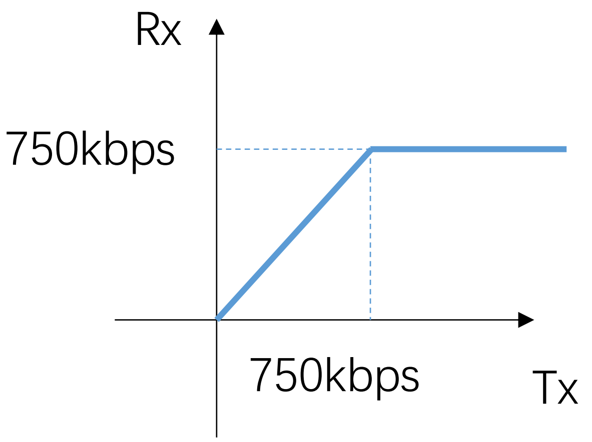
CS120: Computer Networks

Lecture 17. Congestion Control 1

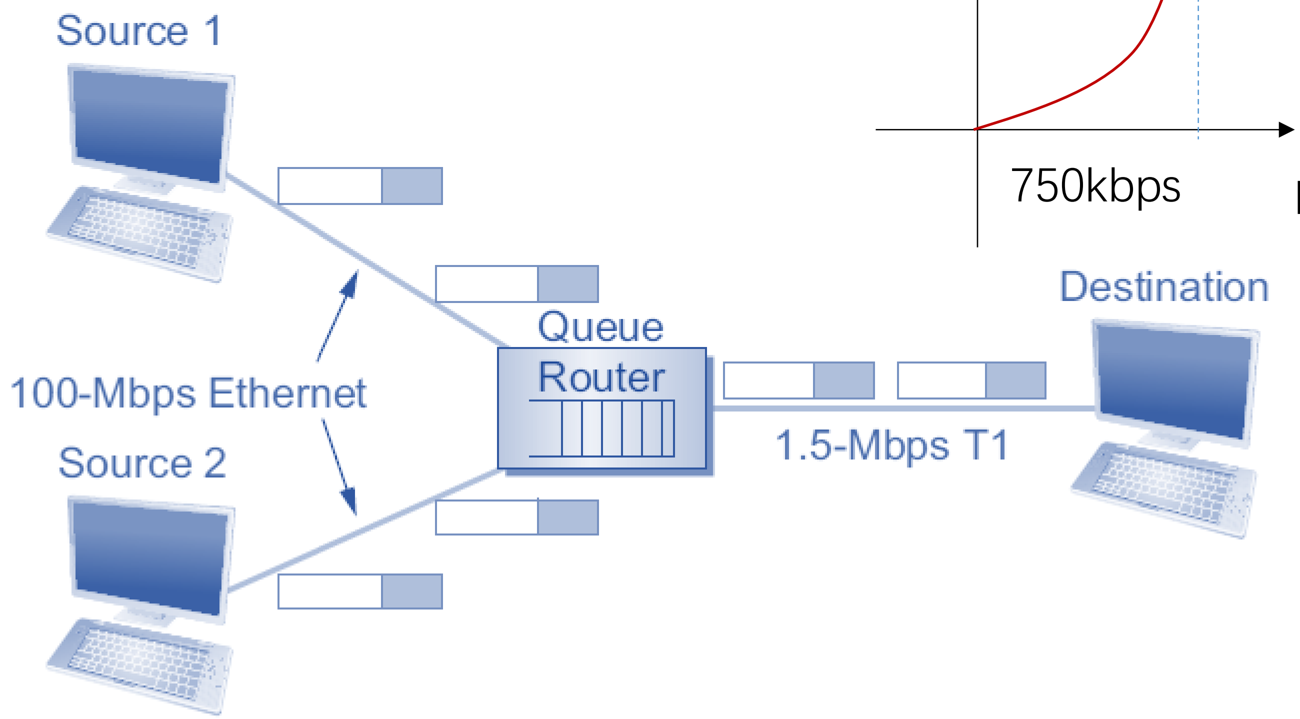
Haoxian Chen

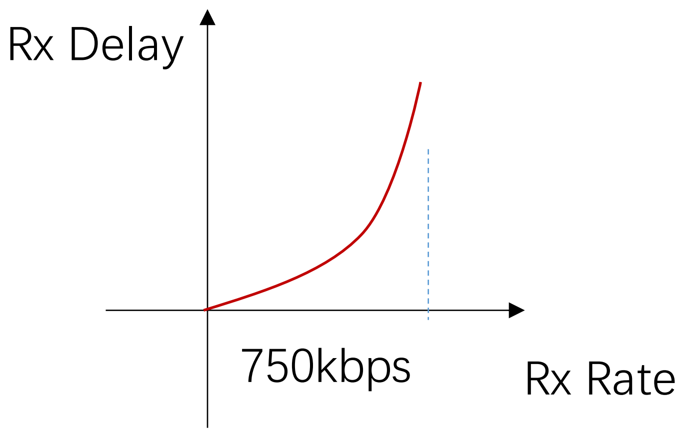
Slides adopted from: Zhice Yang

Congestion in Network



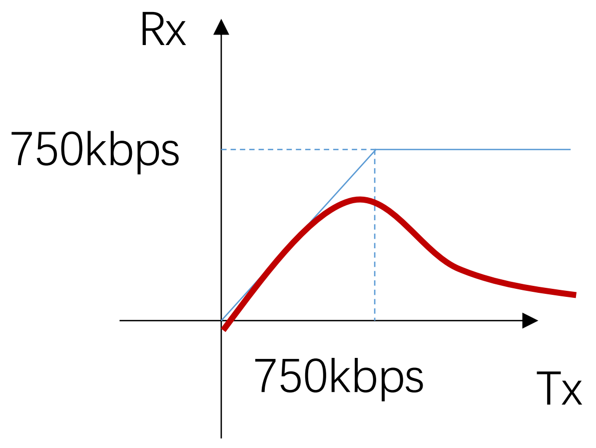
Ideal Case: Infinite Router buffer





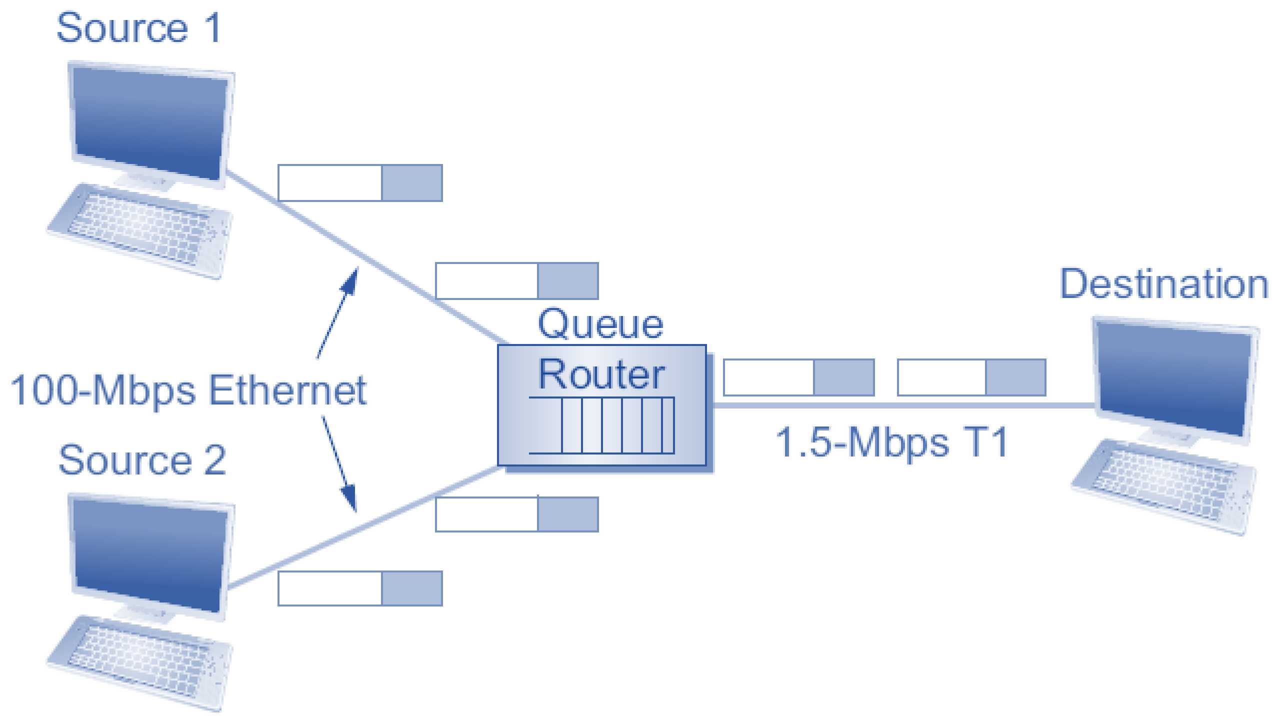
Impact: Network Delay

Congestion in Network



Actual Case: Finite Router buffer

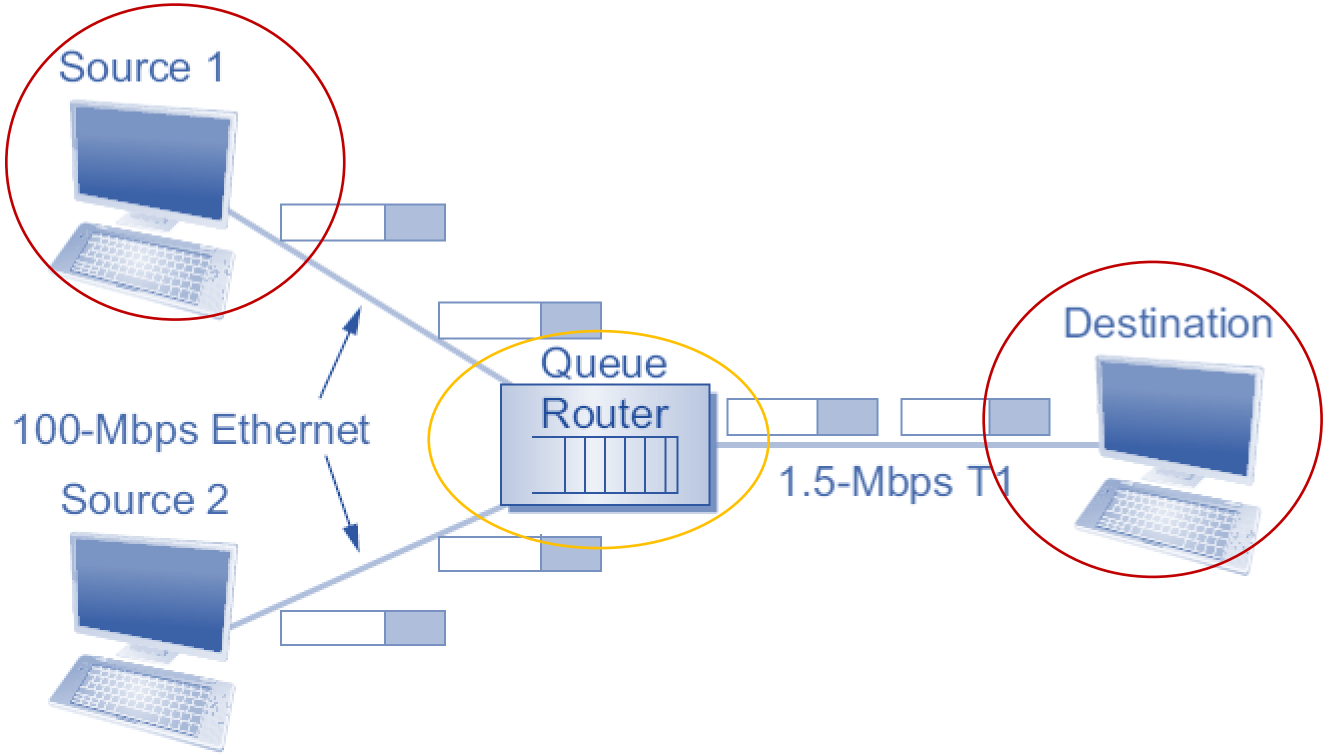
- Packets can be lost (dropped at router) due to full buffers
- Sender does not know when packet has been dropped, retransmissions might be unnecessary



Impact: Retransmissions waste network capacity

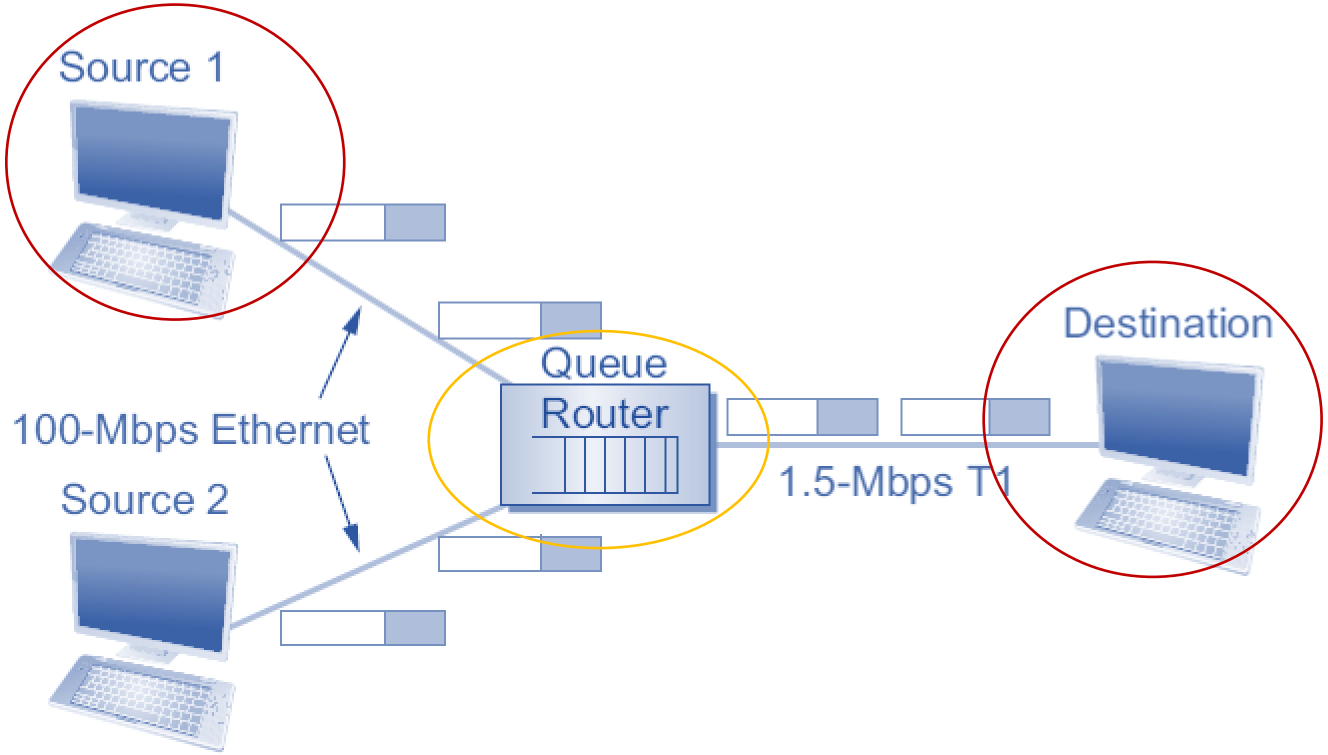
Two Places to Handle Network Congestion

- End hosts
- Routers



Two Places to Handle Network Congestion

- ➢End hosts
- • Routers



Packet Loss v.s. Network Delay

- Packet Loss
  - Packet loss is the indication of congestion
  - When packet loss happens
    - Many arriving packets encounter a full queue
    - Many individual flows lose multiple packets
    - Many flows divide sending rate in half
- Network Delay
  - Increase in network delay is an indicator for possible congestion
  - When network delay increases
    - Some packets are queued in routers
  - Slow down one or two flows before congestion happens

Congestion Control

- Host-based Congestion Control
  - Packet Loss
    - AIMD
    - Slow Start
    - Fast Retransmission
    - Fast Recovery
  - Delay
- Router-based Congestion Control
  - Queuing

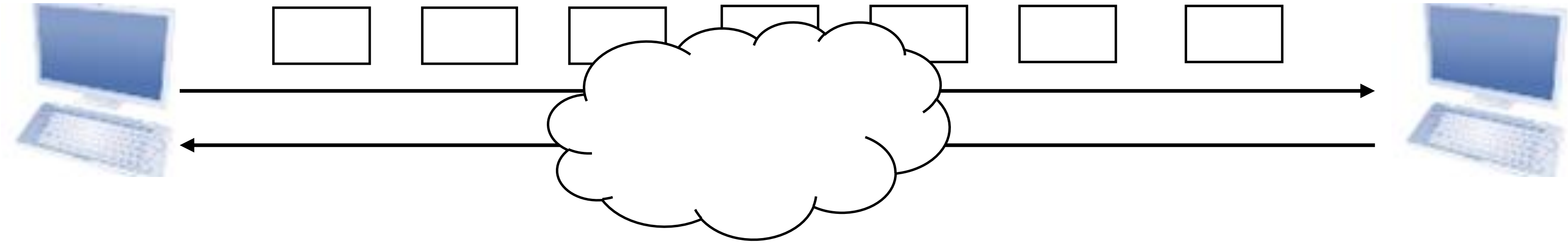
TCP Congestion Control

- Introduced by Van Jacobson through his Ph.D. dissertation work in late 1980s
  - 8 years after TCP became operational
- Basic ideas
  - Each host determines network capacity for itself
    - Leverage feedback
    - Assumption: FIFO or FQ queue in routers
- Challenges
  - Determining the available capacity
  - Adjusting to changes in capacity

Simple Case – Steady Capacity

- In the steady state
  - How to measure the network capacity ?
  - How to pace the sender ?

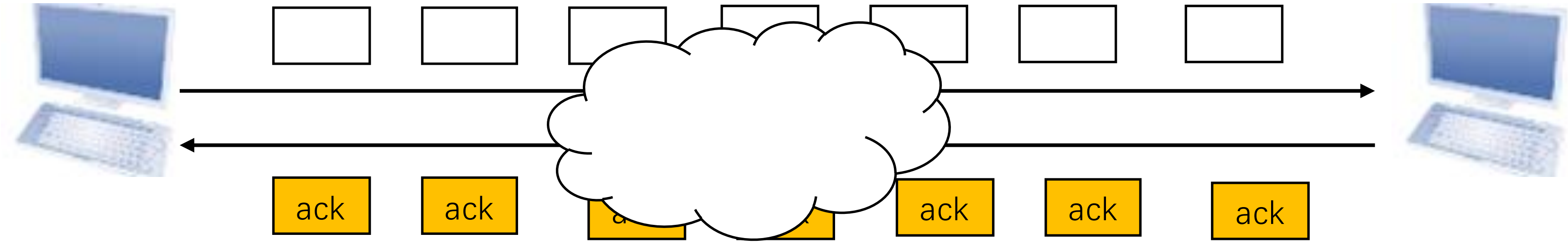
capacity: 7 packets per second (due to congestion)
(physical rate: 100 packets per second)



Simple Case – Steady Capacity

- In the steady state
  - How to measure the network capacity ?
  - How to pace the sender ?

capacity: 7 packets per second (due to congestion)
(physical rate: 100 packets per second)



TCP uses ACKs to estimate the bandwidth and pace the sending, i.e., self-clocking

TCP Congestion Control

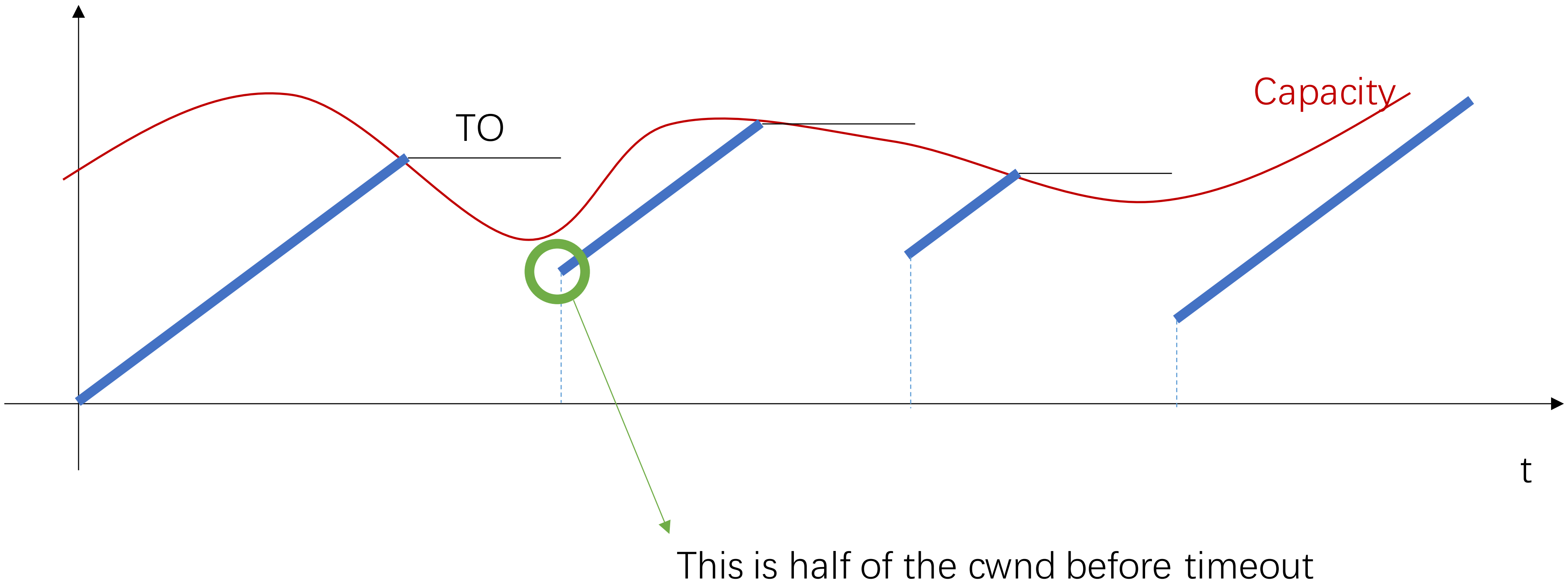
- Objective: Estimate and adapt to (varying) network capacity
- Approach: Adjust Sliding Window according to ACKs
  - MaxWindow = MIN(CongestionWindow, AdvertisedWindow)
  - Decrease CongestionWindow upon detecting congestion
  - Increase CongestionWindow upon lack of congestion
  - CongestionWindow abbr. cwnd (in unit of MSS)
- Basic Components
  - Additive Increase/Multiplicative Decrease (AIMD)
  - Slow Start
  - Fast Retransmission
  - Fast Recovery
- Other Variations

Additive Increase/Multiplicative Decrease (AIMD)

- Intuition: over-sized window is much worse than an under-sized window
  - Over-sized window: packets dropped and retransmitted
  - Under-sized window: somewhat lower throughput
- Additive Increase
  - If successfully received acks of the last window of data
    - cwnd = cwnd+1
- Multiplicative Decrease
  - If packet loss
    - cwnd = cwnd/2

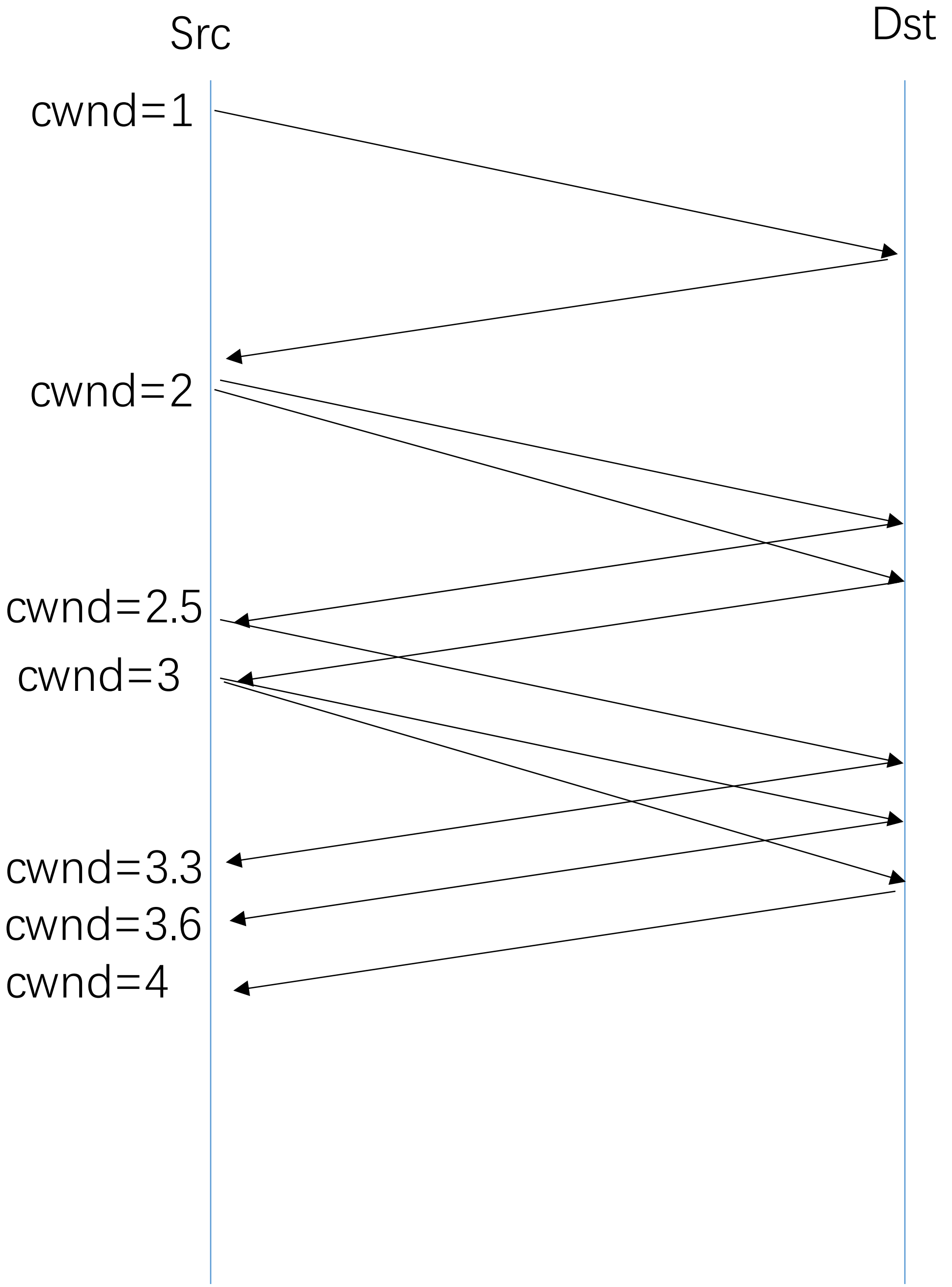
AIMD

- TCP sawtooth pattern



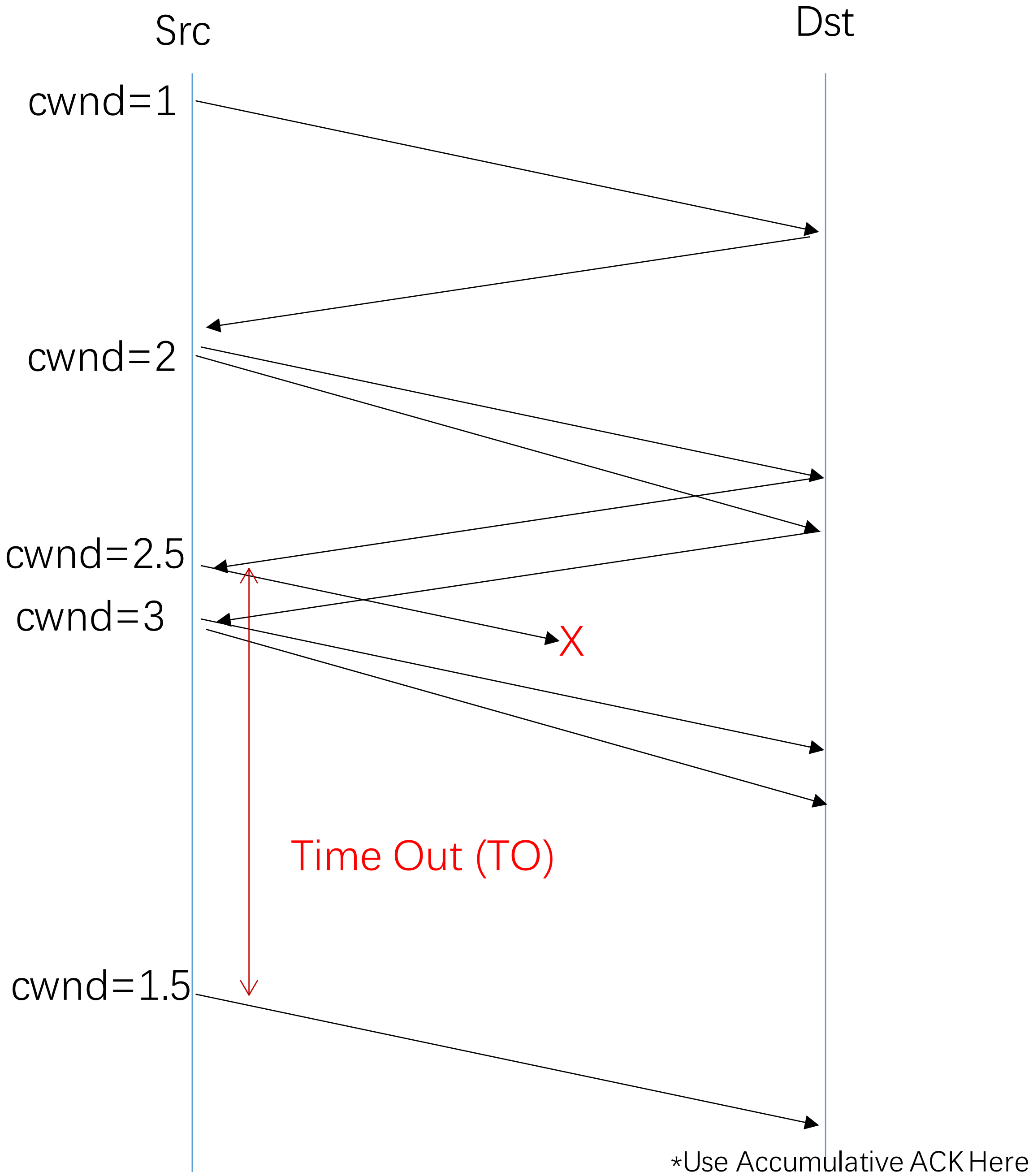
AIMD

- Additive Increase
  - Increment = 1/cwnd
  - cwnd += Increment



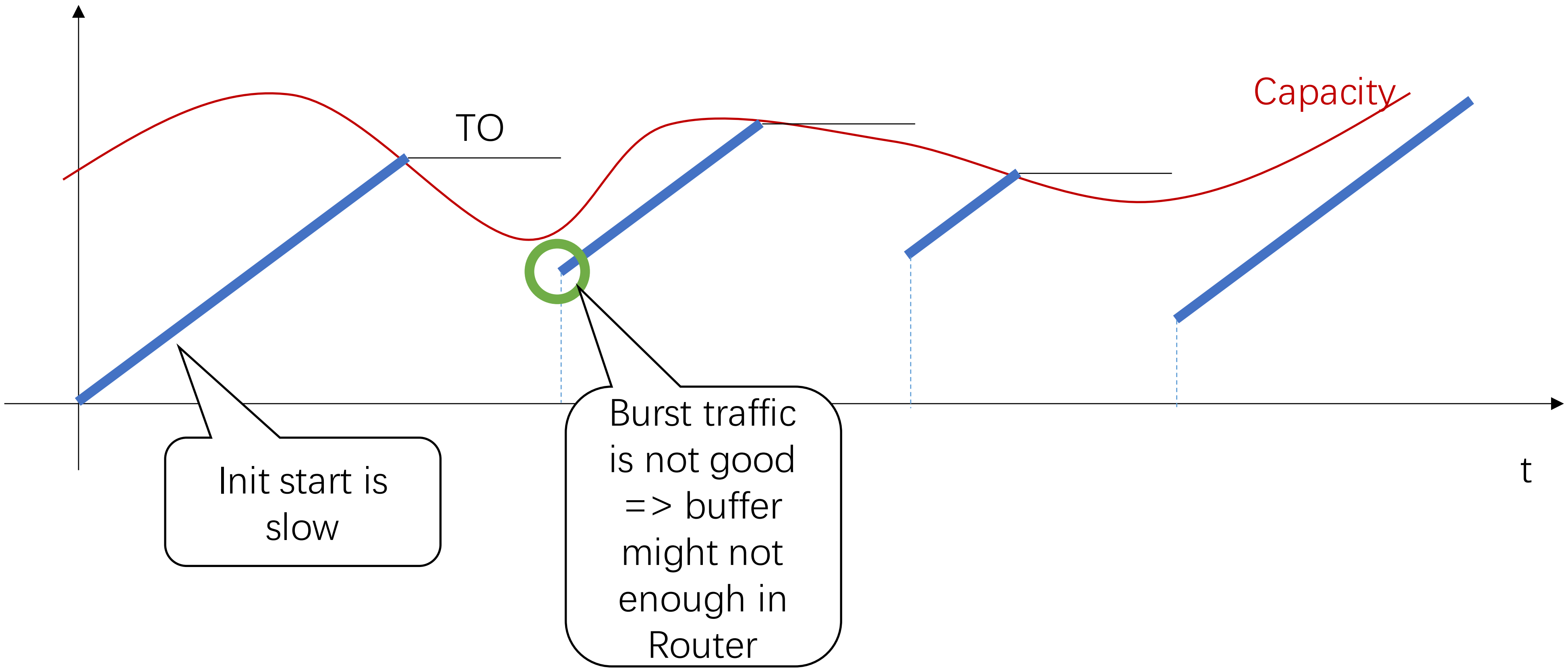
AIMD

- Multiplicative Decrease
  - cwnd = cwnd /2



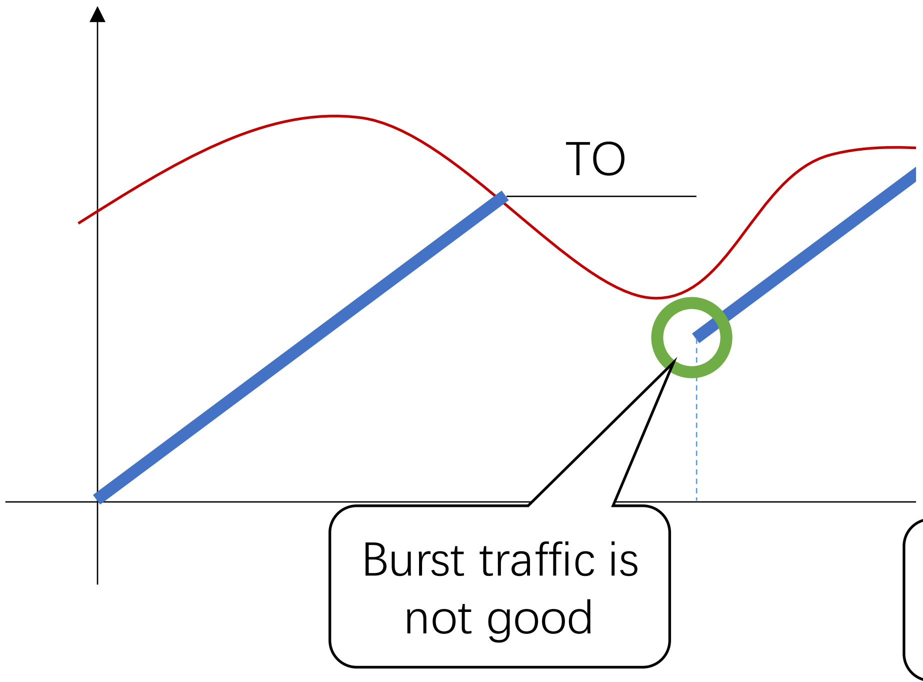
AIMD

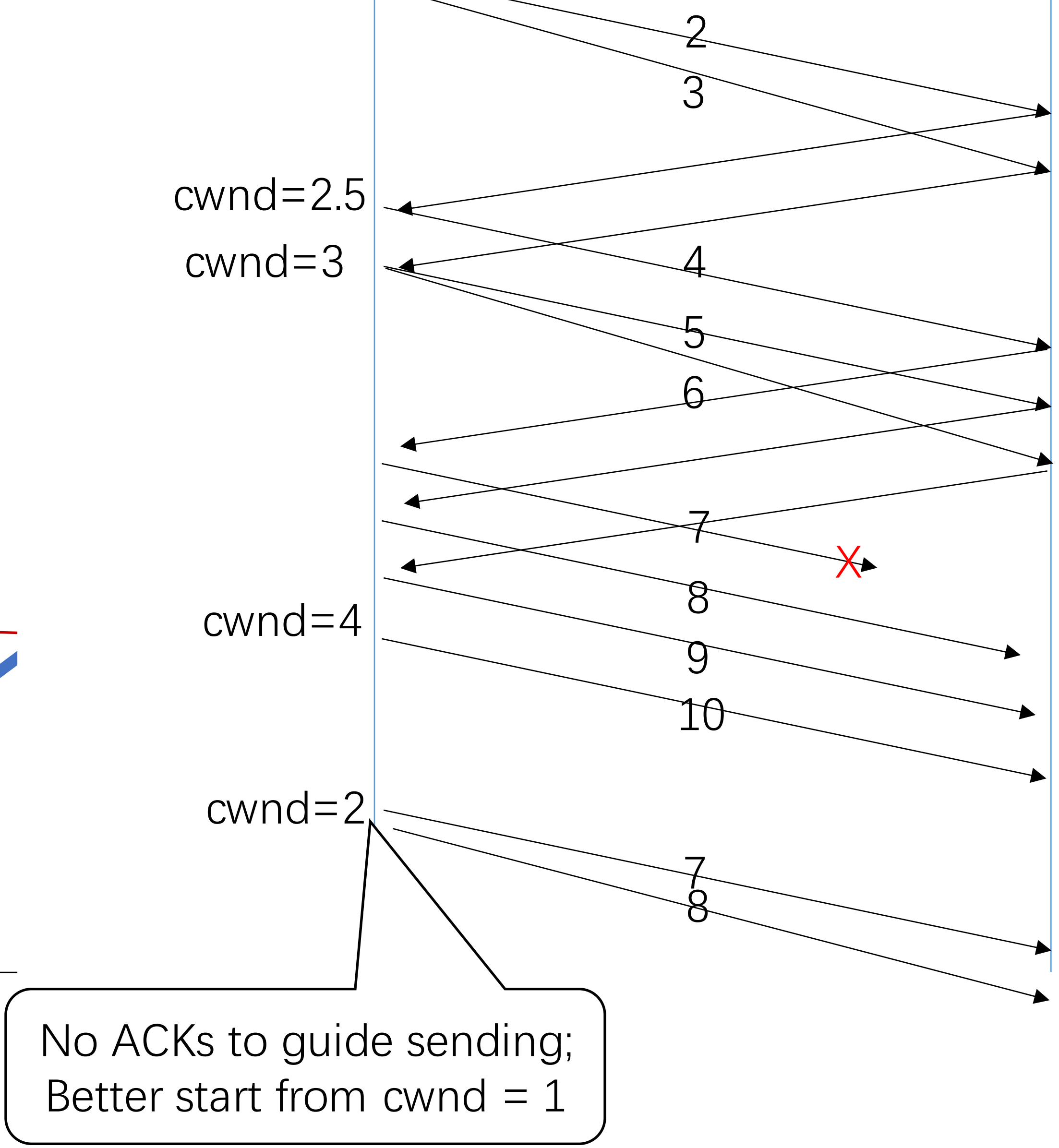
- TCP sawtooth pattern



AIMD

- TCP sawtooth pattern



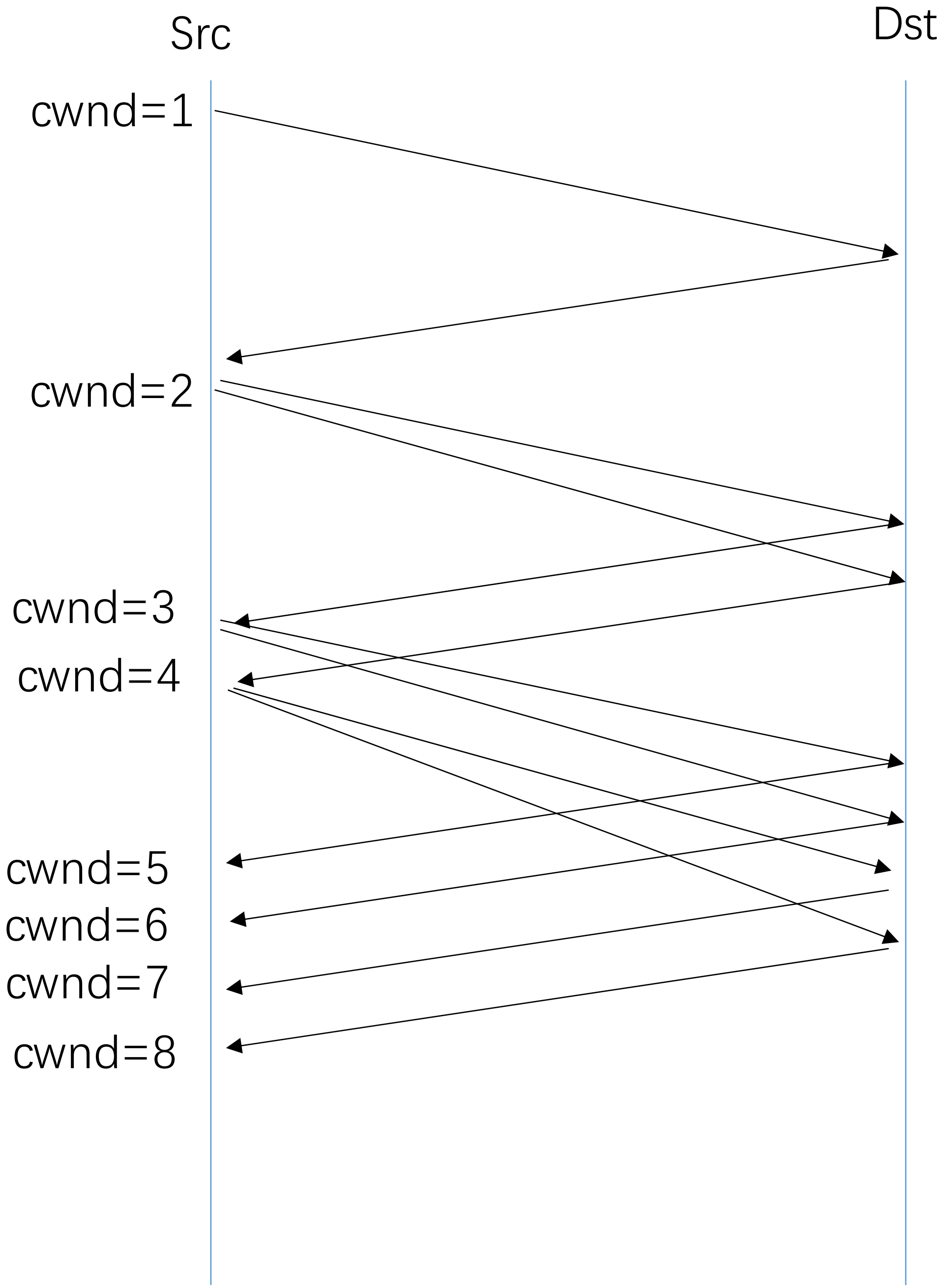


Slow Start

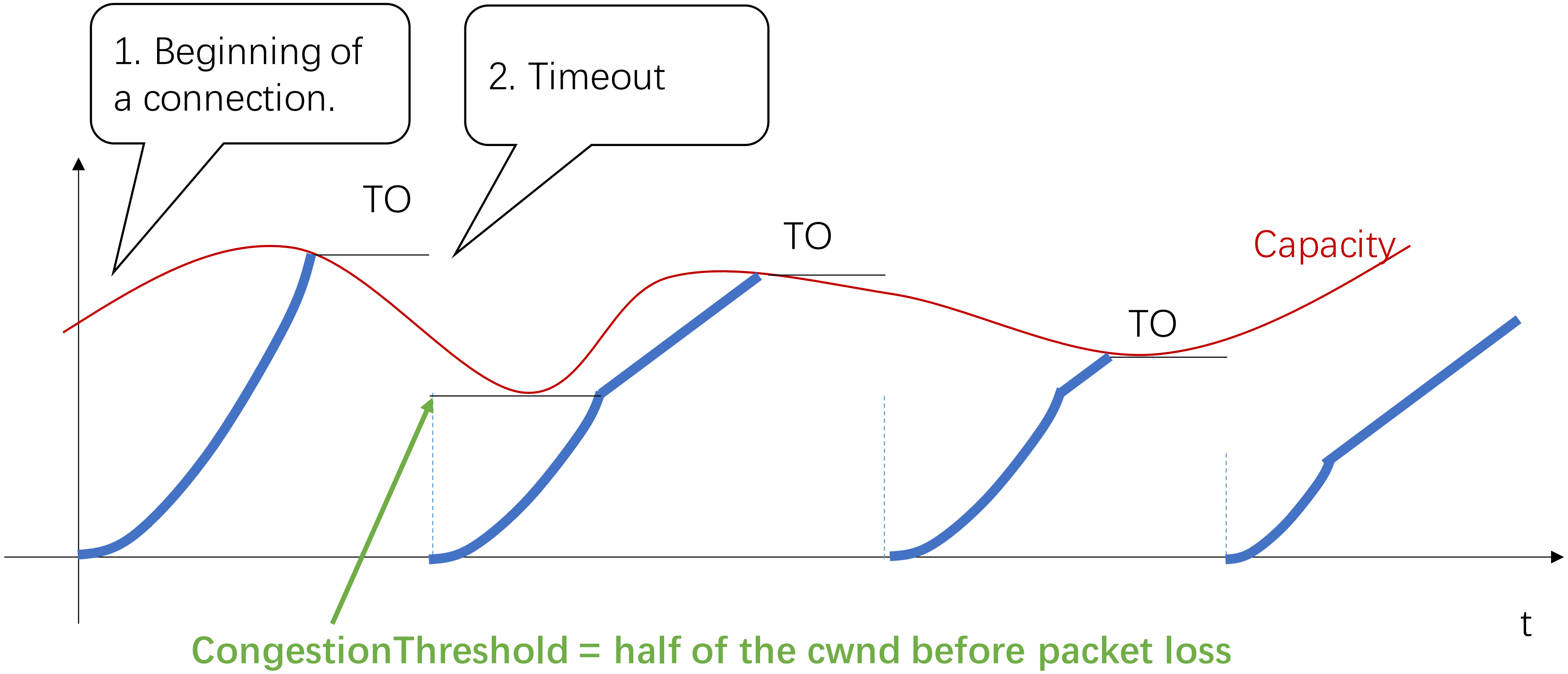
- Intuition: speed up additive Increase when TCP start
- Why “Slow Start”
  - “Slow Start” is not slow compared with additive Increase
  - “Slow Start” is slow compared with sending a whole window’s worth of data (original TCP)
- Double CongestionWindow per round-trip time
  - If successfully received one ack
    - cwnd = cwnd +1
    - Until cwnd == **CongestionThreshold**
      - CongestionThreshold = half of the cwnd before packet loss
  - Then do Additive Increase

Slow Start

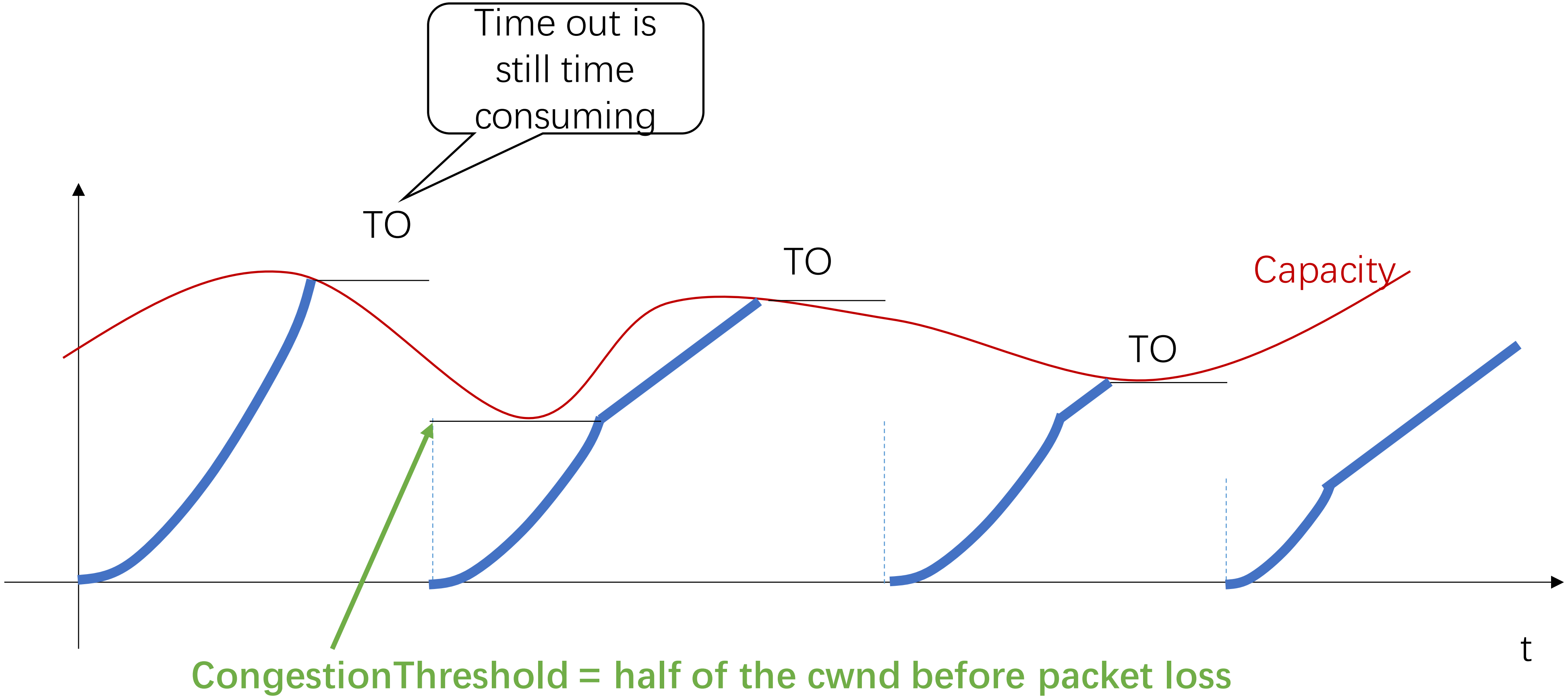
- If successfully received one ack
    - cwnd = cwnd +1



Slow Start runs in two situations



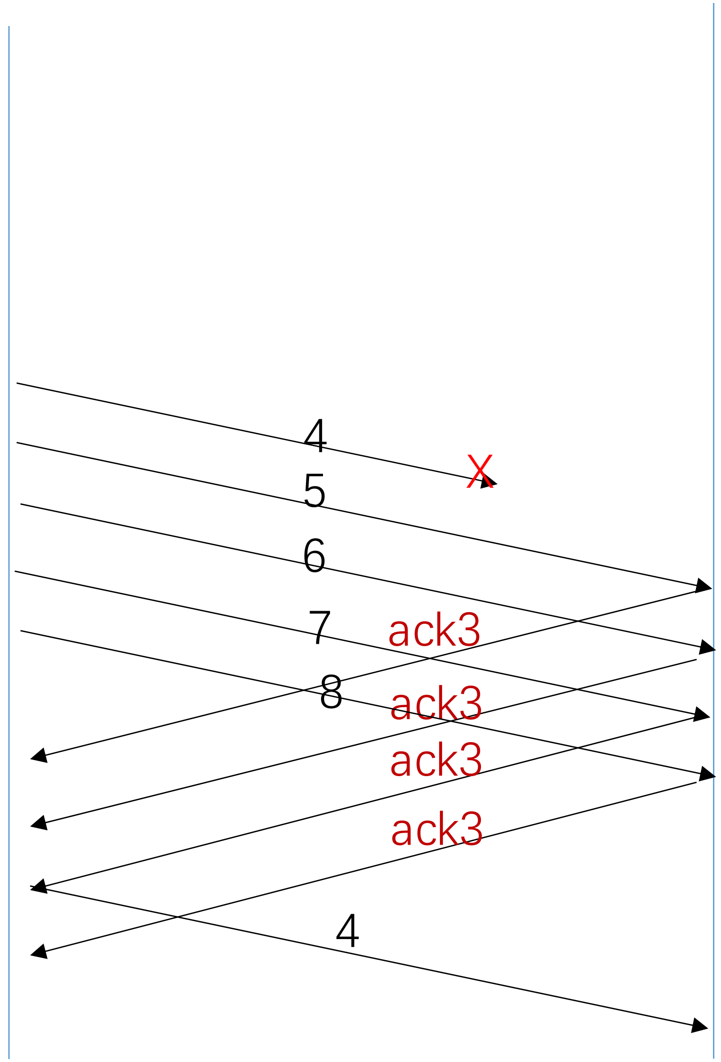
Slow Start



Time out is still time consuming
TO
TO
TO
Capacity
t
CongestionThreshold = half of the cwnd before packet loss

Fast Retransmission

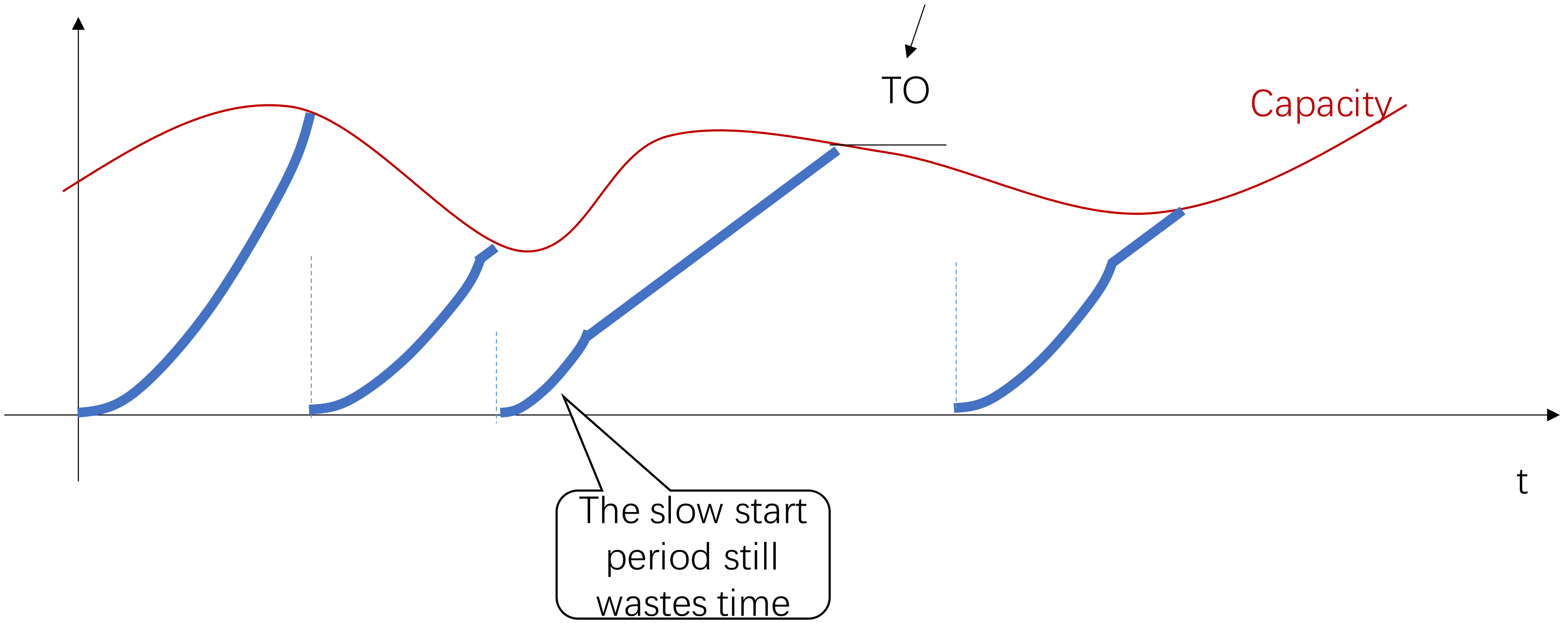
- Intuition: use duplicate ACK to indicate packet loss
- Approach:
  - Receiver replies every TCP segment with acknum = next byte expected
  - Transmitter resends a segment after 3 duplicate acks
    - 3 duplicate acks => possible packet loss
- Throughput Gain: 20%



Fast Retransmission

Timeout still exists

- Too many packet loss
- Window may be too small to generate enough duplicate acks

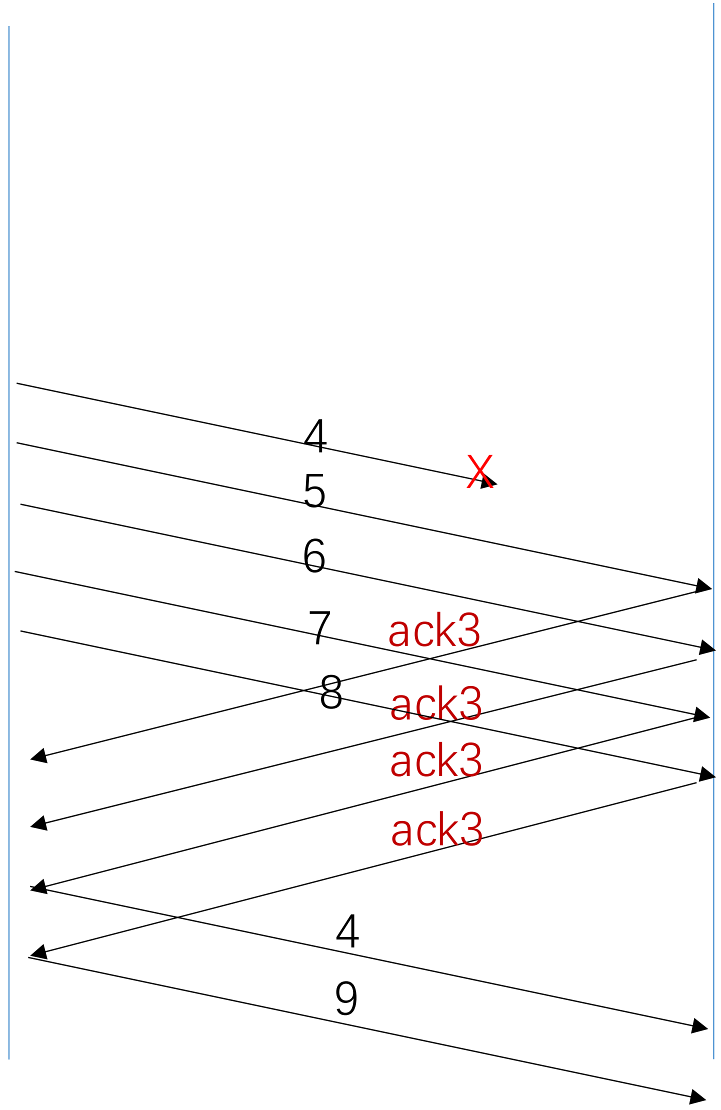


TCP Tahoe

Fast Recovery

Intuition: dupACKs can pace retransmission

- No need to start from window size 1
- Instead, reset CWND to CWND/2



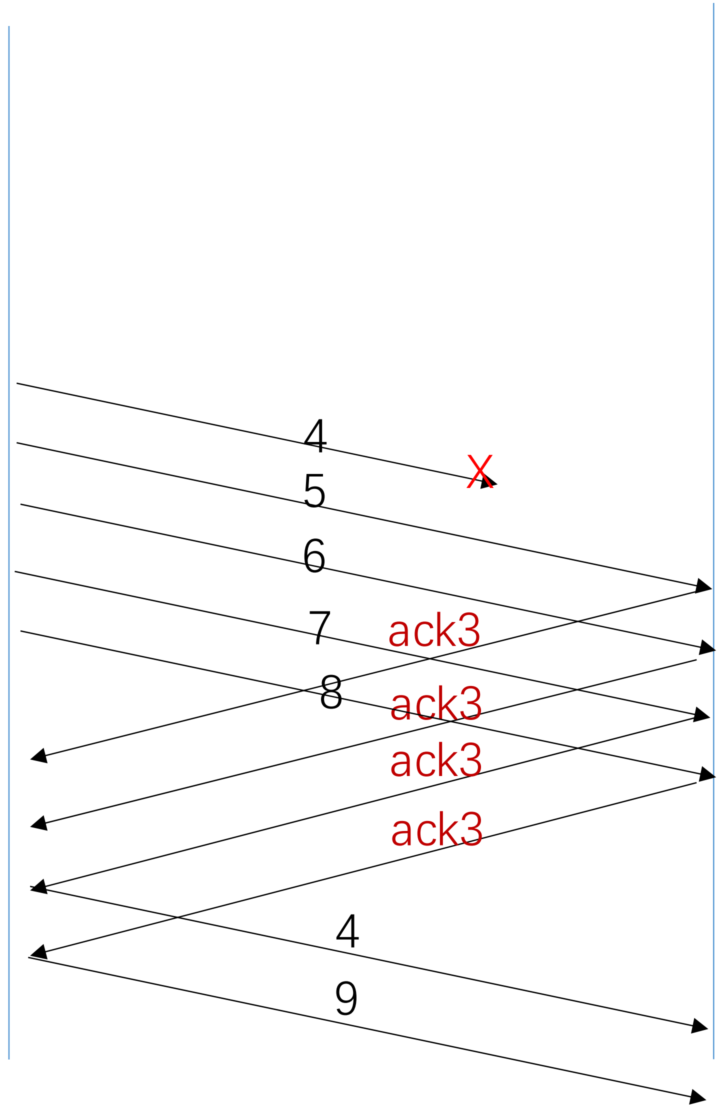
Fast Recovery

But how many ducACKs we have to wait before retransmit?

Recall that we want to keep

Inflight < CWND.

Idea: use dupACK to estimate the number of inflight packets.



CS 120 LEC 17

Fast Recovery

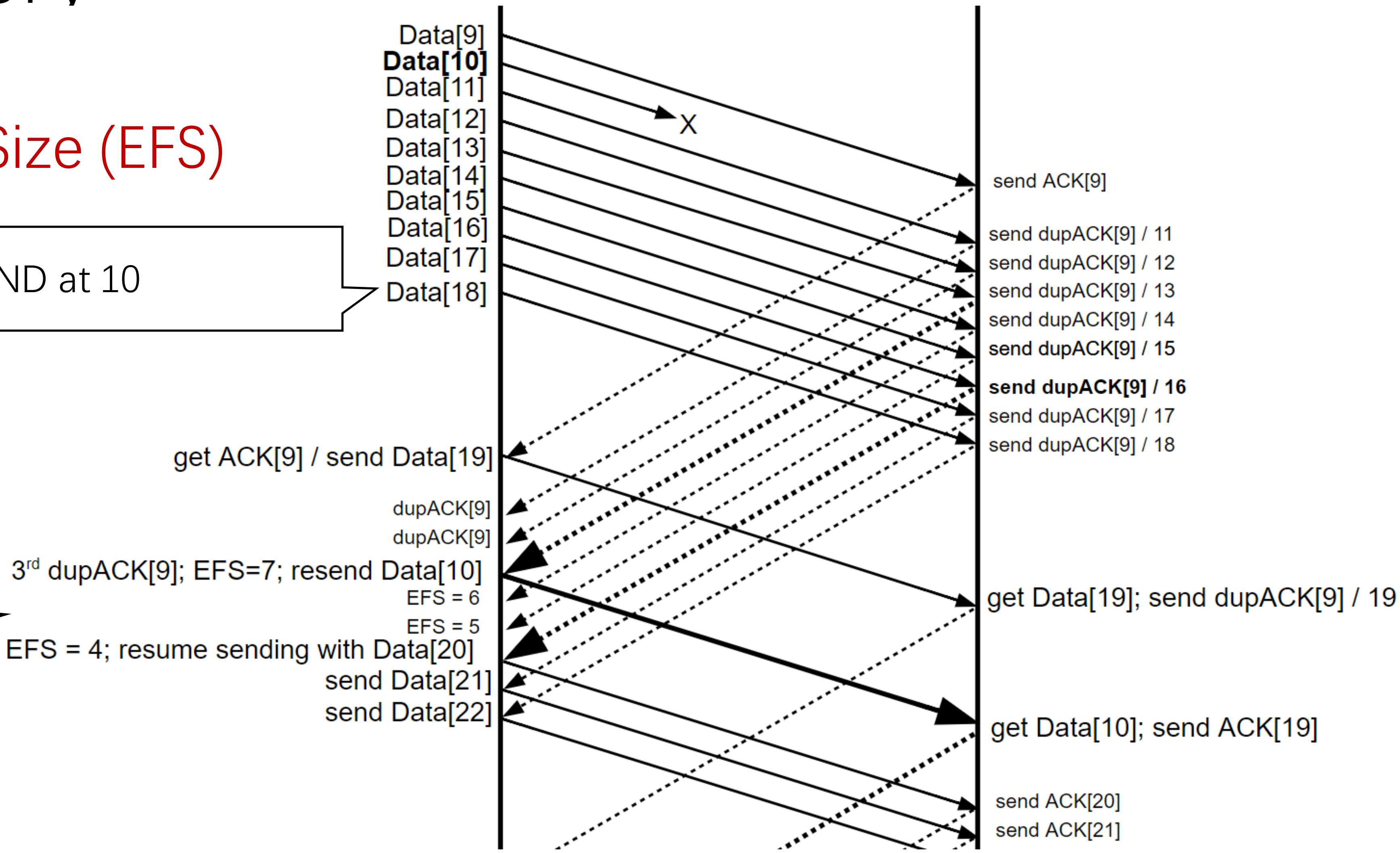
Estimated Flight Size (EFS)

CWND at 10

Reset CWND to 5

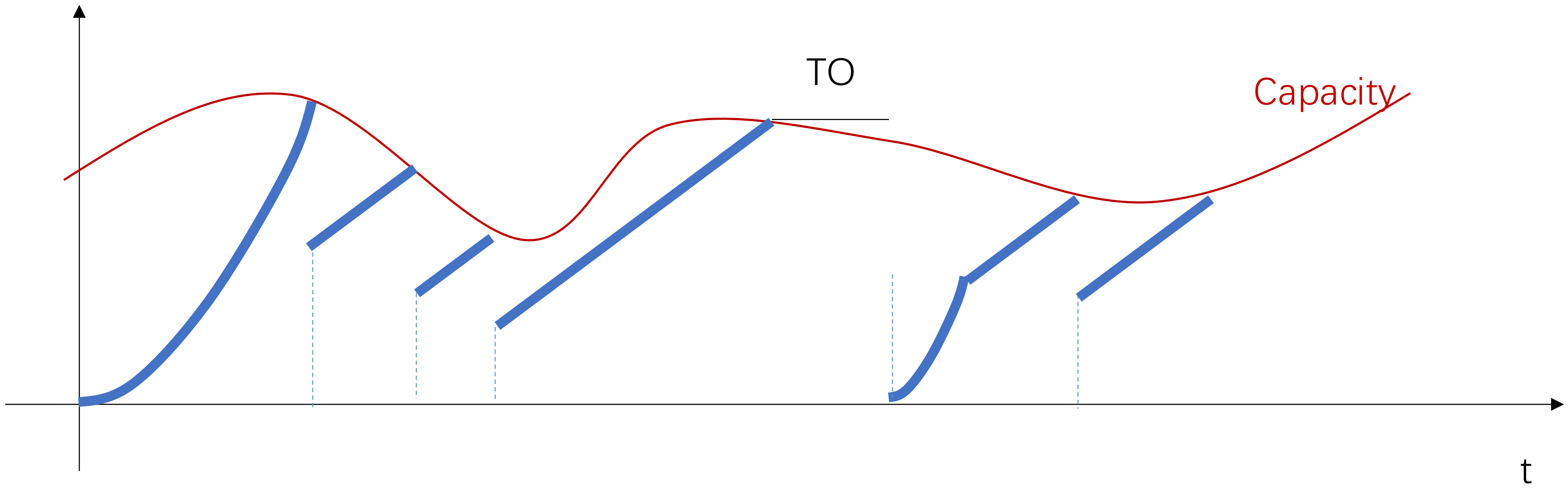
Increment EFS by 1 on receiving each dupACK

Resume sending when EFS < CWND (5)



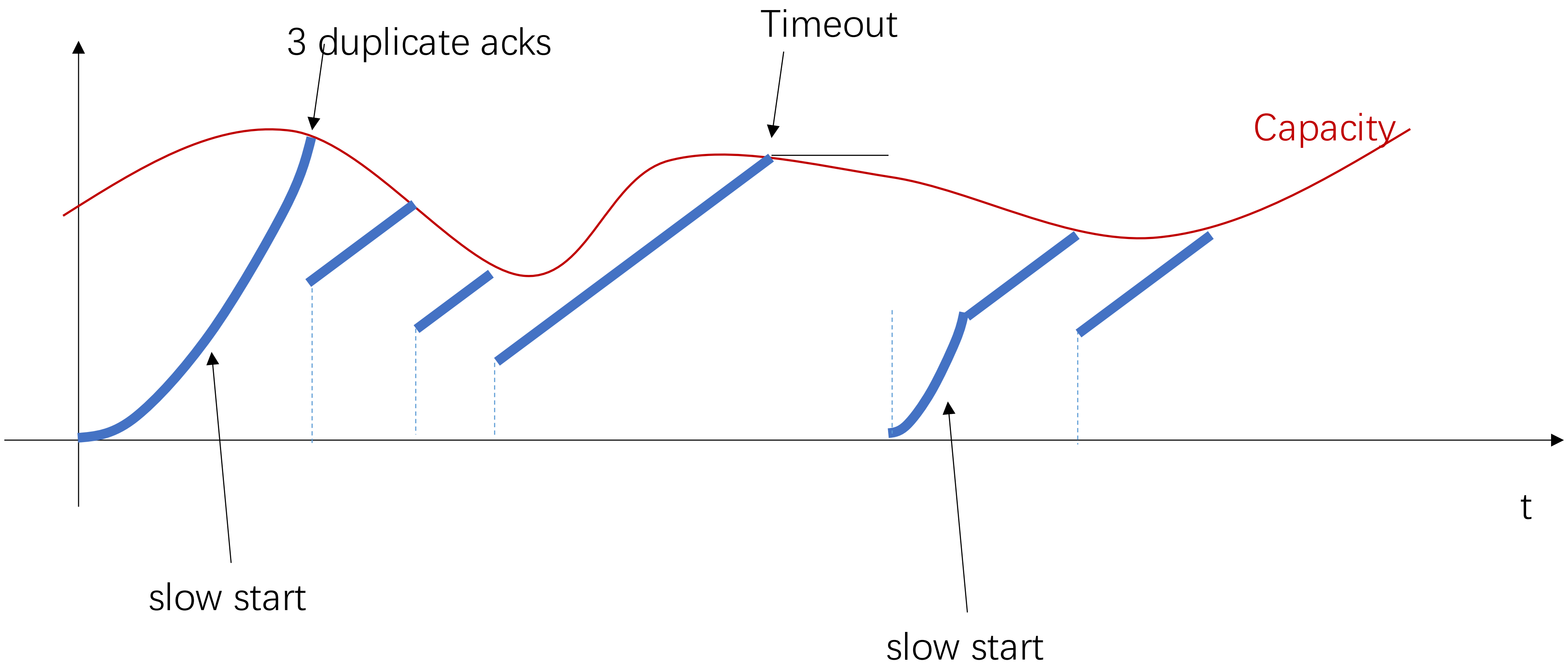
Refer to http://intronetworks.cs.luc.edu/current/html/reno.html

Fast Recovery



TCP Reno

TCP Reno



TCP Congestion Control

- Objective: Estimate and adapt to (varying) network capacity
- Approach: Adjust Sliding Window
  - MaxWindow = MIN(CongestionWindow, AdvertisedWindow)
  - Decrease CongestionWindow upon detecting congestion
  - Increase CongestionWindow upon lack of congestion
- Basic Components
  - Additive Increase/Multiplicative Decrease (AIMD)
  - Slow Start
  - Fast Retransmission
  - Fast Recovery

➢Other Variations

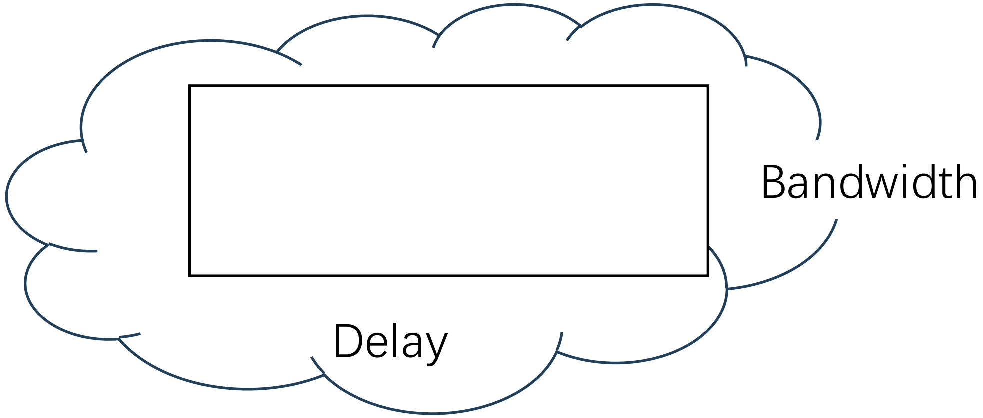
TCP Congestion Control Algorithms

- ref: https://en.wikipedia.org/wiki/TCP_congestion_control

Variant	Feedback	Required changes	Benefits	Fairness
(New)Reno	Loss	-	-	Delay
Vegas	Delay	Sender	Less loss	Proportional
High Speed	Loss	Sender	High bandwidth	
BIC	Loss	Sender	High bandwidth	
CUBIC	Loss	Sender	High bandwidth	
H-TCP	Loss	Sender	High bandwidth	
FAST	Delay	Sender	High bandwidth	Proportional
Compound TCP	Loss/Delay	Sender	High bandwidth	Proportional
Westwood	Loss/Delay	Sender	L	
Jersey	Loss/Delay	Sender	L	
BBR[11]	Delay	Sender	BLVC, Bufferbloat	
CLAMP	Multi-bit signal	Receiver, Routers	V	Max-min
TFRC	Loss	Sender, Receiver	No Retransmission	Minimum delay
XCP	Multi-bit signal	Sender, Receiver, Router	BLFC	Max-min
VCP	2-bit signal	Sender, Receiver, Router	BLF	Proportional
MaxNet	Multi-bit signal	Sender, Receiver, Router	BLFSC	Max-min
JetMax	Multi-bit signal	Sender, Receiver, Router	High bandwidth	Max-min
RED	Loss	Router	Smaller delay	

TCP CUBIC

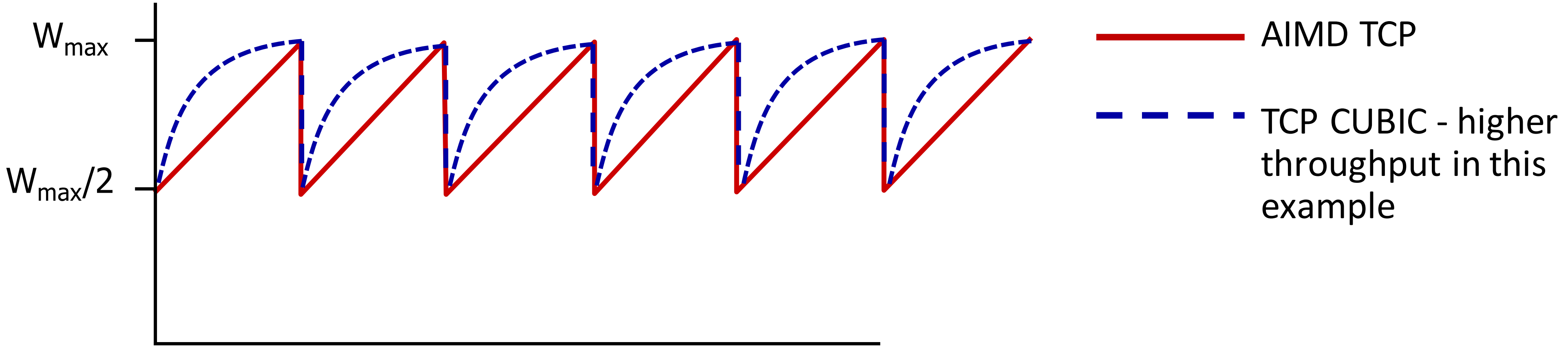
Goal: support network with high bandwidth-delay product.



Original TCP algorithm takes too many round trips to reach available capacity in such “long fact” network.

TCP CUBIC

- A better way than AIMD to "probe" for usable bandwidth
  - Intuition: after cutting rate/window in half on loss, initially ramp to W_{max} faster, but then approach W_{max} more slowly



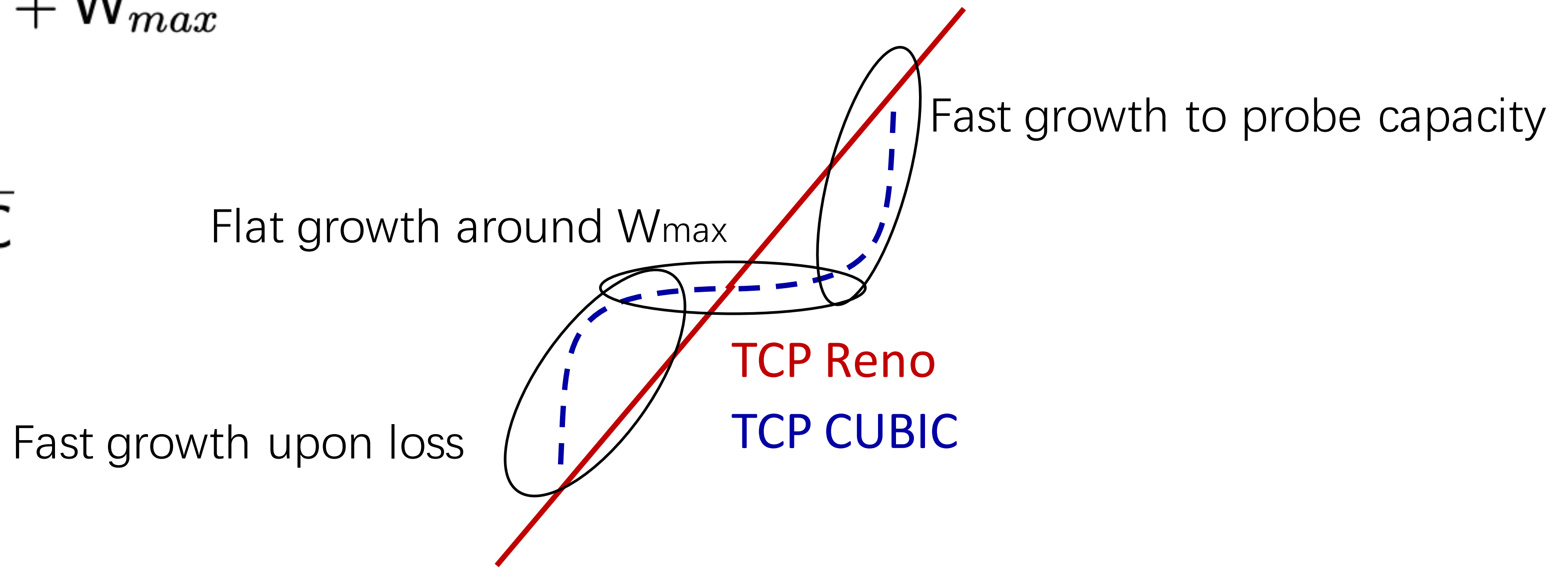
TCP CUBIC

- Why "CUBIC" ?
  - Increase cwnd as a function of the cube of the distance between current time and the estimated time reaching the capacity

$$\mathrm{CWND}(t) = C \times (t - K)^3 + W_{max}$$

where:

$$K = \sqrt[3]{W_{max} \times (1 - \beta)/C}$$



TCP Cubic

Adjust CWND at regular time interval

- As opposed to original TCP, adjust on receiving each ACKs.

- Better fairness between short and long RTT flows
  - short RTT flows receives ACKs more frequently: grows CWND faster in original TCO algorithm.

Demo

- Sliding Window code location

  `/net/ipv4/`

  https://elixir.bootlin.com/linux/latest/source/net/ipv4

- Switching Sliding Window Scheme
  - Show current schemes

  ```
  cat /proc/sys/net/ipv4/tcp_congestion_control
  ```

  - Switch congestion control scheme

  ```
  sysctl net.ipv4.tcp_available_congestion_control
  ```

TCP Congestion in Wireless

- Challenges
  - Timeout doesn’t mean congestions (with very high probability)
    - Reason: wireless channel is not reliable
  - Possible Solutions
    - Error Correction
      - Additional traffic overhead
    - MAC layer retransmission (WiFi)
      - Large End-to-end RTT variance

Reference

- Textbook 6.3
- http://intronetworks.cs.luc.edu/current/html/reno.html