

CS 356: Computer Network Architectures
Lecture 20: Congestion Avoidance
Chap. 6.4 and related papers

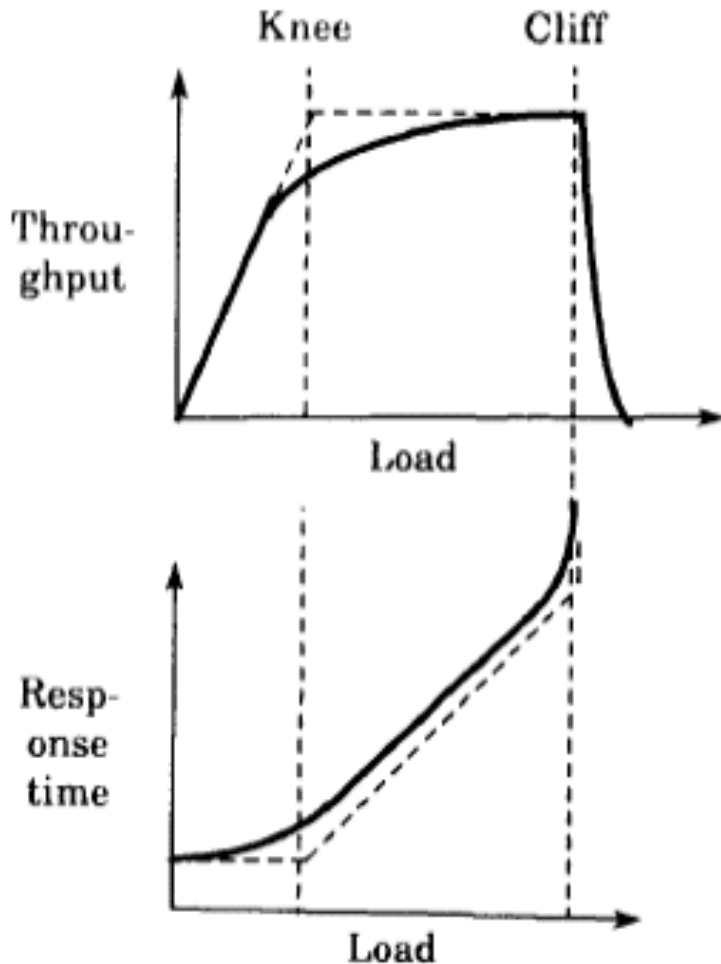
Xiaowei Yang
xwy@cs.duke.edu

TCP Congestion Control

History

- The original TCP/IP design did not include congestion control and avoidance
 - Receiver uses advertised window to do flow control
 - No exponential backoff after a timeout
- It led to congestion collapse in October 1986
 - The [NSFnet](#) phase-I backbone dropped three orders of magnitude from its capacity of 32 kbit/s to 40 bit/s, and continued until end nodes started implementing Van Jacobson's [congestion control](#) between 1987 and 1988.
 - TCP retransmits too early, wasting the network's bandwidth to retransmit packets already in transit and reducing useful throughput (goodput)

Design Goals



- Congestion avoidance: making the system operate around the knee to obtain low latency and high throughput
- Congestion control: making the system operate left to the cliff to avoid congestion collapse

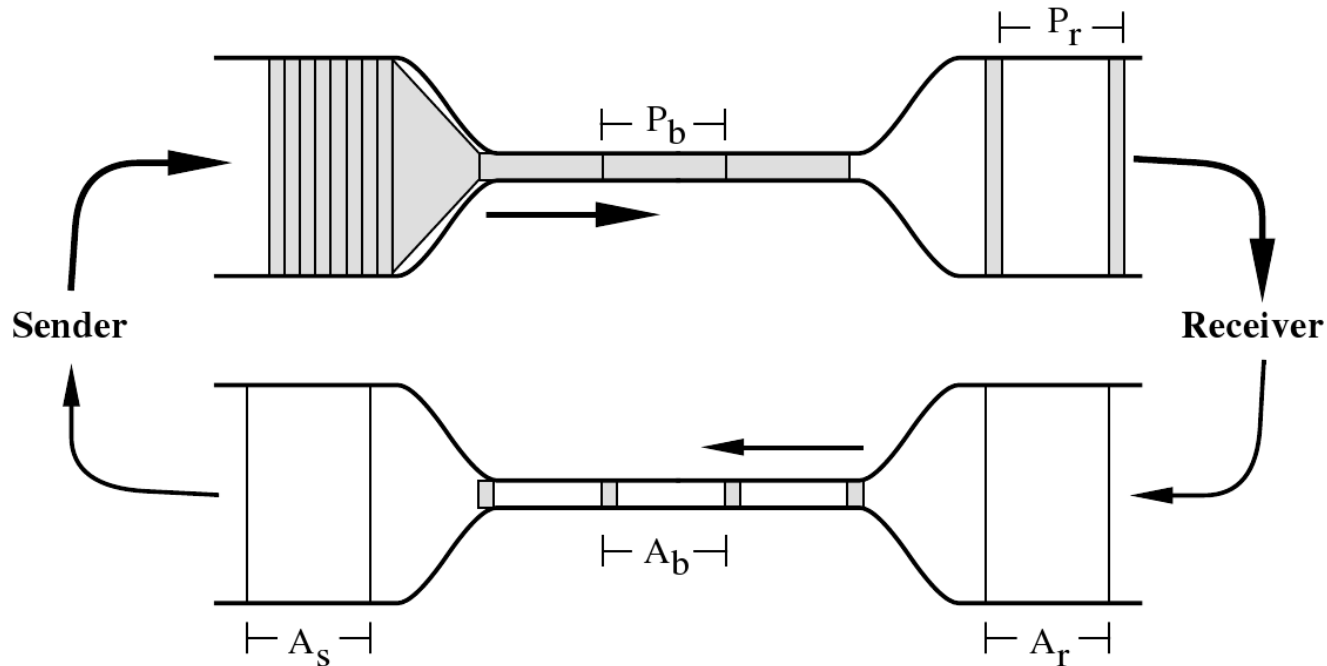
Key Improvements

- RTT variance estimate
 - Old design: $RTT_{n+1} = \alpha RTT + (1 - \alpha) RTT_n$
 - $RTO = \beta RTT_{n+1}$
- Exponential backoff
- Slow-start
- Dynamic window sizing
- Fast retransmit

Challenge

- Send at the “right” speed
 - Fast enough to keep the pipe full
 - But not to overrun the “pipe”
 - Share nicely with other senders

Key insight: packet conservation principle and self-clocking



- When pipe is full, the speed of ACK returns equals to the speed new packets should be injected into the network

Solution: Dynamic window sizing

- Sending speed: SWS / RTT
- → Adjusting SWS based on available bandwidth
- The sender has two *internal* parameters:
 - Congestion Window (**cwnd**)
 - Slow-start threshold Value (**ssthresh**)
- SWS is set to the minimum of (cwnd, receiver advertised win)

Two Modes of Congestion Control

1. Probing for the available bandwidth
 - **slow start** ($\text{cwnd} < \text{ssthresh}$)
2. Avoid overloading the network
 - **congestion avoidance** ($\text{cwnd} \geq \text{ssthresh}$)

Slow Start

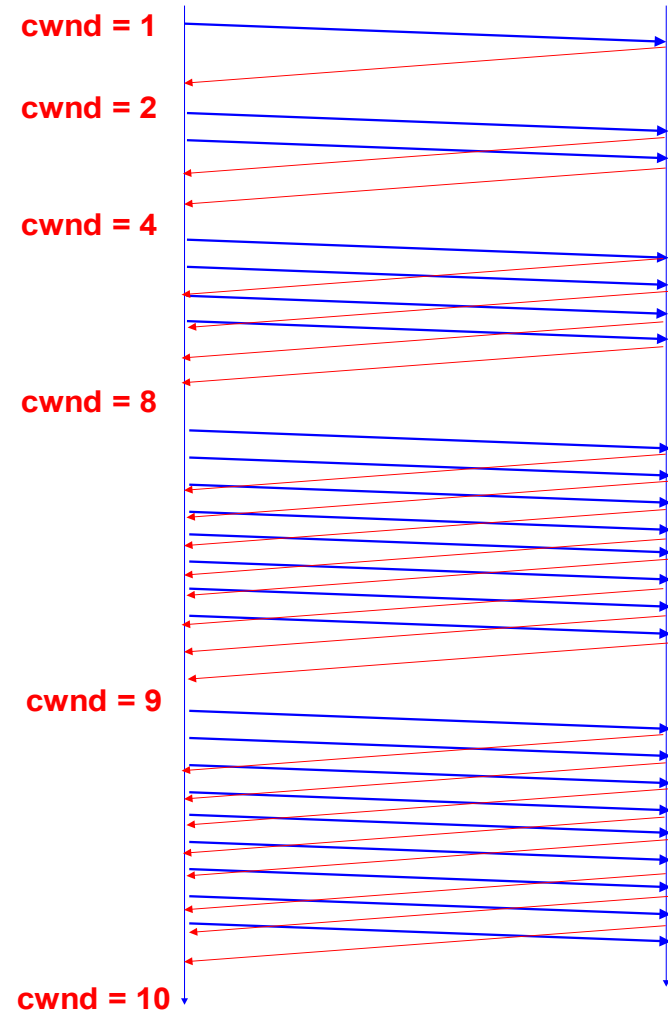
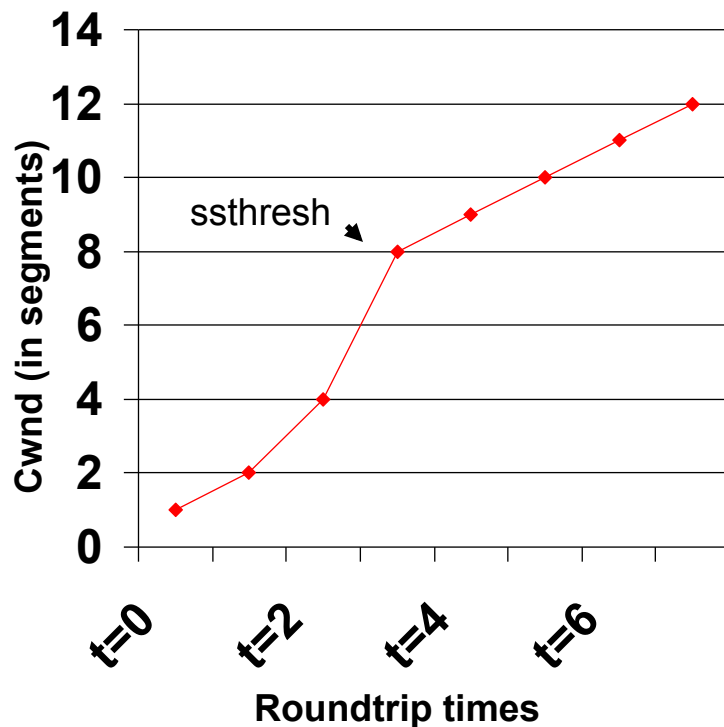
- Initial value: **Set $\text{cwnd} = 1 \text{ MSS}$**
 - Modern TCP implementation may set initial cwnd to larger than 1 (e.g. 2, 4, or 10)
- When receiving an ACK, $\text{cwnd} += 1 \text{ MSS}$
 - If an ACK acknowledges two segments, cwnd is still increased by only 1 segment.
 - Even if ACK acknowledges a segment that is smaller than MSS bytes long, cwnd is increased by 1.
 - Question: how can you accelerate your TCP download?

Congestion Avoidance

- If **cwnd** \geq **ssthresh** then each time an ACK is received, increment cwnd as follows:
 - $\text{cwnd} += \text{MSS} * (\text{MSS} / \text{cwnd})$ (cwnd measured in bytes)
- So *cwnd* is increased by one MSS only if all *cwnd*/MSS segments have been acknowledged.

Example of Slow Start/Congestion Avoidance

Assume *ssthresh* = 8 MSS



Congestion detection

- What would happen if a sender keeps increasing **cwnd**?
 - Packet loss
- TCP uses packet loss as a congestion signal
- Loss detection
 1. Receipt of a duplicate ACK (cumulative ACK)
 2. Timeout of a retransmission timer

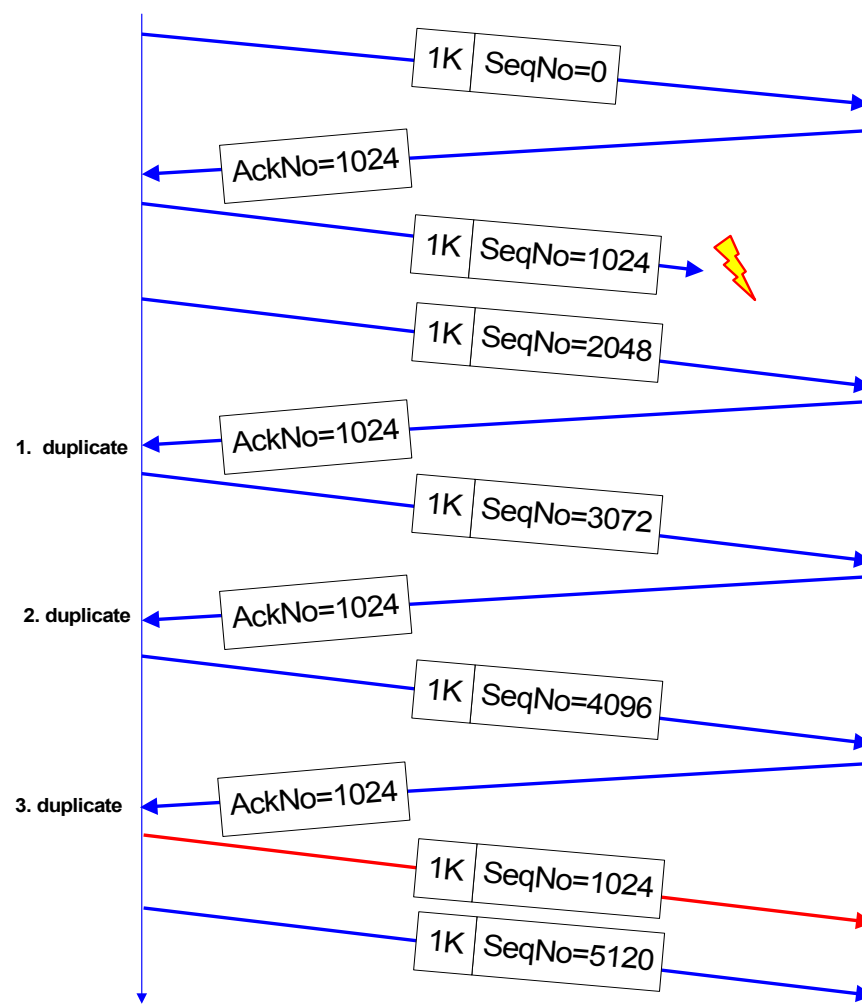
Reaction to Congestion

- Reduce cwnd
- Timeout: severe congestion
 - cwnd is reset to one MSS:
$$\text{cwnd} = 1 \text{ MSS}$$
 - ssthresh is set to half of the current size of the congestion window:
$$\text{ssthresh} = \text{cwnd} / 2$$
 - entering slow-start

Reaction to Congestion

- Duplicate ACKs: not so congested (why?)
- Fast retransmit
 - Three duplicate ACKs indicate a packet loss
 - Retransmit without timeout

Duplicate ACK example



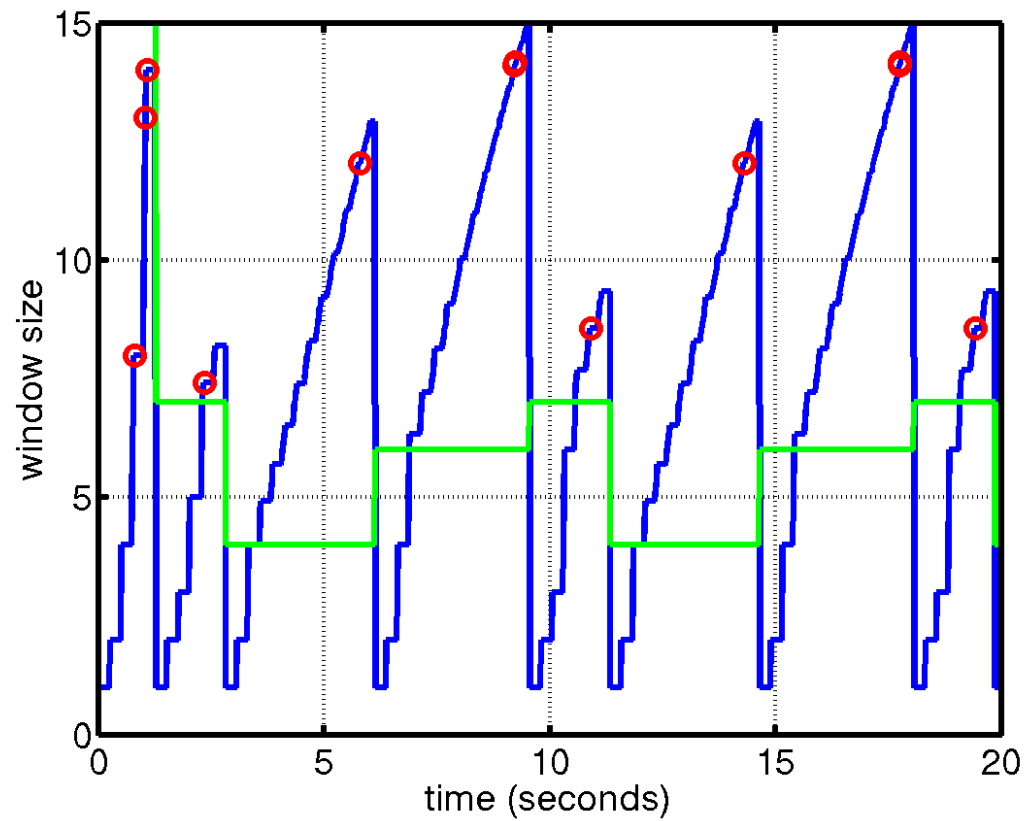
Reaction to congestion: Fast Recovery

- Avoiding slow start
 - $ssthresh = cwnd/2$
 - $cwnd = cwnd + 3MSS$
 - Increase $cwnd$ by one MSS for each additional duplicate ACK
- When ACK arrives that acknowledges “new data,” set:
 - $cwnd = ssthresh$
 - enter congestion avoidance

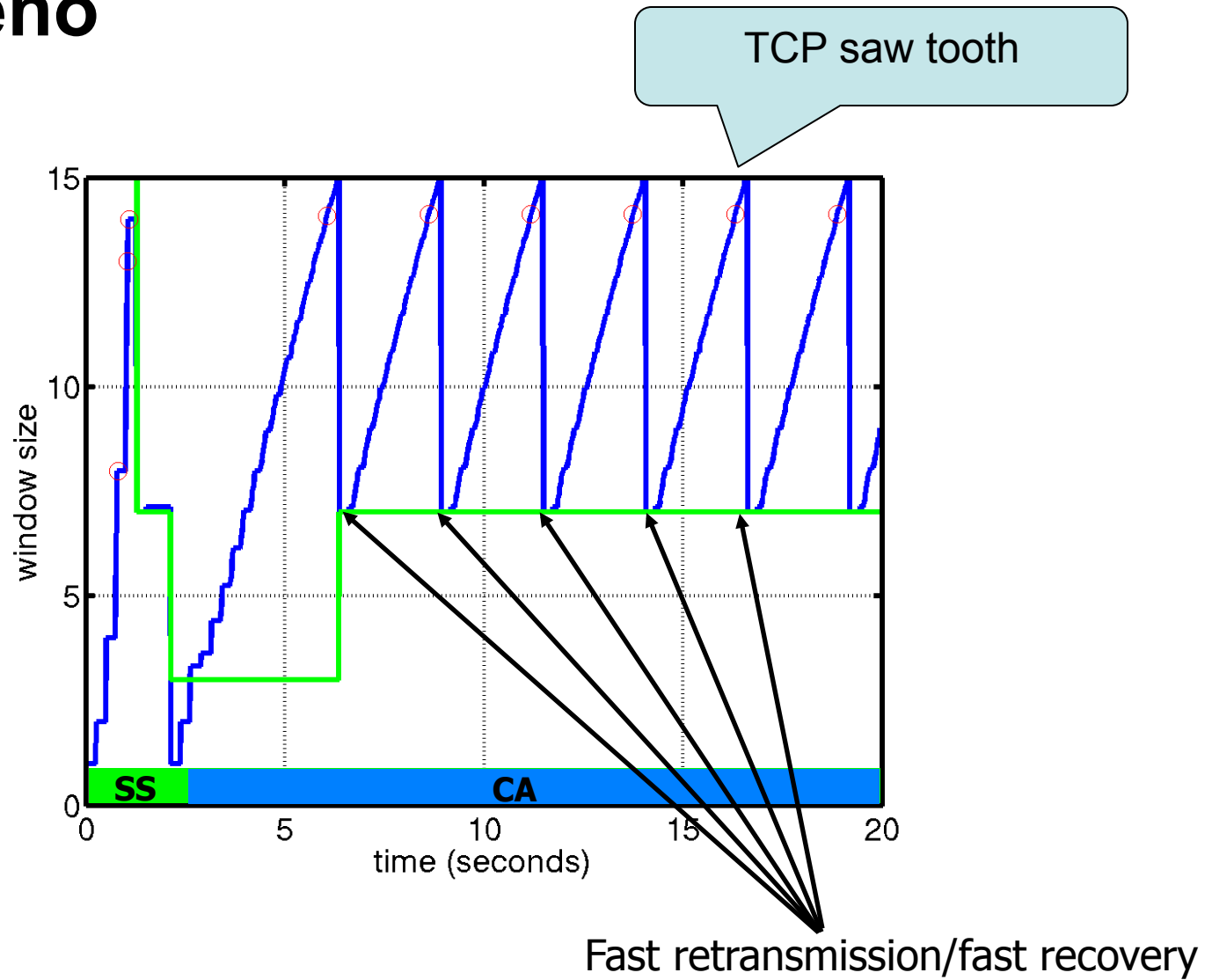
Flavors of TCP Congestion Control

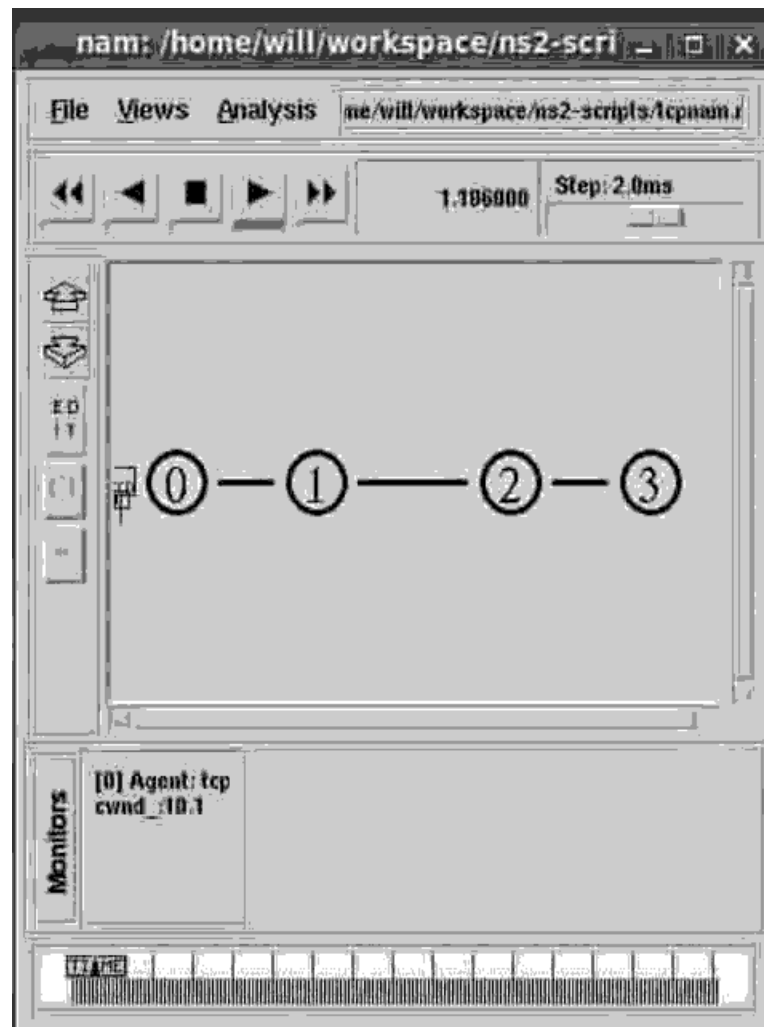
- **TCP Tahoe** (1988, FreeBSD 4.3 Tahoe)
 - Slow Start
 - Congestion Avoidance
 - Fast Retransmit
- **TCP Reno** (1990, FreeBSD 4.3 Reno)
 - Fast Recovery
 - Modern TCP implementation
- **New Reno** (1996)
- **SACK** (1996)

TCP Tahoe



TCP Reno



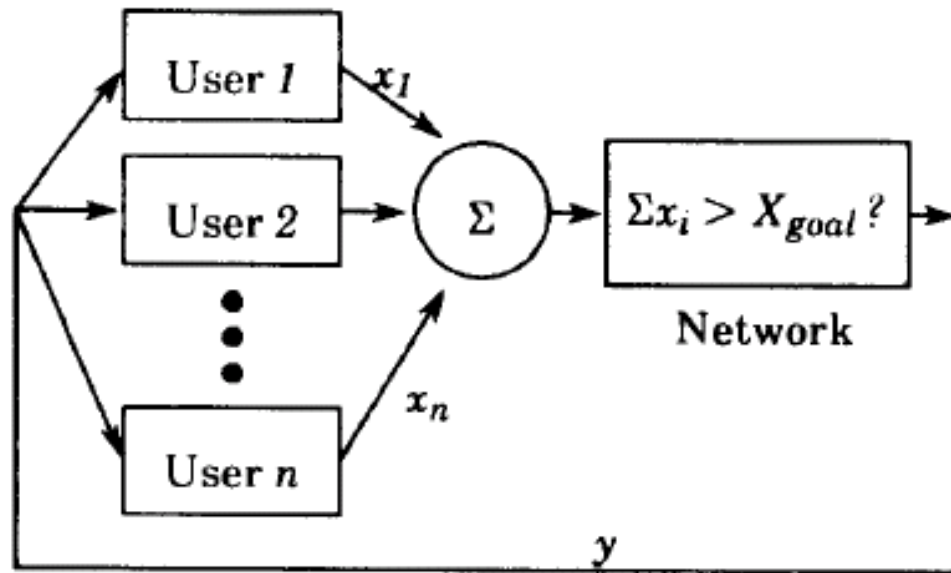


TCP summary

- Connection management
- Flow control
- When to transmit a segment
- Adaptive retransmission
- TCP options
- Modern extensions
- Congestion Control

Theory: why does it work?

Why does it work? [Chiu-Jain]



- A feedback control system
- The network uses feedback y to adjust users' load $\sum x_i$

Goals of Congestion Avoidance

- Efficiency: the closeness of the total load on the resource to its knee
- Fairness:

$$F(\mathbf{x}) = \frac{(\sum x_i)^2}{n(\sum x_i^2)}.$$

- When all x_i 's are equal, $F(\mathbf{x}) = 1$
 - When all x_i 's are zero but $x_j = 1$, $F(\mathbf{x}) = 1/n$
- Distributedness
 - A centralized scheme requires complete knowledge of the state of the system
 - Convergence
 - The system approaches the goal state from any starting state

Metrics to measure convergence

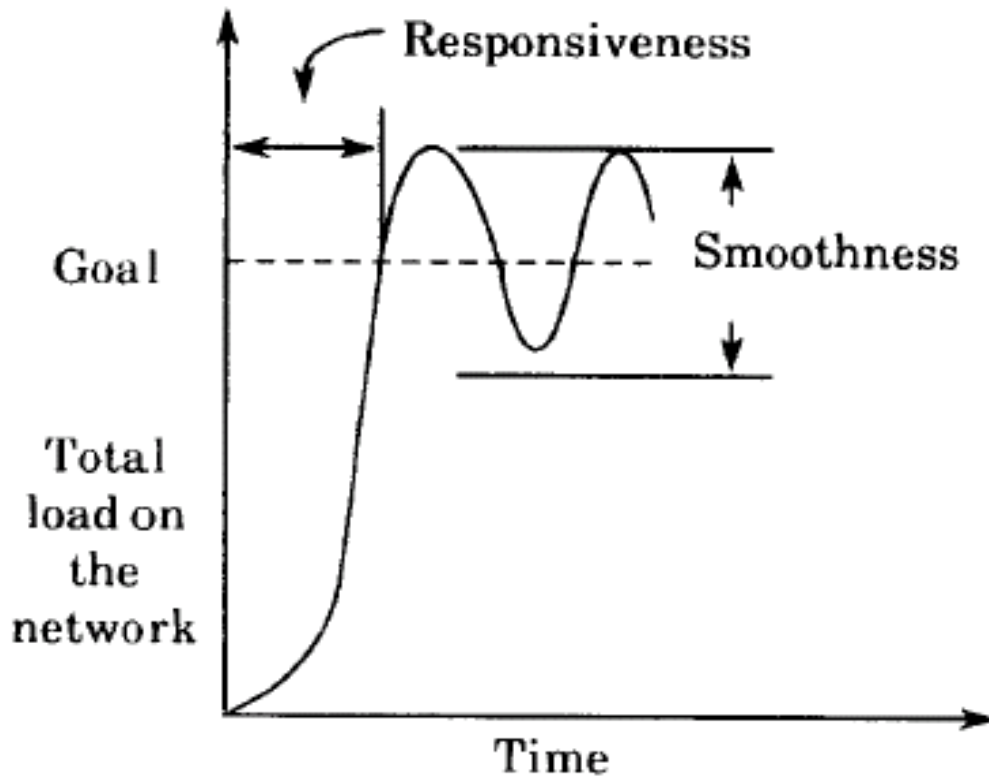


Fig. 3. Responsiveness and smoothness.

- Responsiveness
- Smoothness

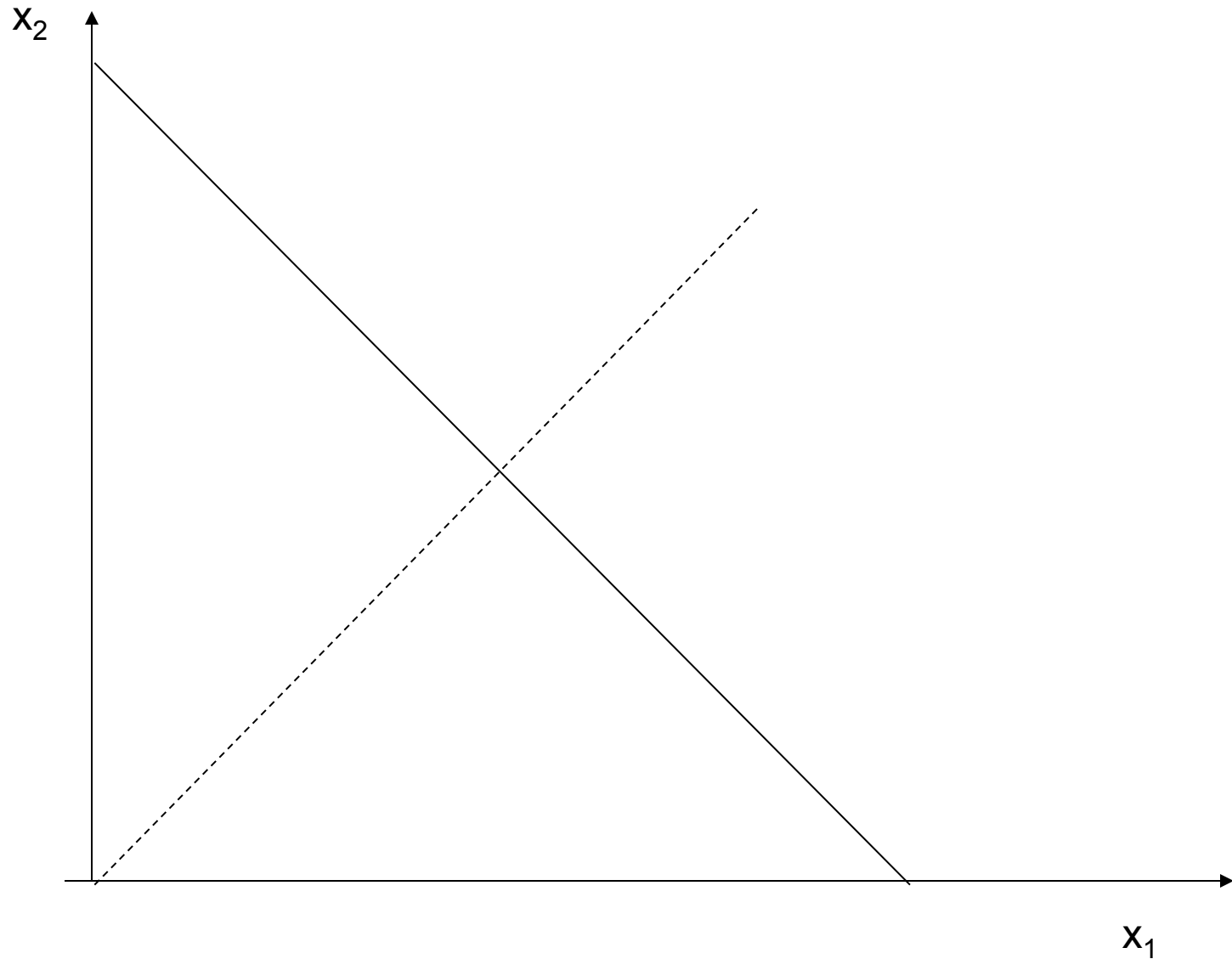
Model the system as a linear control system

$$x_i(t+1)$$

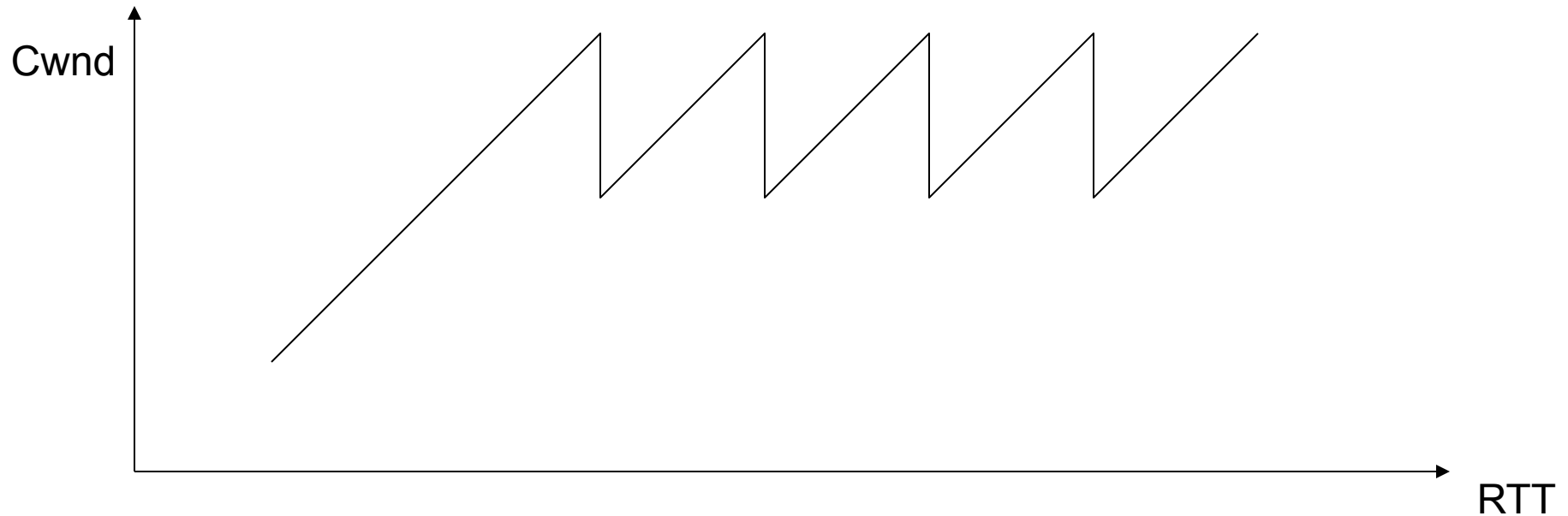
$$= \begin{cases} a_I + b_I x_i(t) & \text{if } y(t) = 0 \Rightarrow \text{Increase,} \\ a_D + b_D x_i(t) & \text{if } y(t) = 1 \Rightarrow \text{Decrease.} \end{cases}$$

- Four sample types of controls
- AIAD, AIMD, MIAD, MIMD
 - A: additive
 - M: multiplicative
 - I: increase
 - D: decrease

Phase plot

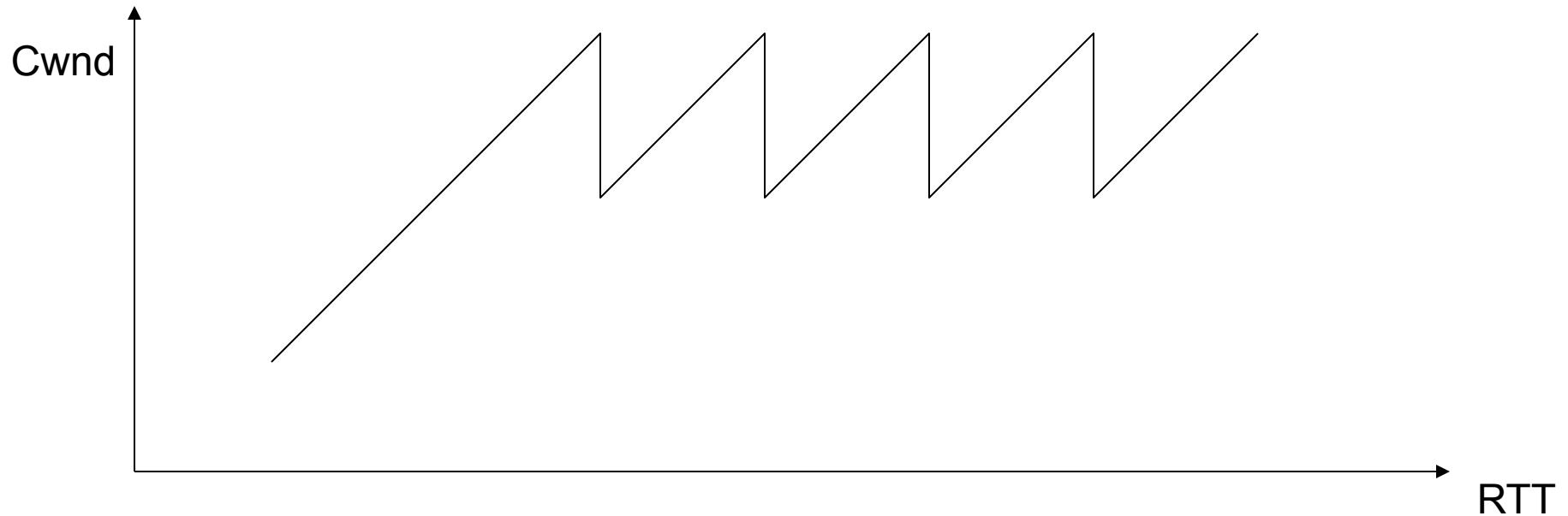


The Sawtooth behavior of TCP



- For every ACK received
 - $Cwnd += 1/cwnd * MSS$
- For every packet lost
 - $Cwnd /= 2$

TCP congestion control is AIMD



- Problems:
 - Each source has to probe for its bandwidth
 - Congestion occurs first before TCP backs off
 - Unfair: long RTT flows obtain smaller bandwidth shares

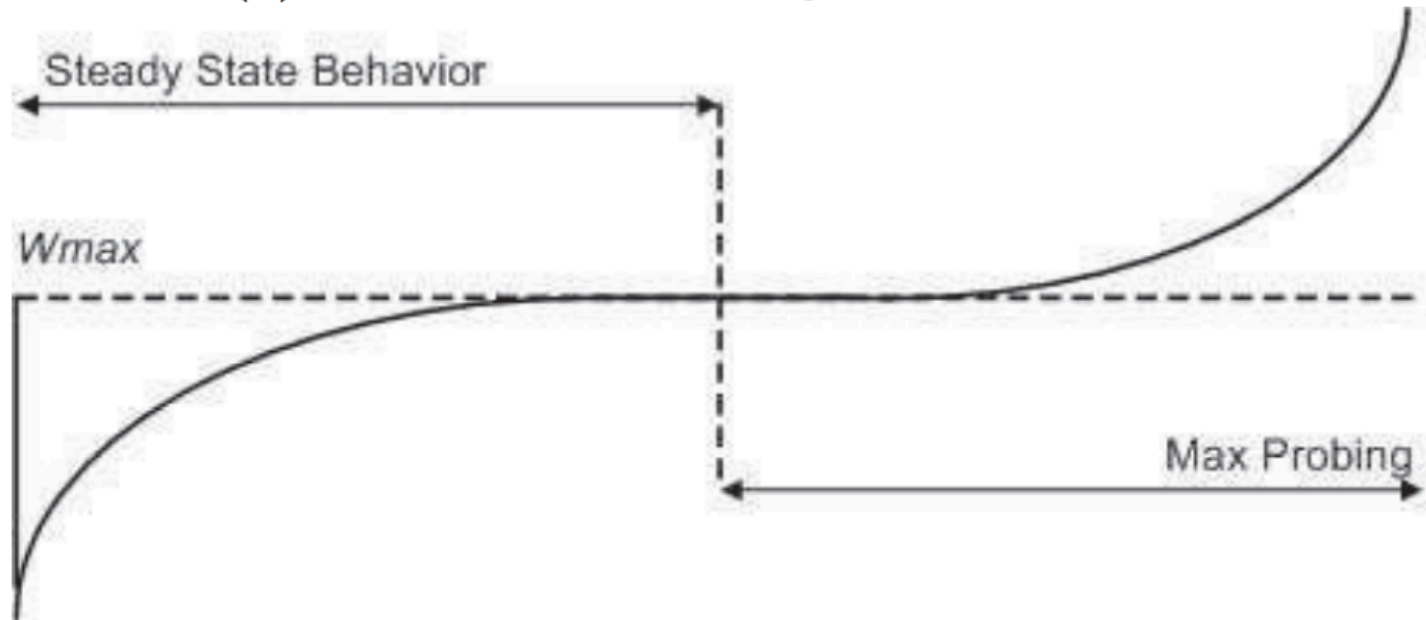
Macroscopic behavior of TCP

- Throughput is inversely proportional to RTT:

$$\frac{\sqrt{1.5} \cdot MSS}{RTT \cdot \sqrt{p}}$$

- In a steady state, total packets sent in one sawtooth cycle:
 - $S = w + (w+1) + \dots (w+w) = 3/2 w^2$
- the maximum window size is determined by the loss rate
 - $1/S = p$
 - $w = \frac{1}{\sqrt{1.5p}}$
- The length of one cycle: $w * RTT$
- Average throughput: $3/2 w * MSS / RTT$

TCP Cubic



(b) CUBIC window growth function.

$$W(t) = C(t - K)^3 + W_{max} \quad K = \sqrt[3]{\frac{W_{max}\beta}{C}}$$

- CUBIC: a new TCP-friendly high-speed TCP variant by S. HaNorth, I. Rhee, and L. Xu
- Implemented in Linux kernel and Windows 10

Summary

- TCP congestion control
- Why it works?
- The macroscopic behavior of TCP
- TCP Cubic