TCP advanced algorithms

MŰEGYETEM 1782

Budapest University of Technology and Economics

Department of
Telecommunications and Media Informatics

What is Congestion?

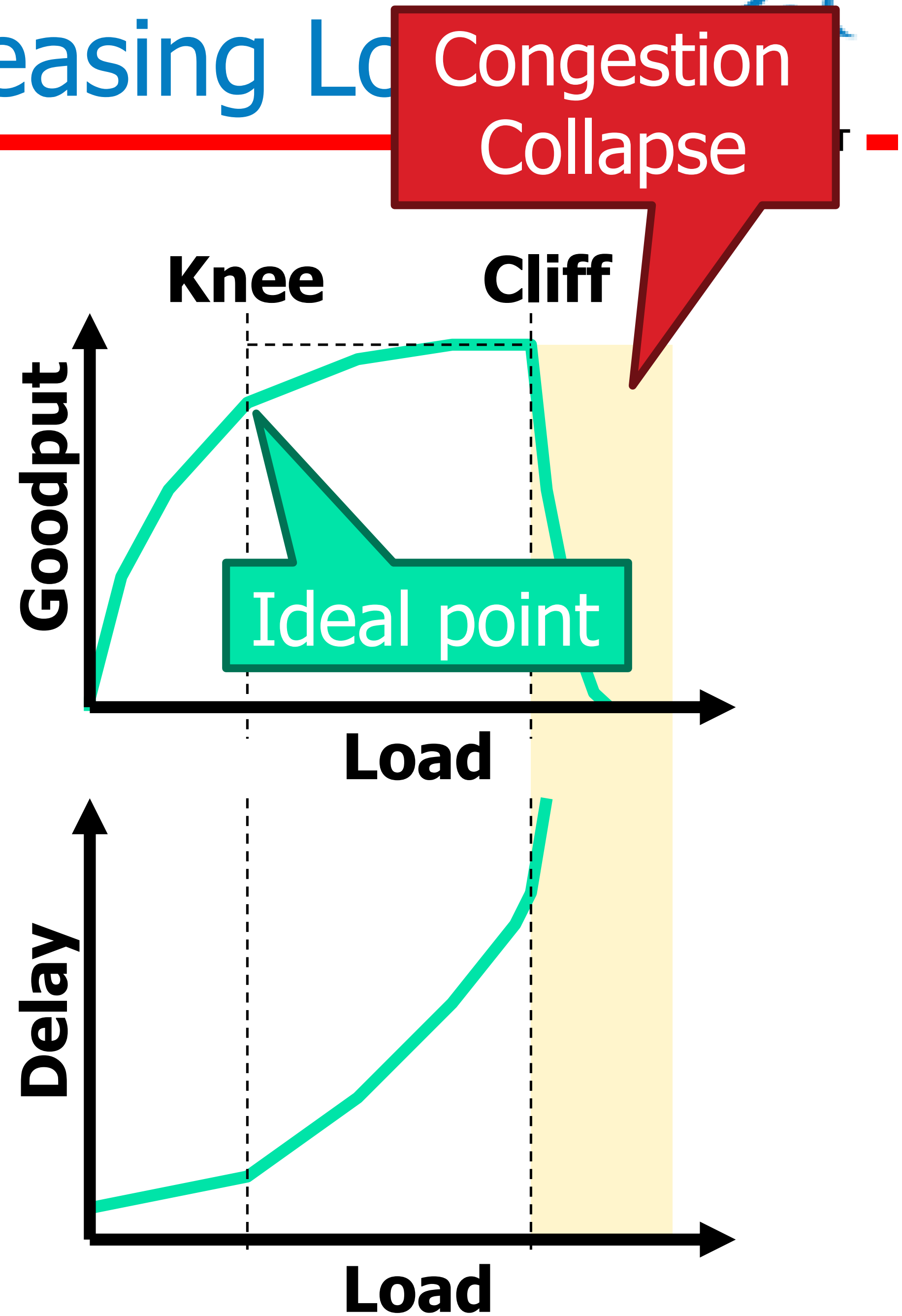
- Load on the network is higher than capacity
  - Capacity is not uniform across networks
    - Modem vs. Cellular vs. Cable vs. Fiber Optics
  - There are multiple flows competing for bandwidth
    - Residential cable modem vs. corporate datacenter
  - Load is not uniform over time
    - 10pm, Sunday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

Why is Congestion Bad?

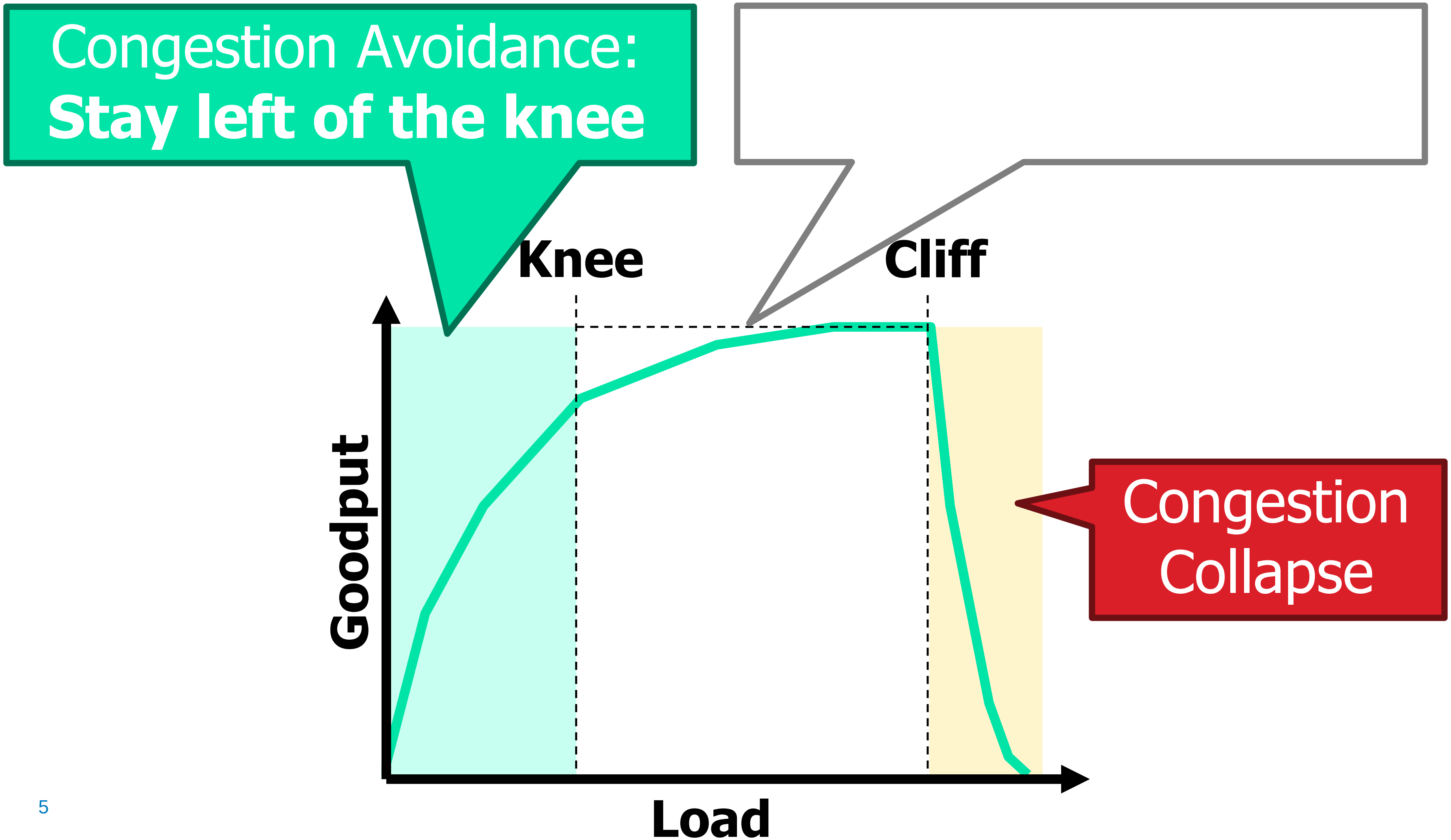
- Results in packet loss
  - Routers have finite buffers, packets must be dropped
- Practical consequences
  - Router queues build up, delay increases
  - Wasted bandwidth from retransmissions
  - Low network goodput

The Danger of Increasing Lo

- Knee – point after which
  - Throughput increases very slow
  - Delay increases fast
- In an M/M/1 queue
  - Delay = 1/(1 – utilization)
- Cliff – point after which
  - Throughput → 0
  - Delay → ∞



Cong. Control vs. Cong. Avoidance



Advertised Window, Revisited

BME-TMIT

- Does TCP's advertised window solve congestion?

NO

- The advertised window only protects the receiver
- A sufficiently fast receiver can max the window
  - 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

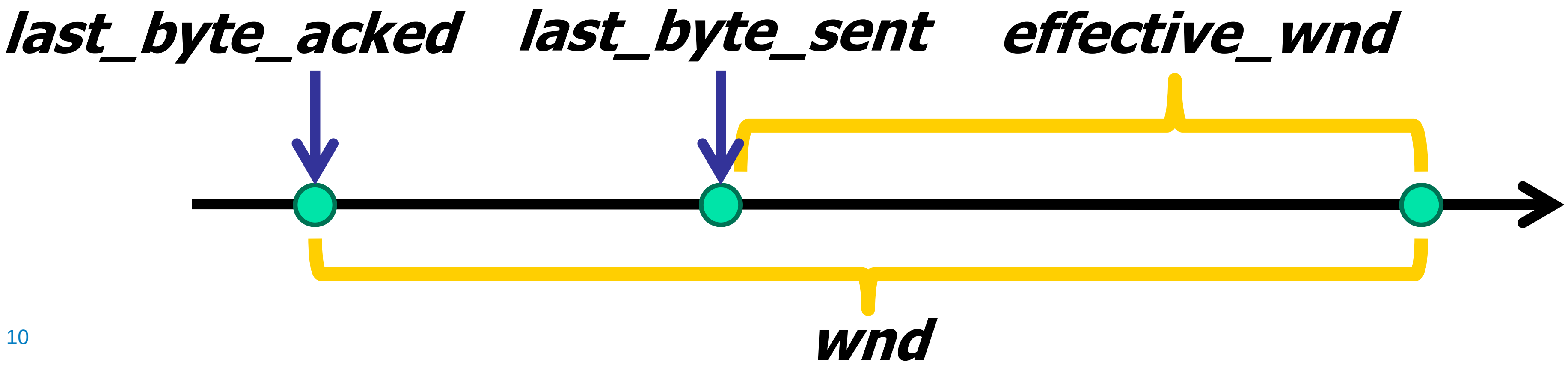
TCP Congestion Control

- Each TCP connection has a window
  - Controls the number of unACKed packets
- Sending rate is ~ window/RTT
- Idea: vary the window size to control the send rate
- Introduce a congestion window at the sender
  - Congestion control is sender-side problem

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 sign[al]
     - 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)

Except on wireless networks

2. Rate adjustment algorithm
   - Modify *cwnd*
   - Probe for bandwidth
   - Responding to congestion

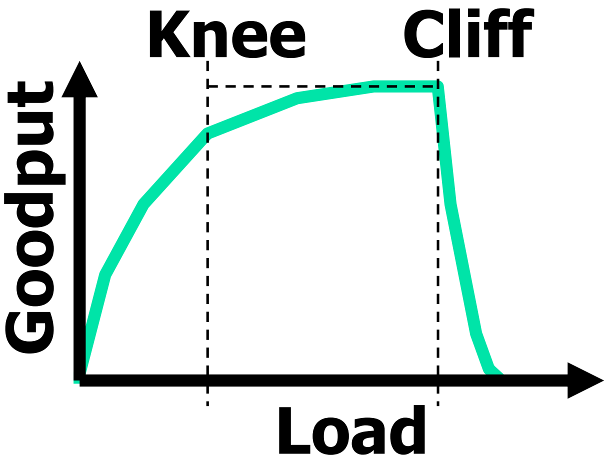
Rate Adjustment

- 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: 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

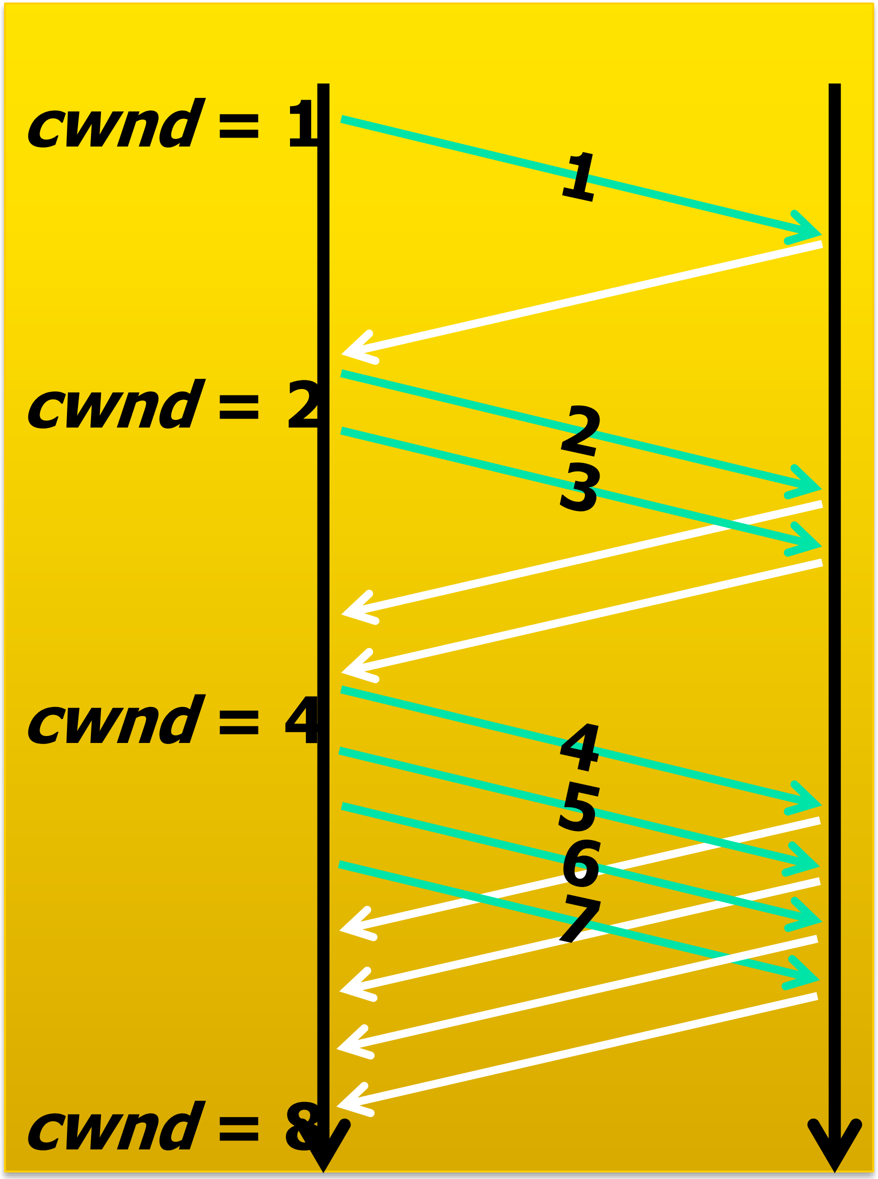
Slow Start



- 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

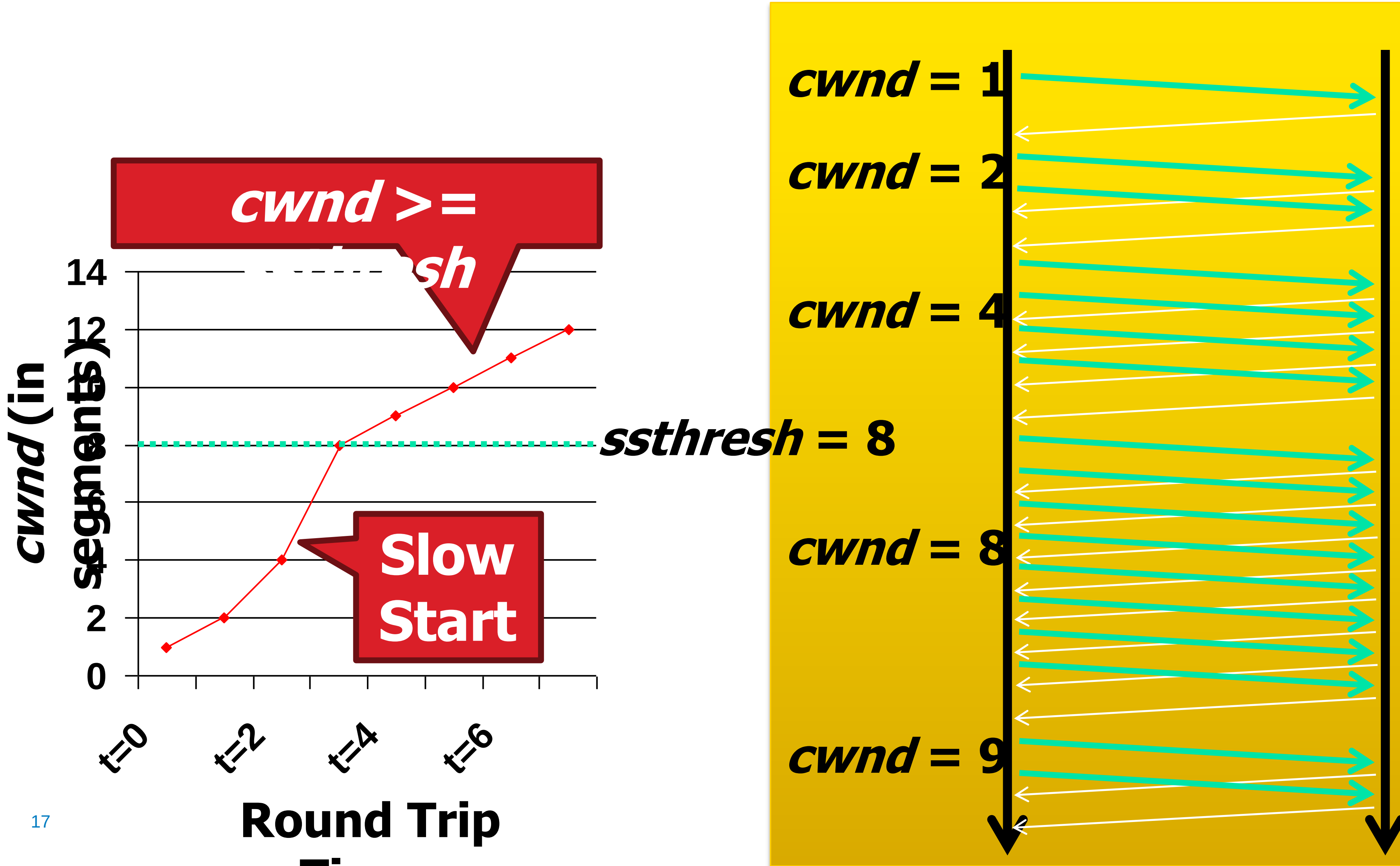
- *cwnd* **grows rapidly**
- **Slows down when…**
  - ***cwnd >= ssthresh***
  - **Or a packet drops**



Congestion Avoidance

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

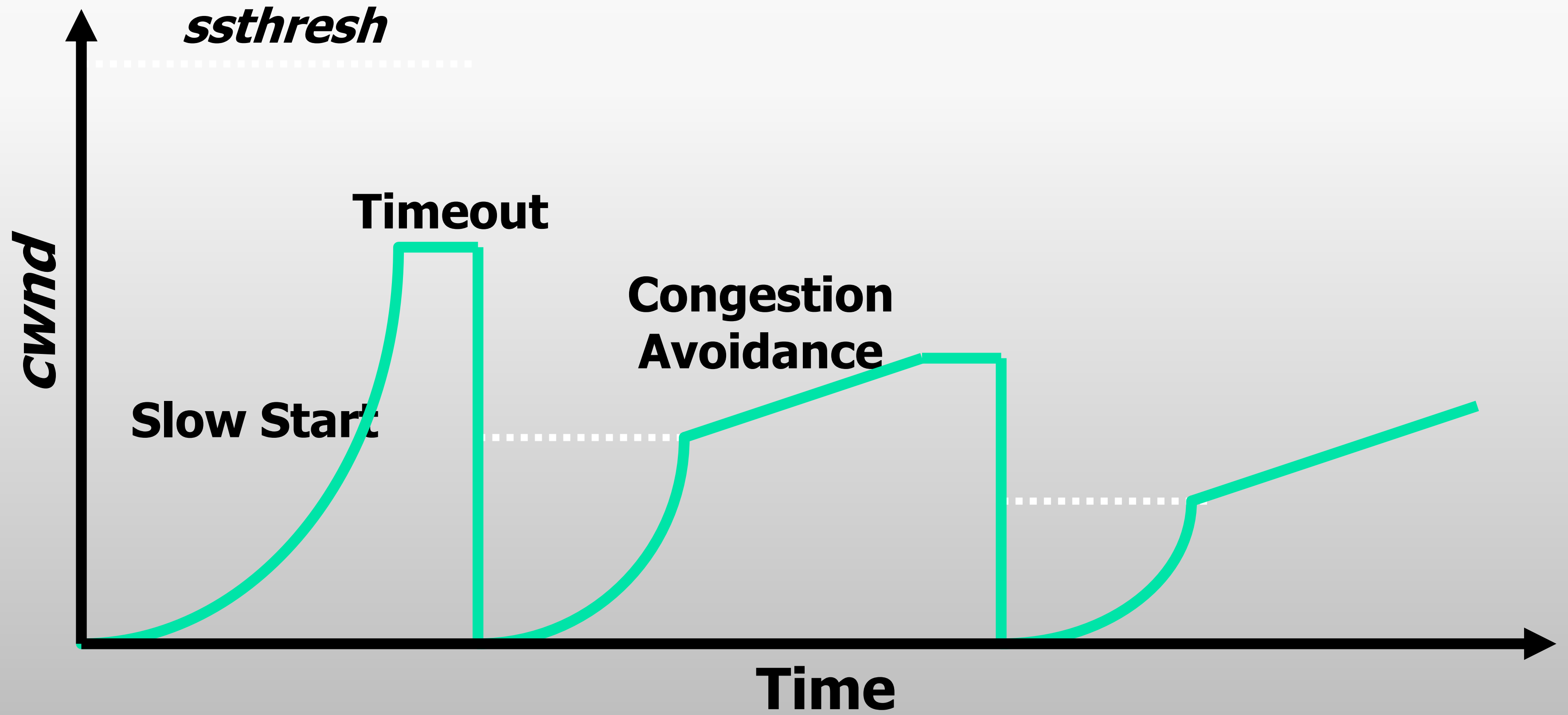
Congestion Avoidance Example



TCP Pseudocode


```
Initially:
        cwnd = 1;
        ssthresh = adv_wnd;
New ack received:
        if (cwnd < ssthresh)
            /* Slow Start*/
            cwnd = cwnd + 1;
        else
            /* Congestion Avoidance */
            cwnd = cwnd + 1/cwnd;
Timeout:
        /* Multiplicative decrease */
        ssthresh = cwnd/2;
        cwnd = 1;
```

The Big Picture



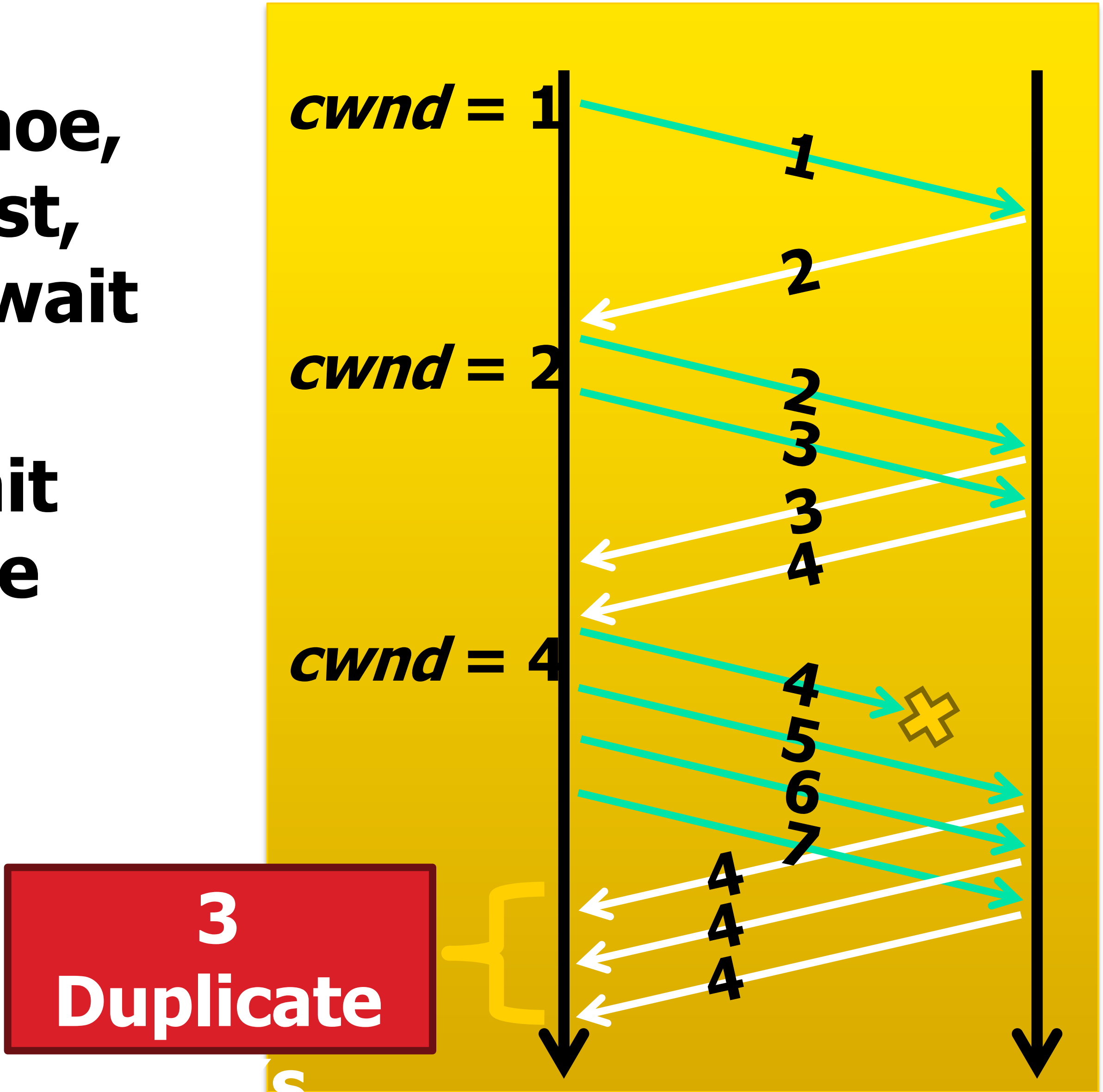
The Evolution of TCP

BME-TMIT

- Thus far, we have discussed TCP Tahoe
  - Original version of TCP
- However, TCP was invented in 1974!
  - Today, there are many variants of TCP
- Early, popular variant: TCP Reno
  - Tahoe features, plus…
  - Fast retransmit
  - Fast recovery

TCP Reno: Fast Retransmit

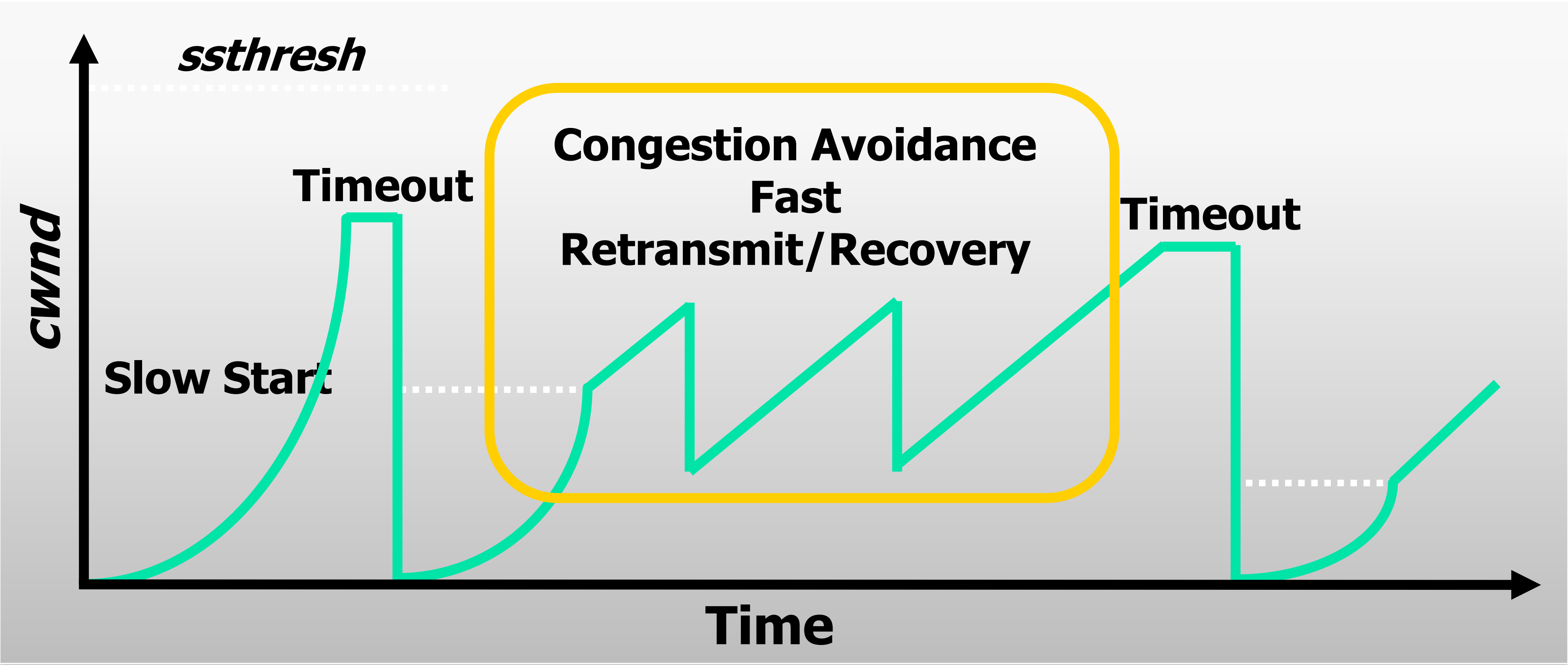
- **Problem: in Tahoe, if segment is lost, there is a long wait until the RTO**
- **Reno: retransmit after 3 duplicate ACKs**



TCP Reno: Fast Recovery

- After a fast-retransmit set *cwnd* to *ssthresh/2*
  - i.e. don't reset *cwnd* to 1
  - Avoid unnecessary return to slow start
  - Prevents expensive timeouts
- But when RTO expires still do *cwnd* = 1
  - Return to slow start, same as Tahoe
  - Indicates packets aren't being delivered at all
  - i.e. congestion must be really bad

Fast Retransmit and Fast Recovery



- At steady state, *cwnd* oscillates around the optimal window size
- TCP always forces packet drops

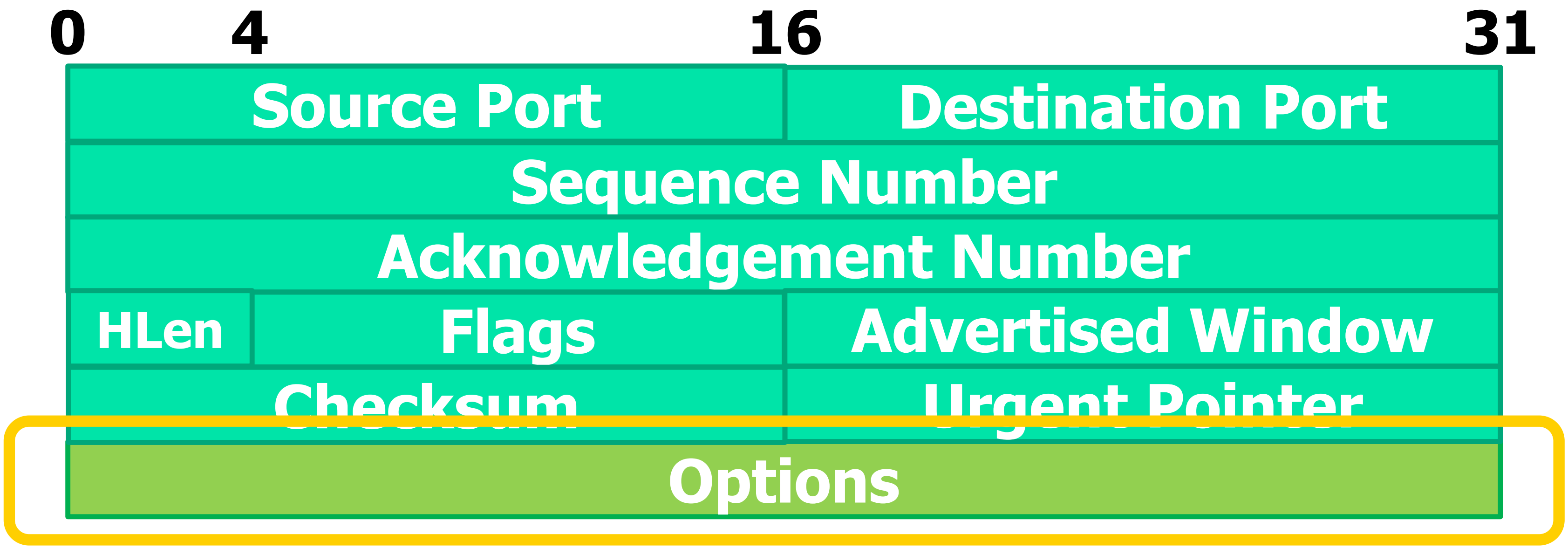
Many TCP Variants…

- Tahoe: the original
  - Slow start with AIMD
  - Dynamic RTO based on RTT estimate
- Reno: fast retransmit and fast recovery
- NewReno: improved fast retransmit
  - Each duplicate ACK triggers a retransmission
  - Problem: >3 out-of-order packets causes pathological retransmissions
- Vegas: delay-based congestion avoidance
- And many, many, many more…

High Bandwidth-Delay Product

- Key Problem: TCP performs poorly when
  - The capacity of the network (bandwidth) is large
  - The delay (RTT) of the network is large
  - Or, when bandwidth * delay is large
    - b * d = maximum amount of in-flight data in the network
    - a.k.a. the bandwidth-delay product
- Why does TCP perform poorly?
  - Slow start and additive increase are slow to converge
  - TCP is ACK clocked
    - i.e. TCP can only react as quickly as ACKs are received
    - Large RTT → ACKs are delayed → TCP is slow to react

Common TCP Options



- Window scaling
- SACK: selective acknowledgement
- Maximum segment size (MSS)
- Timestamp

Window Scaling

- Problem: the advertised window is only 16-bits
  - Effectively caps the window at 65536B, 64KB
  - Example: 1.5Mbps link, 513ms RTT

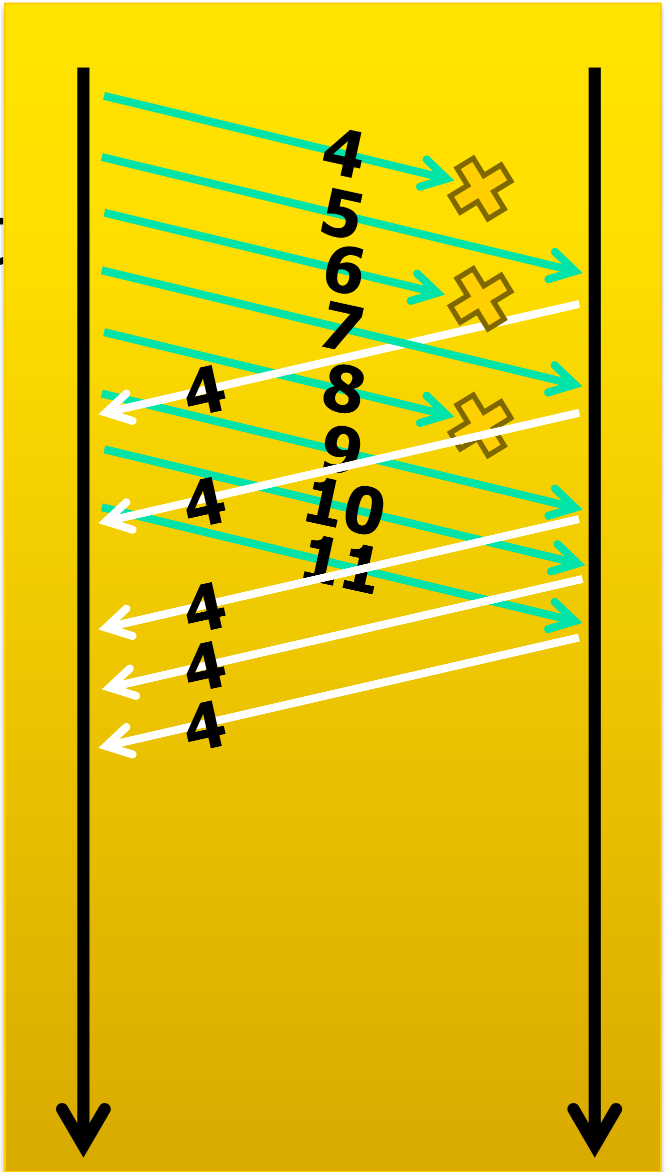
$$(1.5\text{Mbps} * 0.513\text{s}) = 94\text{KB}$$

64KB / 94KB = 68% of maximum possible speed

- Solution: introduce a window scaling value
  - *wnd = adv_wnd << wnd_scale;*
  - Maximum shift is 14 bits, 1GB maximum window

SACK: Selective Acknowledgment

- Problem: duplicate ACKs only tell us about 1 missing packet
  - Multiple rounds of dup ACKs needed to fill all holes
- Solution: selective ACK
  - Include received, out-of-order sequence numbers in TCP header
  - Explicitly tells the sender about holes in the sequence



Other Common Options

- Maximum segment size (MSS)
  - Essentially, what is the hosts MTU
  - Saves on path discovery overhead
- Timestamp
  - When was the packet sent (approximately)?
  - Used to prevent sequence number wraparound
  - PAWS algorithm

Thank You!

- End -

MŰEGYETEM 1782

Budapest University of Technology and Economics

Department of
Telecommunications and Media Informatics