COSC264
Introduction to Computer Networks and the Internet

TCP Congestion Control

Dr. Barry Wu
Wireless Research Centre
University of Canterbury
barry.wu@canterbury.ac.nz

TCP reliable data transfer: a summary

- TCP creates rdt service on top of IP’s unreliable service
- Pipelined segments
- Cumulative ACKs
- TCP uses single retransmission timer
- Retransmissions are triggered by:
  - timeout events
  - duplicate ACKs (*fast retransmit*)

TCP Round Trip Time and Timeout

Setting the timeout

- To measure the variability of the RTT;
- first estimate of how much SampleRTT deviates from EstimatedRTT:

```
DevRTT = (1-β)*DevRTT +
               β*|SampleRTT-EstimatedRTT|

(typically, β = 0.25)
```

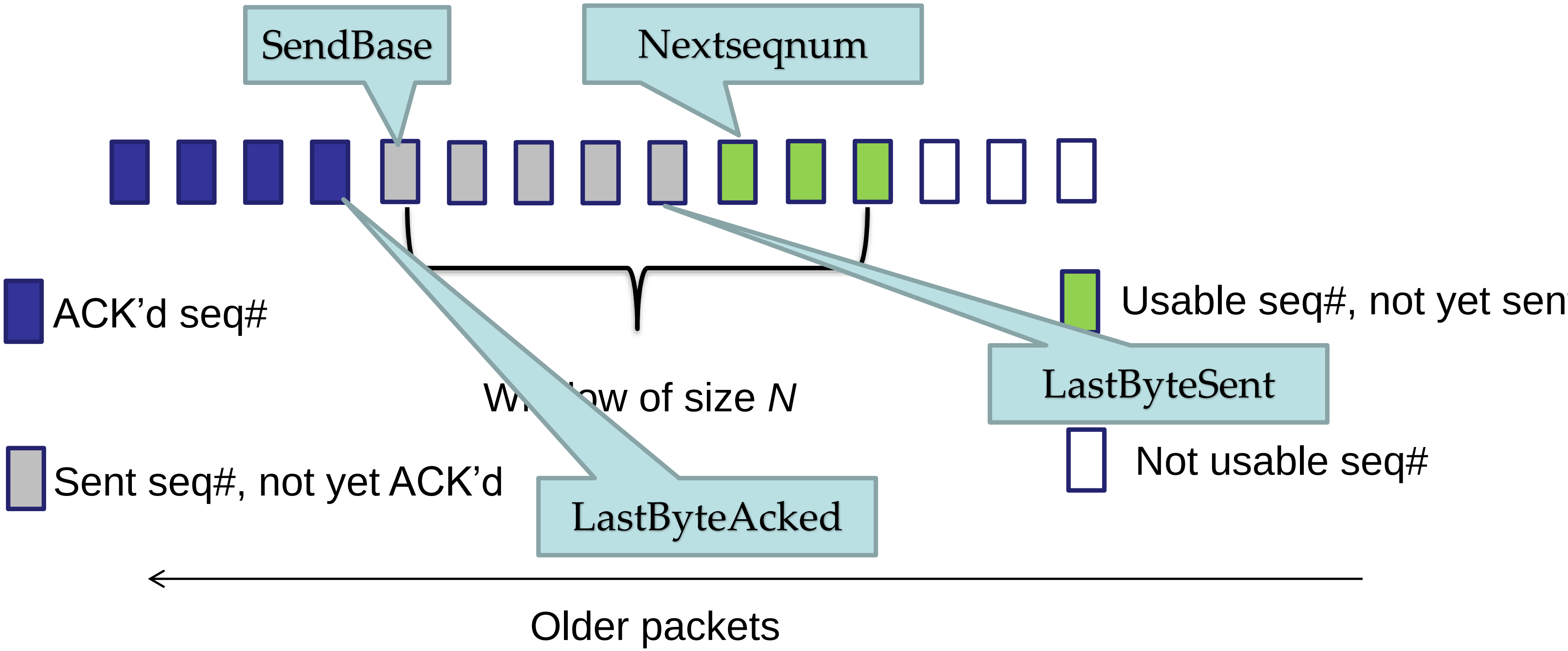
Then set timeout interval:

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

Tricks for reliable data transfer

Mechanism	**Use, Comments**
Checksum	To detect bit errors
Timer	To timeout/retransmit a packet (lost/premature packet)
Sequence number	To detect a lost packet (gap in seq#) To detect a duplicate packet (duplicate seq#)
Acknowledgement	To notify successful reception (individual/cumulative)
Negative ACK	To notify unsuccessful reception
Window, pipelining	To improve sender utilisation (utilisation vs performance)

How to set the window size?



`LastByteSent-LastByteAcked ≤ min{CongWin, RcvWindow};`

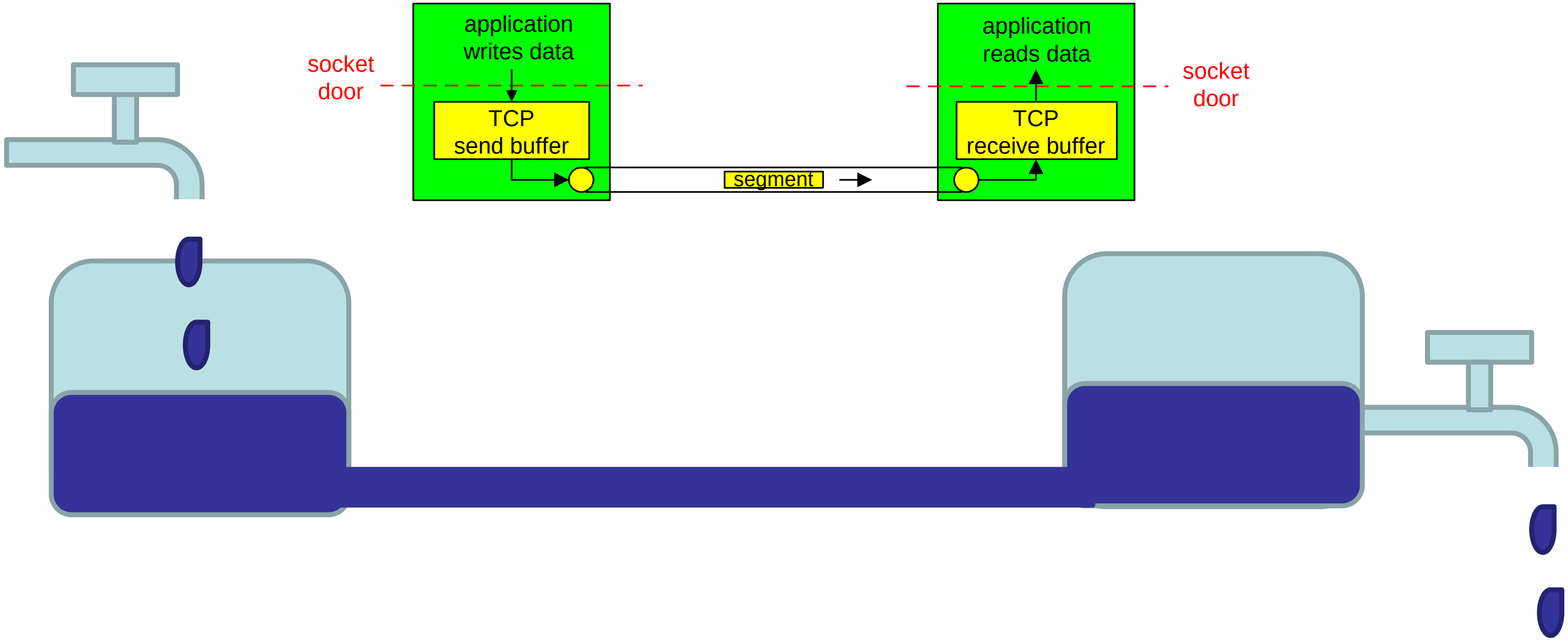
CongWin for congestion control; RcvWindow for flow control;

Outline

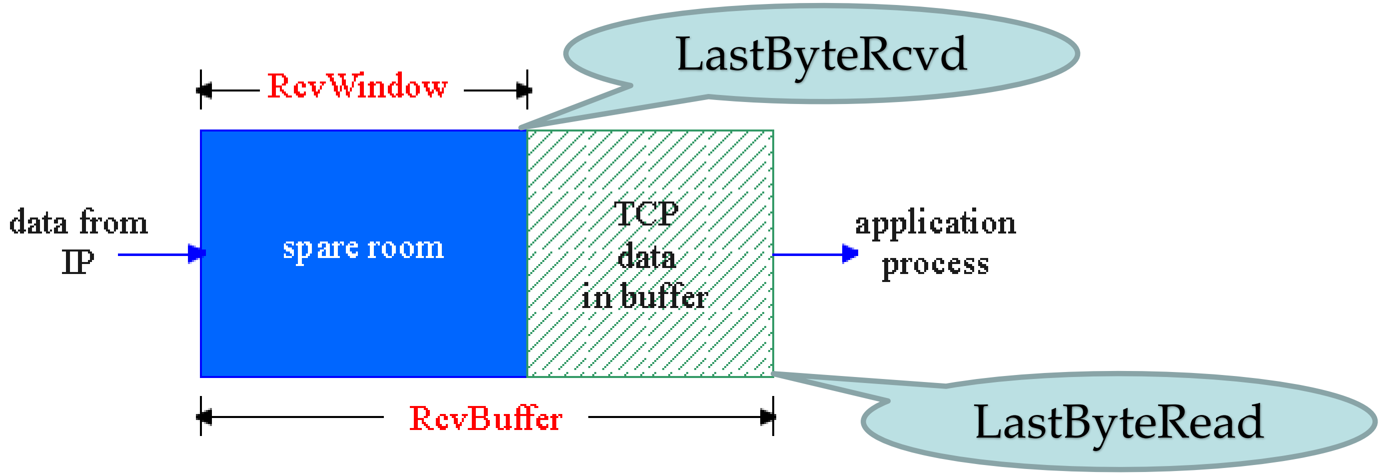
- TCP Flow control
- Principles of general congestion control
- TCP congestion control

Flow control

- Flow control is a speed-matching service – matching the rate at which the sender is sending against the rate at which the receiving is reading.



TCP Flow control: how it works



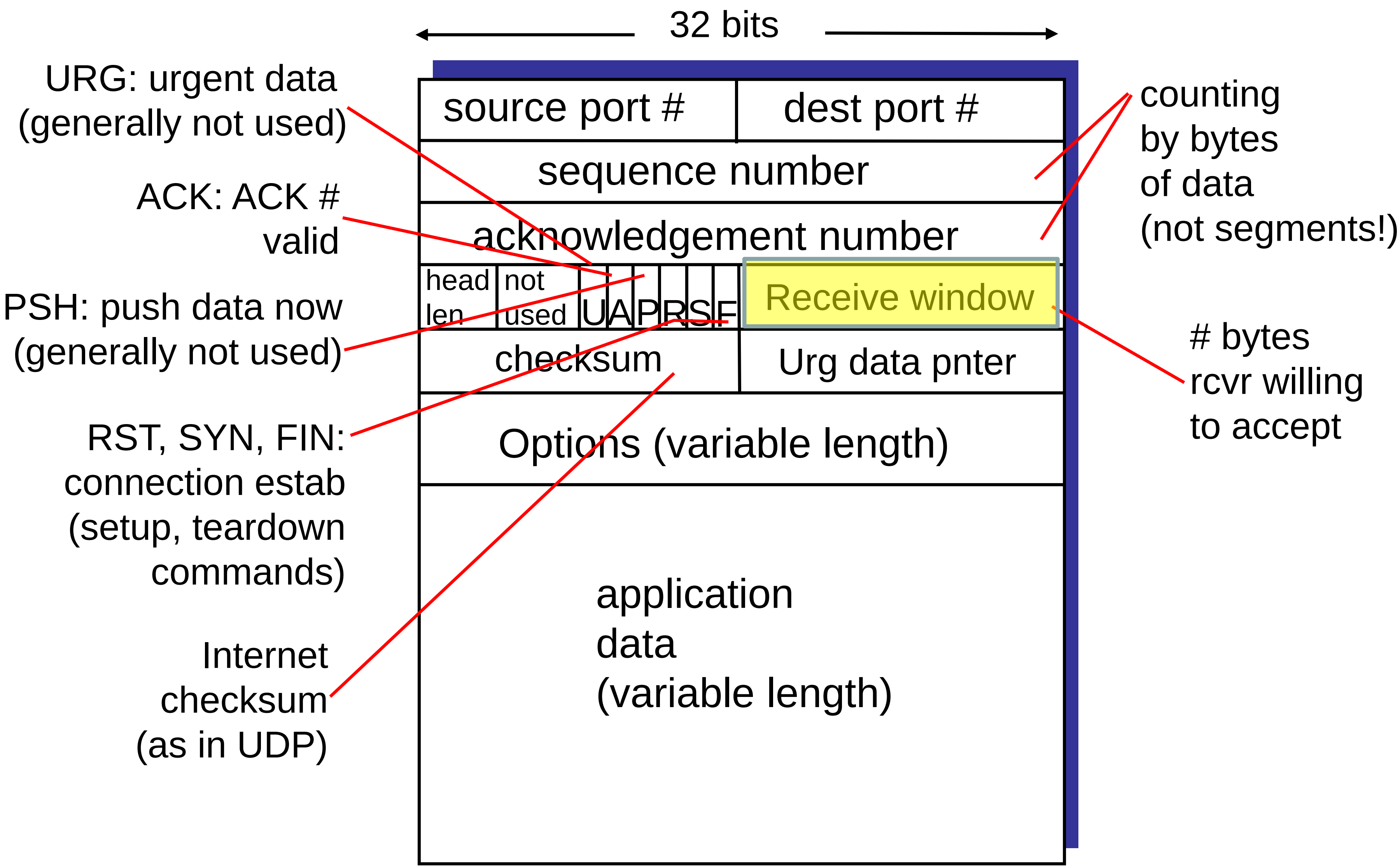
- spare room in buffer

= RcvWindow

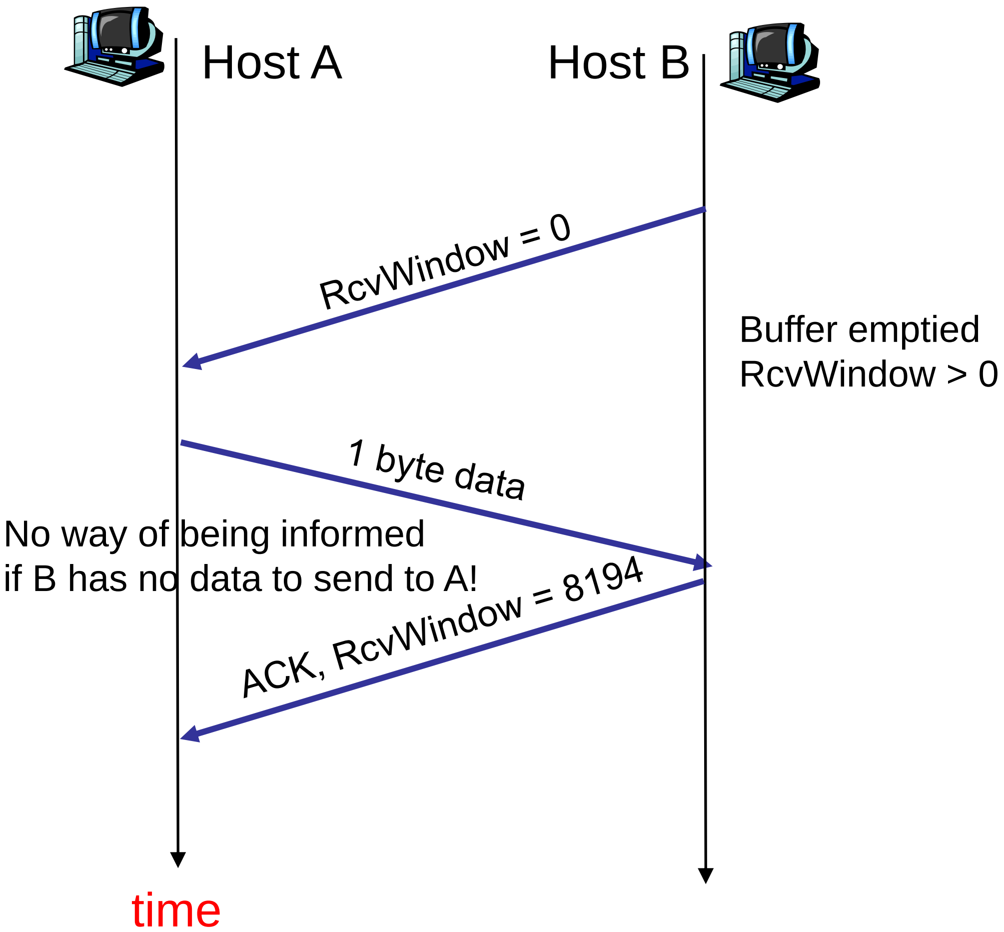
= RcvBuffer-[LastByteRcvd - LastByteRead]

The receiver calculates RcvWindow and tells the sender!

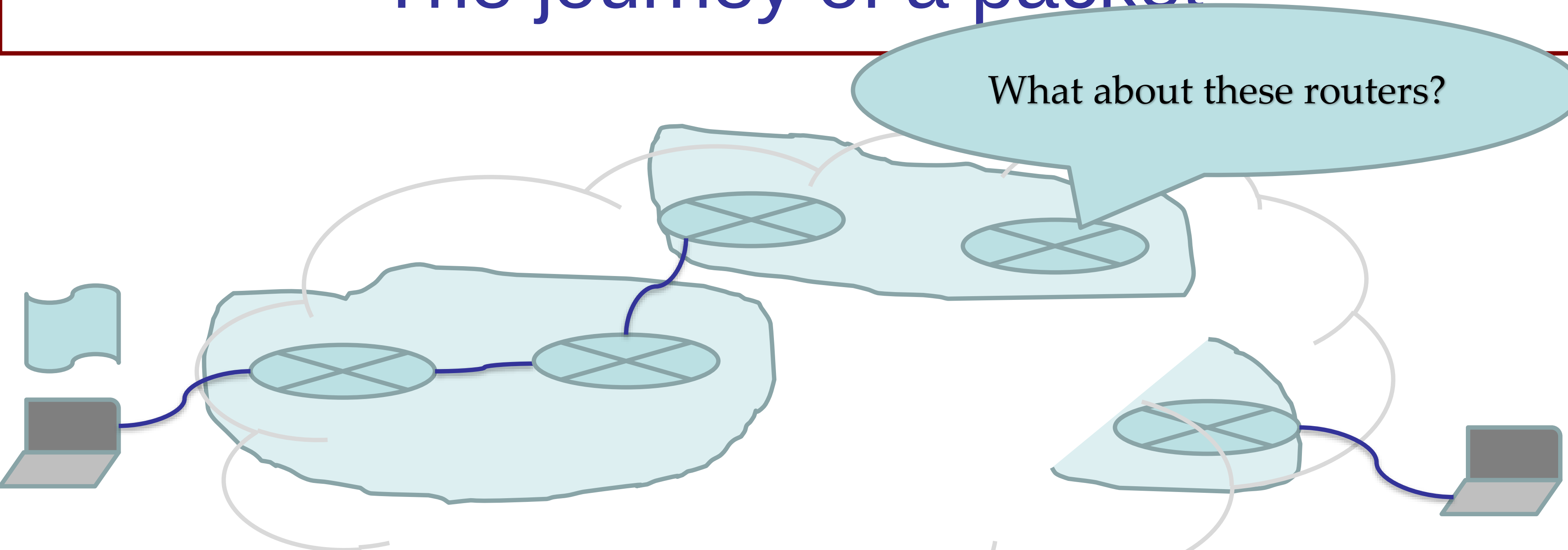
TCP segment structure



Pathological scenario



The journey of a packet



Problems	Causes	Solutions
Bit error	e.g., signal attenuation/noise	Error detection and correction
Buffer overflow	e.g., Speed-mismatch; Too much traffic;	Flow control and congestion control
Lost packet	e.g., buffer overflow at host/router	Acknowledgement and retransmission (ARQ) - RDT
Out of order	e.g. an early packet gets lost and retransmitted; a later one arrives first.	Acknowledgement and retransmission (ARQ) - RDT

Packet Retransmission (a panacea in RDT) *treats the symptom* of packet loss; packet loss typically results from overflowing router buffers as the network becomes congested; (congestion control *treats the cause* of packet loss.)

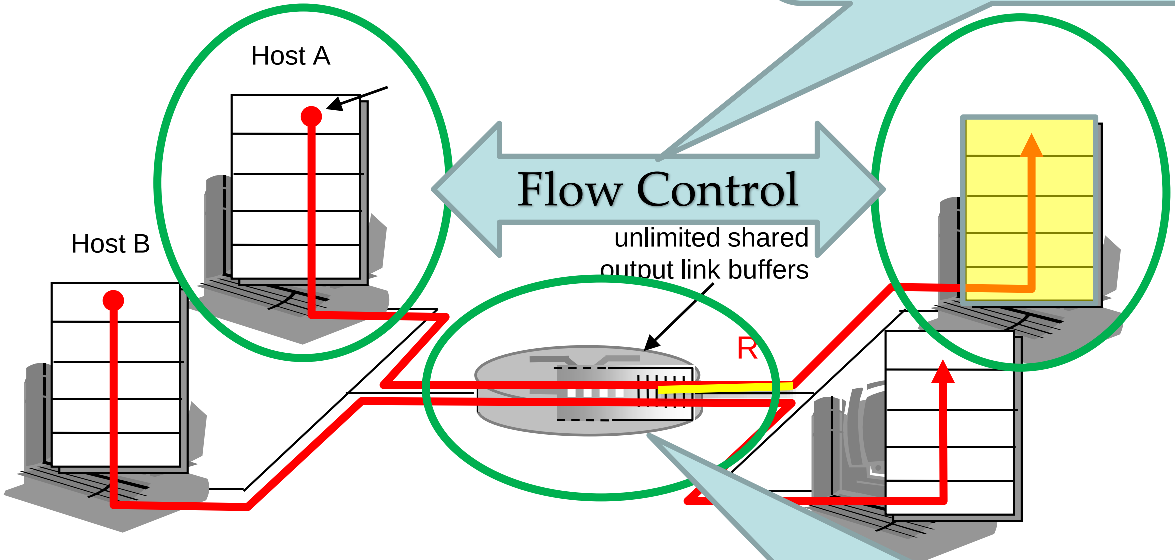
Outline

- TCP Flow control
- Principles of general congestion control
- TCP congestion control

Flow control vs Congestion control

Congestion control is not so much a service provided to the invoking application; it is a service to the network *as a whole*.

Transport-layer flow control – to avoid buffer overflow at the *receiver*;



Congestion control – to avoid router buffer overflow caused by congestion in the network!

Approaches towards congestion control

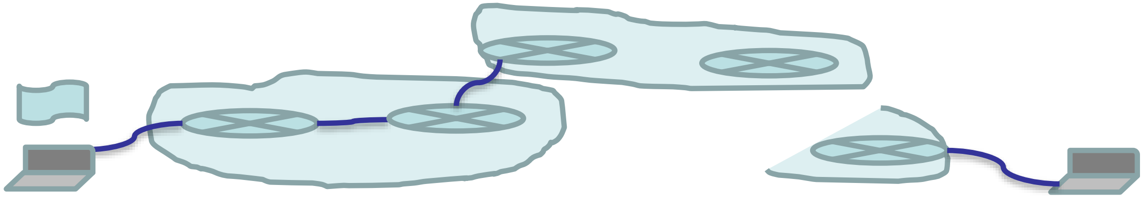
Two broad approaches towards congestion control:

Network-assisted congestion control:

- routers provide feedback to end systems
  - single bit indicating congestion
  - explicit rate sender should send at

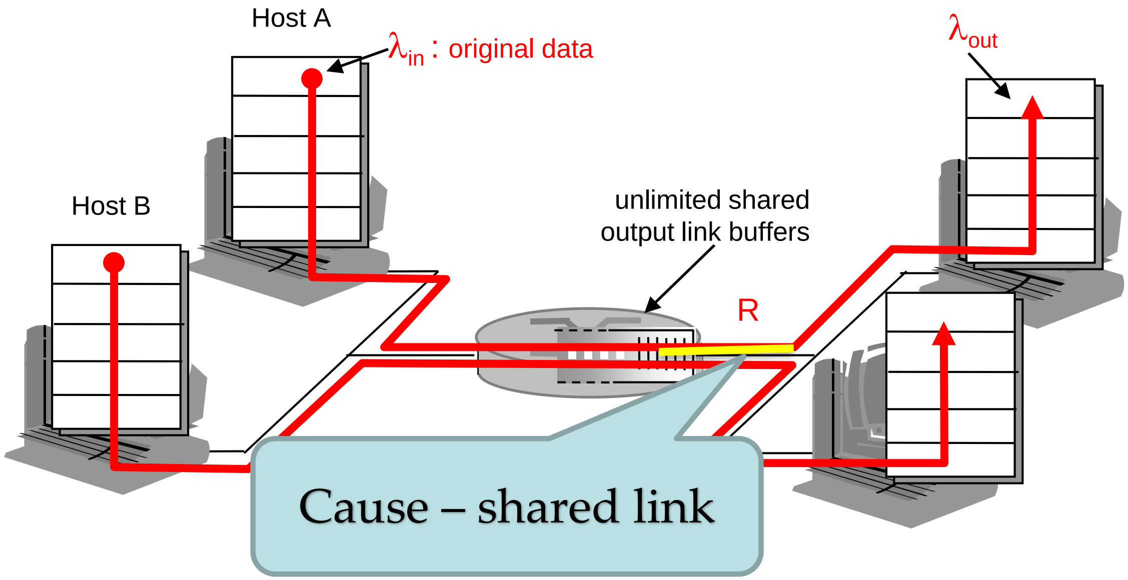
End-end congestion control:

- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- *approach taken by TCP*



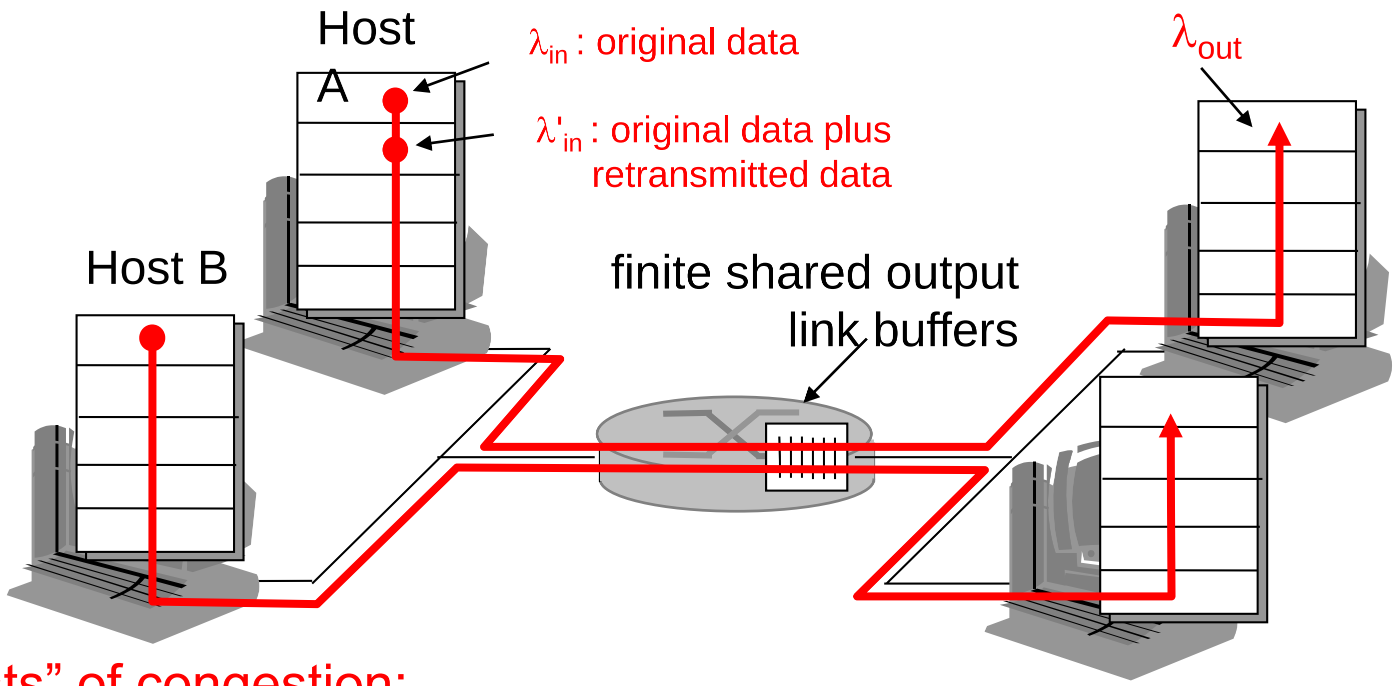
Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- no retransmission
- maximum achievable throughput (R)
- (*costs*) large delays when congested



Causes/costs of congestion: scenario 2

- one router, *finite* buffers
- sender retransmission of lost packet

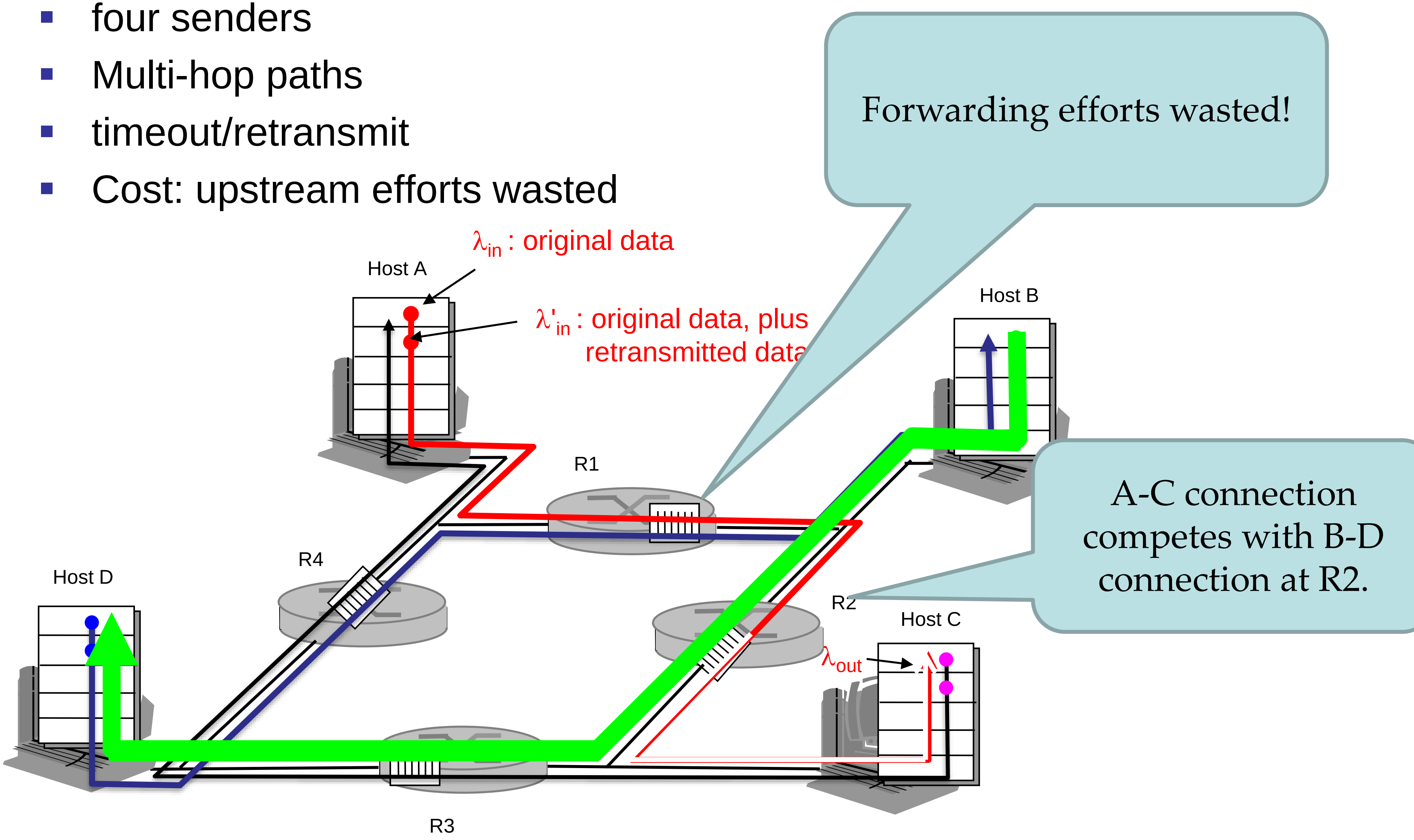


"costs" of congestion:

- more work (retrans) in order for compensation for lost pkts due to buffer overflow.
- unneeded retransmissions: link carries multiple copies of pkt (premature timeout)

Causes/costs of congestion: scenario 3

- four senders
- Multi-hop paths
- timeout/retransmit
- Cost: upstream efforts wasted



Principles of Congestion Control

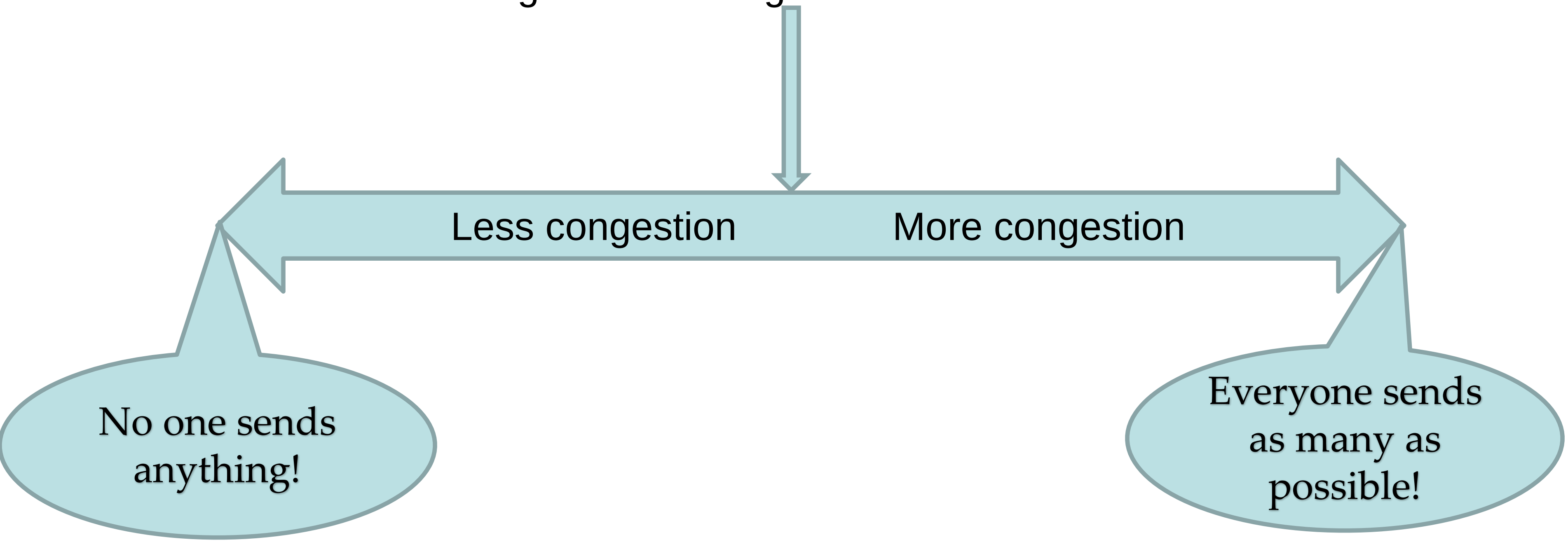
Congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- Manifestations (symptoms):
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)

Outline

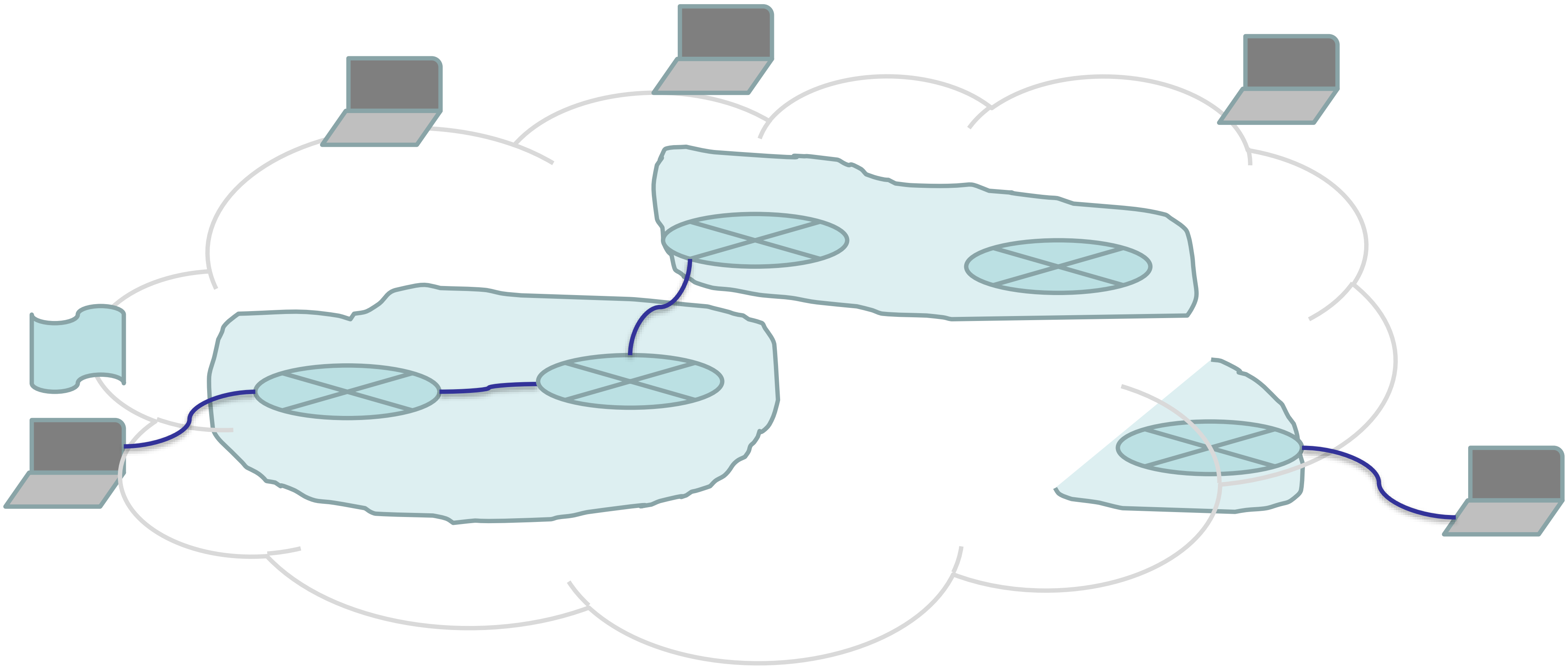
- TCP Flow control
- Principles of general congestion control
- TCP congestion control

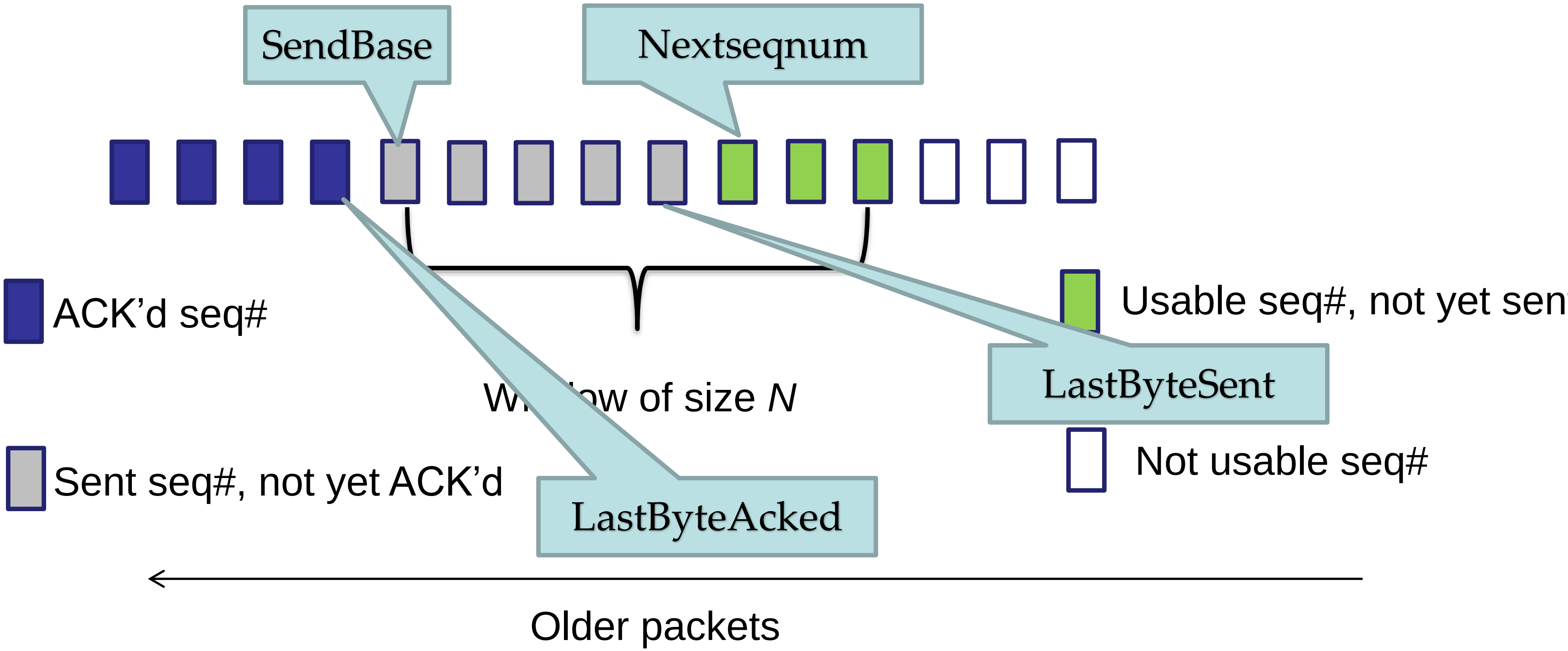
Everyone enjoys a share of the resources
without causing network congestion.



Challenge: *there is no predefined share for everyone!*

The share for everyone is dynamic!





`LastByteSent-LastByteAcked ≤ min{CongWin, RcvWindow};`

CongWin for congestion control; RcvWindow for flow control;

TCP Congestion Control

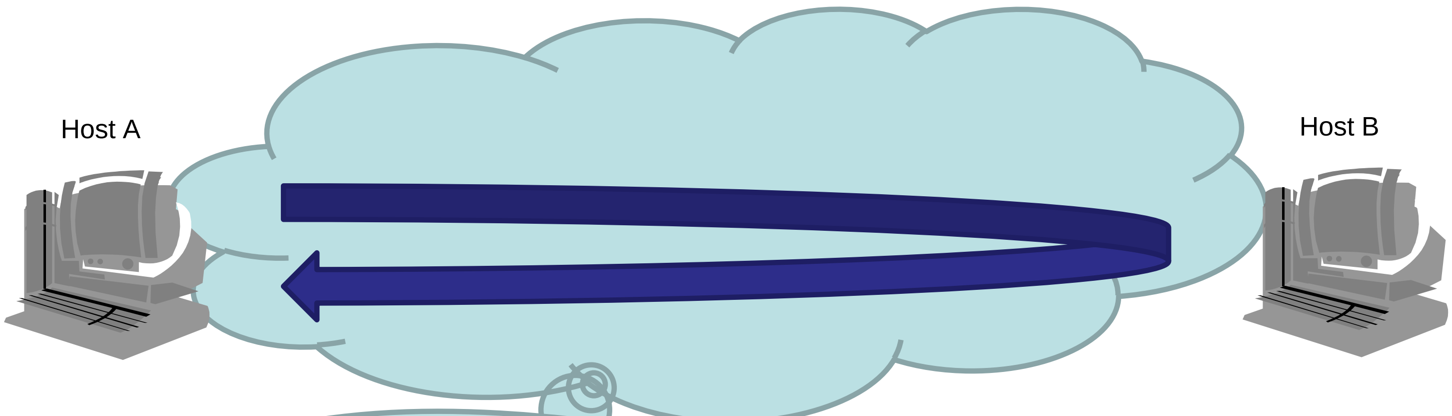
- end-end control (no network assistance)
- sender limits transmission:

  LastByteSent-LastByteAcked

  ≤ min{CongWin, RcvWindow}

For flow control

- Assumptions
  - **RcvWindow > CongWin;**
  - **Negligible transmission delay;**



Within one RTT, the sender sends at most **CongWin** bytes.

Basic strategy

- Roughly,

$$\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

- By adjusting **`CongWin`** *according to network congestion*, the sender can adjust its rate at which it sends data into its connection.

How to detect congestion?

- loss event = timeout *or* 3 duplicate ACKs
- TCP sender reduces rate (**`CongWin`**) after loss event

How to adjust `CongWin`

- Take the arrival of ACKs as an indication of being congestion-free:
  - If ACKs (for previously unacknowledged segments) arrive fast, increase **`CongWin`** quickly;
  - If ACKs (for previously unacknowledged segments) arrive slow, increase **`CongWin`** slowly;
  - If there is timeout/duplicate ACKs, decrease **`CongWin`**.

The celebrated TCP congestion control algorithm

Additive increase, multiplicative-decrease

AIMD

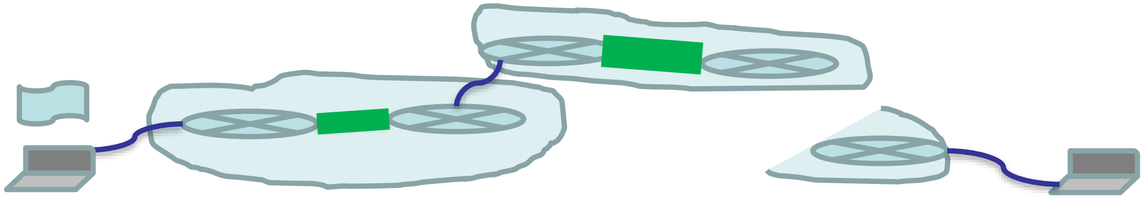
Slow start

Reaction to timeout

TCP Slow Start

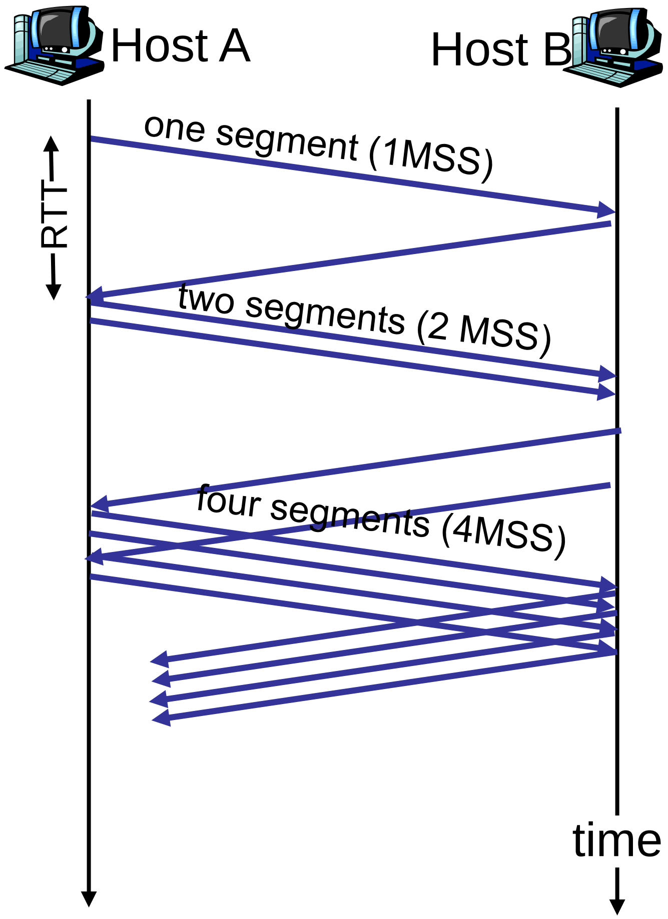
- When connection begins, **`CongWin`** = 1 MSS (Maximum Segment Size)
  - Example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 20 kbps
- available bandwidth may be >> MSS/RTT
  - desirable to quickly ramp up to respectable rate

- When connection begins, increase rate ***exponentially*** fast until first loss event
- "Start from slow"



TCP Slow Start (more)

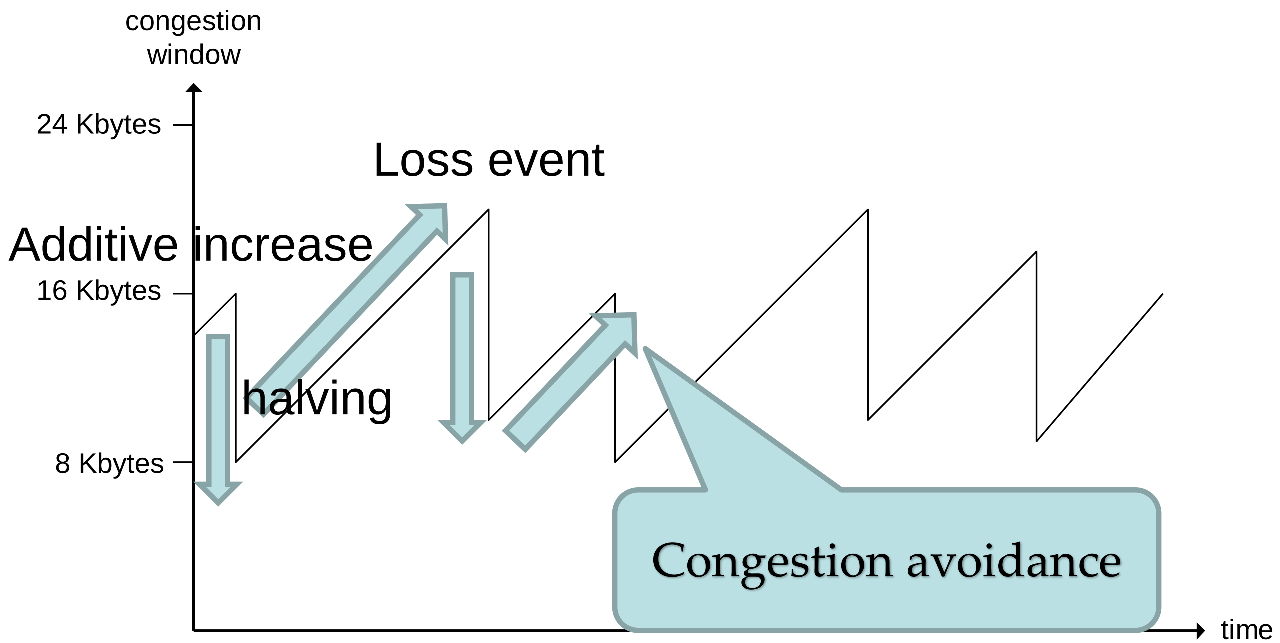
- When connection begins, increase rate exponentially until *first loss event*:
  - double **`CongWin`** every RTT
    - o CongWin is typically set to 1 MSS initially;
    - o done by incrementing **`CongWin`** by **1MSS for every ACK received;**
- <u>Summary:</u> *initial rate is slow* but ramps up exponentially fast



TCP AIMD (saw-toothed patter)

<u>multiplicative decrease:</u> cut **`CongWin`** in half after loss event

<u>additive increase:</u> increase **`CongWin`** by 1 MSS (max seg. Size) **every RTT** in the absence of loss events



Long-lived TCP connection

A common approach for Additive-Increase

- Sender increases its `CongWin` by

  $1\ MSS \times \frac{MSS}{CongWin}$ bytes whenever a new ACK arrives.

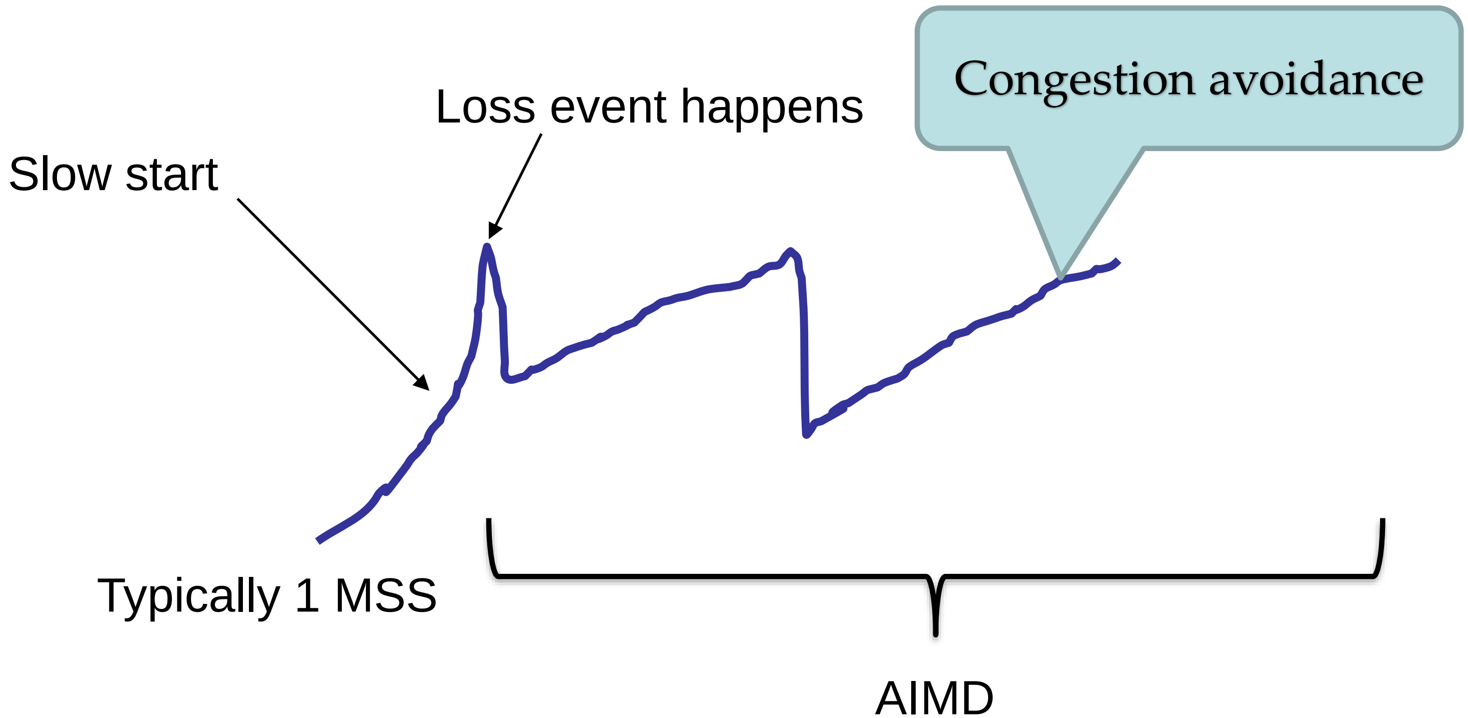
  After ($\frac{CongWin}{MSS}$) segments are sent and ACK'd within one RTT, `CongWin` will be increased by

  $$1\ MSS \times \frac{MSS}{CongWin} \times \frac{CongWin}{MSS} = 1\text{ MSS}$$

An example

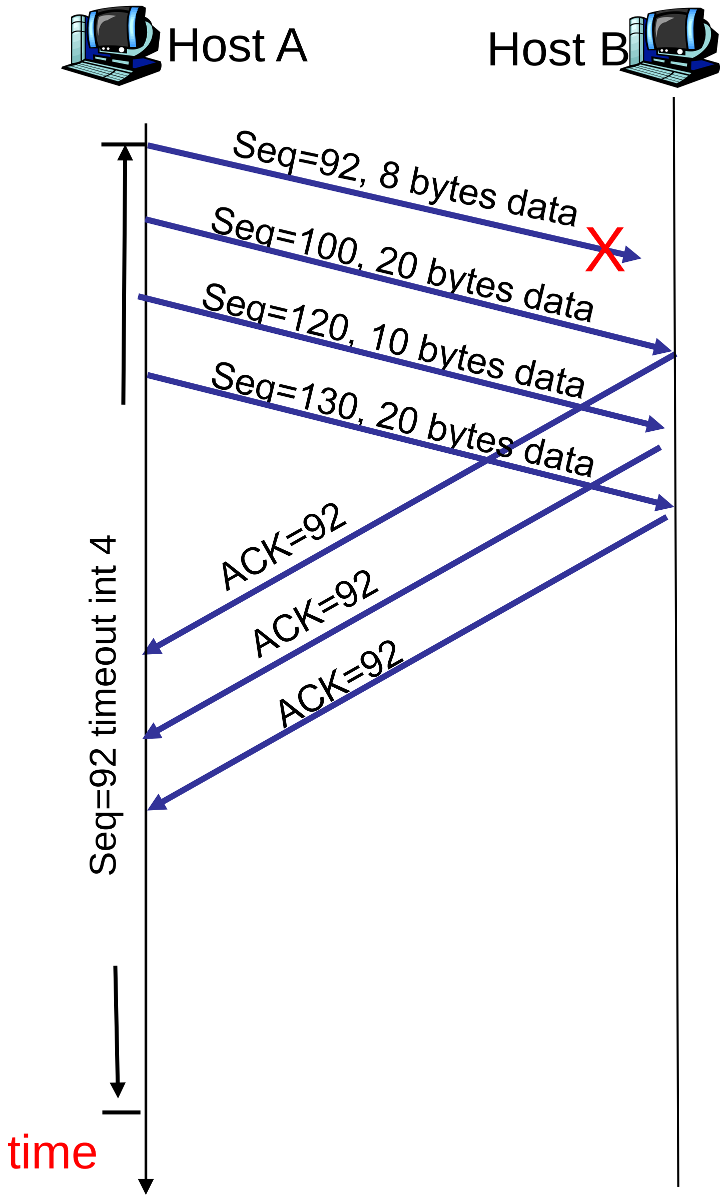
- Init. CongWin = 1 MSS = 500 bytes; (typical values of MSS are 1460, 536, 512)
- After sending one segment (1MSS) and receiving one ACK within one RTT, CongWin = 1 MSS + 1 MSS * (1 MSS/1 MSS) = 2 MSS; *Now it can send two segments;*
- After sending two segments (1MSS each) and receiving two ACKs within one RTT, CongWin = 2 + 1 MSS*(1 MSS/2 MSS) *2 = 3 MSS;
- Continue; CongWin = 3 + 1 MSS*(1 MSS/3 MSS) *3 = 4 MSS;

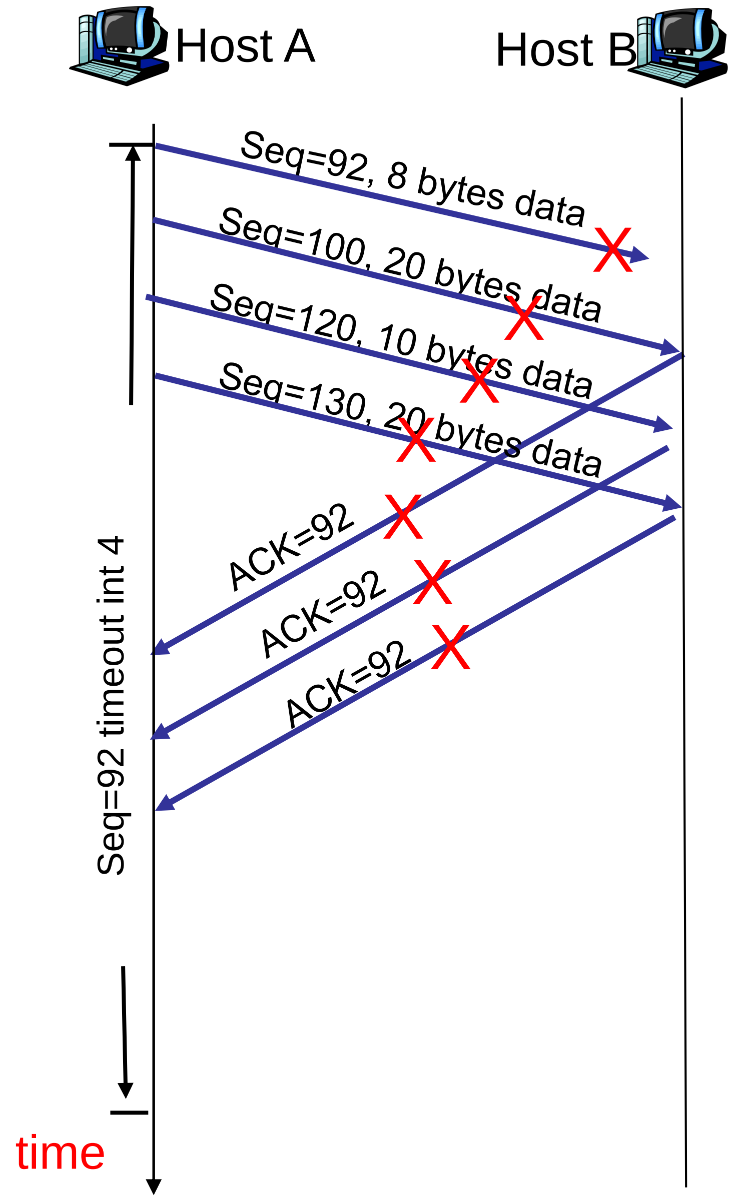
Tricks so far



TCP has some refined actions towards a loss event, depending on whether it is a timeout or 3 dup-ACKs.

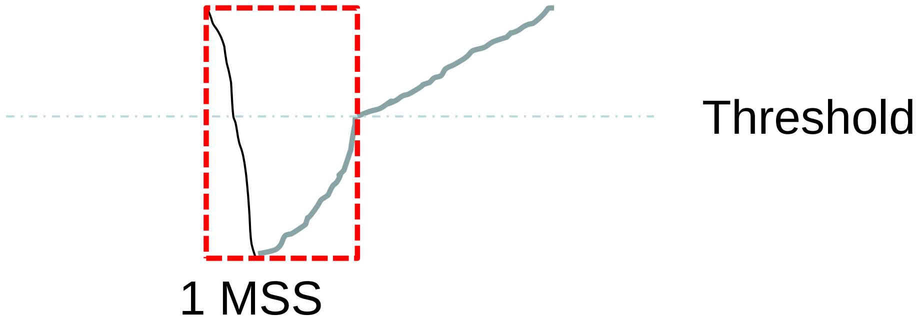
- 3 dup ACKs indicates network capable of delivering some segments
- timeout is “more alarming”;


Host A
Host B
Seq=92, 8 bytes data
Seq=100, 20 bytes data
Seq=120, 10 bytes data
Seq=130, 20 bytes data
ACK=92
ACK=92
ACK=92
Seq=92 timeout int 4
time

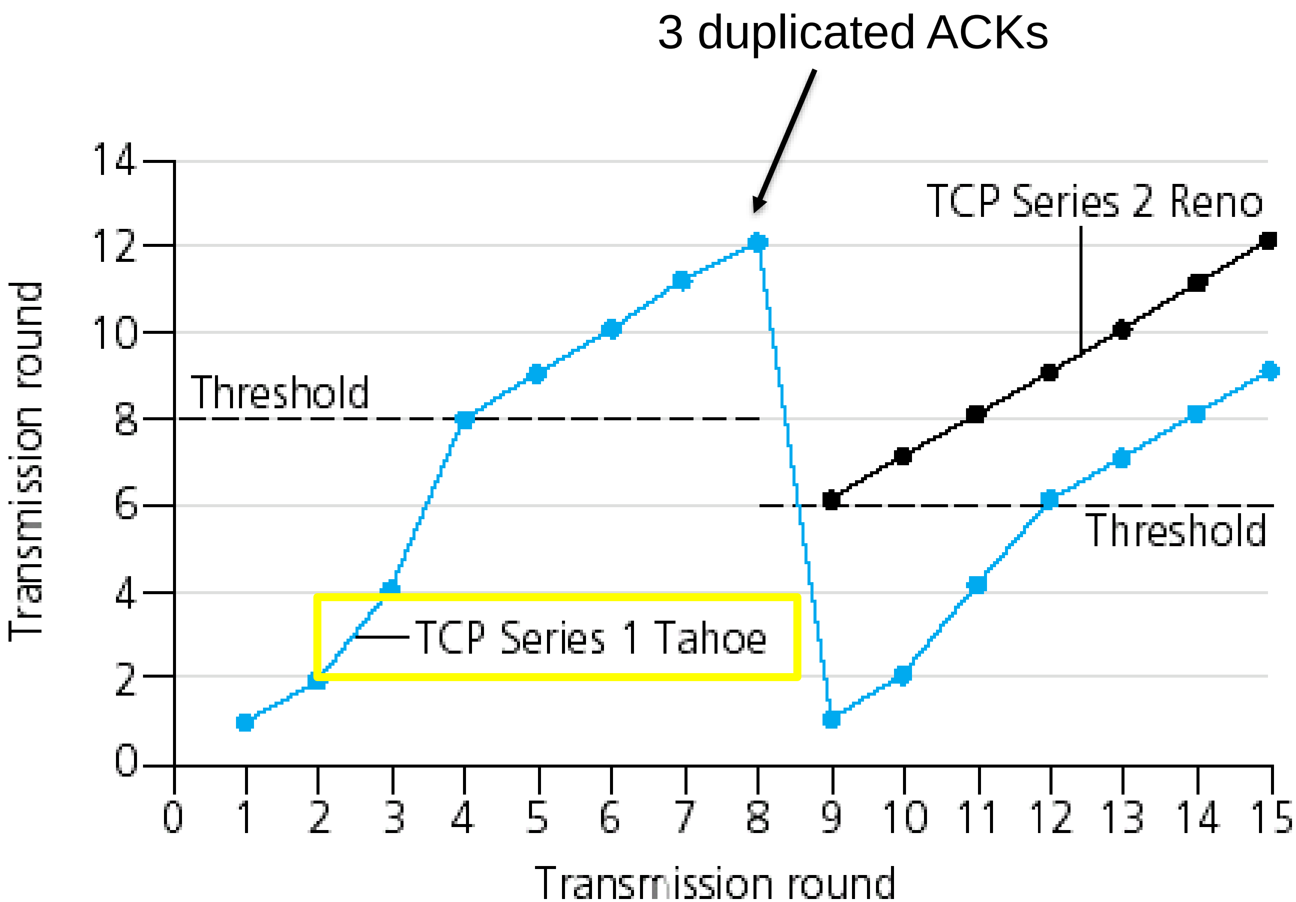

Host A
Host B
Seq=92, 8 bytes data
Seq=100, 20 bytes data
Seq=120, 10 bytes data
Seq=130, 20 bytes data
ACK=92
ACK=92
ACK=92
Seq=92 timeout int 4
time

Refinement – Reaction to timeout

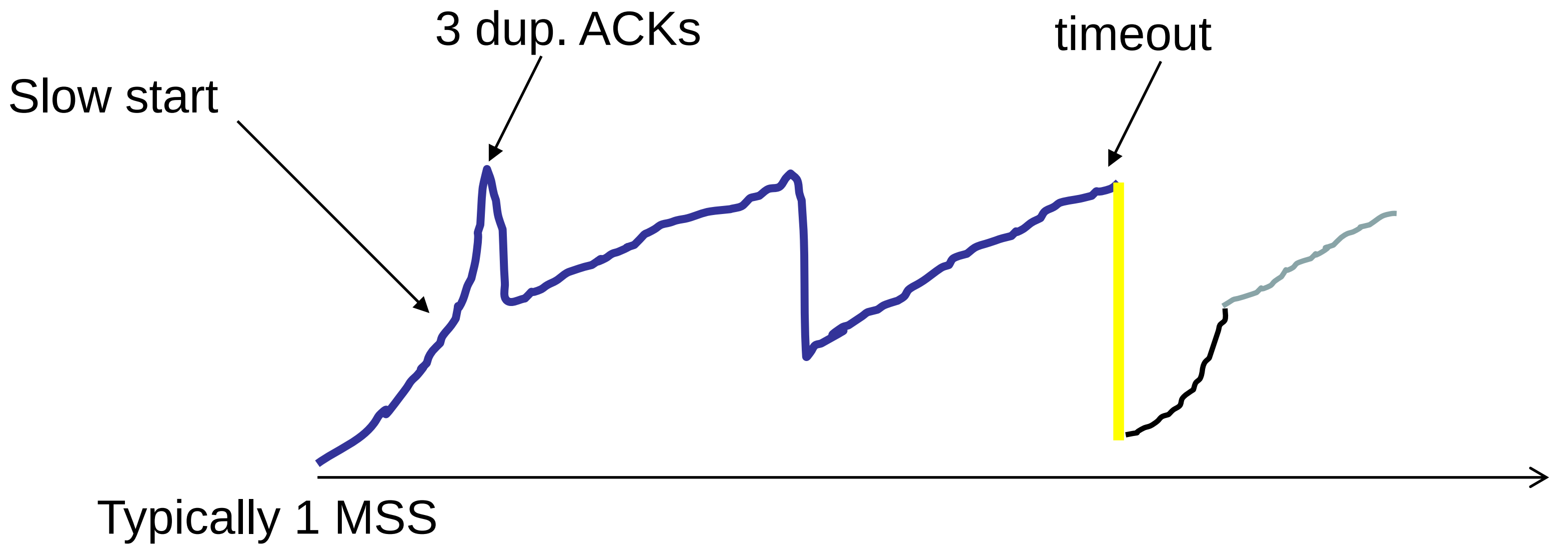
- After 3 dup ACKs:
  - **`CongWin`** is cut in half
  - window then grows linearly
- <u>But</u> after timeout event:
  - **`CongWin`** is set to 1 MSS;
  - window then grows exponentially
  - to a threshold (half of the **`CongWin`** before timeout), then grows linearly



TCP Tahoe vs TCP Reno

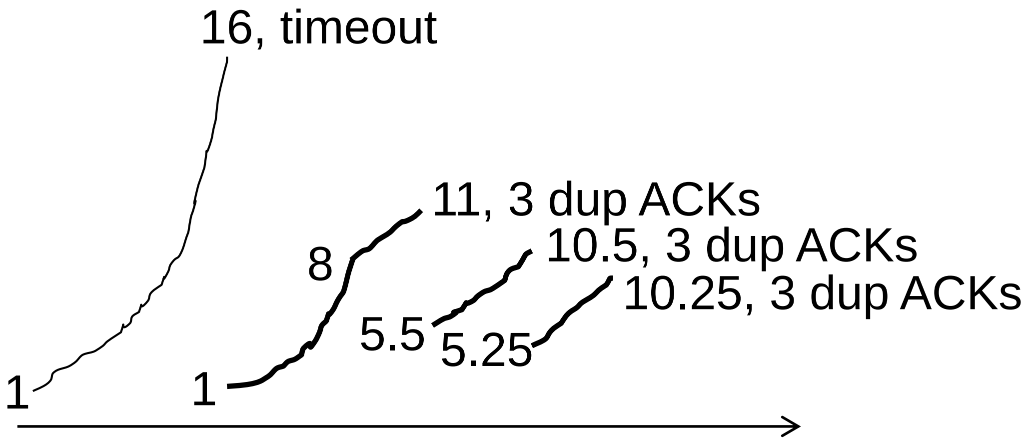


Tricks so far



An example

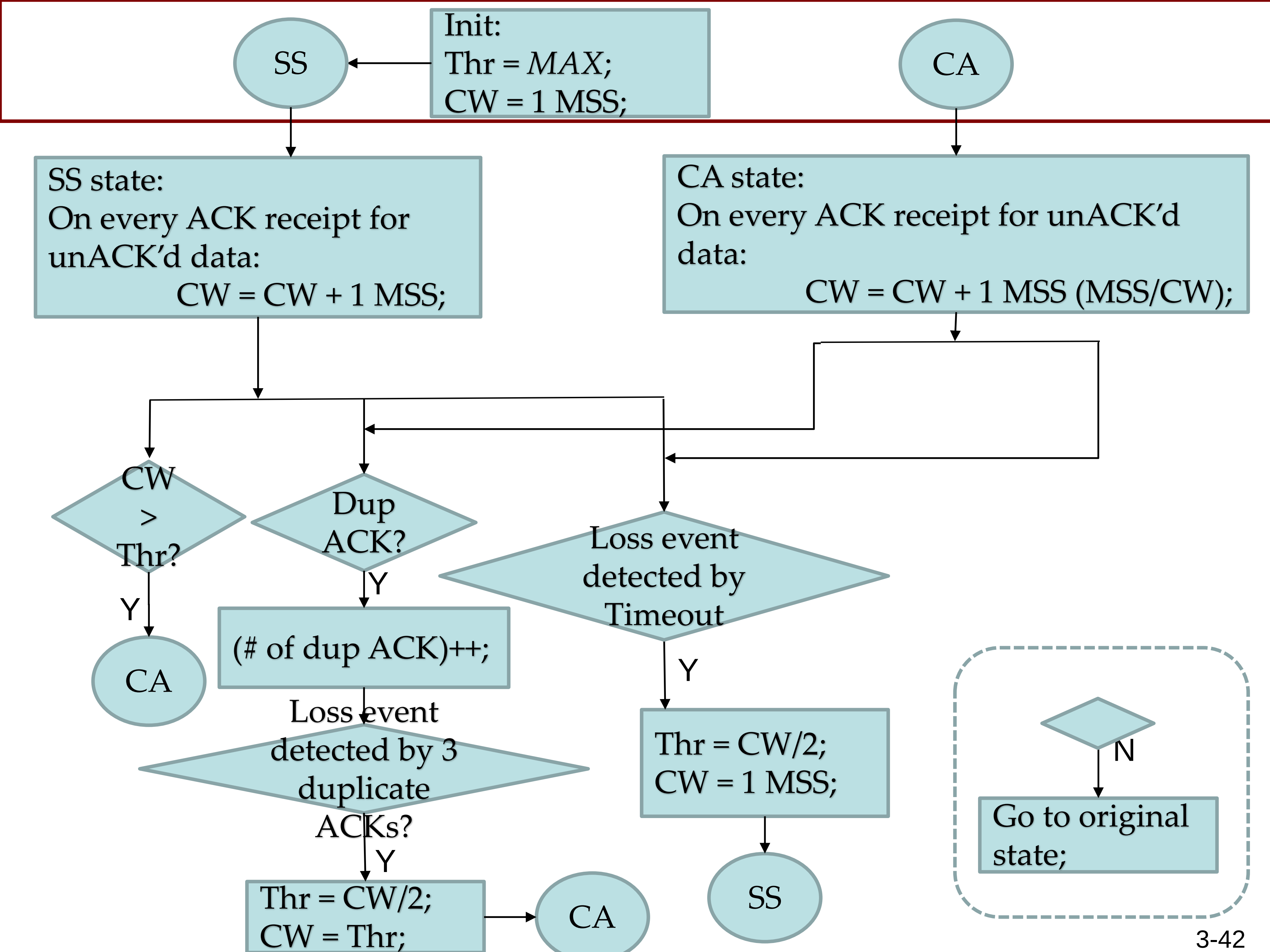
Suppose we know for a host its share is 10 MSS/RTT for a certain time;
Assume that whenever the host sends at a rate >= 16 MSS/RTT, there is a timeout;
Whenever the rate is over 10 MSS/RTT, there are 3 dup ACKs;



10.25, 10.125, 10.0625, …

TCP sender congestion control - Reno

Event	State	TCP Sender Action	Commentary
ACK receipt for previously unacked data	Slow Start (SS)	CongWin = CongWin + MSS (for each effective ACK), If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
ACK receipt for previously unacked data	Congestion Avoidance (CA)	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
Loss event detected by triple duplicate ACK	SS or CA	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
Timeout	SS or CA	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Enter slow start
Duplicate ACK	SS or CA	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed


Init:
Thr = MAX;
CW = 1 MSS;
SS
CA
SS state:
On every ACK receipt for unACK'd data:
CW = CW + 1 MSS;
CA state:
On every ACK receipt for unACK'd data:
CW = CW + 1 MSS (MSS/CW);
CW > Thr?
Y
CA
Dup ACK?
Y
(# of dup ACK)++;
Loss event detected by 3 duplicate ACKs?
Y
Thr = CW/2;
CW = Thr;
CA
Loss event detected by Timeout
Y
Thr = CW/2;
CW = 1 MSS;
SS
N
Go to original state;

Summary

- TCP flow control
- Principles of general congestion control
- TCP congestion control

References

- [KR3] James F. Kurose, Keith W. Ross, *Computer networking: a top-down approach featuring the Internet*, 3rd edition.
- [PD5] Larry L. Peterson, Bruce S. Davie, *Computer networks: a systems approach*, 5th edition
- [TW5] Andrew S. Tanenbaum, David J. Wetherall, *Computer network*, 5th edition
- [LHBi]Y-D. Lin, R-H. Hwang, F. Baker, *Computer network: an open source approach*, International edition

Acknowledgements

- All slides are developed based on slides from the following two sources:
  - Dr DongSeong Kim's slides for COSC264, University of Canterbury;
  - Prof Aleksandar Kuzmanovic's lecture notes for CS340,Northwestern University, https://users.cs.northwestern.edu/~akuzma/classes/CS340-w05/lecture_notes.htm