Module 3: Bulk Transfers Under TCP

GÉANT2

OVERVIEW OF BULK TRANSFERS

Goal: transfer a large set of data reliably with the maximum throughput offered by the network at any given time.

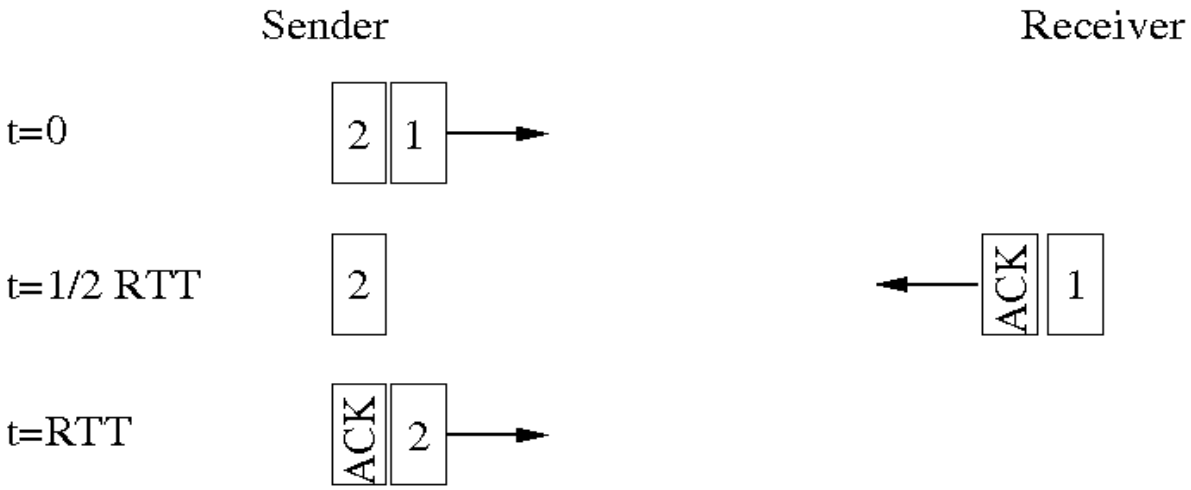
Basic service provided by TCP

- Receiver acknowledges receipt of data, sender retransmits lost data.
- Sender numbers bytes so receiver can reconstruct proper order.
- Sender probes network for maximum data rate, backs off during congestion.

Most common applications used for bulk transfers: HTTP, FTP.

WINDOW-BASED TRANSMISSION

Trivial file transfer protocol (TFTP, RFC1350): send one packet, wait for acknowledgement (uses UDP).



Throughput: 1 packet/Round Trip Time (RTT). Can be increased by sending more data before expecting acknowledgement. This is called the “window”.

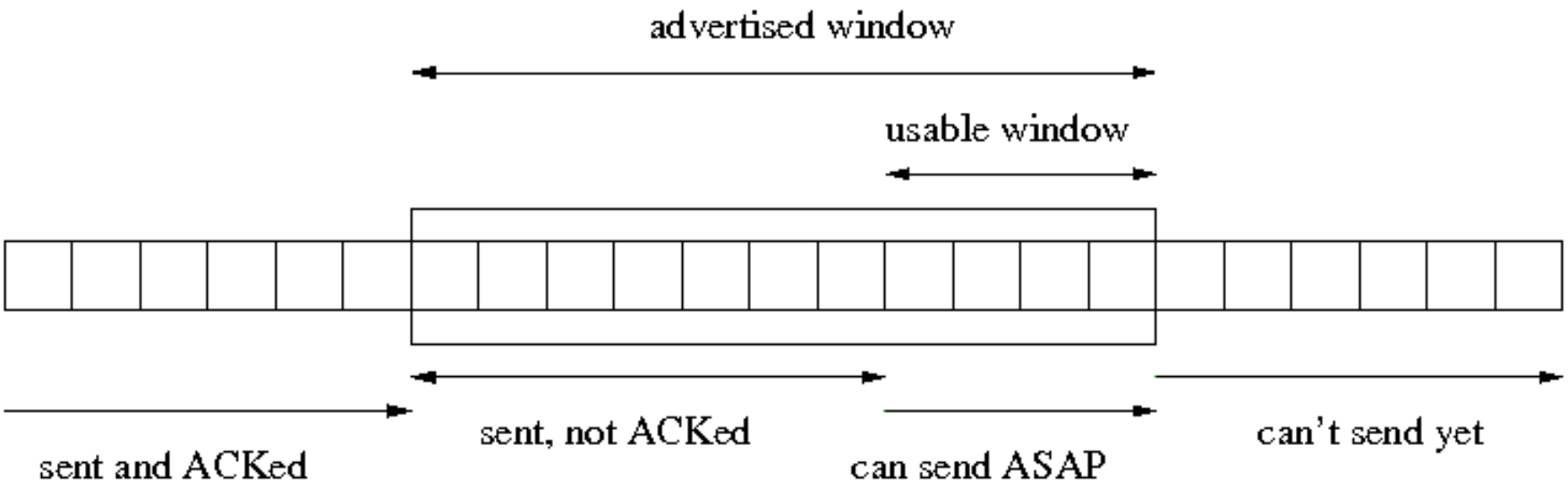
HOW DOES WINDOW-BASED TRANSMISSION WORK?

Basic concepts:

- A packet is ‘in flight’ if it has been sent but not yet acknowledged.
- Only one ‘window-full’ of data can be in flight at any given time.
  - This is the maximum amount of data that can be in transit on the network path.
- The window ‘slides’.
  - As one packet is acknowledged and ‘leaves’ the window, another can be transmitted.
- The sender must buffer a packet until the receiver acknowledges receipt.
  - Needed for re-transmission in case of loss or corruption.
  - Means that the sender’s buffer limits the window size.

Window-based transmission is a key element of TCP.

THE SLIDING WINDOW



The window moves to the right as data is acknowledged.

Advertised window = buffer available at receiver: sender may only send as much as the receiver can accept (receiver-based flow-control).

WINDOW SIZE AND THROUGHPUT

Window size is critical to throughput.

In the absence of packet loss:

Throughput = Window / RTT.

Throughput is bounded by the bandwidth of the bottleneck link of a network path, i.e.

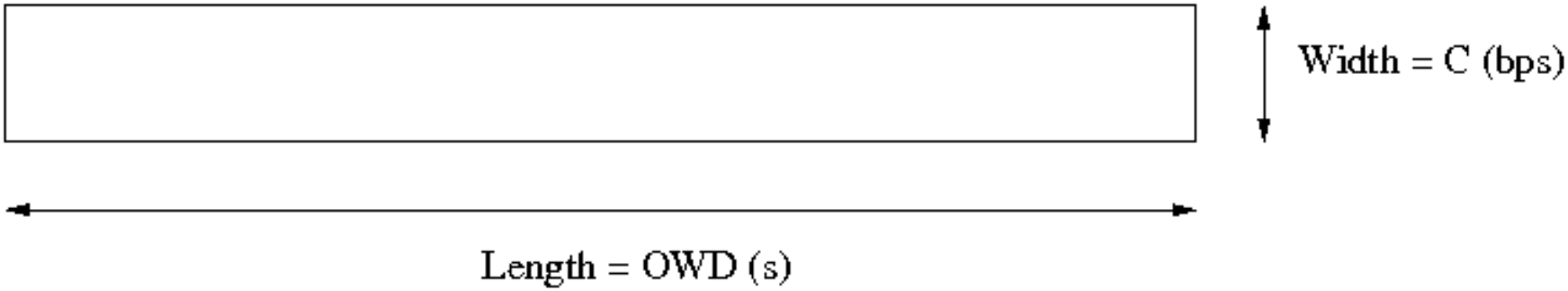
bottleneck bandwidth * RTT ≥ Window.

Equality is required to make optimal use of the link.

LINKS, PATHS AND PIPES

Link capacity C (aka bandwidth, often loosely called link speed): number of bits the link can transport per unit time, measured in bits per second (bps), e.g. FE = 100Mbps, GE = 1Gbps.

Path: concatenation of links, characterized by OWD (one-way delay) and bottleneck capacity. Can be viewed as a “pipe”:



“Filled pipe” can hold C * OWD bits, e.g.

C	OWD	Size of pipe
100Mbps	50ms	625KB
1Gbps	50ms	6.25MB

WINDOW SIZE CALCULATION

How large must the window be to keep the pipe filled? First ACK received after one RTT → window must be no smaller than the **bandwidth-delay product** (BDP):

- Window size (bytes) ≥ Bandwidth (bytes/sec) * round-trip time (sec).

Networks with large BDP are called ‘Long Fat Networks’ (LFNs or “elephants”).

C	RTT	BDP
100Mbps	100ms	1.25MB
1Gbps	100ms	12.5MB

HOW IS THE WINDOW SIZE DETERMINED?

The window size is equal to the smallest of:

- The sender's buffer size (static).
- The receiver's advertised window size (bounded by static buffer size).

Need to guess BDP beforehand, tune buffers. TCP uses an additional *congestion window* to probe for effective BDP, more on this later.

Note: window limited to 64KiB in original TCP. Need to use *window scaling option* (RFC1323) for windows up to 2^{30} bytes.

WINDOW SIZE AND PERFORMANCE

Window size is the most important factor influencing throughput in high-performance networks.

- Degradation can occur if the Window size is too big or too small

Window size too small:

- Unused capacity.

Window size too big:

- Can monopolise system memory in popular web or file servers.
  - May run out of buffer space to open new connections.
  - May starve other processes of access to fast memory.
- Can result in uneven bursts of traffic.
  - When large bursts of traffic combine with a network bottleneck, the result can be buffer overflows with resultant packet loss and retransmission overheads.

TUNING YOUR HOST FOR MAXIMUM TRANSMISSION RATES (1)

Tuning techniques and options vary between operating systems. However, you may be able to:

- Change the TCP buffer size (TCP stack size).
- Use Large Send Offload (LSO) to reduce load on CPU
  - Increases the amount of data that can be sent by the CPU to the network adapter for transmission.
  - Also known as TCP Segmentation Offload and TCP Multidata Transmit.
- Use Interrupt Coalescence
  - Increases amount of incoming data that can be buffered on the network adapter before it is sent to the CPU.
  - Decreases the number of ‘interrupts’ that the CPU has to deal with.

– Continued on next slide…

TUNING YOUR HOST FOR MAXIMUM TRANSMISSION RATES (2)

Tuning techniques and options vary between operating systems. However, you may be able to:

- Use Checksum Offload.
  - Network adapter can verify / generate TCP checksums itself.
  - Reduces load on CPU.
- Use TCP Offload Engines (TOEs).
  - Moves all TCP processing onto the network adapter.
  - LSO and checksum offload are ‘subsets’ of TOE.
  - Frees up CPU.
  - Requires driver support in the Operating System.

You can find detailed information about tuning different operating systems in the PERT Knowledge Base.

ACTIVITY

Exercise:

- Tuning your laptop's TCP Stack.

THE GOALS OF EARLY TCP (1)

Basic TCP functionality has changed little since RFC 793, published in 1981. The main goals of TCP were defined as:

- Reliability:
  - Each octet is given a sequence number. The receiver uses these to:
    - Eliminate duplicate octets.
    - Re-order out-of-sequence octets.
  - Each segment must be acknowledged by the receiver:
    - Cumulative ACKs: acknowledges all data up to this point (implies that loss on ACK-path is less critical).
    - If acknowledgement does not arrive before a timeout, the sender re-transmits the segment. Requires estimation of RTT.
  - Each segment is given a checksum:
    - Receiver uses this to spot corrupted or damaged segments.
      - » Continued on next slide.

THE GOALS OF EARLY TCP (2)

The main goals of TCP according to RFC 793 (continued):

- Flow Control.
  - Receiver governs amount of data sent by sender through mechanism of window size.
- Multiplexing.
  - Use of ports makes many simultaneous connections possible from one server.
- Connections.
  - Established between a pair of sockets and the TCP implementation on each side.

‘CONGESTION COLLAPSE’ IN THE LATE 1980s

By the late 1980s computer networks were experiencing ‘explosive growth’, but TCP could not cope with the increasing traffic.

Often internet gateways would drop 10% of incoming packets due to local buffer overflows.

From late 1986 onwards, the internet experienced a series of ‘congestion collapses’.

Van Jacobson and others identified aggressive sending strategy of early TCP as the source of the problem.

WHY DID CONGESTION COLLAPSE OCCUR? (1)

Window size was equal to the lesser of:

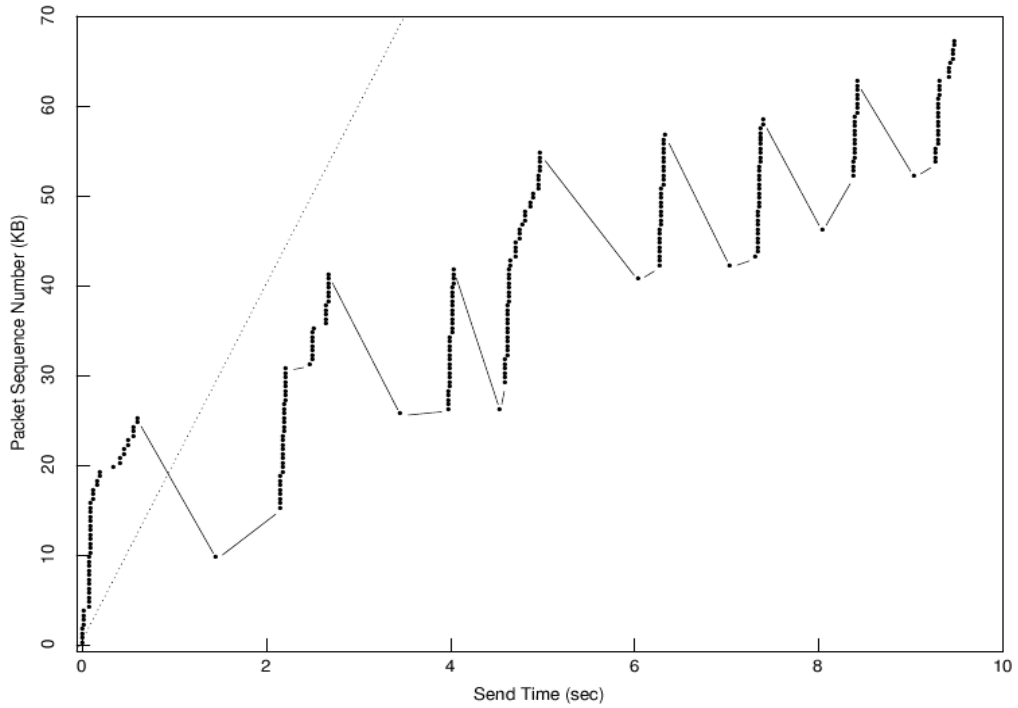
- Sender's buffer size.
- Receiver's advertised window size.

Window size was determined by capacity at both ends of the link, but it did not allow for bottlenecks between them. Neither did the window size take network congestion into account. The results:

- On connection start-up, bottleneck gateways could be overwhelmed with packets.
- Buffers overflowed.
- Packets were lost and had to be retransmitted.
- This created a vicious circle.

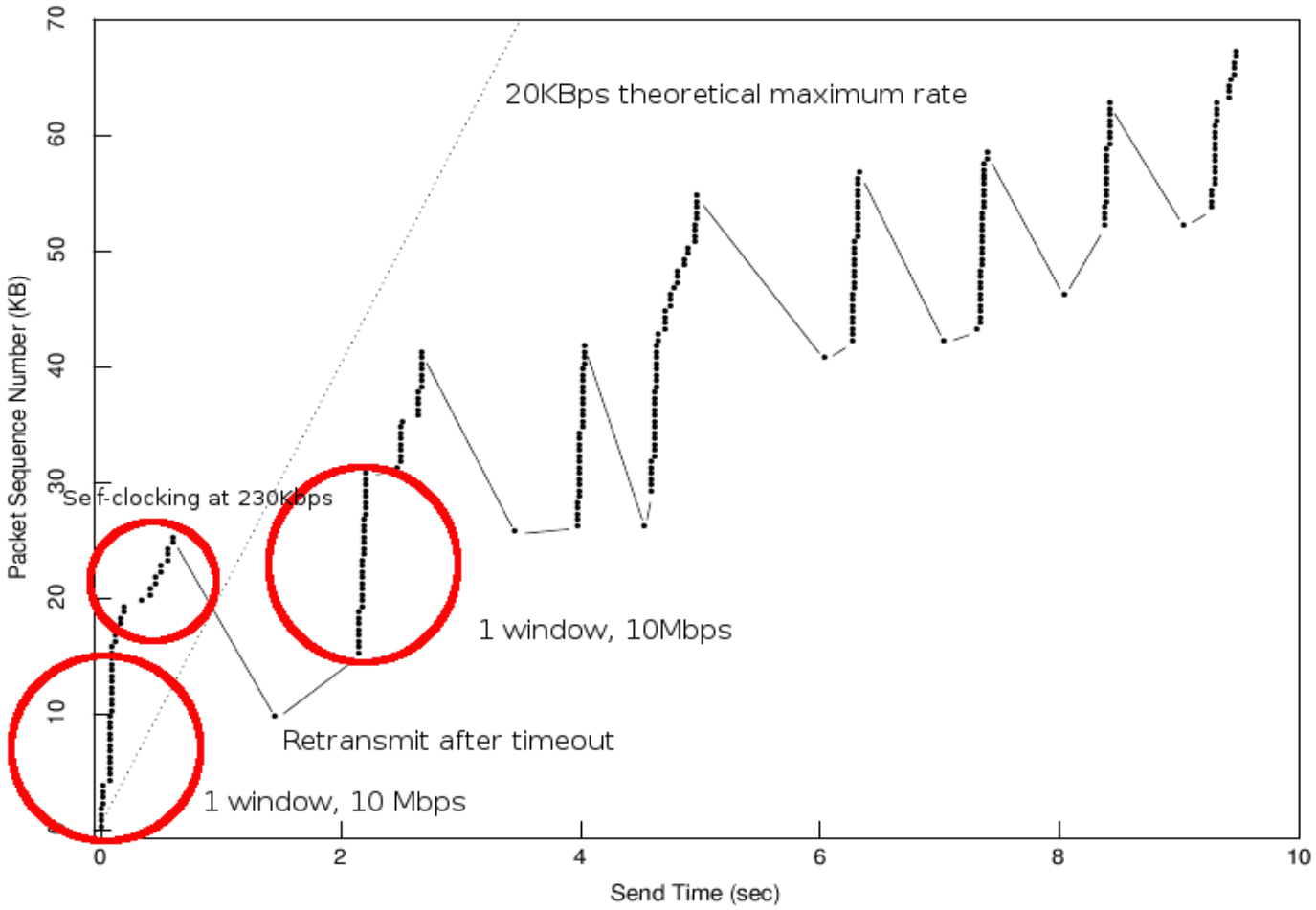
WHY DID CONGESTION COLLAPSE OCCUR? (2)

Early TCPs started by sending an entire window in a burst, overloading router queues immediately. Ethernet connected to a 230Kbps link with buffer (15kiB) < window (16kiB):



Van Jacobson, “Congestion Avoidance and Control”, ACM SIGCOMM Computer Communication Review, 1988, Volume 18, Issue 4, pp. 314-329.

WHY DID CONGESTION COLLAPSE OCCUR? (3)



INTRODUCTION OF CONGESTION CONTROLS

In the late 1980s, TCP evolved to prevent congestion collapse. TCP Tahoe introduced new flow control mechanisms and TCP Reno refined them:

- The Congestion Window
  - Makes window size dependent upon a connection's available bandwidth.
- Slow Start and Congestion Avoidance
  - Increase window size to probe a connection for available bandwidth.
- Congestion Control Mechanisms
  - Decrease the TCP transmission window when congestion is detected.

AVOIDING CONGESTION: THE CONGESTION WINDOW

The transmission window size was made equal to the lesser of:

- The sender's buffer size.
- The receiver's advertised window size.
- **The Congestion Window.**

TCP implementations could dynamically resize the congestion window to:

- Probe a connection for available bandwidth using 'slow start'.
- Slowdown throughput growth before collapse occurred using 'congestion avoidance'.
- Reduce the rate at which data was sent when congestion was detected.

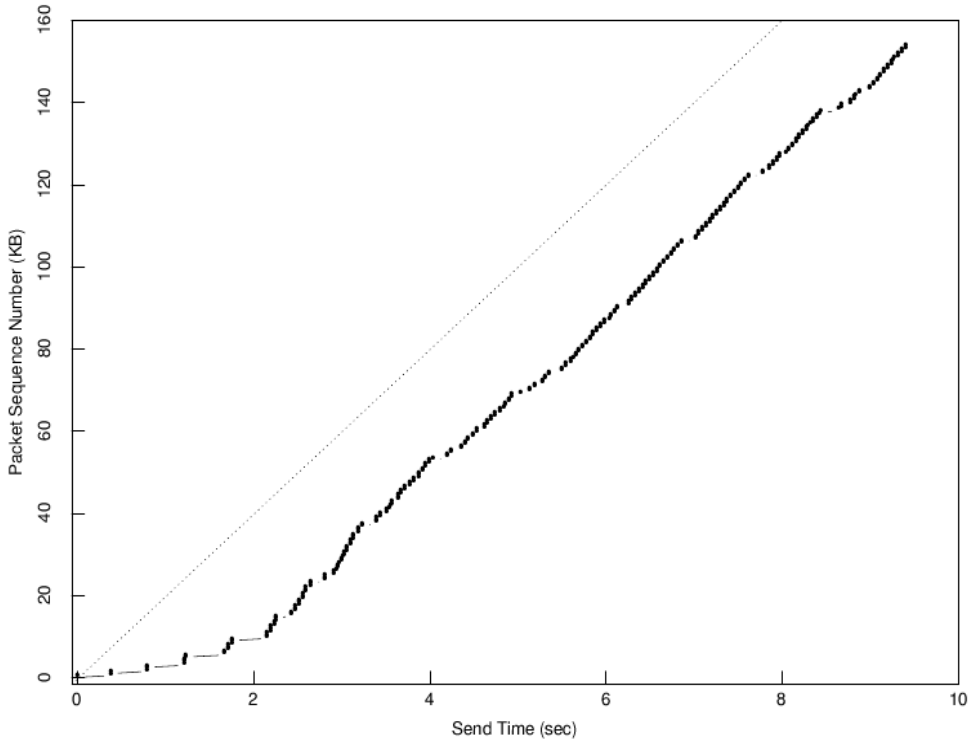
SLOW START

Slow start aims to avoid overwhelming bottlenecks with packets on start-up. It starts small and increases the rate of flow exponentially.

- On start-up, the congestion window is set to the Maximum Segment Size (MSS) of the connection.
- On each ACK, congestion window is increased by 1 MSS
  - Doubles congestion window every RTT.
- Congestion window continues to increase at same rate until:
  - The receiver's advertised window size is reached, or
  - Congestion is detected in the connection, or
  - There is no traffic waiting to take advantage of increased window, or
  - The slow-start threshold is crossed.

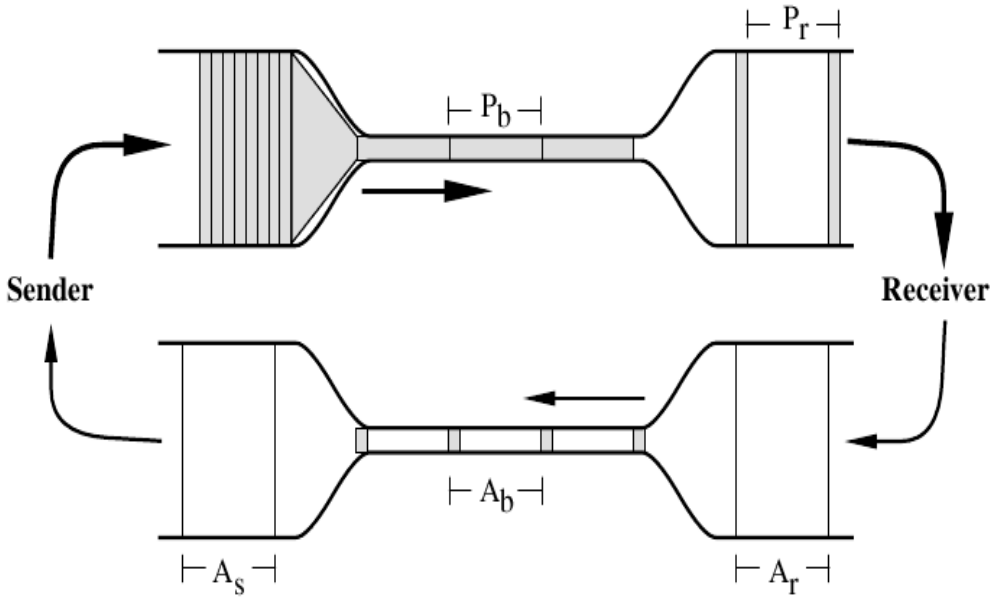
CONGESTION COLLAPSE FIXED

Same transmission as on slide 18 but with slow start (window-limited transfer, self-clocking)



DIGRESSION: SELF-CLOCKING

Sender has a window in flight. ACK spacing dictated by bottleneck bandwidth → window moves at the same clock.



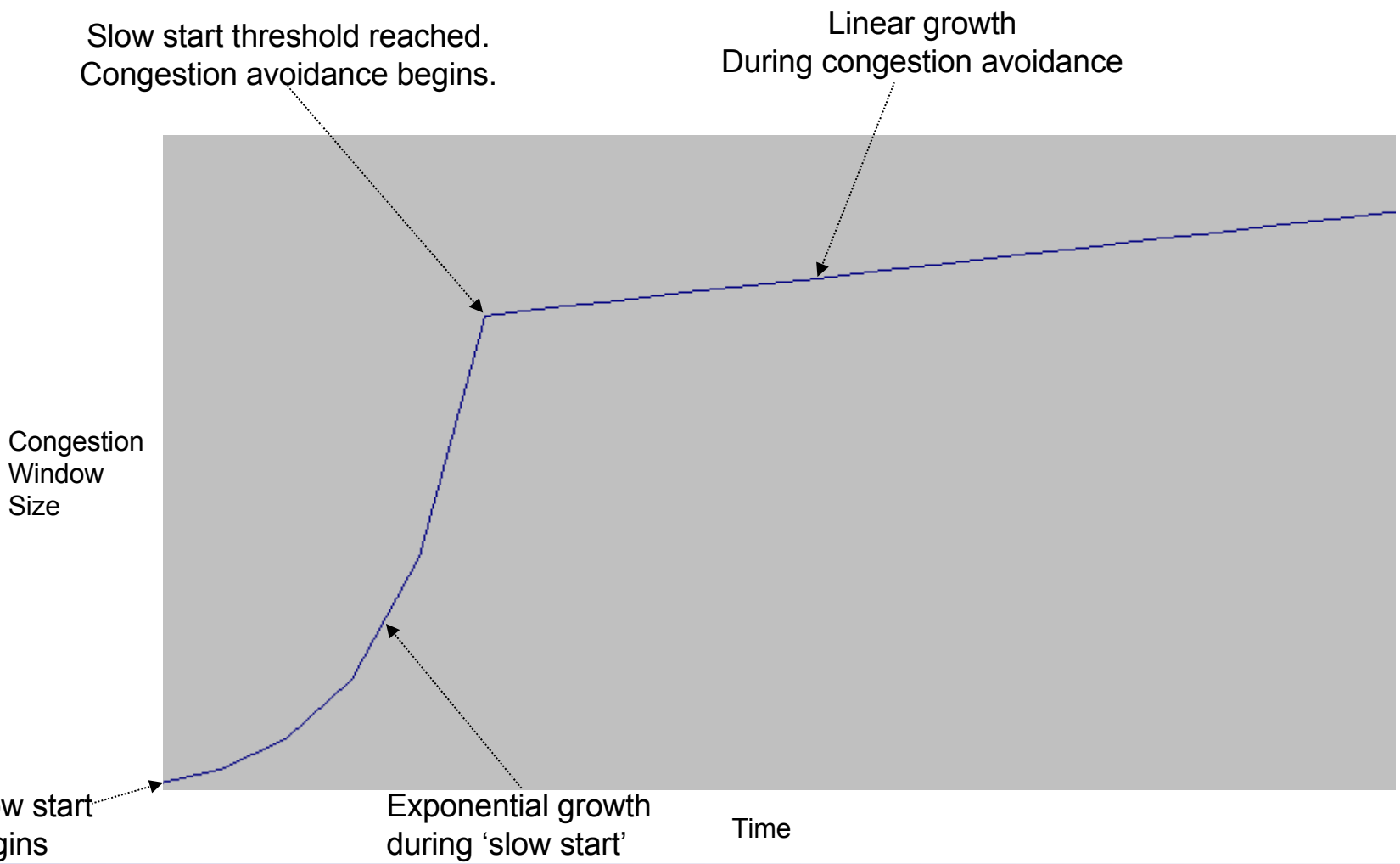
CONGESTION AVOIDANCE AND THE SLOW START THRESHOLD

The slow-start threshold (ssthresh) is a TCP variable that is used to slow exponential congestion-window growth before congestion occurs (set to “infinity” on session start).

When the congestion window exceeds ssthresh, TCP’s strategy changes from ‘slow start’ to ‘congestion avoidance’.

- During congestion avoidance, the congestion window is still increased in response to each ACK, but only in a slower, linear fashion (*additive increase*).
  - cwnd = cwnd + (1/cwnd)
  - Results in an increase of 1 MSS per RTT (cwnd ACKs, each adding 1/cwnd).

SLOW START, THE SLOW START THRESHOLD AND CONGESTION AVOIDANCE: AN EXAMPLE



DEALING WITH CONGESTION: PRINCIPLES

When congestion is detected, the sender should respond by reducing its share of the available bandwidth.

- This means that it must reduce its rate of packet transmission.
- In practical terms, this means that it must reduce its transmission window size.

Heavy congestion is deduced if a Retransmission Time-Out (RTO) occurs.

- In other words, if a packet is not acknowledged before the timeout, it is deemed lost due to congestion.

Lighter congestion is deduced if three duplicate ACKs are received.

- This indicates that packets have arrived out of order due to congestion or that some may have been lost, but that others are still ‘getting through’.

TCP'S RESPONSE TO CONGESTION

TCP's response depends upon whether heavy or lighter congestion was detected.

- Heavy congestion (an ACK times out):
  - Reduce congestion window to 1 MSS.
  - Reduce ssthresh to half the flight-size when the ACK timed out (*multiplicative decrease* → exponential backoff, no less will do).
  - Enter slow start mode.
- Lighter Congestion (three duplicate ACKs are received):
  - Perform fast retransmit and fast recovery. Goal is to
    - Re-send lost packet.
    - Set the sending rate to half of that when the loss was detected.
    - Keep packets flowing until the re-transmitted packet is acknowledged

FAST RETRANSMIT AND FAST RECOVERY

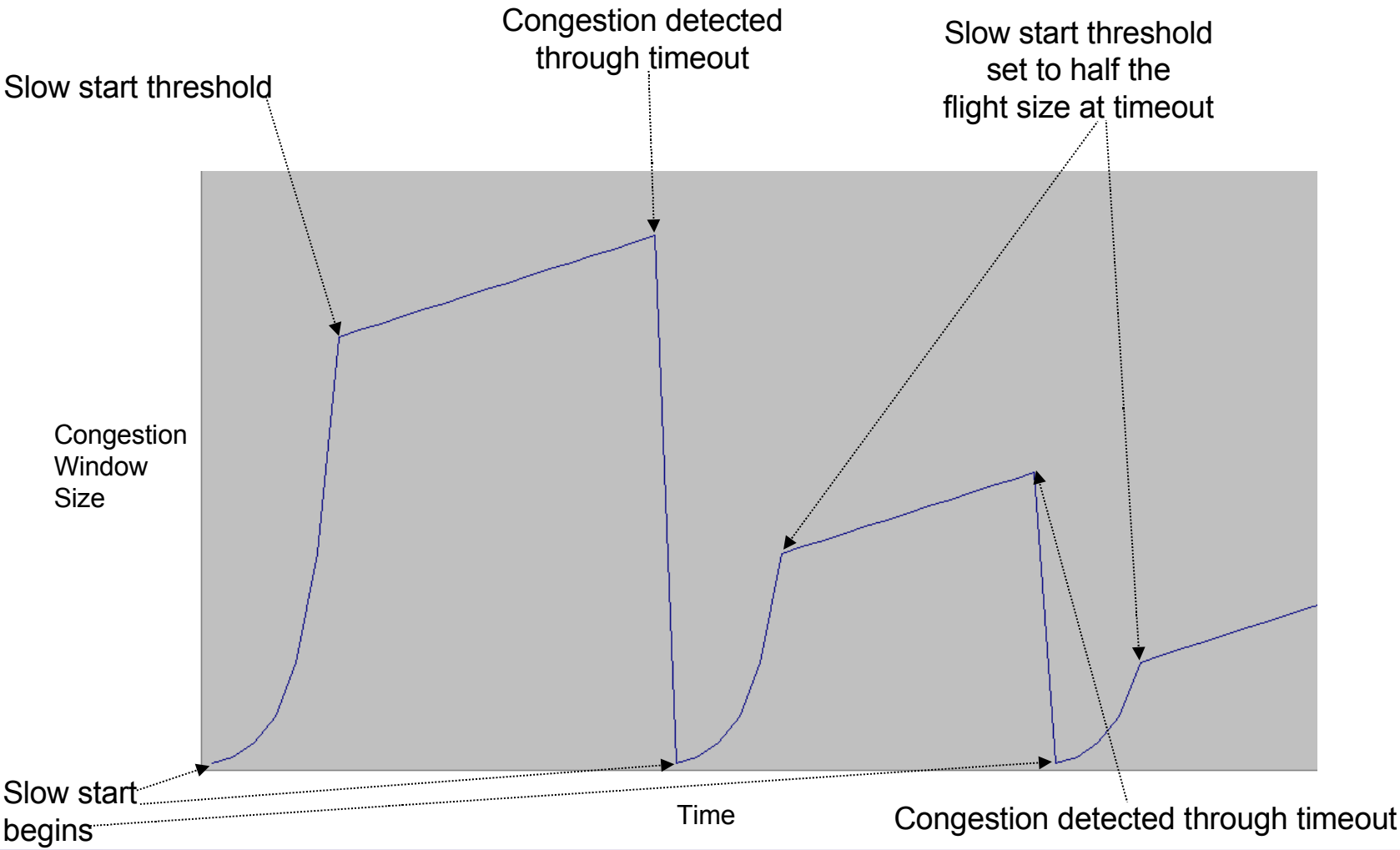
Fast Retransmit:

- Retransmit lost segment immediately (do not wait for RTO).

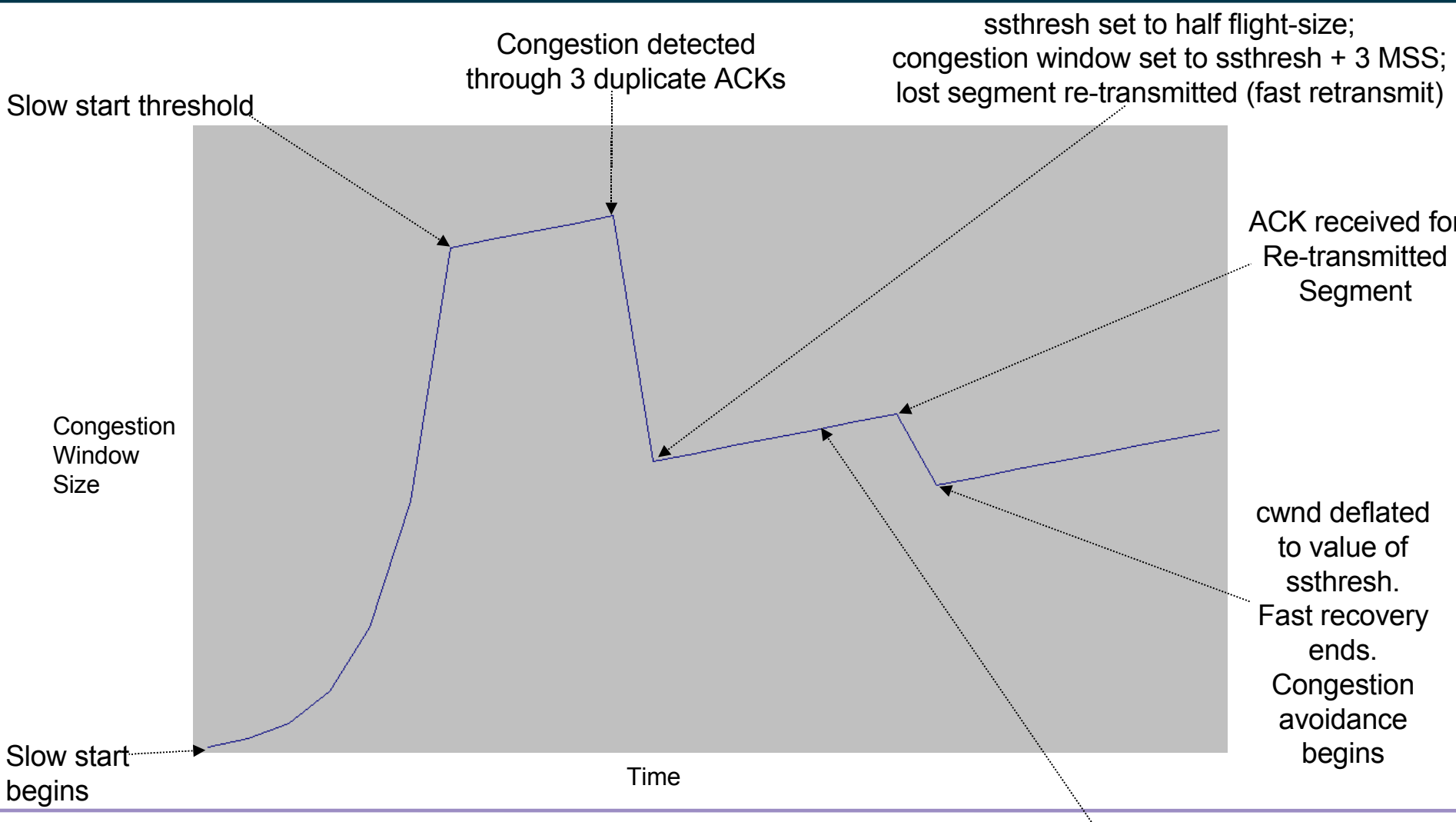
Fast Recovery, keep pipe from draining:

- Reduce ssthresh to half the flight-size.
- Duplicate ACKs don't move the window. Inflate congestion window artificially to keep packets flowing:
  - Set congestion window to ssthresh + 3 MSS (initial 3 duplicate ACKs).
  - Increase congestion window by 1 MSS for each extra duplicate ACK received.
- On acknowledgement of retransmitted segment, reduce congestion window to value of ssthresh and enter congestion avoidance mode.

THE WINDOW CLOSES AGGRESSIVELY IN RESPONSE TO RETRANSMISSION TIMEOUT



THE WINDOW CLOSES LESS IN RESPONSE TO DUPLICATE ACKS



PROBLEMS WITH TCP CONGESTION CONTROL

Fast recovery does not handle multiple packet loss well.

- Assumes that a single packet was lost per RTT, due to cumulative nature of ACKs.
- If two or more packets are lost at the same time, the losses are dealt with in a ‘serial’ fashion.
  - E.g. Fast recovery reduces the cwnd to half the flight size and then repeats this operation for each lost packet.
  - Can lead to ‘ACK starvation’ and deadlock.

Modern TCPs use

- Modified fast recovery algorithm using “partial ACKs”.
- Selective acknowledgements (SACK, RFC2018).

TROUBLE WITH THE ELEPHANTS (1)

Problem with ‘traditional TCP’ on Long Fat Networks (LFNs):

- BDP is high. Therefore:
- Optimal congestion window is big.
- Packet loss exponentially reduces congestion window.
- Recovery from congestion takes too long, because of large congestion window and high round-trip time.
- Given realistic rates of packet loss, traditional TCP cannot maintain an optimal-sized average congestion window in an LFN. Therefore:
- Throughput is sub-optimal.

TROUBLE WITH THE ELEPHANTS (2)

An example:

- Consider a path of 1 Gbps at 100 ms RTT.
  - BDP = 12.5 MB or 8333 1500 byte packets.
- It takes log(8333)/log(2) = 13 RTTs (1.3 seconds) to inflate congestion window to equal BDP during slow-start.
- It takes 4166 RTTs (416 seconds) to inflate congestion window from half maximum size back to maximum size during congestion avoidance.
- With a bit error rate of 10^{-11}, there will be a corrupt packet in every 1000^{th} window → unlikely to ever reach maximum window (also see last exercise).

HIGH-SPEED TCP VARIANTS

Recent research has focused on optimising TCP's performance on (LFNs).

- Current congestion algorithms limit efficiency of network resource utilisation.
- New TCP variants introduce changes to the way that the congestion window is calculated.

High-Speed TCP Variants fall into two (orthogonal) categories:

- Explicit congestion control protocols.
  - These rely on feedback from routers.
- Implicit congestion control protocols.
  - Deduce congestion by measuring loss / delay etc.

EXPLICIT CONGESTION CONTROL PROTOCOLS (1)

'Traditional TCP' congestion control mechanisms rely on information gathered by the sender and the receiver only.

- I.e. they use information from the two ends of the path to deduce that the middle is congested.

By contrast, explicit congestion control protocols rely on routers in 'the middle' of the path to supply information.

- Routers return information (sometimes via the receiver) about:
  - Congestion that has been experienced, or
  - The optimal 'share' of capacity that should be allocated to the path.
- This information is often held within packet-headers.
- In response, the sender re-sizes its congestion window.

EXPLICIT CONGESTION CONTROL PROTOCOLS (2)

Advantage

- Congestion notification is usually quicker and more sensitive.

Disadvantage

- Routers do more work and need to support Explicit Congestion Control Protocols; often they do not.

Examples of Explicit Congestion Control Protocols:

- Source Quench (now obsolete).
- Explicit Congestion Notification (ECN)
  - More information about each of these is available in the PERT Knowledge Base.

IMPLICIT CONGESTION CONTROL PROTOCOLS (1)

Implicit congestion control protocols seek to ‘re-engineer’ TCP’s congestion control algorithms: slow start, AIMD (additive increase, multiplicative decrease). Many different approaches

- High Speed TCP (HS-TCP)
  - Calculate AIMD parameters as a function of cwnd to achieve the maximum window at a target packet drop rate.
- Hamilton TCP (H-TCP)
  - Calculate AIMD parameters as a function of
    - Time since last congestion event.
    - Throughput before congestion event.
    - Maximum and minimum round-trip times

IMPLICIT CONGESTION CONTROL PROTOCOLS (2)

Advantages:

- Facilitate a bigger average congestion window size over Long Fat Networks (LFNs).
  - And therefore higher throughput.
- Do not require router support.

Disadvantages:

- Limited real-world implementations.
- Some variants can starve traditional TCP connections of bandwidth when run in parallel with them.
  - Termed ‘efficient but unfriendly’.

NETWORK TUNING FOR BETTER CONGESTION CONTROL (1)

Router buffers

- When incoming traffic exceeds outbound capacity, routers buffer packets in a queue.
- When buffer is full, newly arriving packets are simply dropped.
- Whole bursts of packets can be dropped.
  - Can lead to synchronised traffic bursts and lower throughput.
- Packets can remain in buffers for a long period.
  - Leads to increased one-way delay and round-trip times.

In response to these issues, router buffers can be *actively managed*.

NETWORK TUNING FOR BETTER CONGESTION CONTROL (2)

Active Queue Management (AQM)

- Network nodes send congestion signals to avoid buffers filling (ECN approach).
- Most common form of Active Queue Management is Random Early Detection (RED).
  - RED is supported by many routers, but is not active by default.
  - Needs to be tuned.
  - Samples queue-size over time.
  - Can drop packets to keep queue-size small.
  - Short queue keeps one-way delay and round-trip time low.
  - Also helps to avoid synchronisation effects and related throughput degradation.

NETWORK TUNING FOR BETTER CONGESTION CONTROL (3)

Sizing of Network Buffers

- Approach 1: Traditional Wisdom
  - Suggests that network node should be able to buffer an end-to-end round-trip time's worth of line-rate traffic.
  - This is to accommodate bursts.
- Approach 2: Suggested by recent research
  - Suggests that much smaller buffers are sufficient when there is a high degree of multiplexing of TCP streams.

Simple Model for TCP Throughput

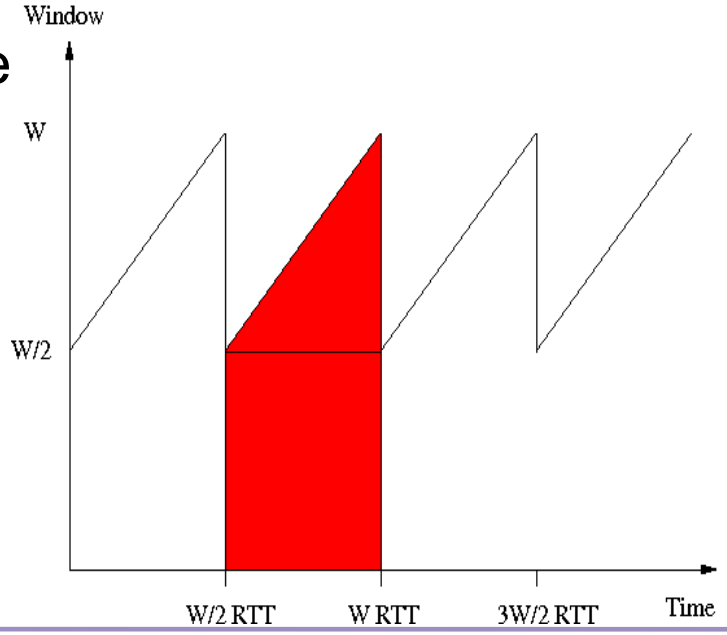
Path with constant RTT and average loss rate p. Assume

- Perfectly periodic loss events, i.e. sequences of 1/p packets (of size MSS) followed by one lost packet.
- Every packet is acknowledged, i.e. window increases by 1 per RTT, cycle lasts W/2 RTT seconds.

Throughput = Data per cycle / cycle time

Data per cycle (units of MSS): $\frac{3}{2}\left(\frac{W}{2}\right)^2 = \frac{1}{p}$

$$T = \frac{MSS}{p}\,\frac{1}{\frac{W}{2}RTT} = \frac{MSS}{RTT}\,\frac{2}{pW} = \frac{MSS}{RTT}\sqrt{\frac{3}{2p}}$$



ACTIVITY

Exercise

- Explore performance over two contrasting paths:
  - A long path with low levels of packet loss.
  - A short path experiencing high levels of packet loss.