Congestion Control

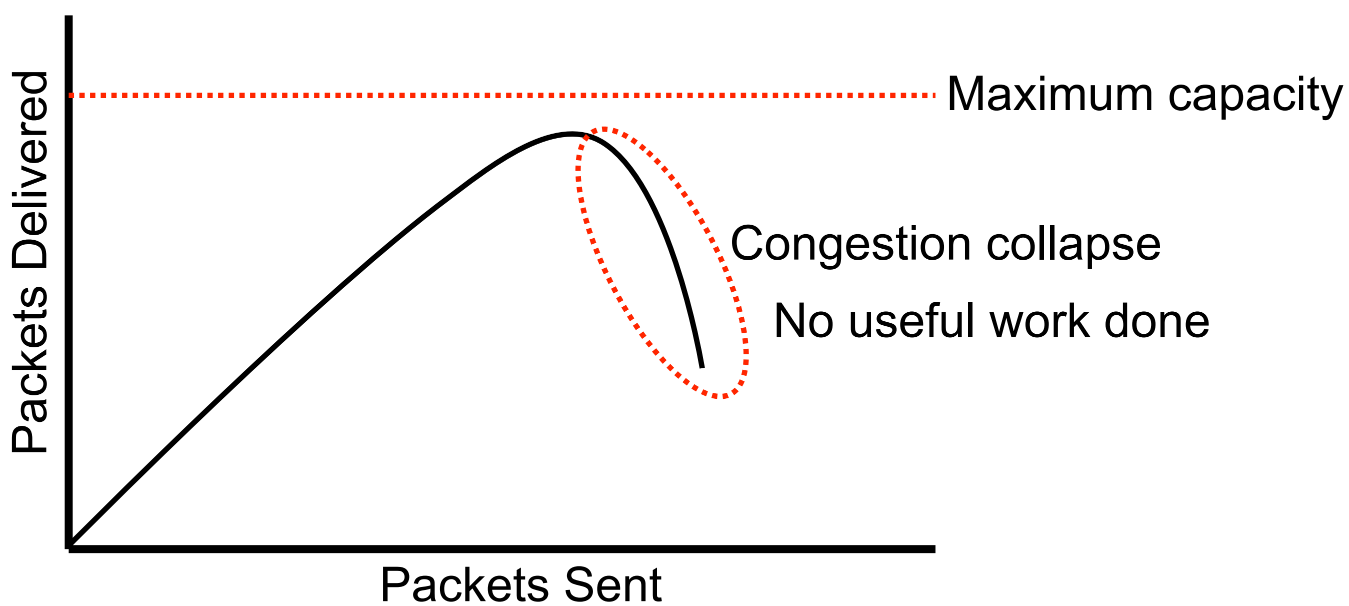
Networked Systems 3
Lecture 14

Lecture Outline

- Congestion control principles
- Congestion control in the Internet
  - TCP congestion control
  - Alternative approaches

What is Congestion Control?

- Adapting speed of transmission to match available end-to-end network capacity
  - Analogous to flow control on a single link
- Preventing congestion collapse of a network



Occurred in the Internet in 1986, before congestion control added

Network or Transport Layer?

- Can implement congestion control at either the network or the transport layer
  - Network layer – safe, ensures all transport protocols are congestion controlled, requires all applications to use the same congestion control scheme
  - Transport layer – flexible, transports protocols can optimise congestion control for applications, but a misbehaving transport can congest the network

Congestion Control Principles

- Two key principles, first elucidated by Van Jacobson in 1988: ["Congestion Avoidance and Control", Proc. ACM SIGCOMM'88]
  - Conservation of packets
  - Additive increase/multiplicative decrease in sending rate

Source: PARC

Van Jacobson

- Together, ensure stability of the network

Conservation of Packets

- The network has a certain capacity
  - The *bandwidth x delay* product of the path
- When in equilibrium at that capacity, send one packet for each acknowledgement received
  - Total number of packets in transit is constant
    - "ACK clocking" – each acknowledgement "clocks out" the next packet
  - Automatically reduces sending rate as network gets congested and delivers packets more slowly

AIMD Algorithms

- Adjust sending rate according to an additive increase/multiplicative decrease algorithm
  - Start slowly, increase gradually to find equilibrium
    - Add a small amount to the sending speed each time interval without loss
    - For a window-based algorithm $w_i = w_{i-1} + \alpha$ each RTT, where $\alpha = 1$ typically
  - Respond to congestion rapidly
    - Multiply sending window by some factor $\beta < 1$ each interval loss seen
    - For a window-based algorithm $w_i = w_{i-1} \times \beta$ each RTT, where $\beta = {}^{1}/_{2}$ typically
- Faster reduction than increase → stability

How to Adapt Transmission?

- For sliding window protocols:
  - Acknowledge each packet, only send new data when an acknowledgement received
  - Adjust size of window, based on AIMD rules

- Other types of protocol should do something similar

Congestion in the Internet

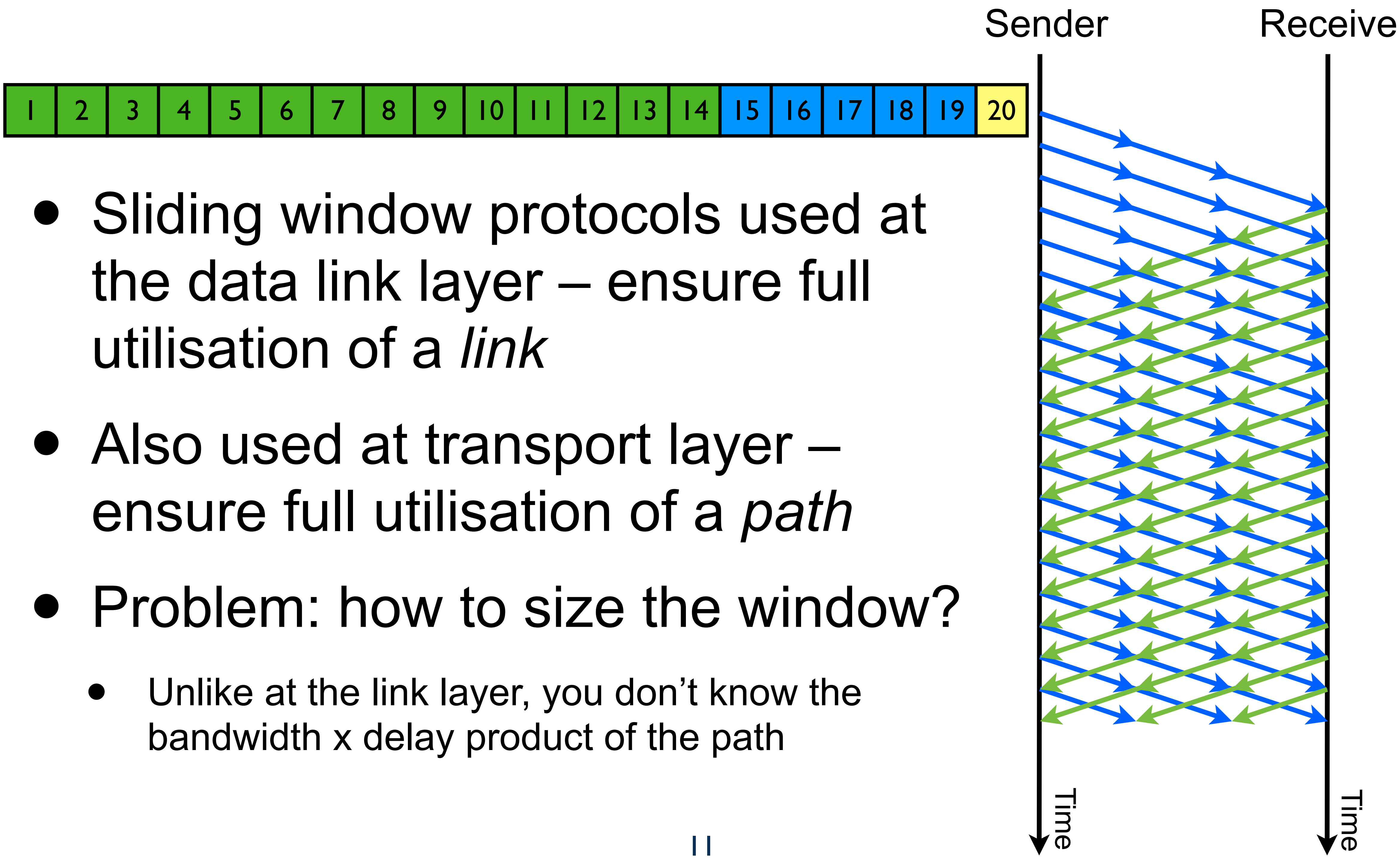
- Congestion control provided by transport layer
  - Dominant protocol is TCP
  - Others try to be “TCP Friendly”
- Network layer signals congestion to transport
  - Packets discarded on congestion
    - Note: implications for wireless Internet
  - Modern TCP also has ECN bits, but not widely used

TCP Congestion Control

- TCP is a sliding window protocol, measuring the window size in bytes
  - Plus *slow start* and *congestion avoidance*
  - Gives an approximately equal share of the bandwidth to each flow sharing a link

  - *“The world’s most baroque sliding-window protocol” – Lloyd Wood*

TCP Congestion Control



- Sliding window protocols used at the data link layer – ensure full utilisation of a *link*
- Also used at transport layer – ensure full utilisation of a *path*
- Problem: how to size the window?
  - Unlike at the link layer, you don't know the bandwidth x delay product of the path

TCP Congestion Control

- Issues with transport layer sliding window protocols:
  - How to choose initial window?
  - How to find the link capacity?
    - Slow start to estimate the bottleneck link capacity
    - Congestion avoidance to probe for changes in capacity

Choosing the Initial Window

- How to choose initial window size, W_{init}?
  - No information → need to measure path capacity
  - Start with a small window, increase until congestion
    - W_{init} of one packet per round-trip time is the only safe option – equivalent to a stop-and-wait protocol – but is usually overly pessimistic
    - TCP uses a slightly larger initial window:
      $W_{init} = \min(4 \times MSS, \max(2 \times MSS, 4380 \text{ bytes}))$ packets per RTT
    - Example: an Ethernet with MTU of 1500 bytes, TCP/IP headers of 40 bytes → $W_{init} = \min(4 \times 1460, \max(2 \times 1460, 4380)) = 4380$ bytes = 3 packets per RTT

MSS = Maximum Segment Size
(MTU minus TCP/IP header size)

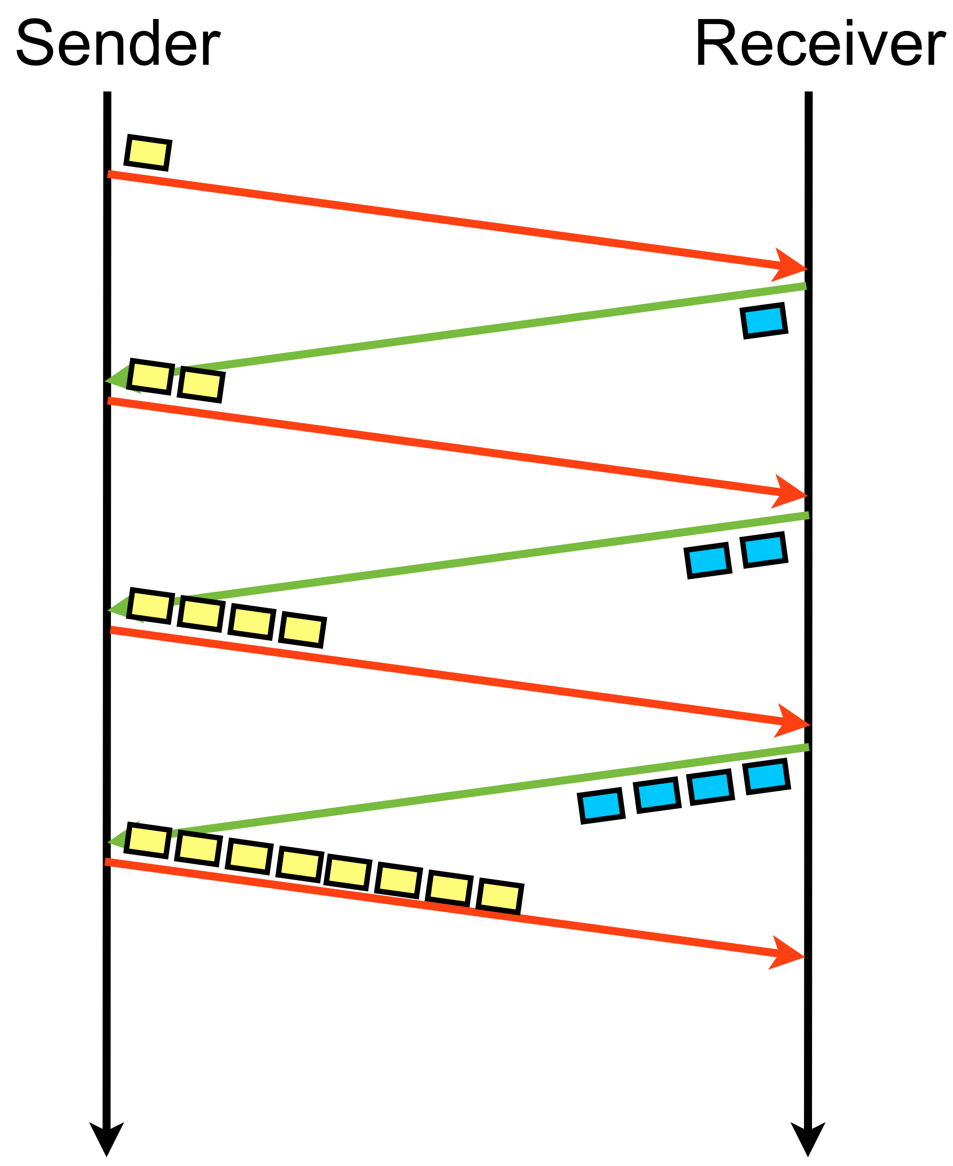
Finding the Link Capacity

- The initial window allows you to send
- How to choose the right window size to match the link capacity? Two issues:
    - How to find the correct window for the path when a new connection starts – *slow start*
    - How to adapt to changes in the available capacity once a connection is running – *congestion avoidance*

Slow Start

- Initial window, W_{init} = 1 packet per RTT
  - Or similar… a “slow start” to the connection
- Need to rapidly increase to the correct value for the network
  - Each acknowledgement for new data increases the window by 1 packet per RTT
  - On packet loss, immediately stop increasing window

Slow Start



- Two packets generated per acknowledgement
- The window doubles on every round trip time – until loss occurs
- Rapidly finds the correct window size for the path

Congestion Avoidance

- Congestion avoidance mode used to probe for changes in network capacity
  - E.g. is sharing a connection with other traffic, and that traffic stops, meaning the available capacity increases
- Window increased by 1 packet per RTT
  - Slow, additive increase in window: $w_i = w_{i-1} + 1$
  - Until congestion is observed → respond to loss

Detecting Congestion

- TCP uses cumulative positive ACKs → two ways to detect congestion
  - Triple duplicate ACK → packet lost due to congestion
  - ACKs stop arriving → no data reaching receiver; link has failed completely somewhere
    - How long to wait before assuming ACKs have stopped?
    - T_{rto} = max(1 second, average RTT + (4 x RTT variance))
      - Statistical theory: 99.99% of data lies with 4σ of the mean, assuming normal distribution (where variance of the distribution = σ^2)

Responding to Congestion

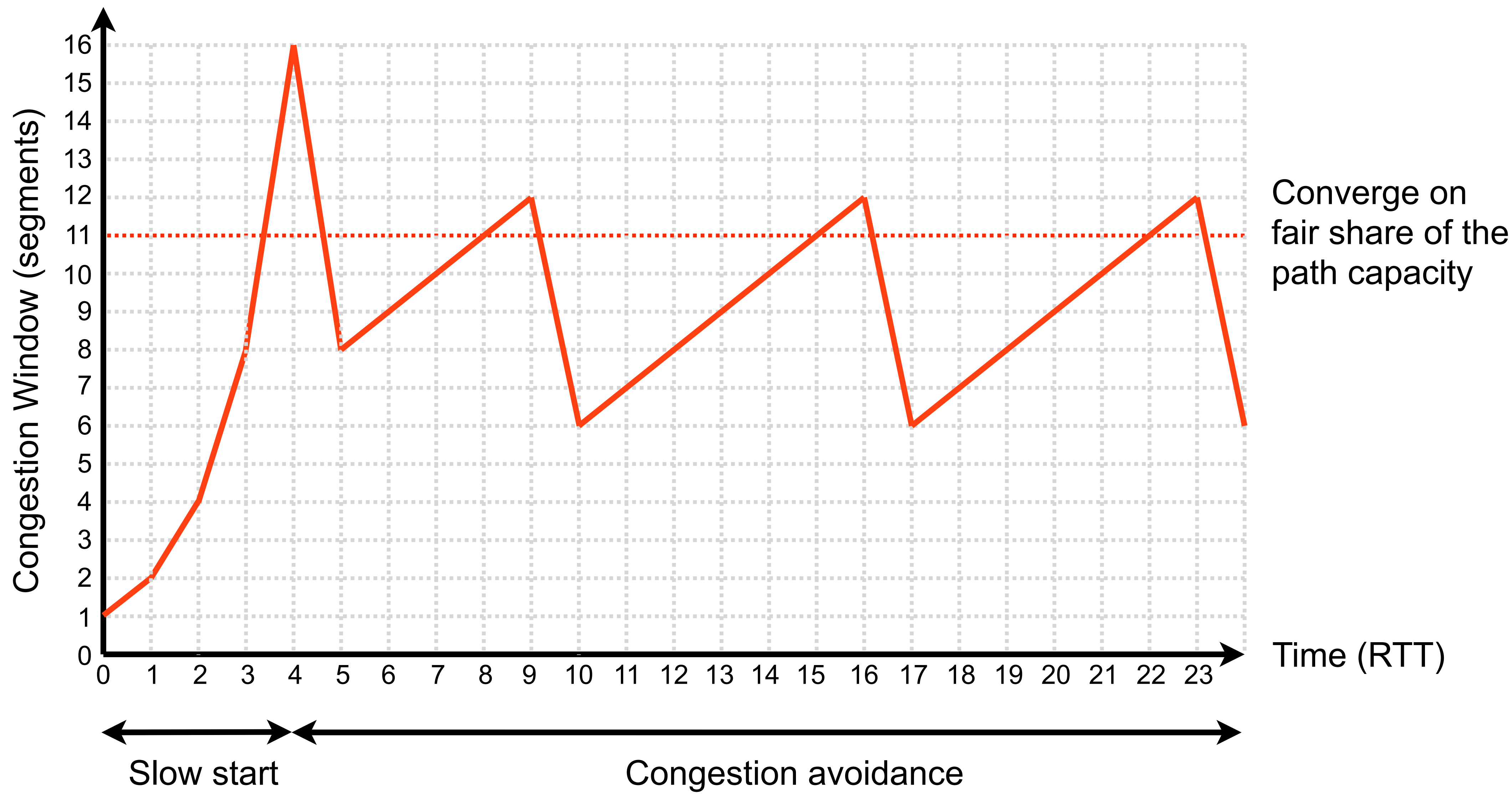
- If loss detected by triple-duplicate ACK:
  - Transient congestion, but data still being received
  - Multiplicative decrease in window: $w_i = w_{i-1} \times 0.5$
  - Rapid reduction in sending speed allows congestion to clear quickly, avoids congestion collapse

Responding to Congestion

- If loss detected by time-out:
    - No packets received for a long period of time – likely a significant problem with network (e.g., link failed)
    - Return to initial sending window, and probe for the new capacity using slow start
    - Assume the route has changed, and you know nothing about the new path

Congestion Window Evolution

Typical evolution of TCP window, assuming W_{init} = 1



The Limitations of TCP

- TCP assumes loss is due to congestion
  - Too much traffic queued at an intermediate link → some packets dropped
  - This is not always true:
    - Wireless networks
    - High-speed long-distance optical networks
  - Much research into improved versions of TCP for wireless links

Other Congestion Control

- TCP is not appropriate for all applications
- But need to be *TCP Friendly*:
  - Avoid congestion collapse
  - Avoid gratuitous unfairness
- Streaming media applications prefer something with a smoother response function
  - Lots of research ongoing, but no accepted standards

Questions?