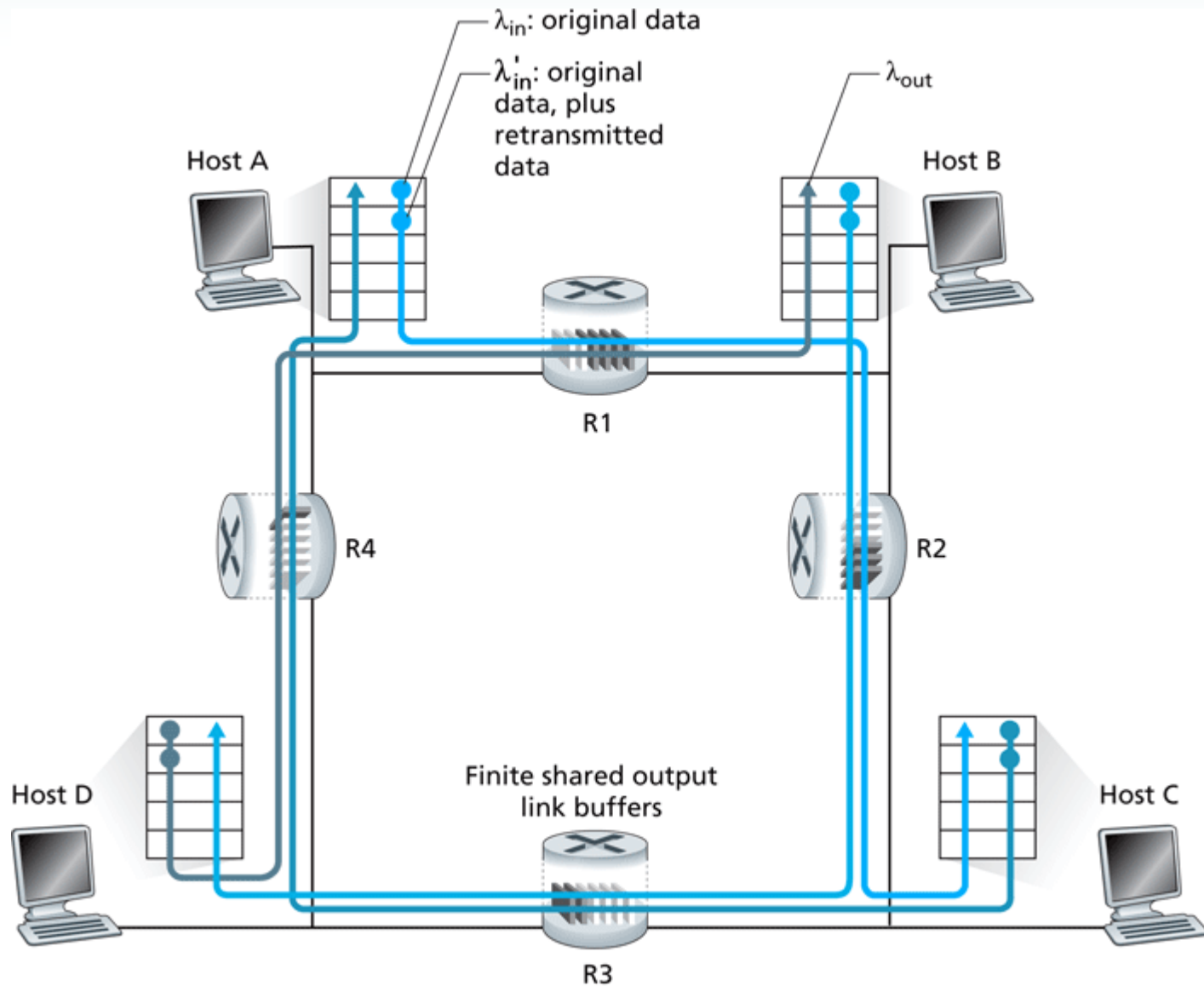


Congestion Scenario 2

- Lets assume a sender times out too early and retransmits a packet even though the original packet was sent by the router to destination
 - The output link from the router will be used to send the original and retransmitted packet
 - But the retransmitted packet will be ignored (discarded) by the destination

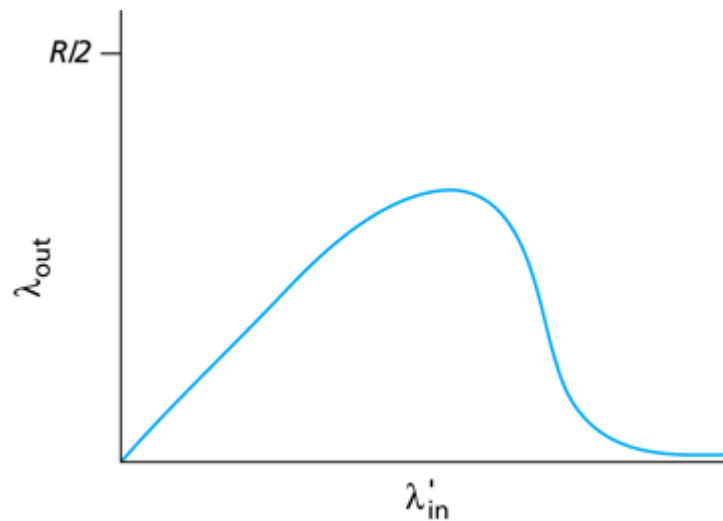


Congestion Scenario 3



Congestion Scenario 3

- Throughput can go to 0 with large amount of traffic
- Network spends all the time sending unneeded/wasted packets



Costs of Congestion

- Large queuing delays are experienced as the sending rate nears the output link capacity at a router
- Sender must perform retransmissions in order to compensate for dropped (lost) packets due to buffer overflow
- Router may send unneeded copies of packets if sender retransmits due to large delays (but not lost packets)
- With multiple routers in a path, if a packet is dropped by a router, all links leading up to that router have been wasted

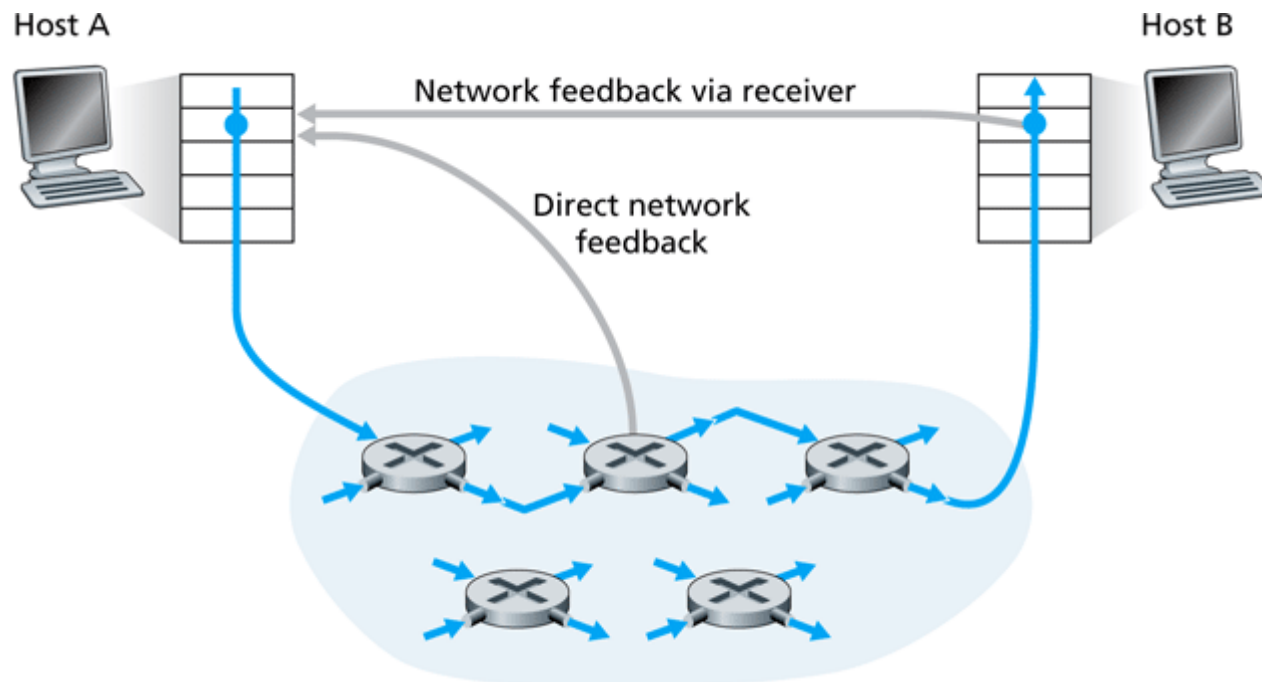
Approaches to Congestion Control

Approaches to Congestion Control

- End-to-end Congestion Control: Transport Layer
 - Network layer provides no feedback on congestion in network
 - End systems (source/destination hosts) infer congestion based on detected events such as packet loss and/or delay
- Network Assisted Congestion Control: Network Layer
 - Network devices (mainly routers) provide explicit feedback to the source host about congestion
 - Routers may provide direct feedback to source
 - Feedback from routers may be provided via the destination host
 - Feedback may be:
 - Backpressure: router A tells previous router B to slow down; router B tells previous router C to slow down; and so on
 - Explicit signalling: routers or destination host send special packets to source informing it of congestion and/or indicate the appropriate rate

Network Assisted Congestion Control

- Feedback may come direct from routers, or via the destination (receiver)



- ATM is an example network technology using Network Assisted Congestion Control

TCP Congestion Control

TCP Congestion Control

- TCP sender limits the rate at which it sends based on perceived network congestion
- How does TCP sender limit its sending rate?
- How does TCP sender perceive there is network congestion?
- How does TCP sender respond to congestion?
 - TCP congestion control algorithm

Limiting the TCP Sending Rate

- Amount of bytes TCP sender can send is limited by Advertised Window from Flow Control
 - Outstanding Bytes \leq Advertised Window
- In fact, TCP sender also maintains Congestion Window:
 - Outstanding Bytes \leq min (Advertised Window, Congestion Window)
- When an ACK is received, more bytes can be sent by TCP sender
- Assume the Advertised Window is very large (buffer at receiver is very large)
 - Sending rate \approx Congestion Window/RTT
- By adjusting the Congestion Window, TCP sender can adjust its sending rate

Perceiving Network Congestion

- TCP sender assumes a loss indicates increased network congestion
 - A loss:
 - TCP sender times out: has not received ACK within timeout period
 - TCP sender receives 3 duplicate ACKs
 - Is this a valid assumption?
 - Most packet losses occur at routers, i.e. congestion
 - However, in some networks (e.g. wireless), packets may be lost due to link errors, not congestion
- TCP sender assumes arrival of ACKs indicates decreased network congestion
 - The faster the arrival rate of ACKs, the large decrease in congestion assumed

TCP Congestion Control Algorithm

- Three main components:
 1. Additive Increase, Multiplicative Decrease (AIMD)
 2. Slow start (SS)
 3. Reaction to Loss Events
- Terminology
 - Maximum Segment Size (MSS): determined or assumed by TCP sender for network path; measured in bytes
 - Congestion Window (cwnd): measured in bytes
 - Round Trip Time (RTT): time from sending a segment, until corresponding ACK is received
- We will assume:
 - TCP receiver sends an ACK for every segment received

AIMD

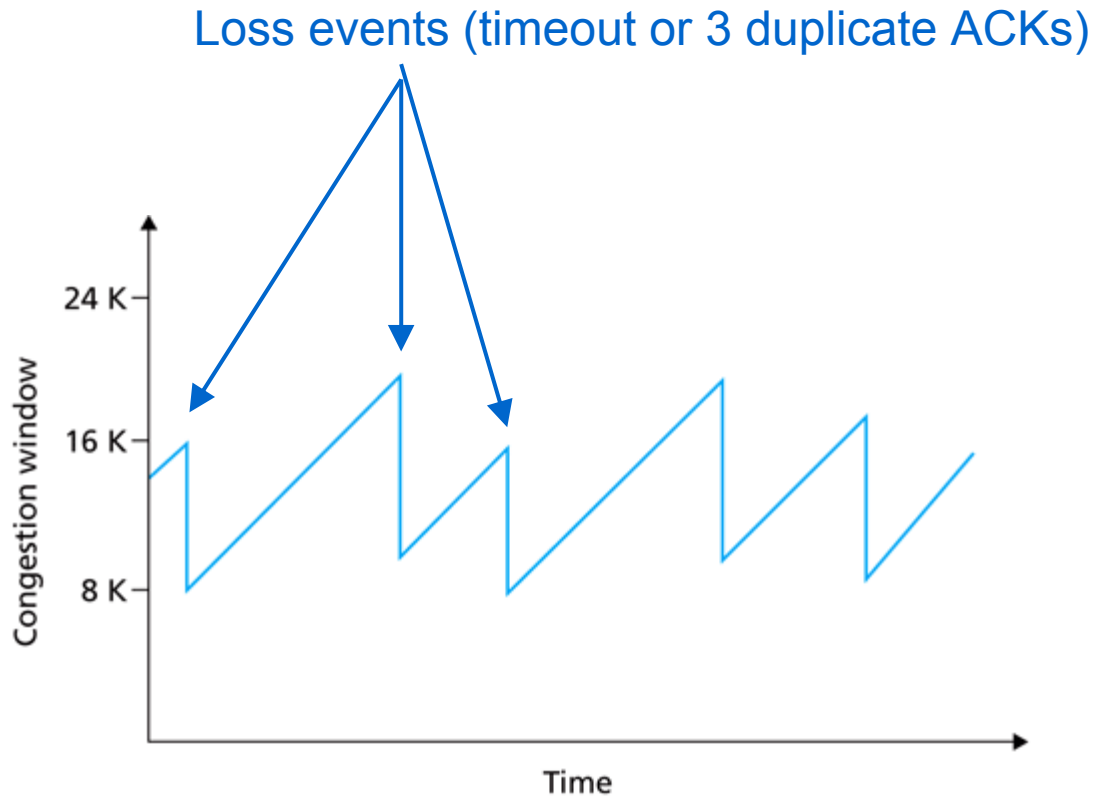
- Additive Increase

- If no congestion detected, then TCP sender assumes there is available (unused) capacity in the network; hence increases its sending rate
 - However, TCP slowly increases its sending rate
- TCP sender aims to increase Congestion Window by 1 x MSS every RTT
- One approach:
 - For every new ACK received, increase Congestion Window (cwnd) by:
 - $cwnd_{new} = cwnd_{old} + MSS * MSS / cwnd_{old}$
- Additive Increase phase is also called Congestion Avoidance

- Multiplicative Decrease

- If congestion detected, then TCP sender decreases its sending rate
- TCP sender aims to half the Congestion Window for each loss
 - For each loss detected:
 - $cwnd_{new} = cwnd_{old} / 2$
 - cwnd is not decreased to less than 1 MSS
 - (This is not entirely accurate – see Reaction to Loss Events)

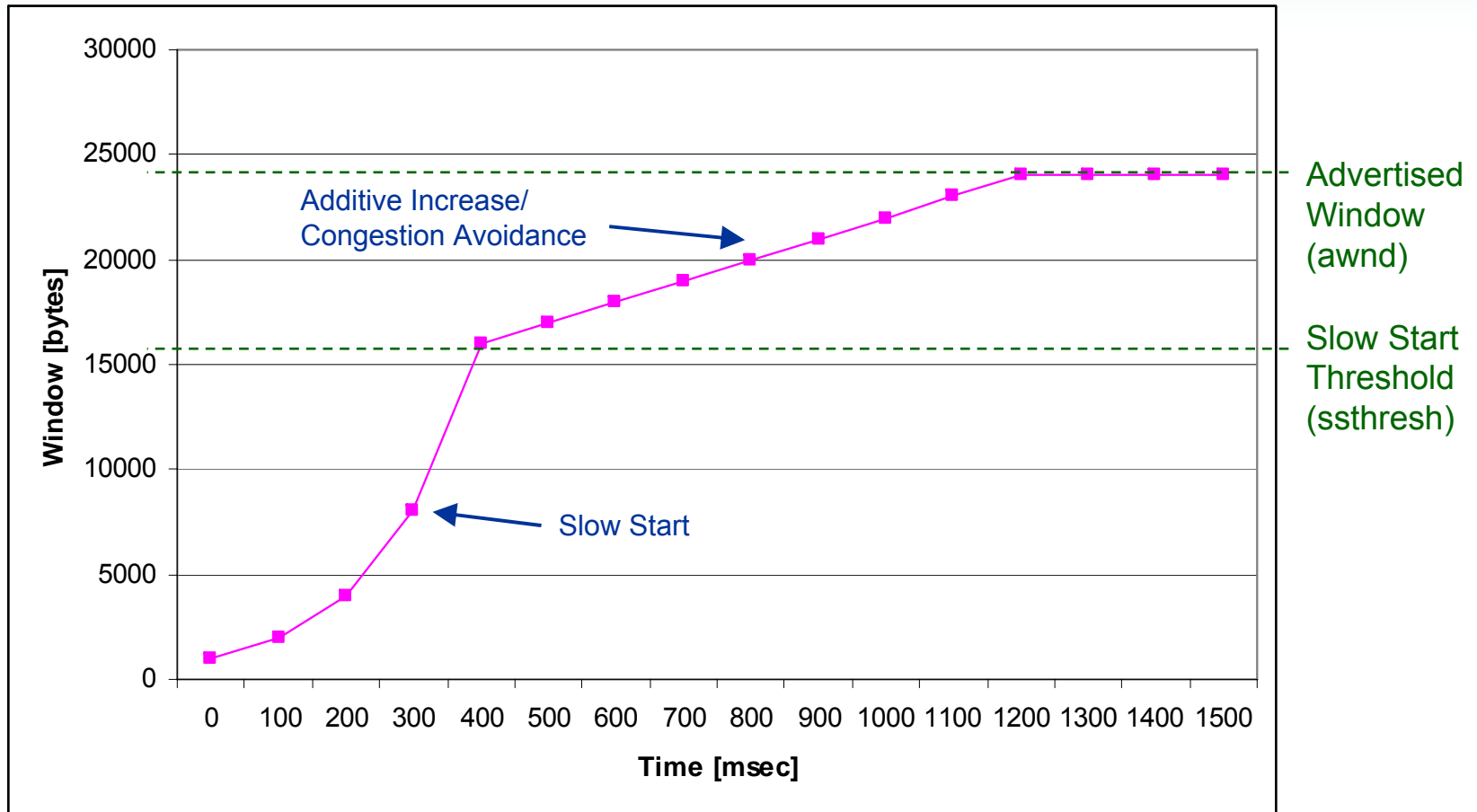
AIMD Example



Slow Start Phase

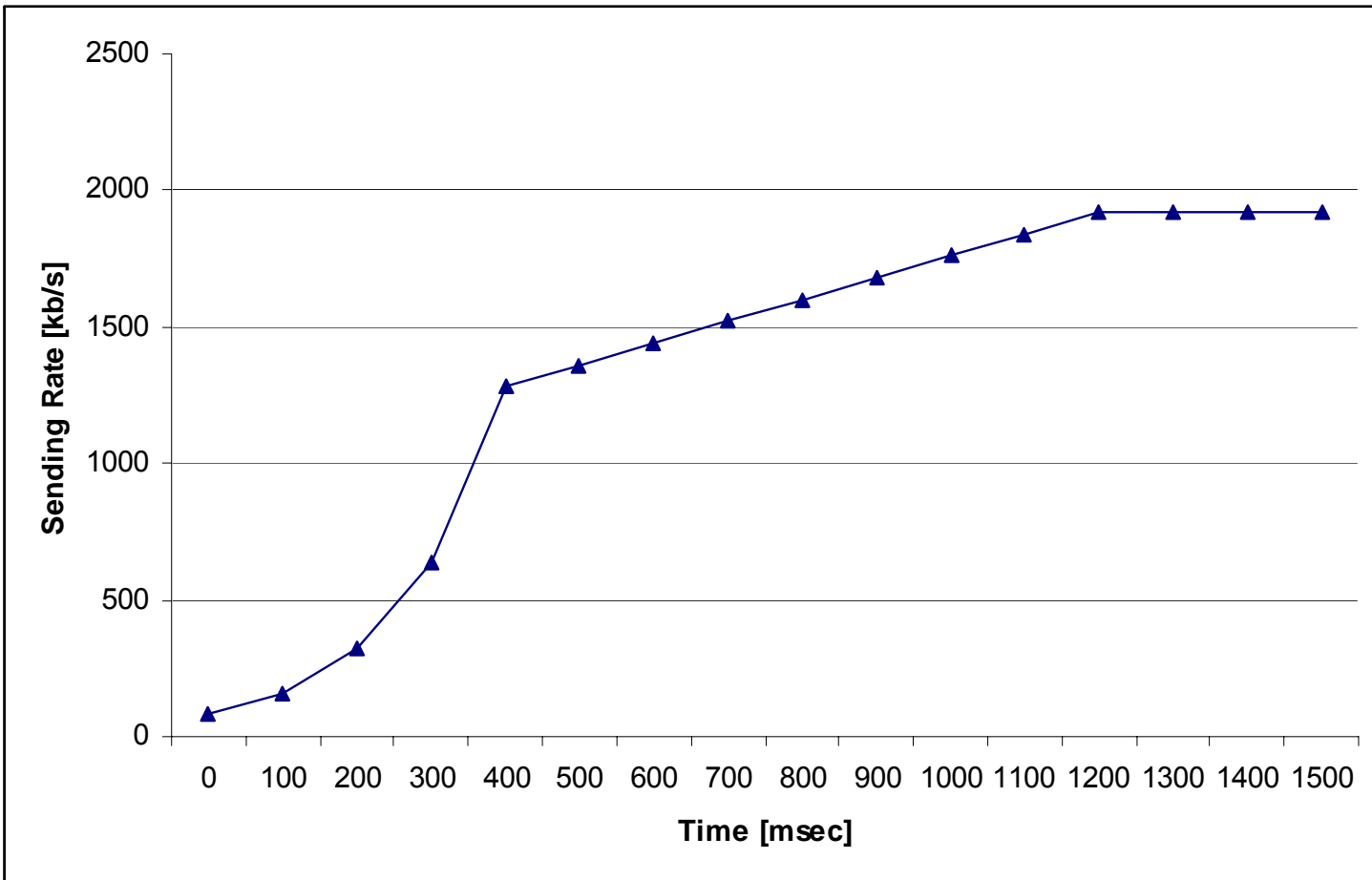
- At **start** of a TCP connection, the TCP sender sends at a **slow** rate
 - By default: $\text{cwnd} = \text{MSS}$
 - E.g. approximate sending rate for $\text{MSS} = 1000$ bytes, $\text{RTT} = 200\text{ms}$ is 40kb/s
- If large capacity is available for the connection, using additive increase (congestion avoidance) will be too slow
- Therefore Slow Start phase involves **very fast** increase of Congestion Window
 - Cwnd is increased exponentially
 - For every ACK received in Slow Start phase, increase cwnd by 1 MSS
 - $\text{cwnd}_{\text{new}} = \text{cwnd}_{\text{old}} + \text{MSS}$
 - Slow Start phase is continued until a loss event (then multiplicative decrease) or Congestion Window reaches a threshold (ssthresh) value (then additive increase)

AIMD and Slow Start: Window Size



MSS = 1000 B; RTT=100ms; ssthresh=16000 B; Advertised Window = 24000 B
Window = min (Congestion Window, Advertised Window)

AIMD and Slow Start: Sending Rate

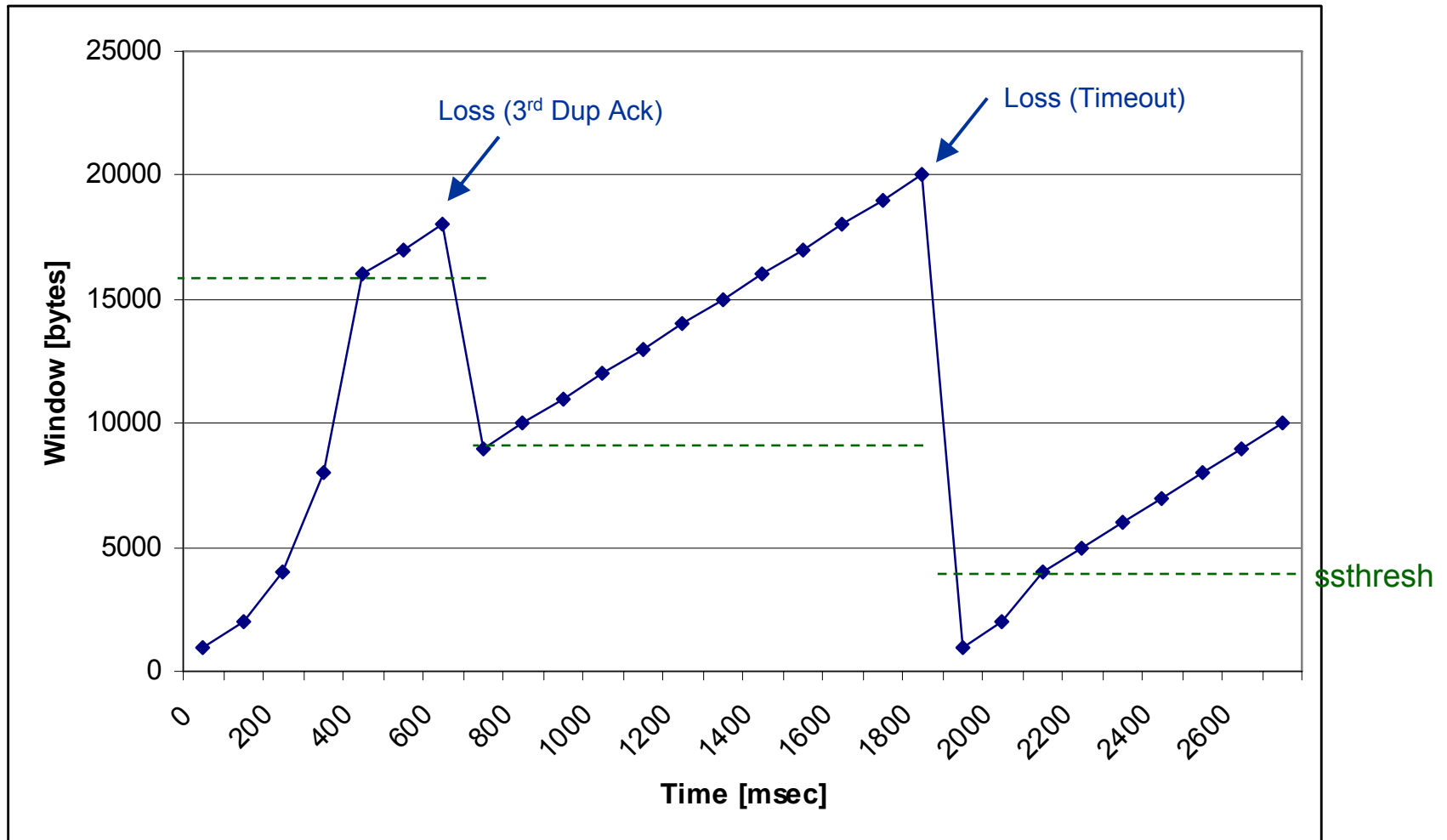


Assumes Sending Rate = Window / RTT

Reaction to Loss Events

- Upon a loss, Multiplicative Decrease halves the current congestion window
- The next action then depends on type of loss event:
 - Loss detected by 3rd Duplicate ACK
 - Slow start threshold is halved
 - $ssthresh_{new} = ssthresh_{old}/2$
 - Congestion window set to slow start threshold
 - $cwnd_{new} = ssthresh$
 - TCP enters Additive Increase (or Congestion Avoidance) phase
 - Loss detected by a timeout
 - Slow start threshold is halved
 - $ssthresh_{new} = ssthresh_{old}/2$
 - Congestion window set to initial value of 1 MSS
 - $cwnd_{new} = MSS$
 - TCP enters Slow Start phase

Reaction to Loss Events



Reaction to Loss Events

- Why?
 - TCP assumes a loss indicates congestion in the network (and therefore slows down)
 - Loss due to 3rd Duplicate ACK
 - Some TCP segments are being delivered (since some ACKs are coming back)
 - TCP assumes small level of congestion, therefore immediately enters Congestion Avoidance phase
 - Loss due to Timeout
 - Most TCP segments were lost (since not even duplicate ACKs are received)
 - TCP assumes heavy congestion, therefore go back to start (of Slow Start) with very slow sending rate

TCP Congestion Control in Practice

- TCP Congestion Control algorithm works well in networks where losses are mainly due to congestion
 - Note that with a congested network, the throughput of TCP connection can be severely limited
- In networks with losses due to errors on links, TCP Congestion Control has problems
 - Example: a wireless link may lose segments due to poor link quality
 - TCP slows down (thinking it is congestion) when it should maintain its sending rate
 - Several variants of TCP have been developed specifically for wireless links
- In high-speed networks (>10Gb/s), TCP may perform poorly even with very few link packet losses

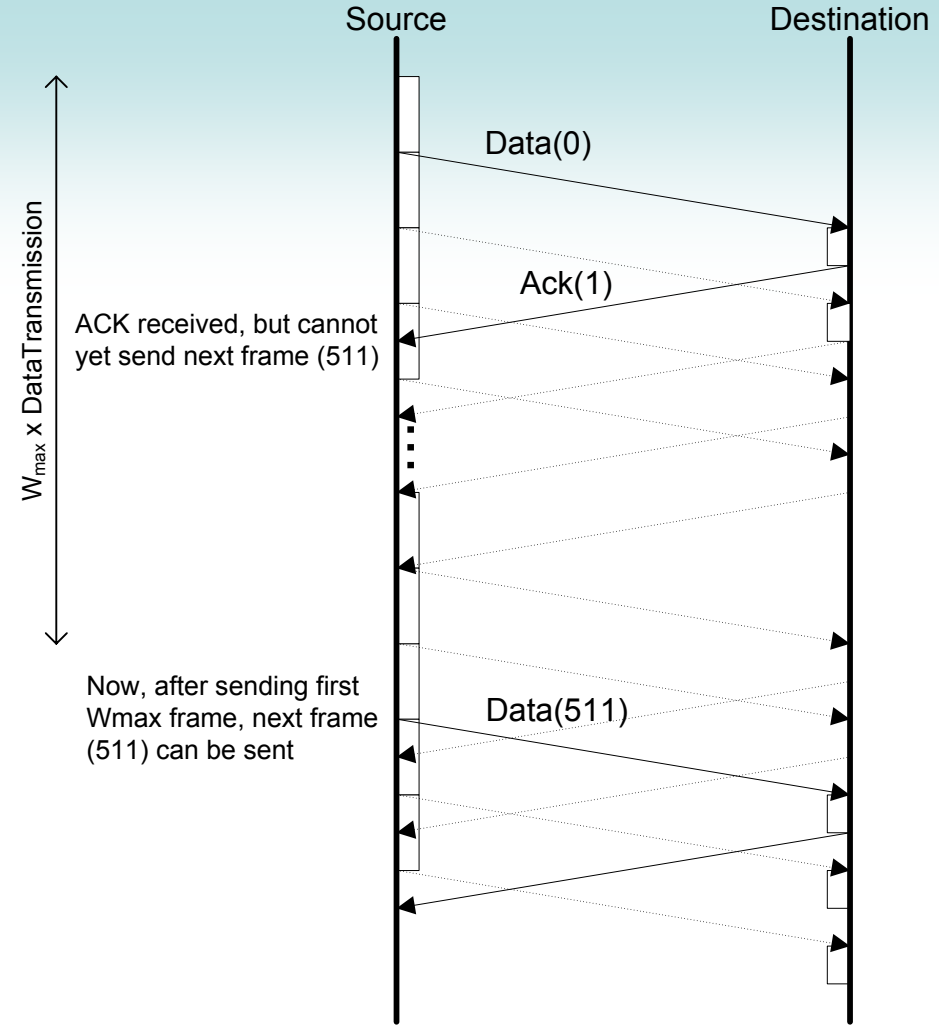
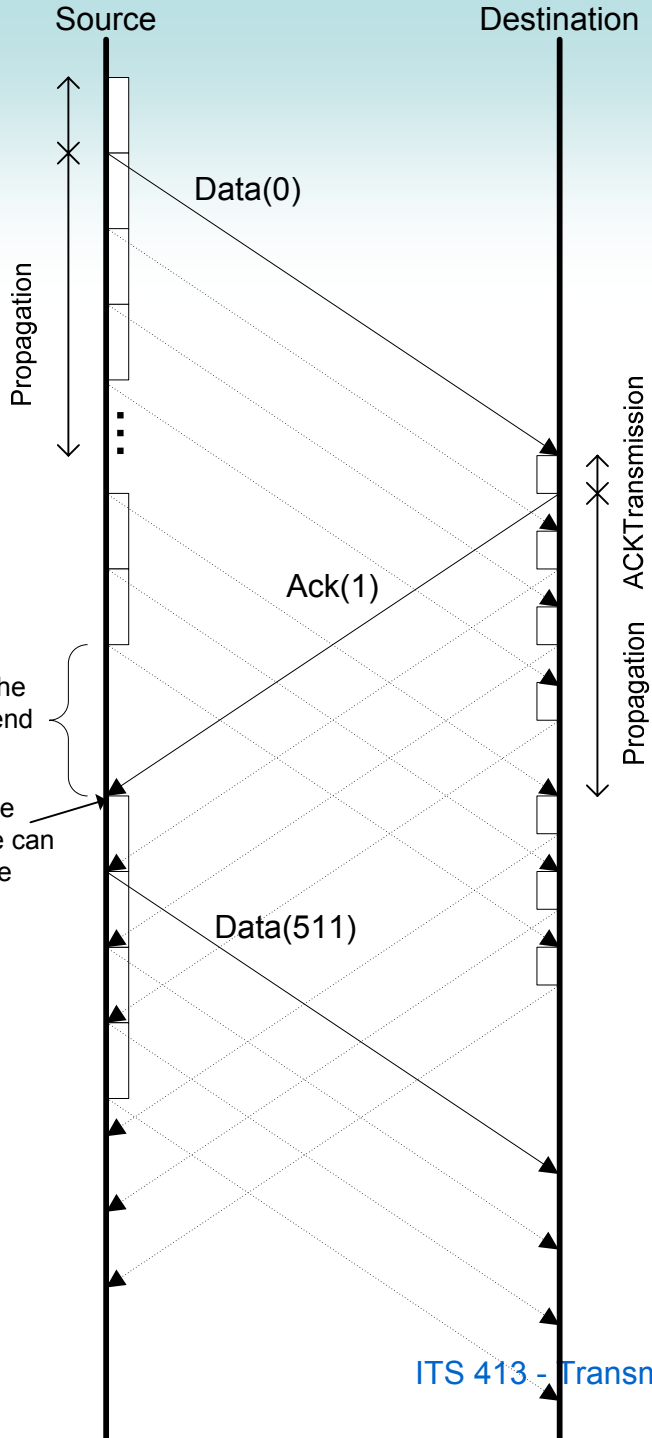
TCP Versions and Options

- TCP RFC 793 (1981)
 - Reliability (sequence numbers), Flow control (receiver window), Connection management
- TCP Tahoe (1988)
 - Adds Slow Start, Congestion Avoidance, Fast Retransmit
- TCP Reno (1990)
 - Adds Fast Recovery
- TCP NewReno (1995)
 - Only halves congestion window once
- Other Options:
 - Selective Acknowledgement (SACK)
 - TCP Vegas
- Some Operating Systems implement their own options/variants

TCP Performance Issues

Sliding Window Performance

- Sliding Window operation:
 - Sender can send W segments before having to wait for ACK
 - When ACK is received then more segments can be sent
- Note: remember TCP actually counts in bytes, not segments. However for simplicity we will often refer to ACK of a segment
- Parameters:
 - Payload: size of data in a segment
 - Header: size of header
 - Rate: data rate of the link/network
 - Prop: propagation delay between sender and receiver
 - W : number of segments that can be sent



Basic TCP Performance

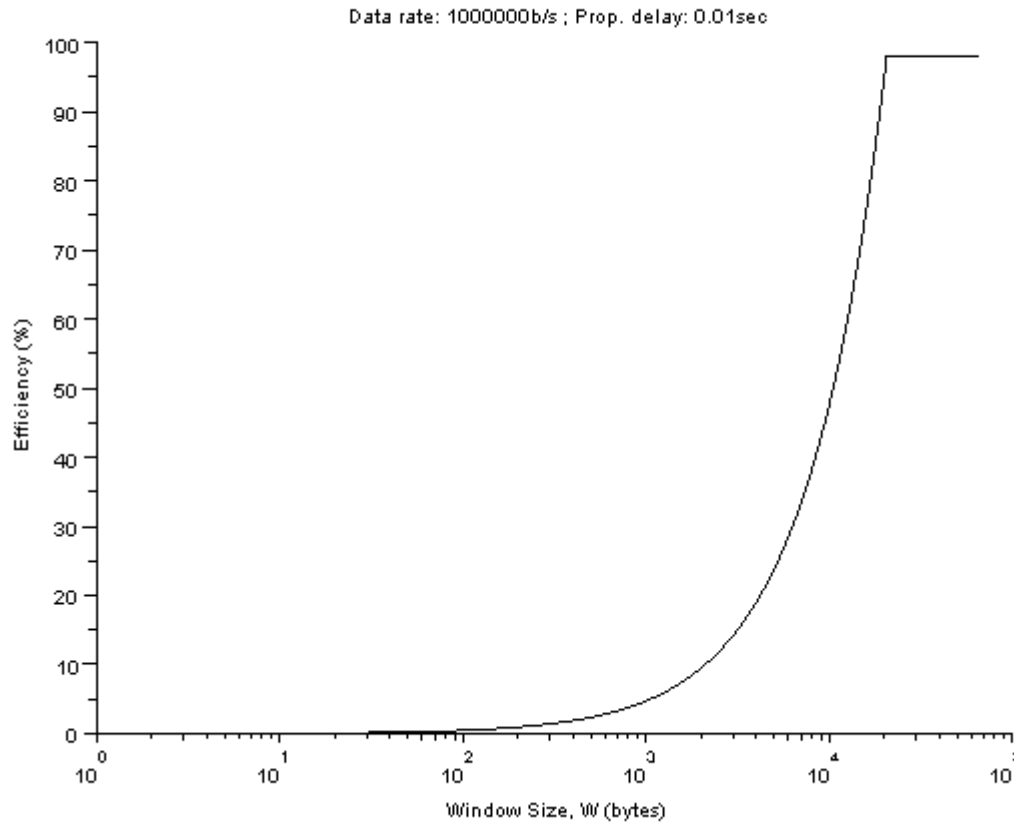
- $\text{Data} = (\text{Payload} + \text{Header}) / \text{Rate}$
- $\text{ACK} = \text{Header} / \text{Rate}$
- If $(\text{Data} + 2 \times \text{Prop} + \text{Ack} > W \times \text{Data})$ Then

$$\text{Throughput} = \frac{W \times \text{Payload}}{\text{Data} + 2 \times \text{Prop} + \text{Ack}}$$

Else

$$\text{Throughput} = \frac{\text{Rate} \times \text{Payload}}{(\text{Payload} + \text{Header})}$$

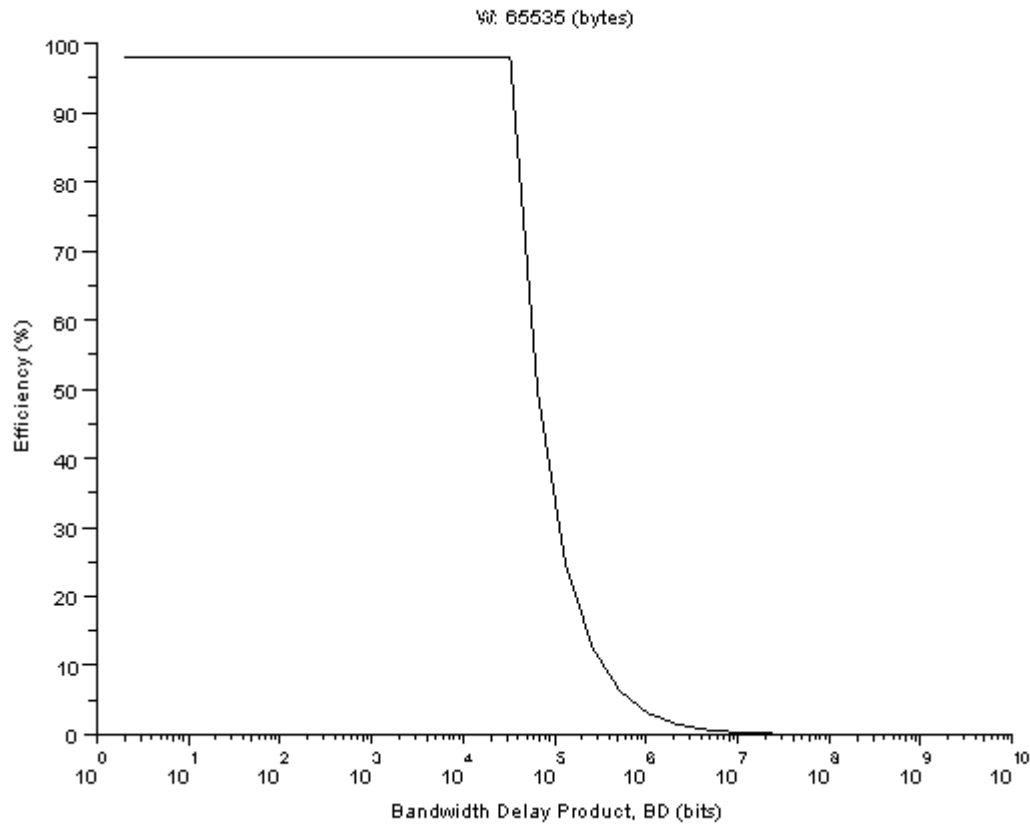
Basic TCP Performance



Basic TCP Performance

- If transmission delay \gg propagation delay, performance is good
- Relationship between transmission and propagation delay often referred to as:
 - Bandwidth Delay Product
 - $BD = \text{Data Rate} * 2 * \text{Propagation Delay}$
- A link or network may be characterised by its Bandwidth Delay product
 - Larger BD leads to lower throughput
- Examples:
 - 1Mb/s ADSL link with 0.1ms propagation: $BD = 100$ bits
 - 100Mb/s Ethernet with 1ms propagation: $BD = 100,000$ bits
 - 1Gb/s satellite link with 250ms propagation: $BD = 250$ Mbits

Basic TCP Performance



Basic TCP Performance

- Consider a high speed satellite link with the satellite in geostationary orbit
 - Data rate: 2Mb/s
 - Propagation delay: 250ms
 - BD: 1,000,000 bits
 - Maximum TCP window size: 65535 Bytes = 524,280 bits
 - Efficiency: 5%
- On high BD links, TCP performs poorly because of the limited maximum window size
- Solution: Allow larger window sizes
 - TCP Window Scaling Option

TCP Window Scaling

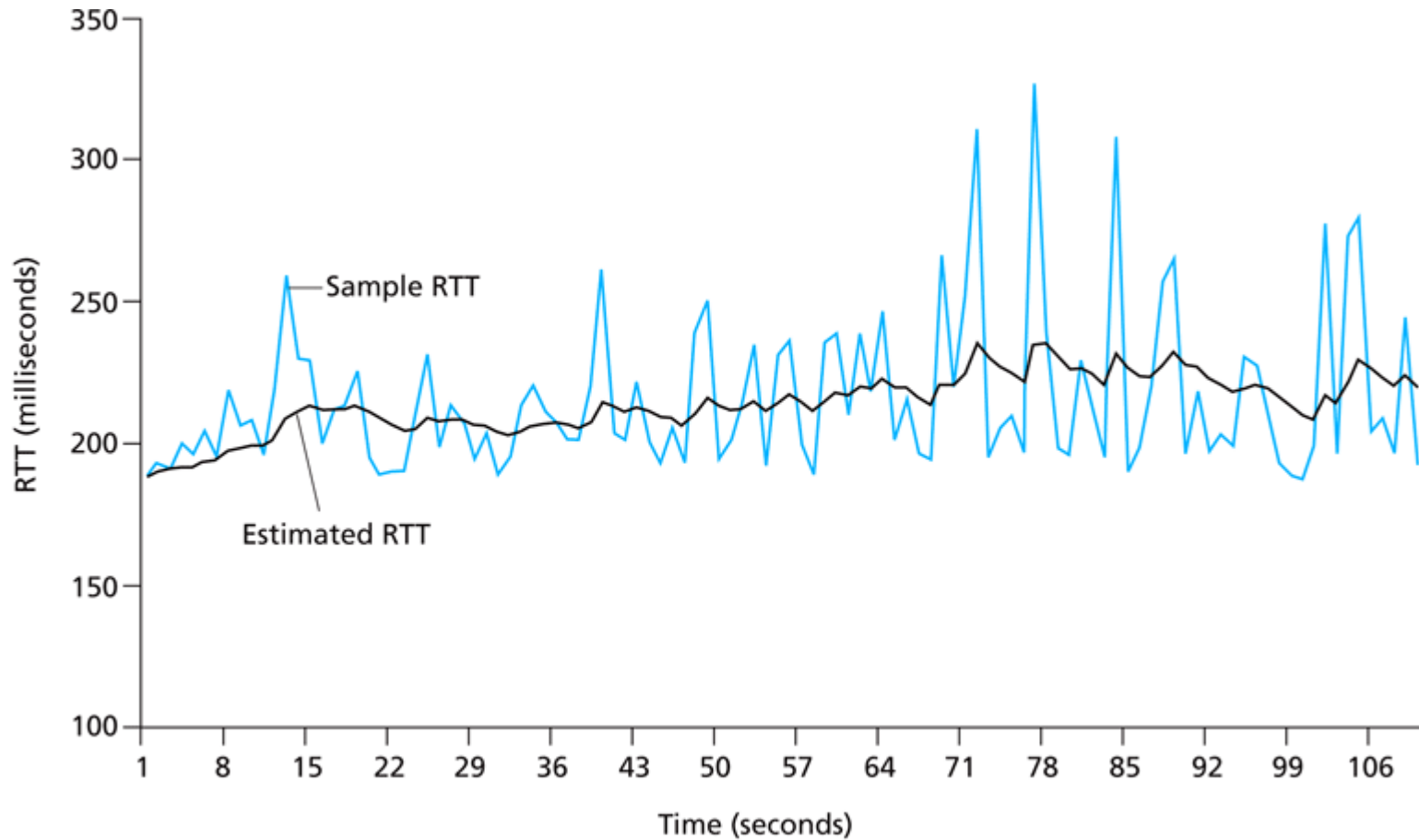
- TCP Segment header has window field of 16 bits
 - Maximum window size: 65535 bytes
- TCP Window Scaling is a TCP option
 - Scale factor: $2^0, 2^1, 2^2, \dots, 2^{14}$
 - Maximum window size: $2^{14} \times 65535$ bytes = 1 GByte
- Allows higher speeds (efficiency) for links with large Bandwidth Delay product
 - Satellite links
 - Long optical links

RTT Estimation

RTT Estimation

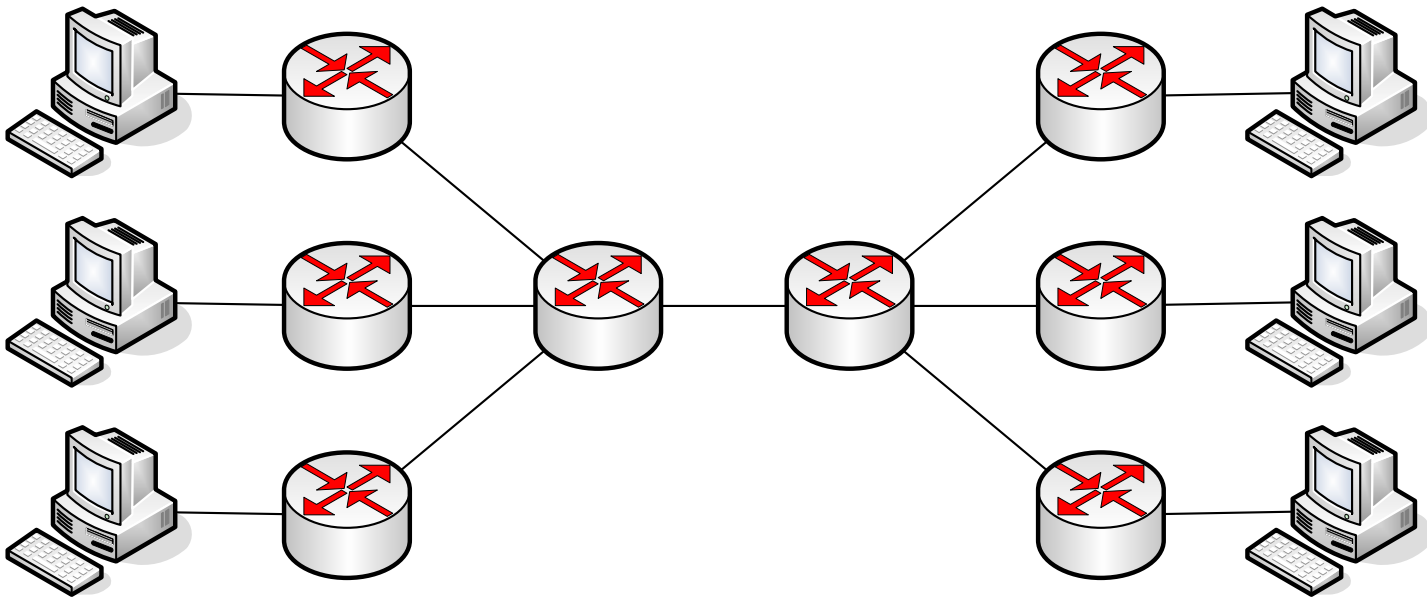
- TCP sender needs to estimate Round-Trip Time (RTT)
 - Why? To determine the appropriate timeout interval.
 - Timeout interval too large: Sender will wait too long before retransmitting
 - Timeout interval too short: Sender may retransmit when data is not lost
- RTT estimation is complex, yet important function in TCP
 - Delay in Internet can vary significantly over time
 - TCP samples the RTT of segments
 - TCP uses a weighted average function to estimate RTT
 - Different algorithms/options used in different versions of TCP and its implementation

RTT Estimation

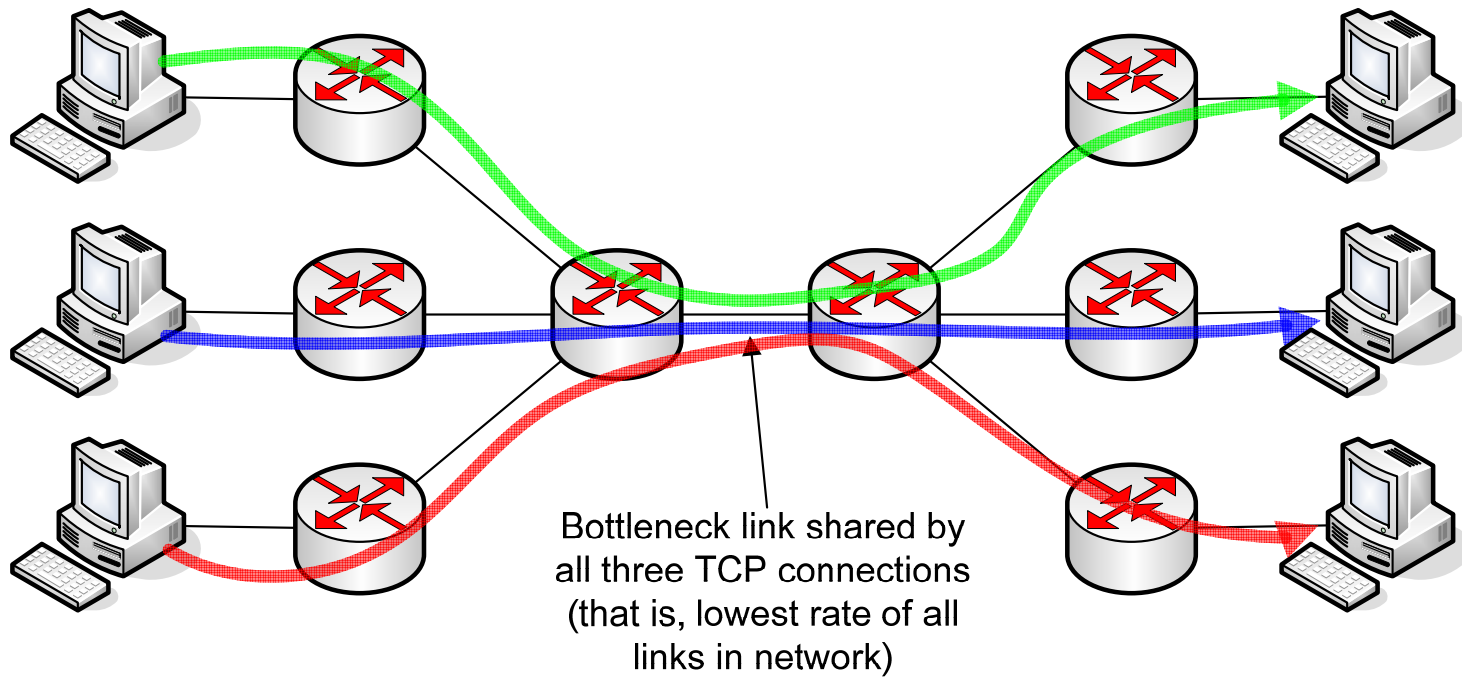


TCP Fairness

Example: TCP Fairness



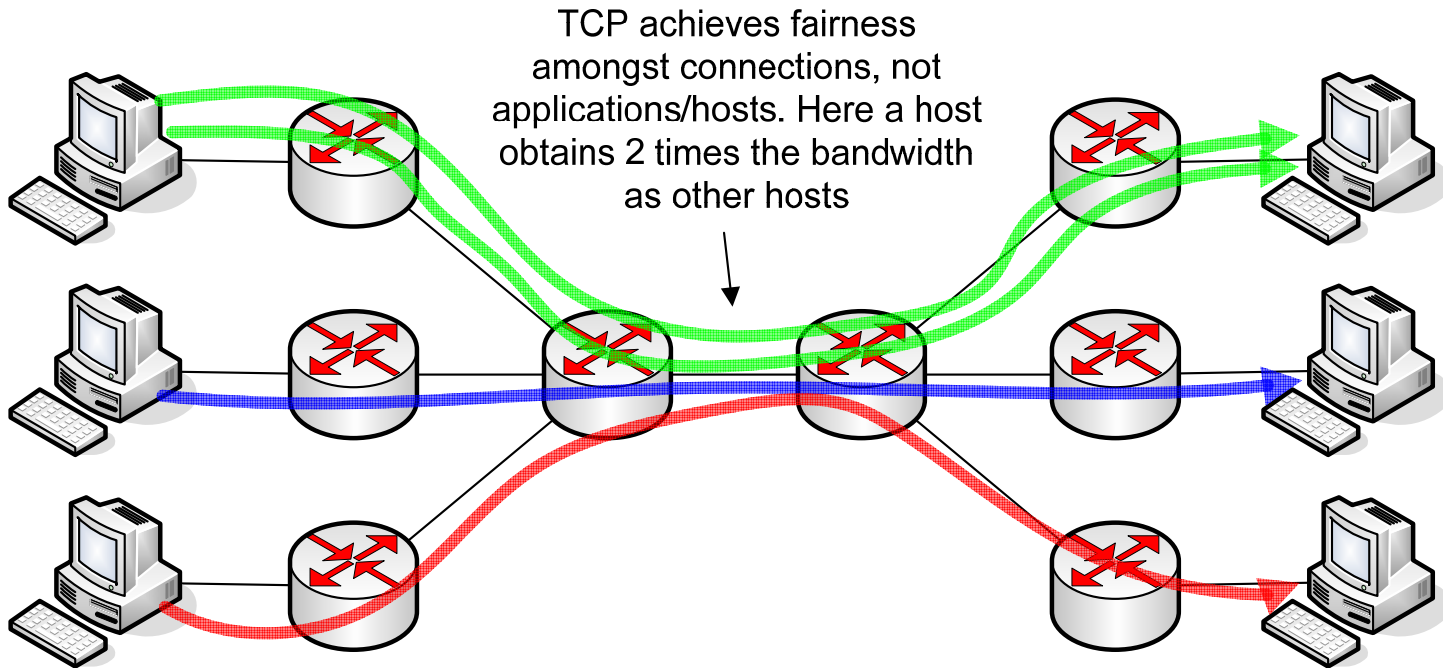
Example: TCP Fairness



TCP Fairness

- If TCP is fair, with N TCP connections using a R bps link
 - Each TCP connection should achieve R/N bps
- Does TCP achieve fairness?
 - In ideal conditions, yes. If all TCP connections have same RTT and same sized segments, with no other traffic, fairness is achieved
 - In practice:
 - If RTT of connections vary, connections with small RTT are able to higher proportion of bandwidth than connections with large RTT
 - If other non-TCP data is also present (such as multimedia using UDP), then TCP connections receive unfair treatment
 - Applications can use multiple TCP connections: each TCP connection gets fair treatment, but the application using multiple connections gets more bandwidth than application using single connection

Example: TCP Fairness



TCP and The Internet

- IP does not include any built-in congestion control mechanisms
 - If every host sent IP datagrams as fast as possible, the Internet would not work
- The Internet relies on TCP mechanisms to avoid collapse
 - TCP comprises about 90% of all traffic on the Internet
 - As a means for congestion control, TCP has been very successful
- But ...
 - If hosts/applications choose not to follow TCP's congestion control rules, then congestion can become a major problem in the Internet
 - Challenges:
 - Web browsers opening many TCP connections at once.
 - Growth of multimedia applications that use UDP.
 - Growth of P2P applications using multiple connections and/or UDP.