CS120: Computer Networks

Lecture 18. Congestion Control 2

Zhice Yang

Congestion Control

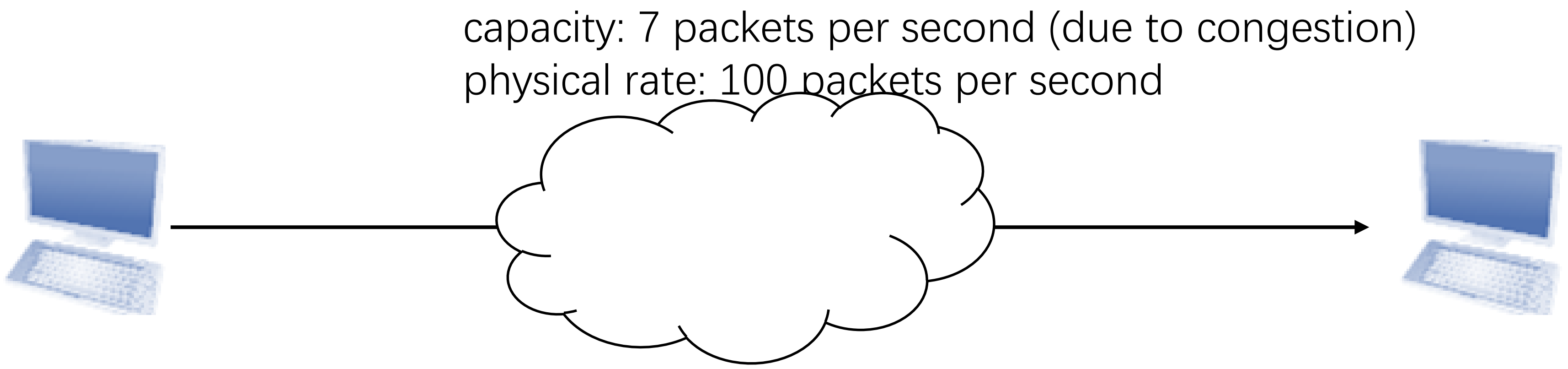
- Queuing
- ➢ Connection Control Methods
  - Congestion Control
  - Congestion Avoidance
- QoS

TCP Congestion Control

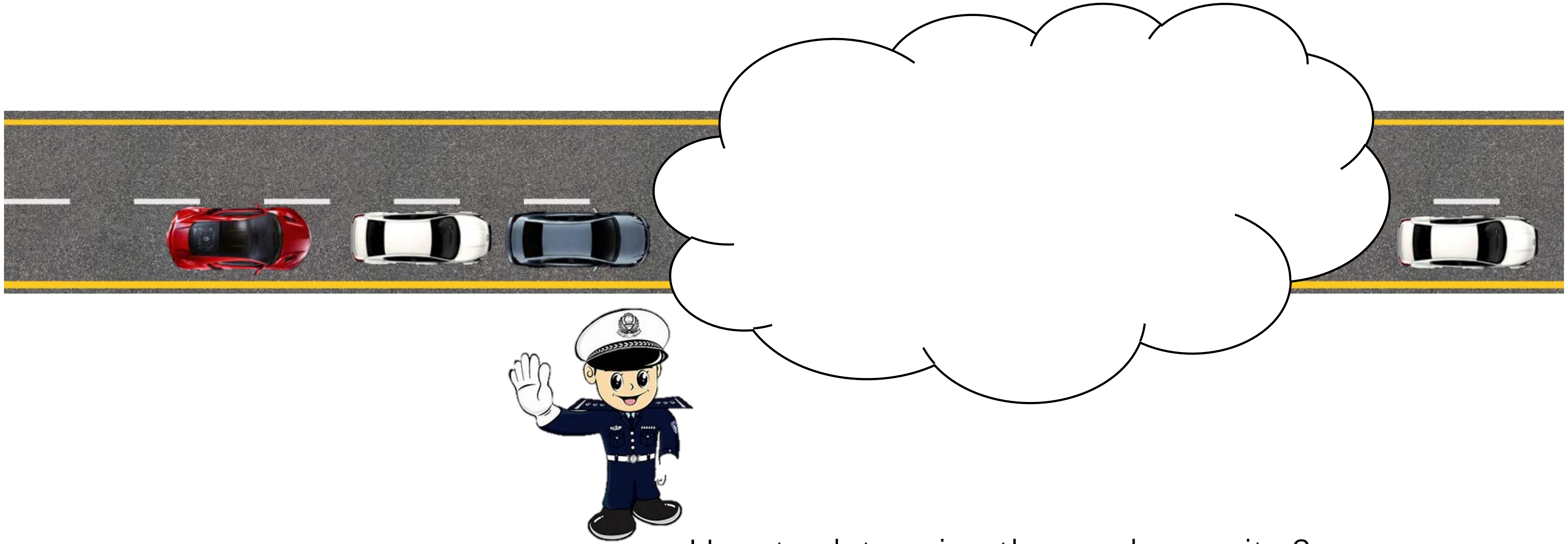
- Introduced by Van Jacobson through his Ph.D. dissertation work in late 1980s
  - 8 years after TCP became operational
- Basic ideas
  - Each host determines network capacity for itself
    - Leverage feedback
    - Assumption: FIFO or FQ queue in routers
- Challenges
  - Determining the available capacity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

Congestion Control with Sliding Window

- Size of sliding window is determined by network capacity: delay*bandwidth
  - Problem
    - bandwidth is unknown
    - How to pace the sender ?

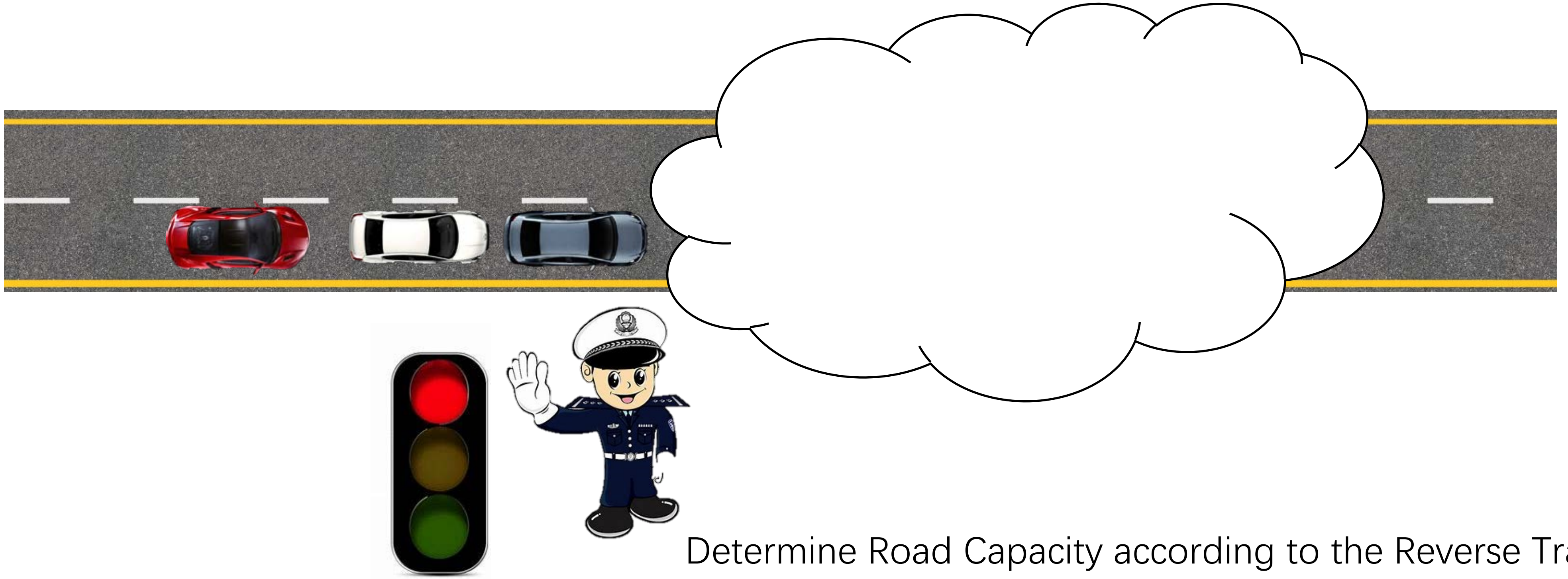


Congestion Control in Road



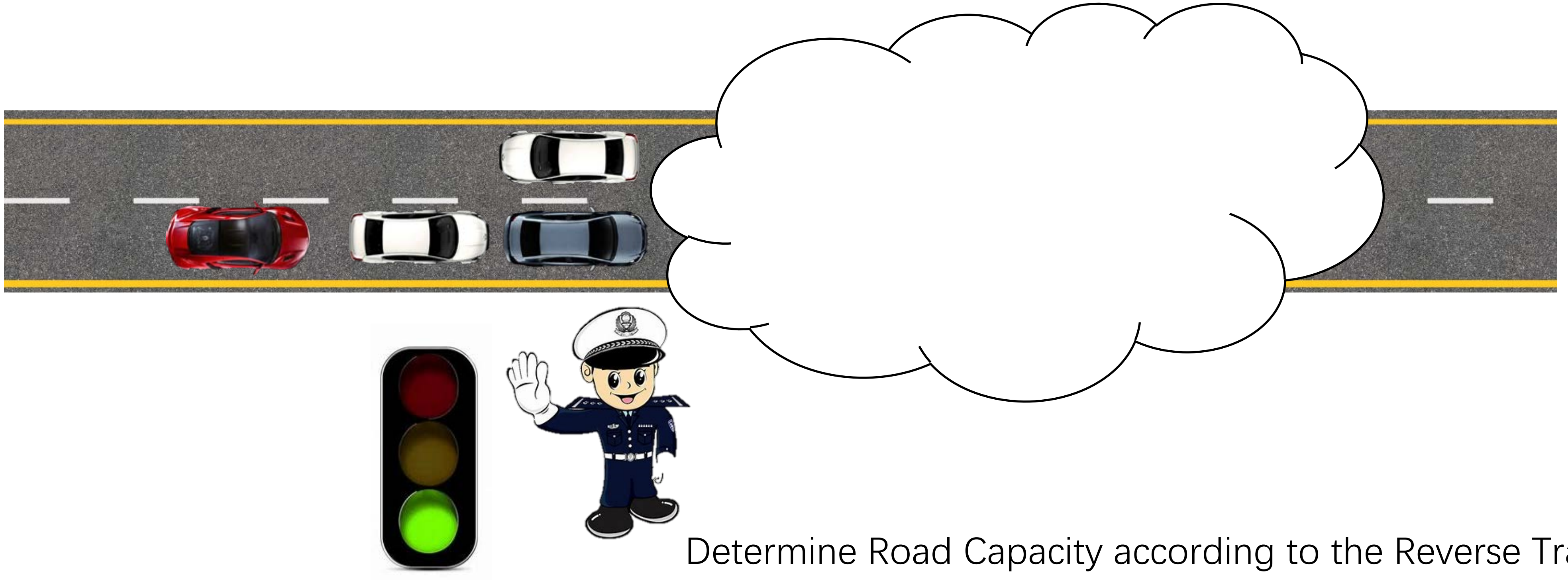
How to determine the road capacity ?

Congestion Control in Road



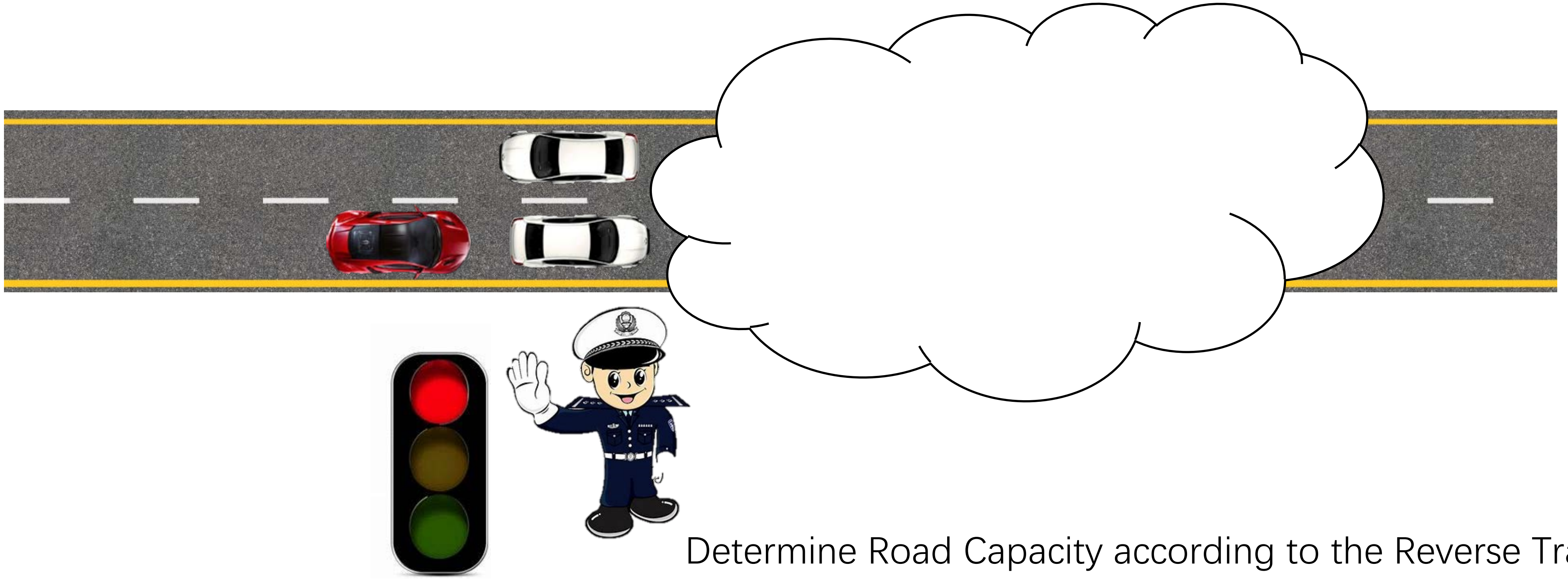
Determine Road Capacity according to the Reverse Traffic

Congestion Control in Road



Determine Road Capacity according to the Reverse Traffic

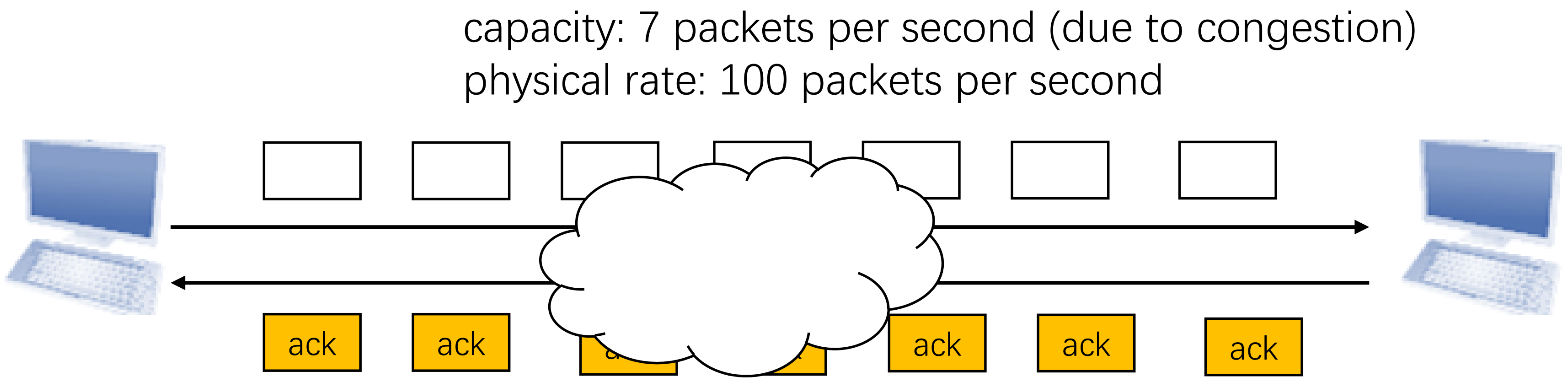
Congestion Control in Road



Determine Road Capacity according to the Reverse Traffic

Congestion Control with Sliding Window

- Size of sliding window is determined by network capacity: delay*bandwidth
  - Problem
    - bandwidth is unknown
    - How to pace the sender ?



Use ACK to estimate congestion and pace the sending

TCP Congestion Control

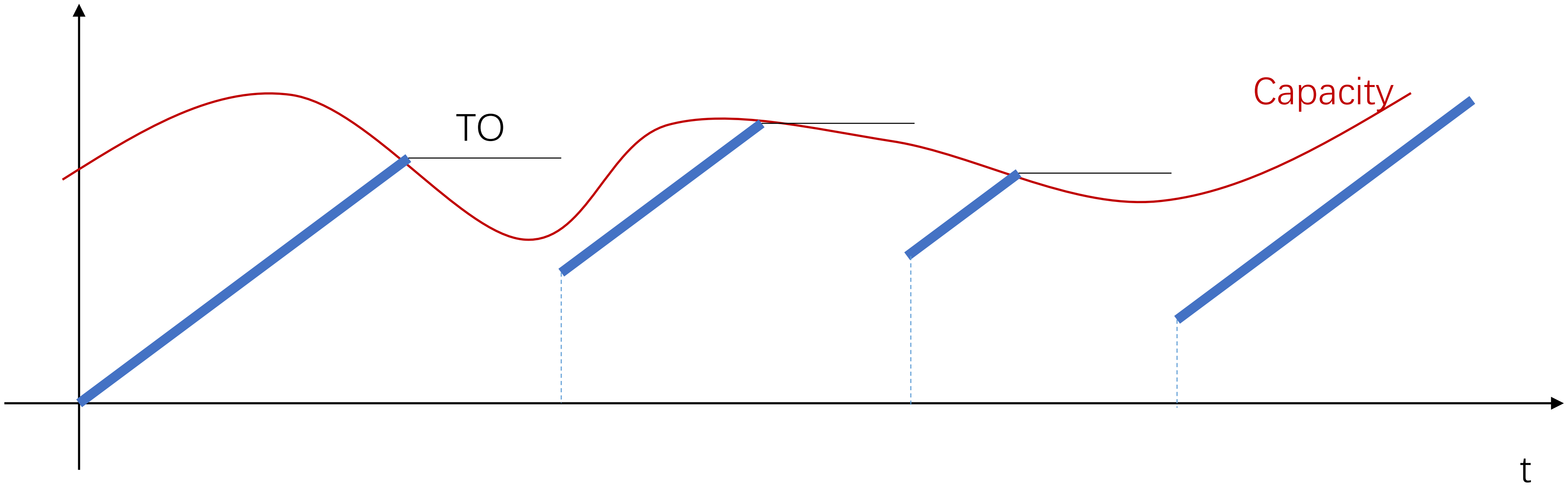
- Objective: Estimate and adapt to (varying) network capacity
- Approach: Adjust Sliding Window
  - MaxWindow = MIN(CongestionWindow, AdvertisedWindow)
  - Decrease CongestionWindow upon detecting congestion
  - Increase CongestionWindow upon lack of congestion
- Basic Components
  - Additive Increase/Multiplicative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

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
    - CongestionWindow = CongestionWindow+1
- Multiplicative Decrease
  - If packet loss
    - CongestionWindow = CongestionWindow/2

AIMD

- TCP sawtooth pattern



Sliding Window in TCP: Adaptive Timeout

- Jacobson/Karels Algorithm Implementation

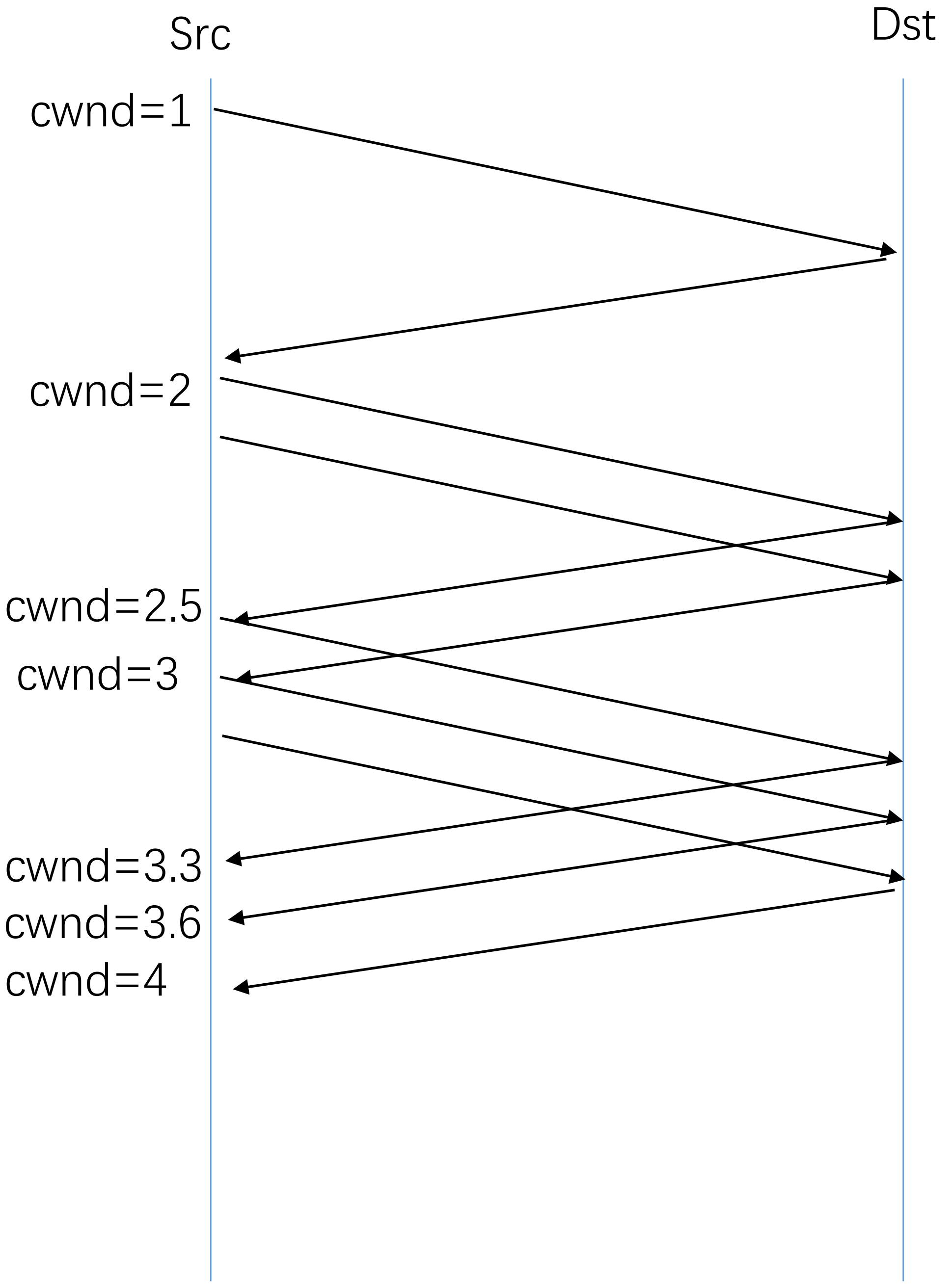
$$\text{Difference} = \text{SampleRTT} - \text{EstimatedRTT}$$
$$\text{EstimatedRTT} = \text{EstimatedRTT} + (\delta * \text{Difference})$$
$$\text{Deviation} = \text{Deviation} + \delta * (|\text{Difference}| - \text{Deviation})$$
$$\text{TimeOut} = \mu * \text{EstimatedRTT} + \varphi * \text{Deviation}$$

⇩

```
SampleRTT -= (EstimatedRTT >> 3);
EstimatedRTT += SampleRTT;
if (SampleRTT < 0)
      SampleRTT = -SampleRTT;
SampleRTT -= (Deviation >> 3);
Deviation += SampleRTT;
TimeOut = (EstimatedRTT >> 3) + (Deviation >> 1);
```

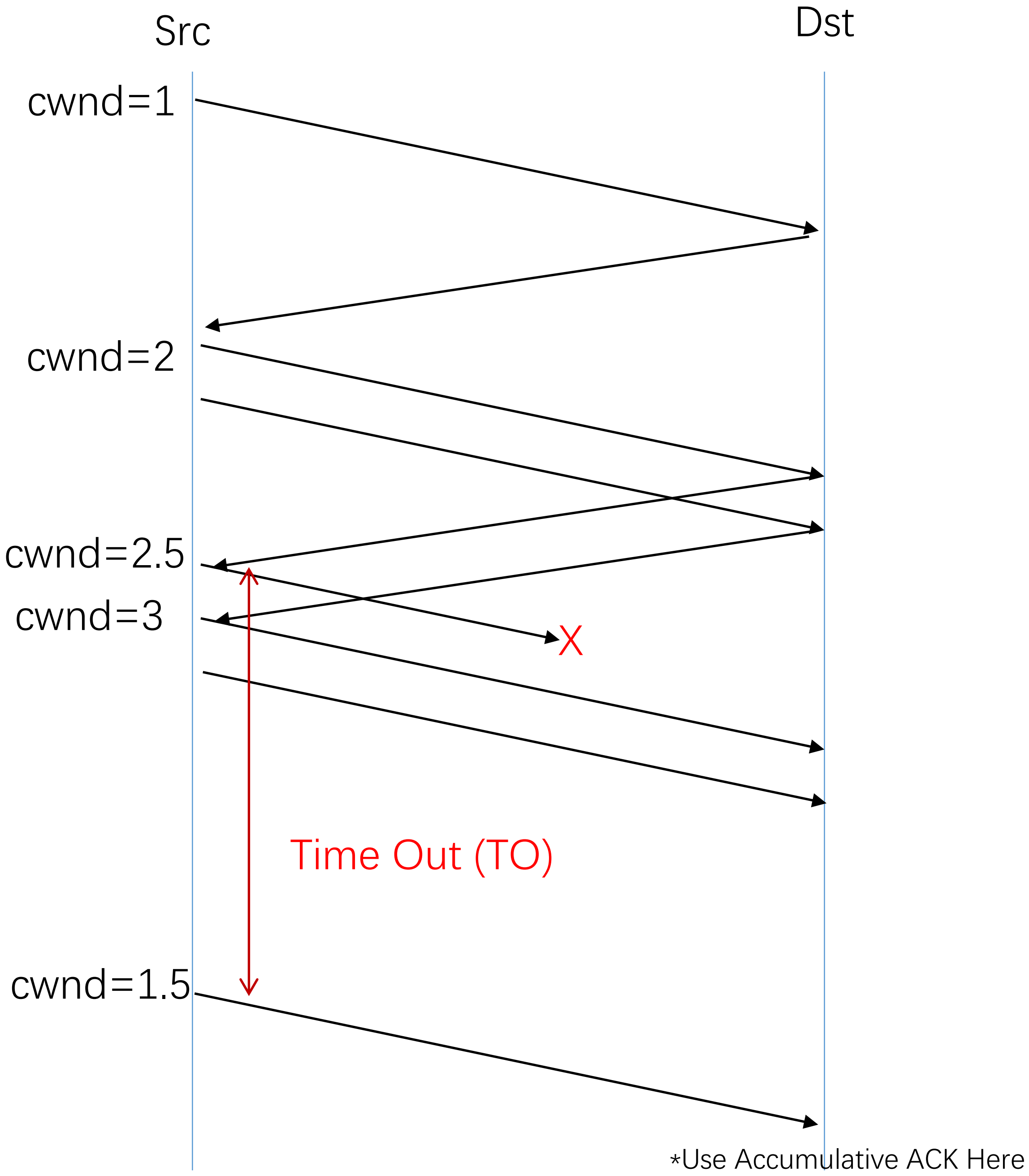
AIMD

- Additive Increase
    - Increment = MSS/CongestionWindow
    - CongestionWindow += Increment



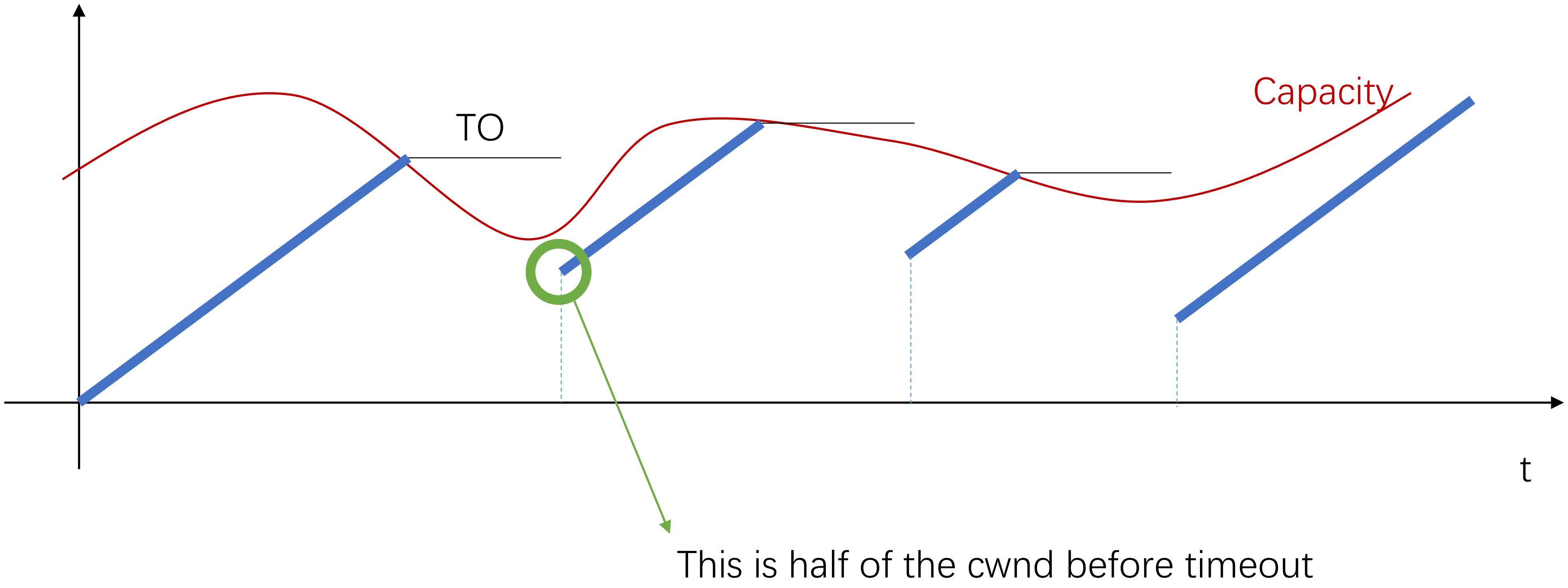
AIMD

- Multiplicative Decrease
  - CongestionWindow = CongestionWindow/2



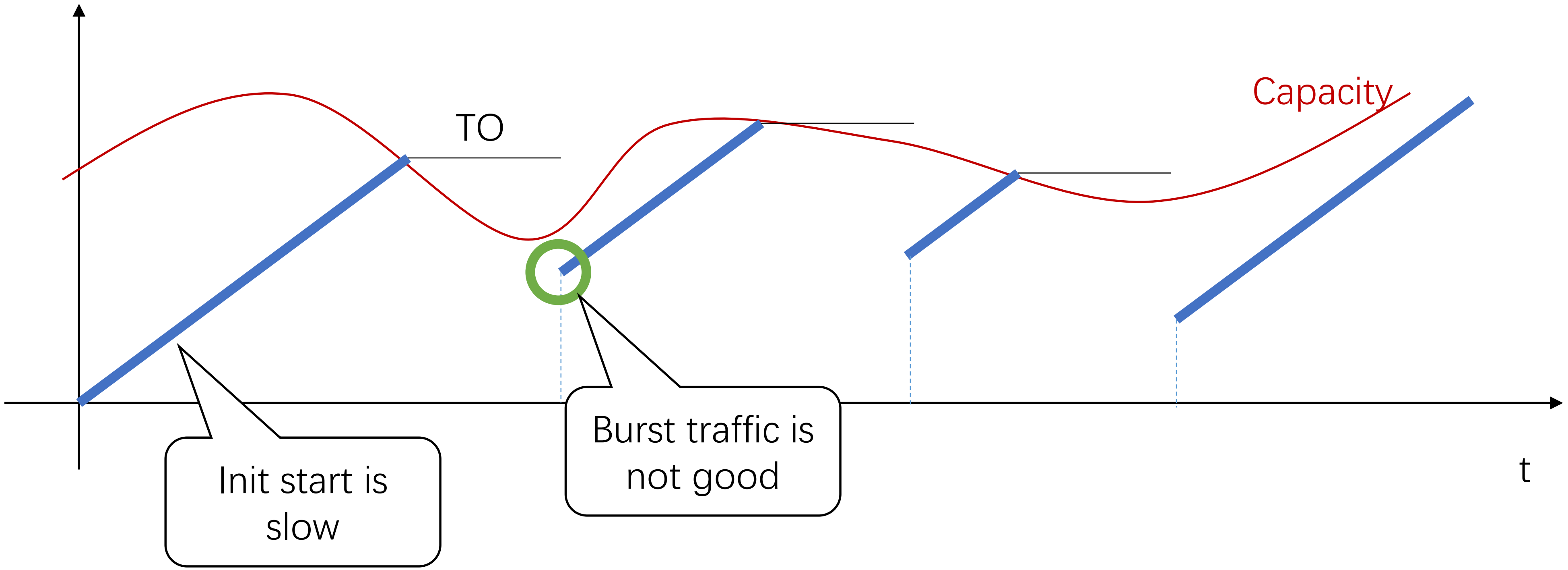
AIMD

- TCP sawtooth pattern



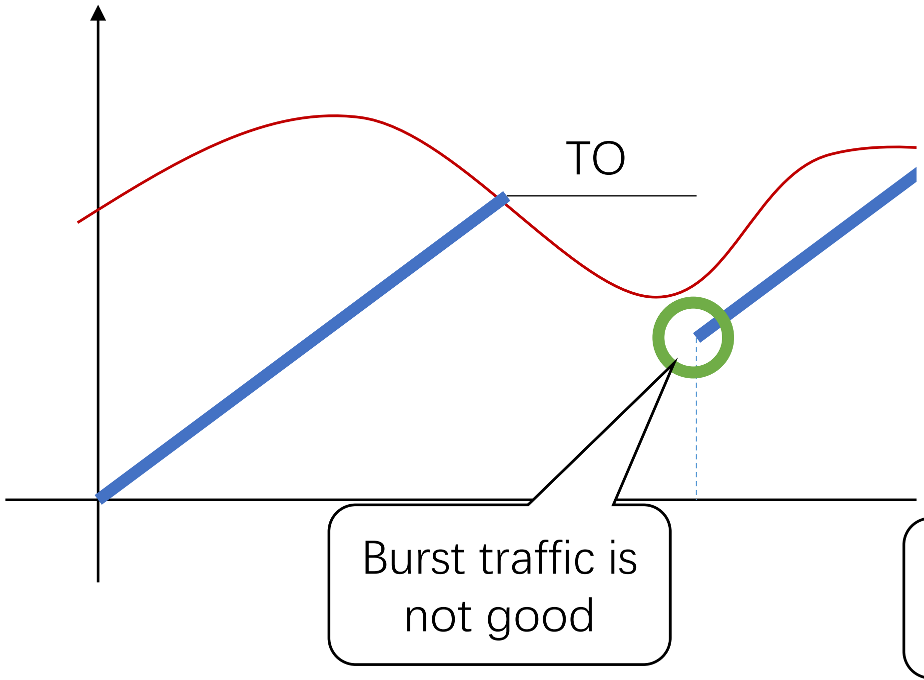
AIMD

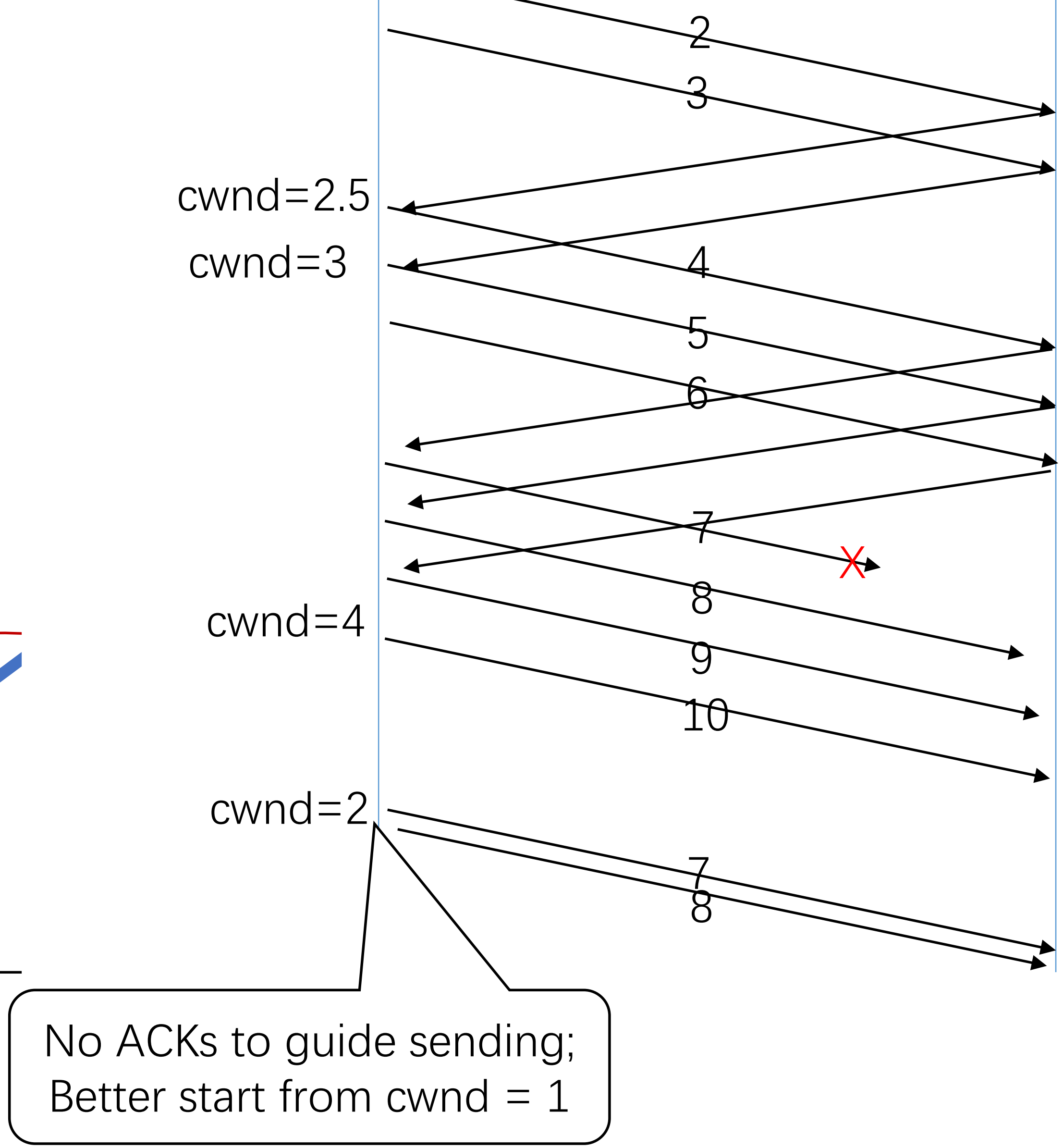
- TCP sawtooth pattern



AIMD

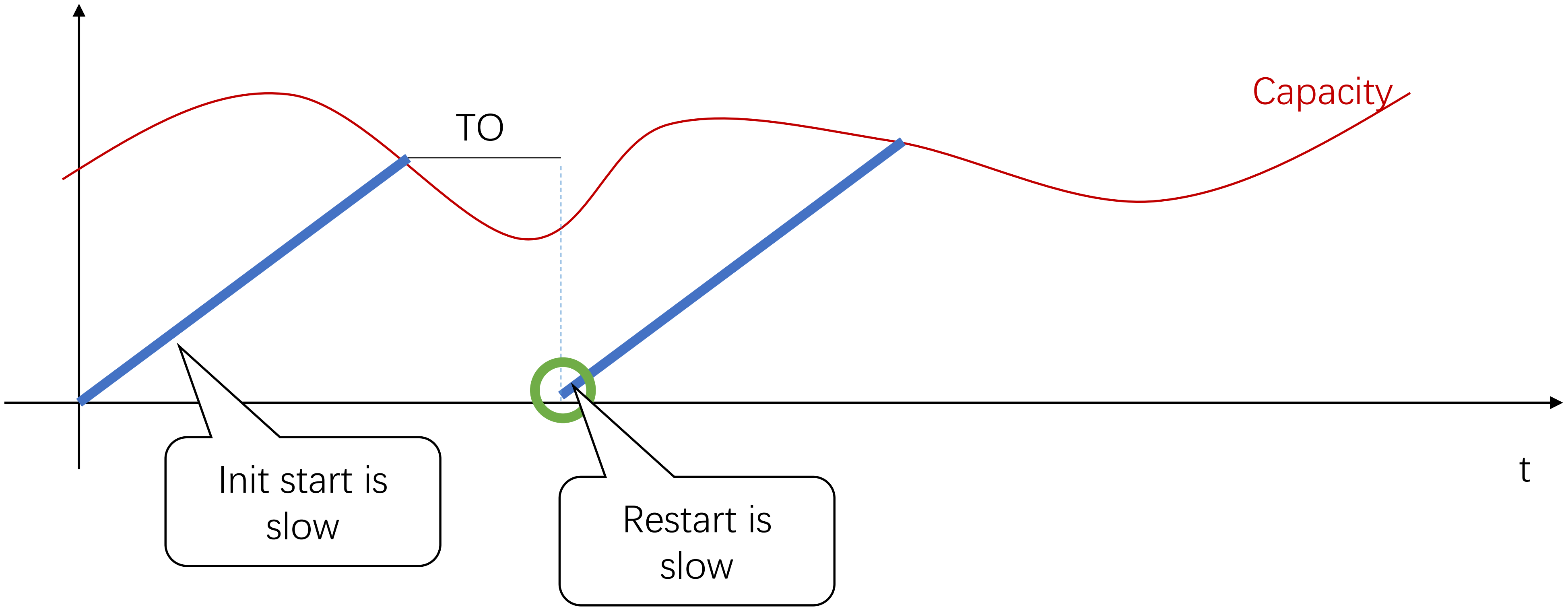
- TCP sawtooth pattern





AIMD

- TCP sawtooth pattern

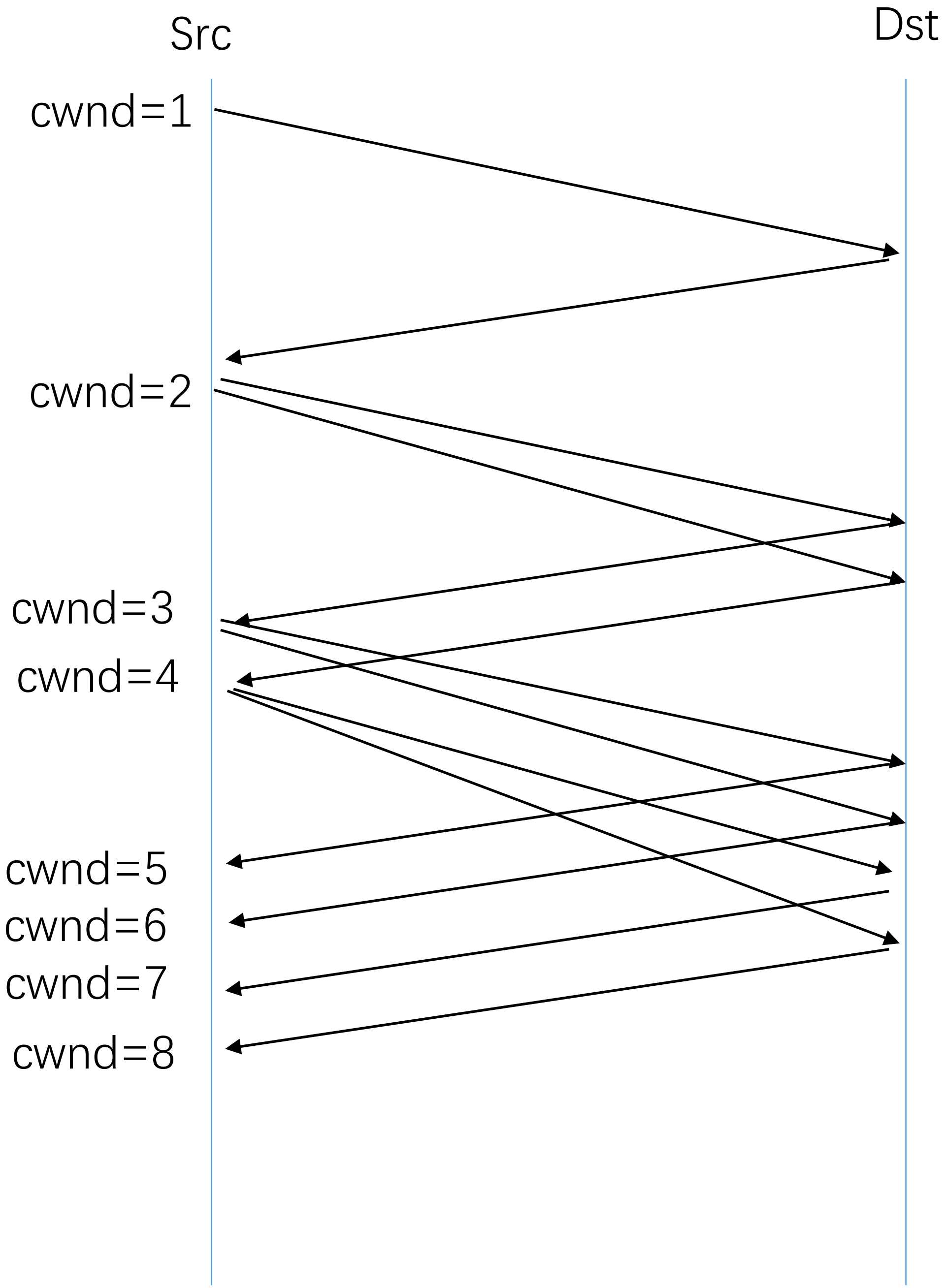


Slow Start

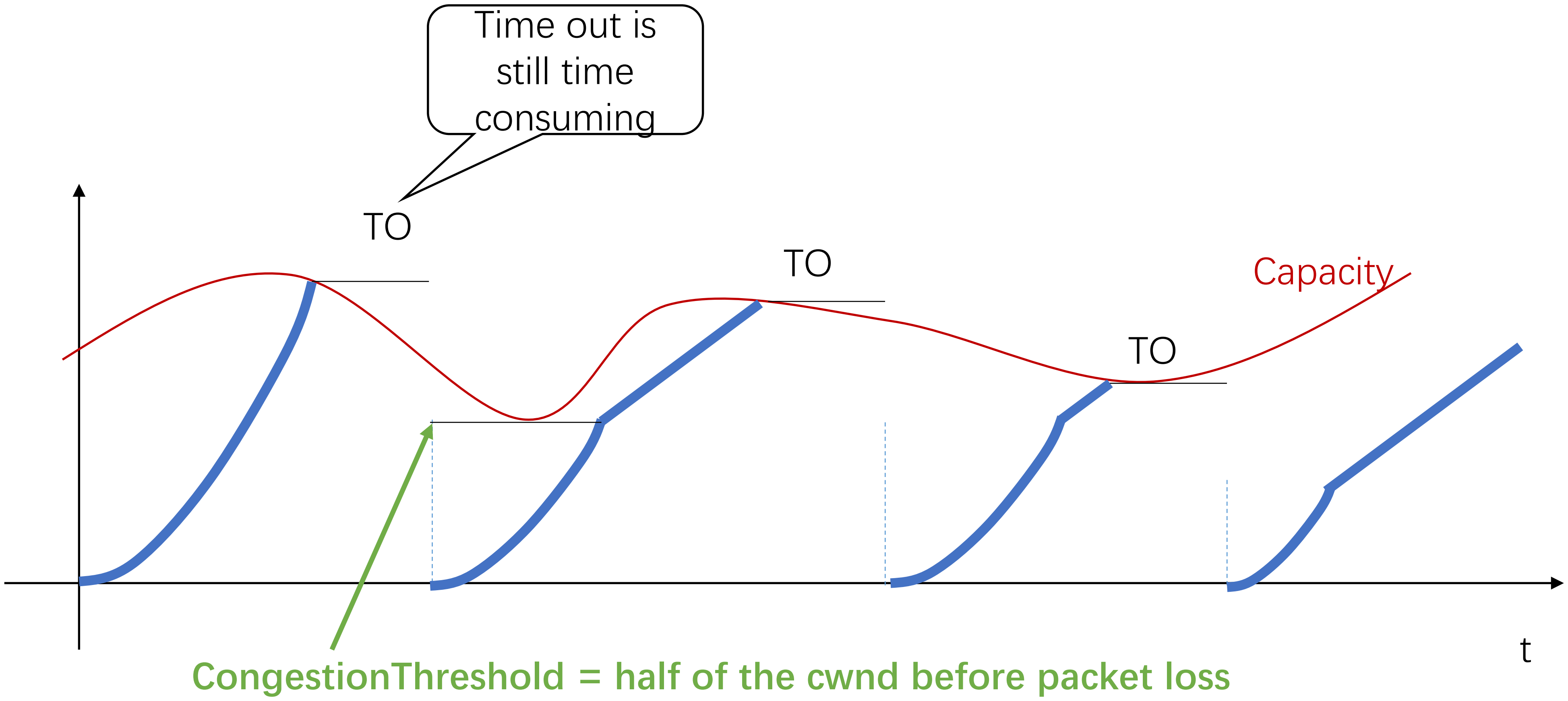
- Intuition: speed up additive Increase when TCP start
- Why "Slow Start"
  - "Slow Start" is not slow compared with additive Increase
  - "Slow Start" is slow compared with sending a whole window's worth of data (Original TCP)
- Double CongestionWindow per round-trip time
  - If successfully received one ack
    - CongestionWindow = CongestionWindow+1
  - Until CongestionWindow == **CongestionThreshold**
  - Then do Additive Increase

Slow Start

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



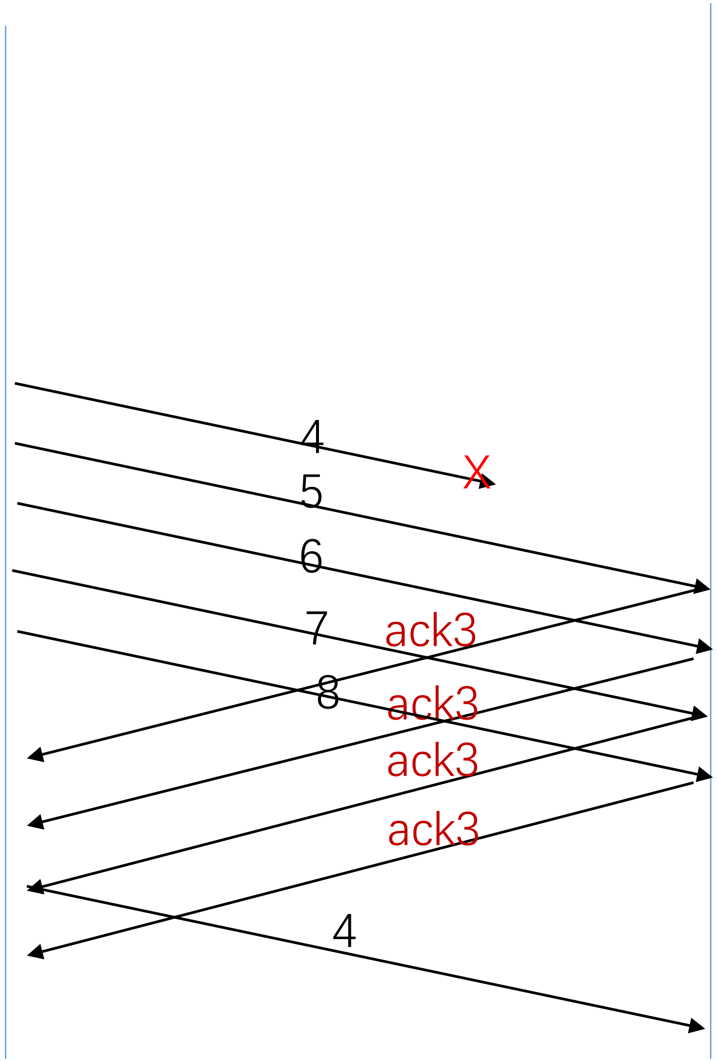
Slow Start



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

Fast Retransmission

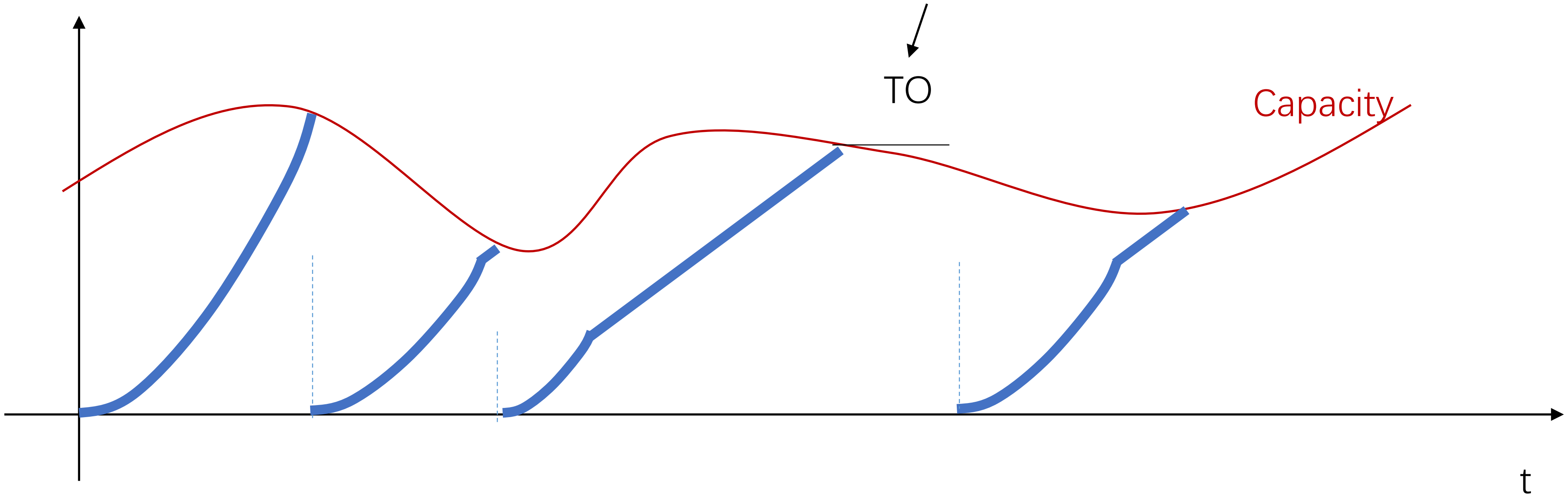
- Intuition: use duplicate ACK to indicate packet loss
- Approach:
  - Receiver replies every TCP segment with acknum = next byte expected
  - Transmitter resends a segment after 3 duplicate acks
    - 3 duplicate acks => possible packet loss
- Throughput Gain: 20%



Fast Retransmission

Timeout still exists

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

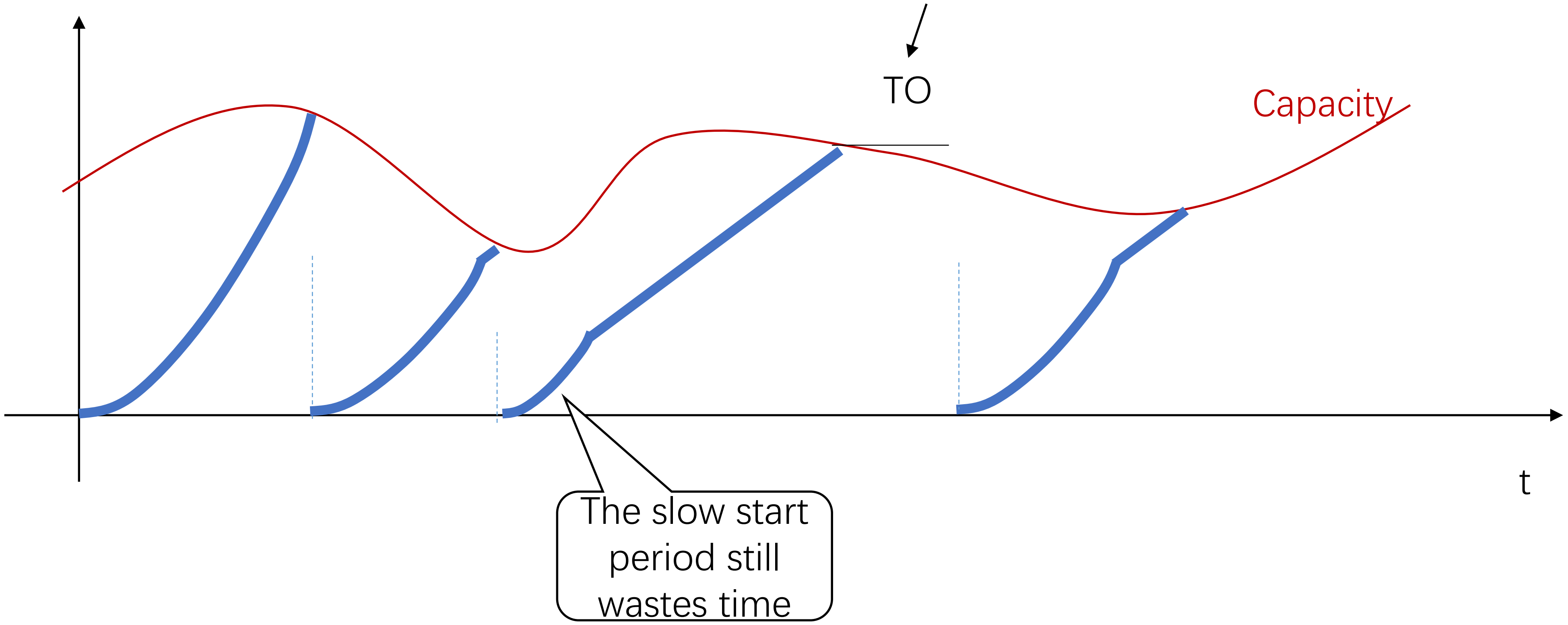


TCP Tahoe

Fast Retransmission

Timeout still exists

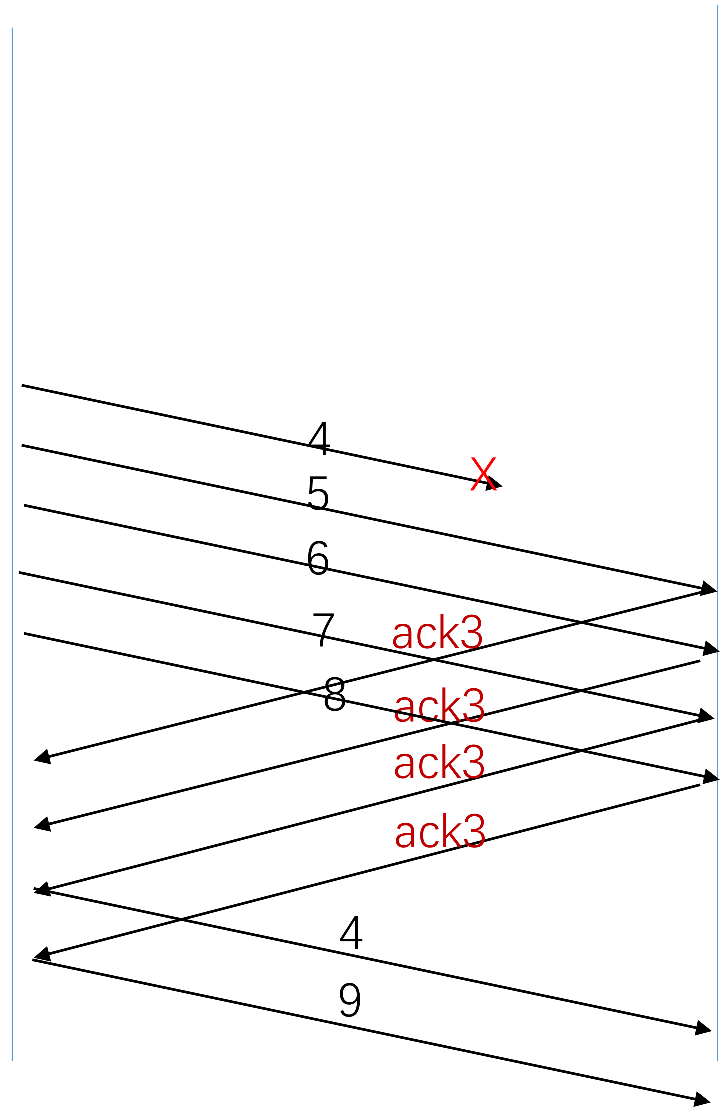
- Too many packet loss
- Window may be too small to generate enough duplicate acks



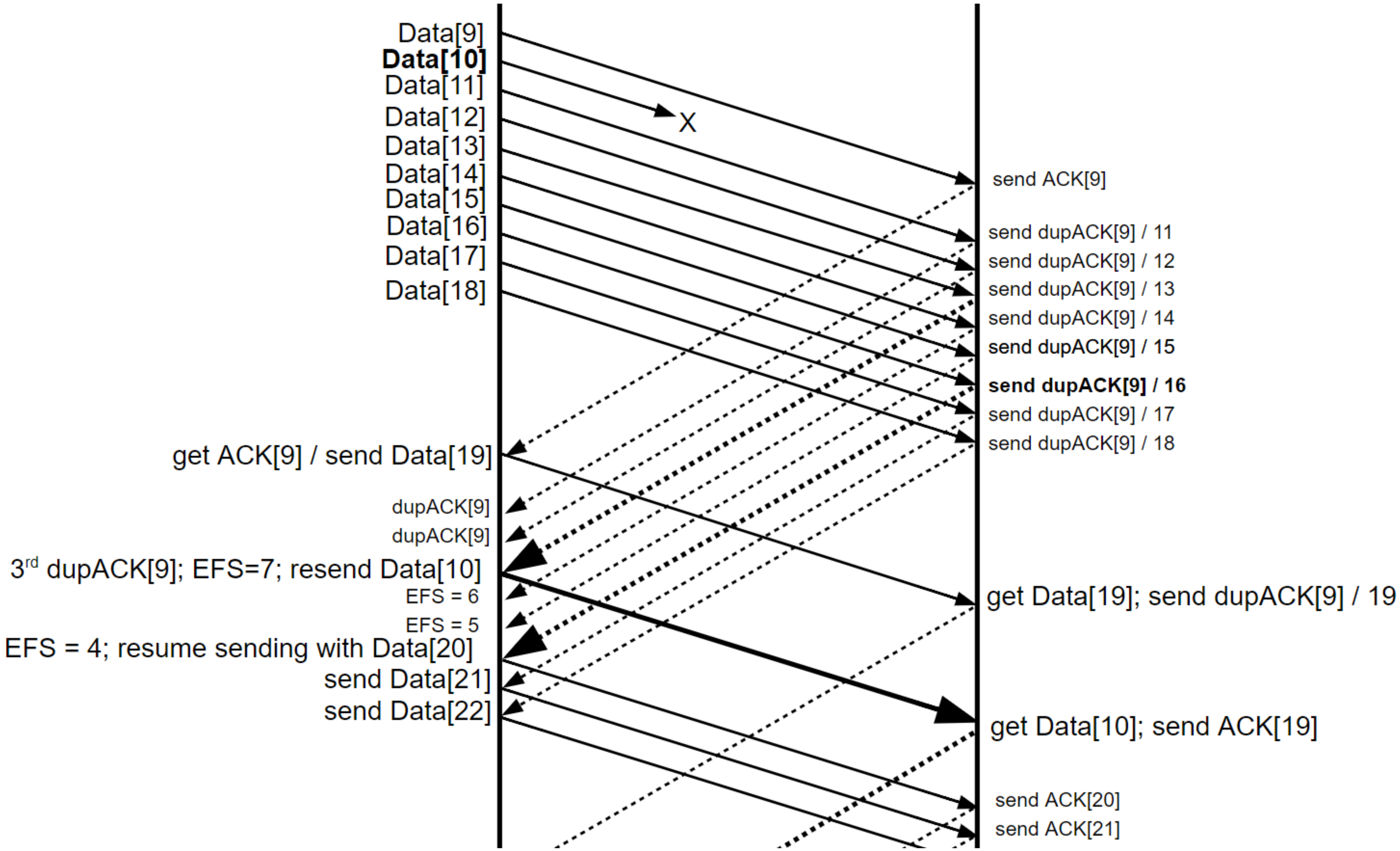
TCP Tahoe

Fast Recovery

- Intuition:
  - Flying acks can be used as clock
    - No need to start from window size 1

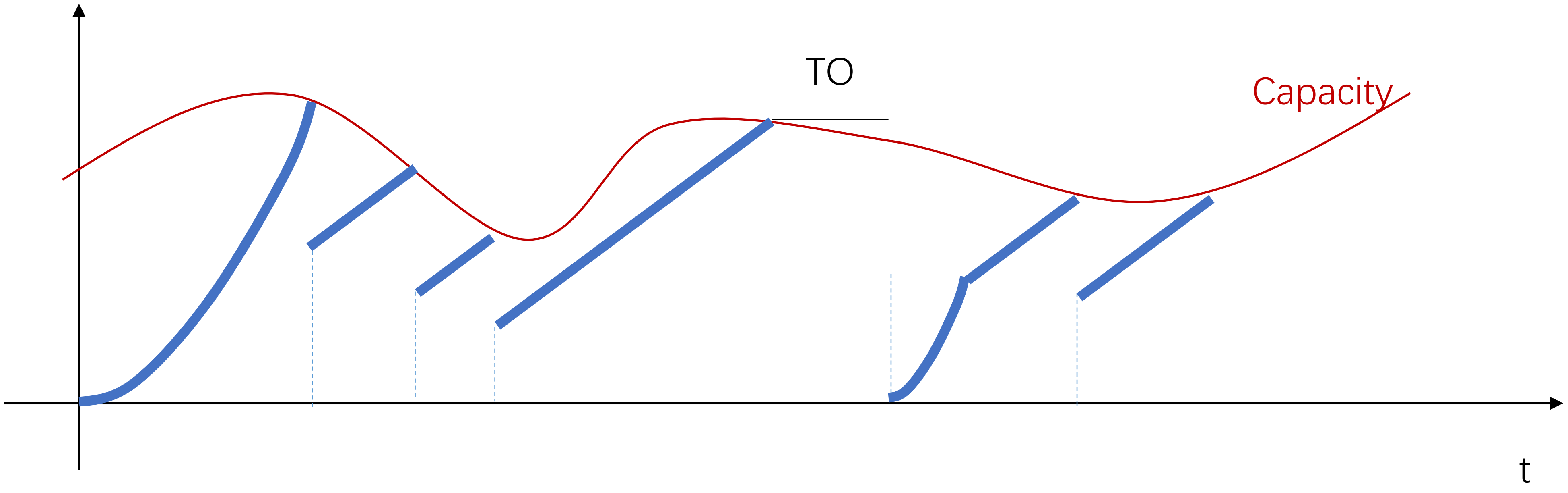


Fast Recovery



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

Fast Recovery



TCP Reno

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

Demo

- http://leeo1116.github.io/TCP_congestion_control/TCP.html

Reference

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