CS120: Computer Networks

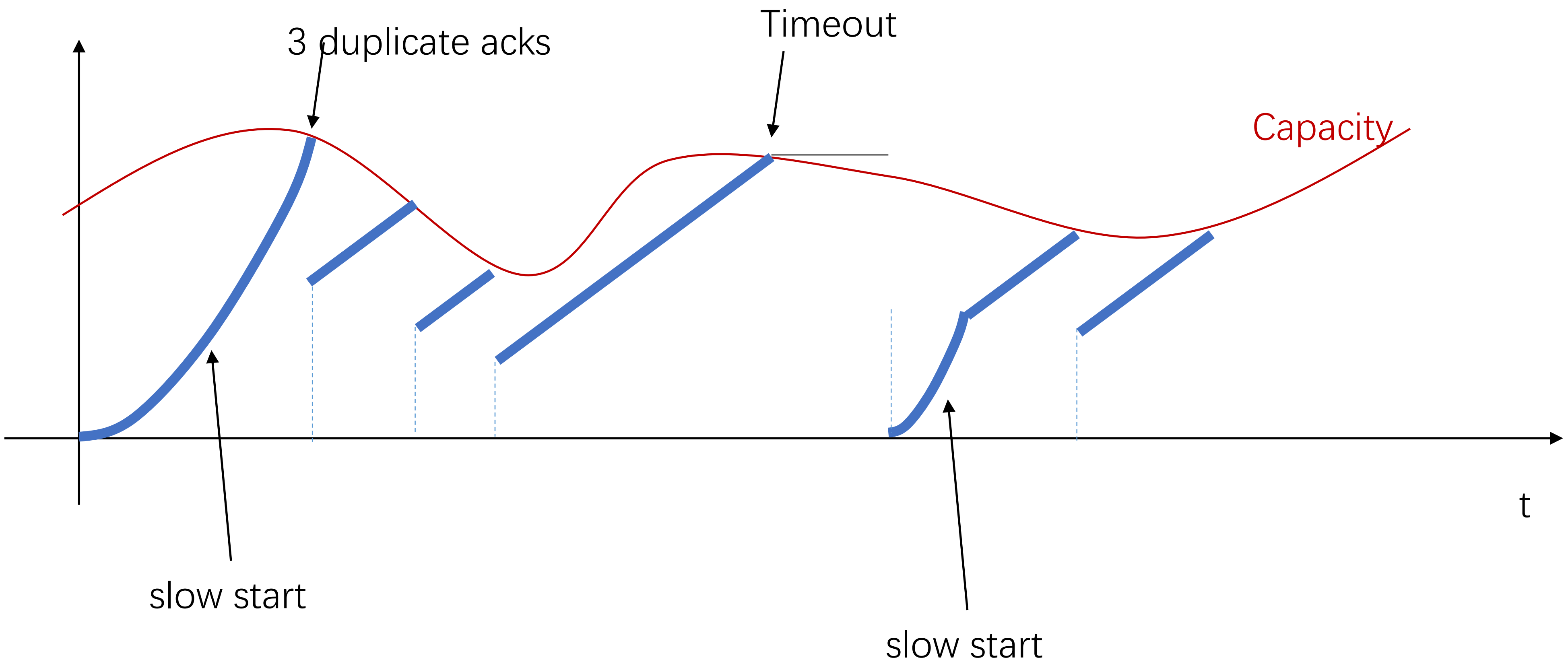
Lecture 19. Congestion Control 3

Zhice Yang

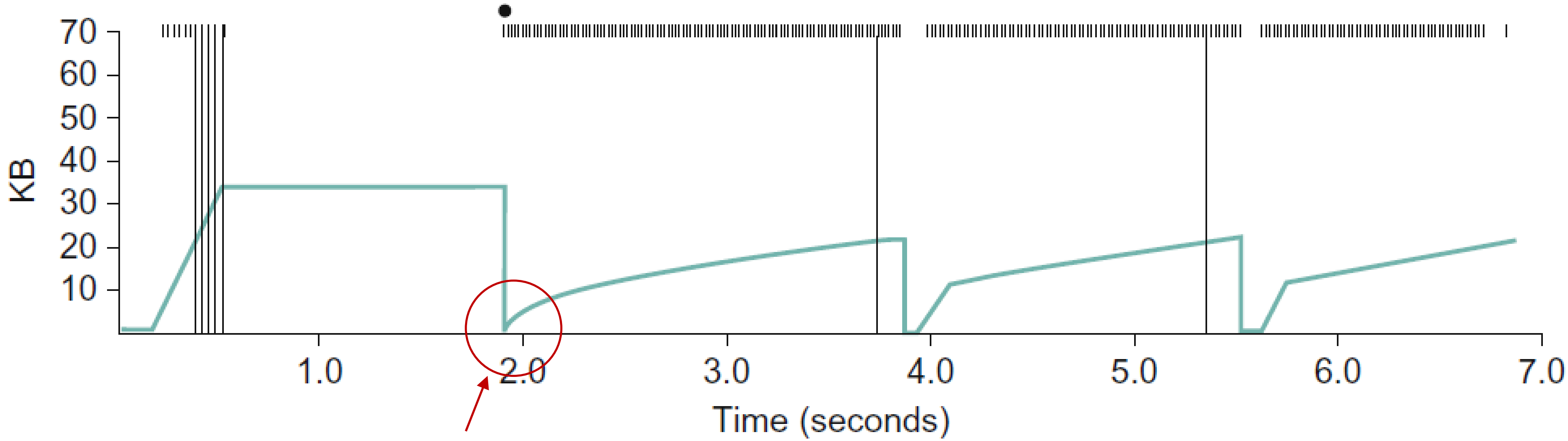
Congestion Control

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

TCP Reno



Figures in Textbook



Hard to Explain

No Fast Recovery in the Simulation, i.e., TCP Tahoe

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	

Demo

- Sliding Window code in TCP

  `/net/ipv4/`

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

- Change Sliding Window
  - Show current congestion control scheme

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

  - Show/change available congestion control scheme

  ```
  sysctl net.ipv4.tcp_available_congestion_control[=XX]
  ```

Congestion Control

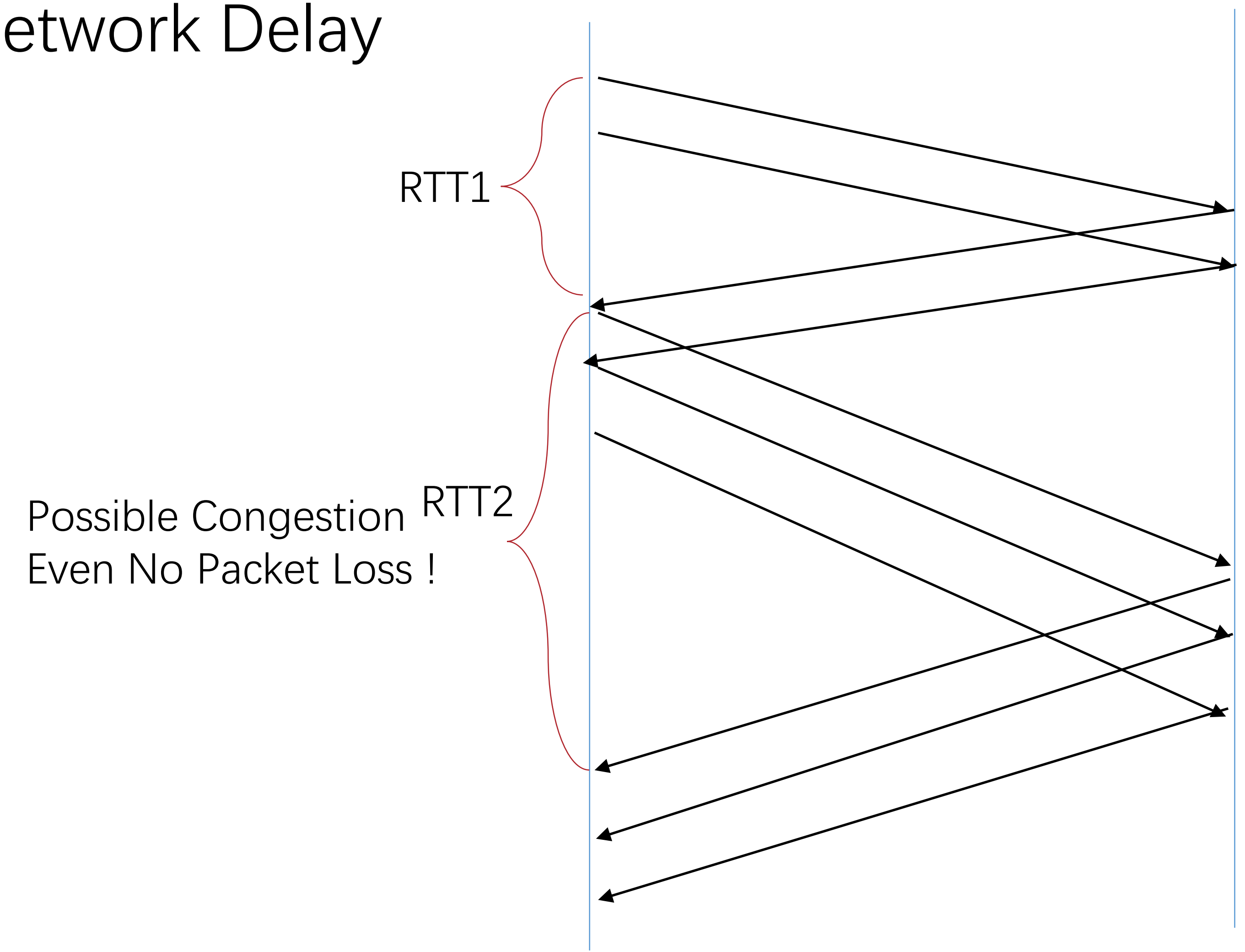
- Queuing
- Connection Control Methods
  - Congestion Control
  - ➢Congestion Avoidance
- QoS

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

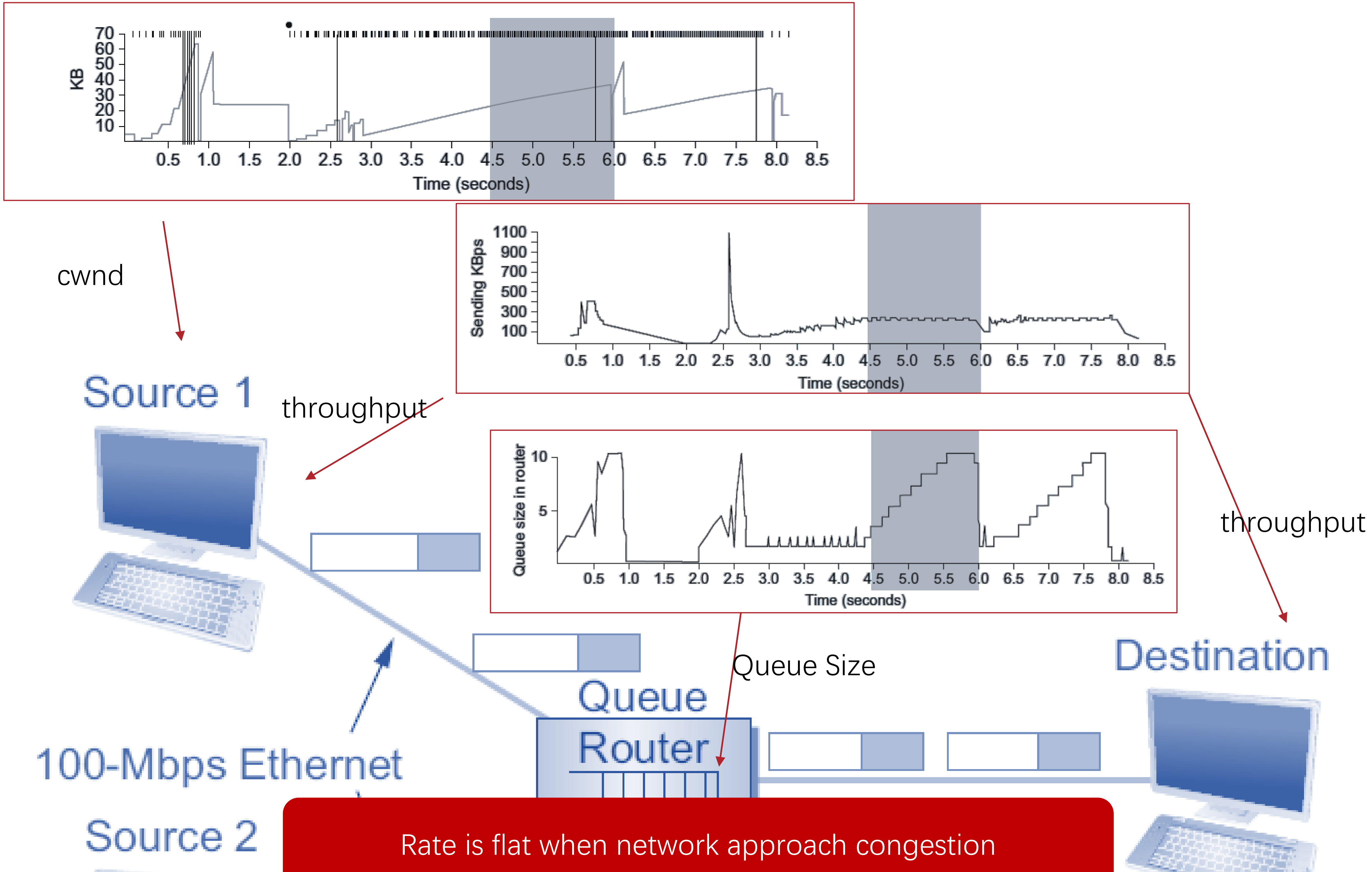
Avoid Congestion

- Feedback: Network Delay



Avoid Congestion

- Feedback: Network Delay
  - Possible Designs
    - Make decision based on current RTT and average RTT
    - Make decision based on current {RTT, Window} and average {RTT, Window}
    - Make decision based rate trend
    - etc…



Rate is flat when network approach congestion

TCP Vegas

- Idea: TCP source uses RTT as the sign to avoid network congestion
  - Compare RTT with BaseRTT
- Method:
  - Reduce rate when congestion is about to happen
    - If Actual RTT >> Base RTT
  - Recover rate soon after bandwidth is available
    - if Actual RTT > Base RTT
    - Keep certain pressure on network

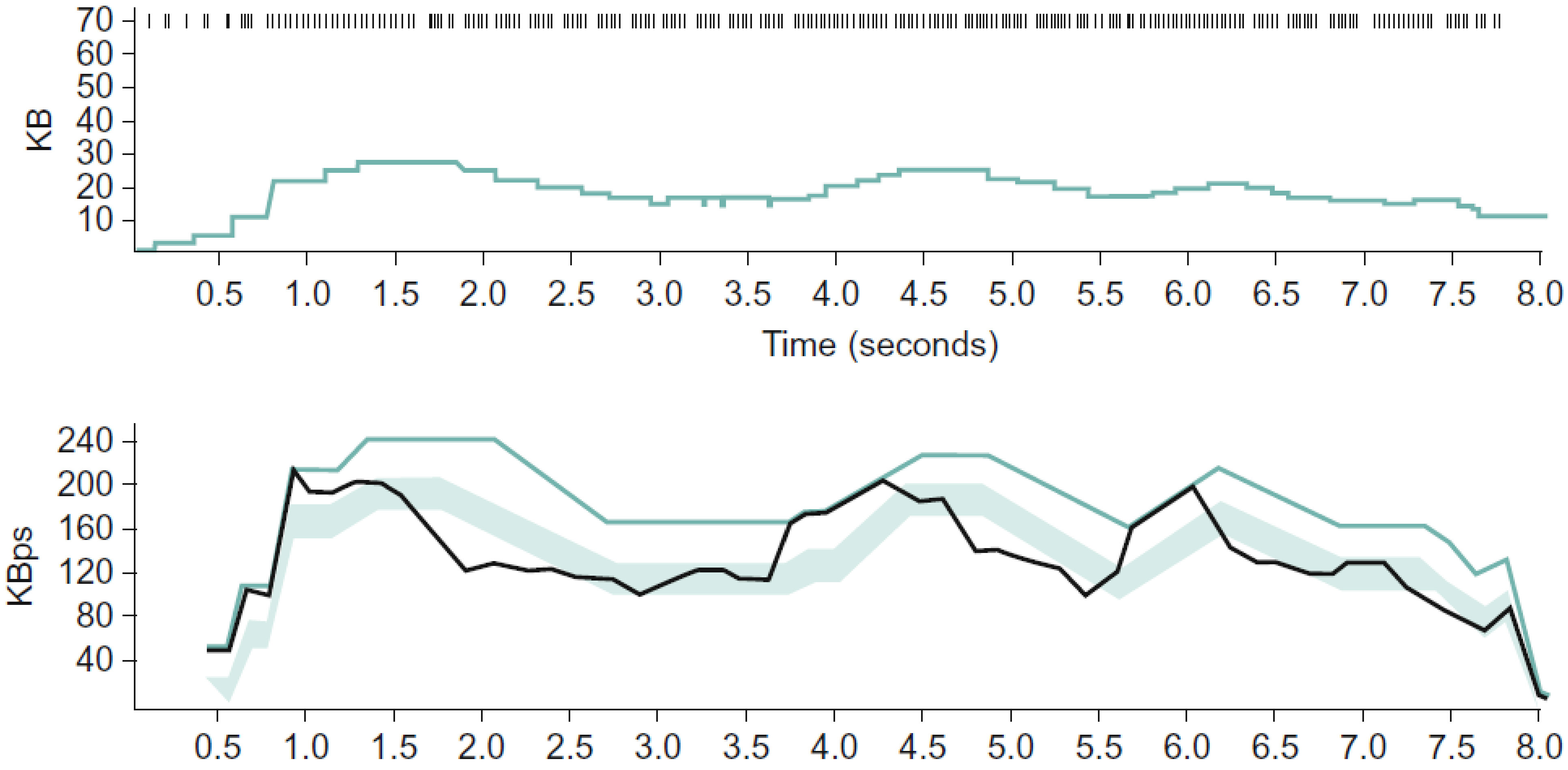
TCP Vegas Algorithm

- BaseRTT: the reference for increases in RTTs
  - The minimum RTT
    - If RTT < BaseRTT
      - BaseRTT = RTT
- ExpectRate:
  - CongestionWindow/BaseRTT
- ActualRate:
  - ActualRTT: according to timestamps
  - ActualRate=CongestionWindow/ActualRTT

TCP Vegas Algorithm

- Source compares ActualRate with ExpectRate
    - if ExpectRate - ActualRate < α
        - cwnd++
    - if α <ExpectRate - ActualRate < β
        - cwnd = cwnd
    - if β <ExpectRate - ActualRate
        - cwnd --

TCP Vegas



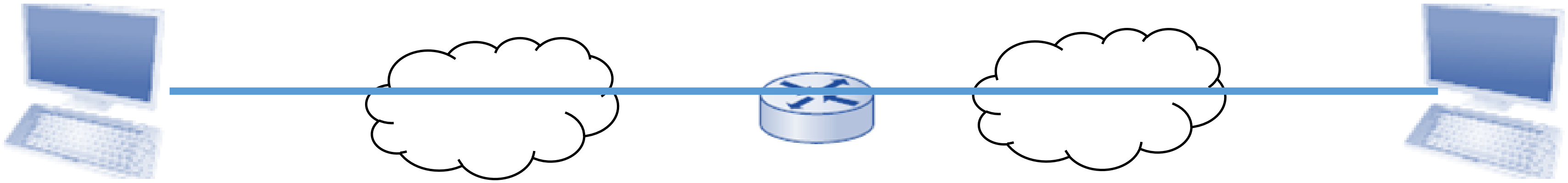
70
60
50
40
30
20
10
KB
0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
Time (seconds)
240
200
160
120
80
40
KBps
0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0

Congestion Control

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

Congestion Control and Resource Allocation

- Two Sides of the Same Coin
  - Control congestion if (and when) is occurs: reactive
  - Pre-allocate resources so at to avoid congestion: proactive
- Resources in Network
  - Bandwidth
  - Router Queue Buffer
- Two Places of Implementation
  - Hosts at the edges of the network (transport protocol)
  - Routers inside the network (queuing discipline)



Congestion Avoidance with Routers

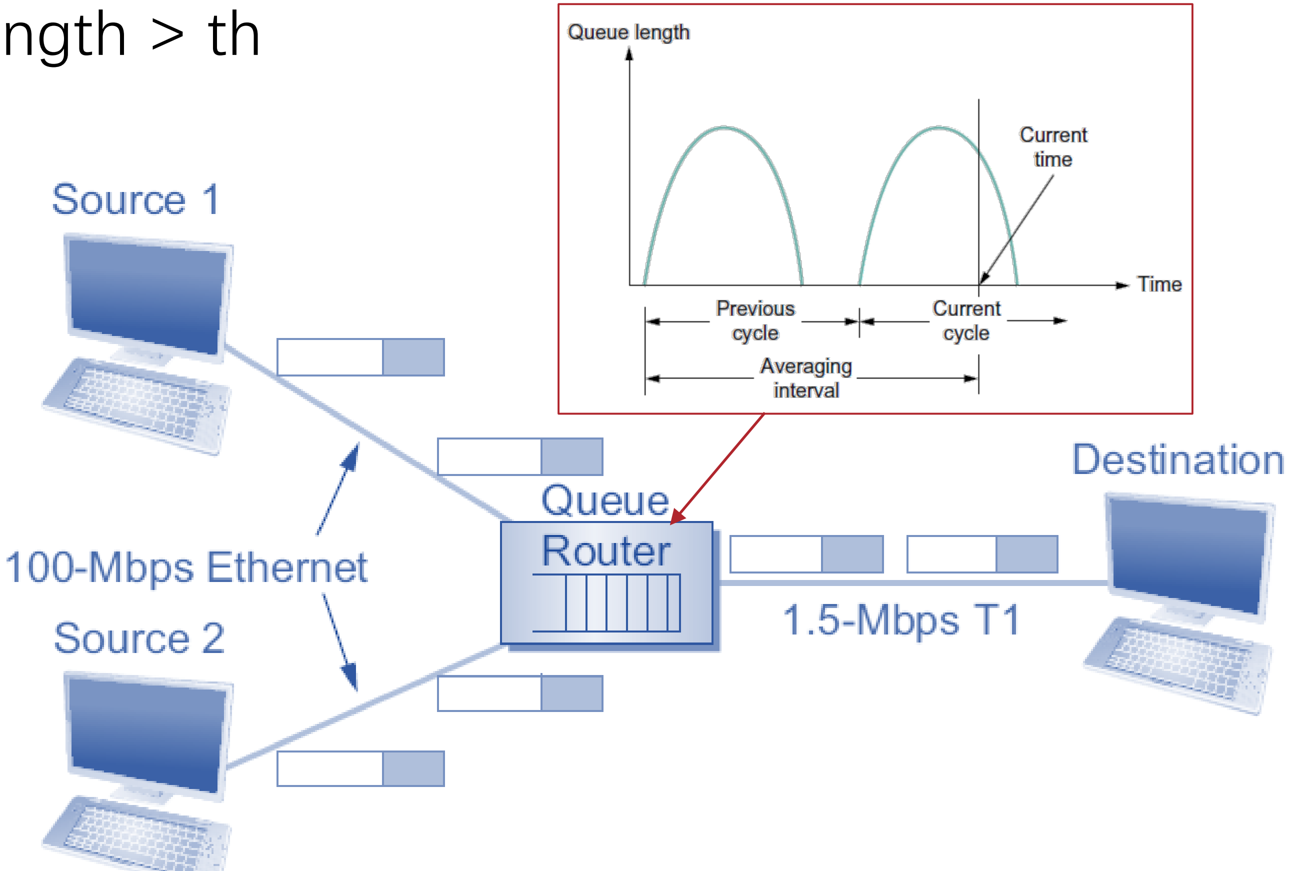
- DECbit
- Random Early Detection (RED)

DECbit

- Developed for the Digital Network Architecture (DNA)
  - Before TCP/IP was “standardized”
- Idea: let routers explicitly indicate congestion
- Approach:
  - Router set congestion bit in passing packets if there is congestion
  - Destination echoes bit back to source through acks
  - Source adjusts cwnd according to congestion bit

DECbit

- Routers determine congestion
  - Monitor average queue length over last busy+idle cycle
  - If average queue length > th
    - set congestion bit

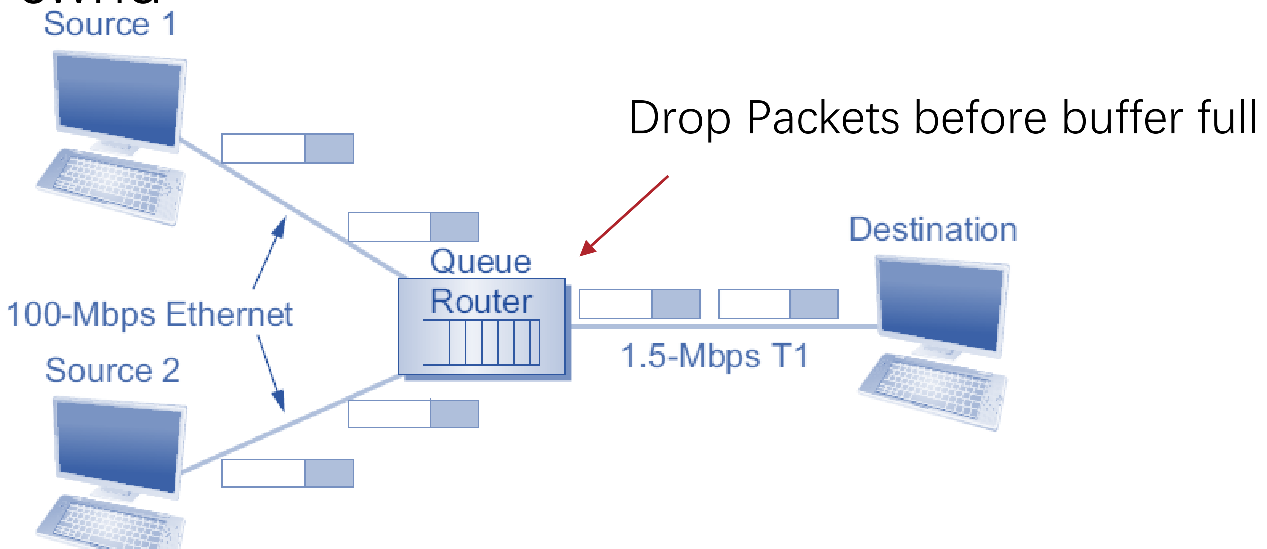


DECbit

- Source reacts to congestion bit
  - If < 50% of last window's packets had bit set
    - cwnd++
  - If > 50% of last window's packets had bit set
    - cwnd*0.875

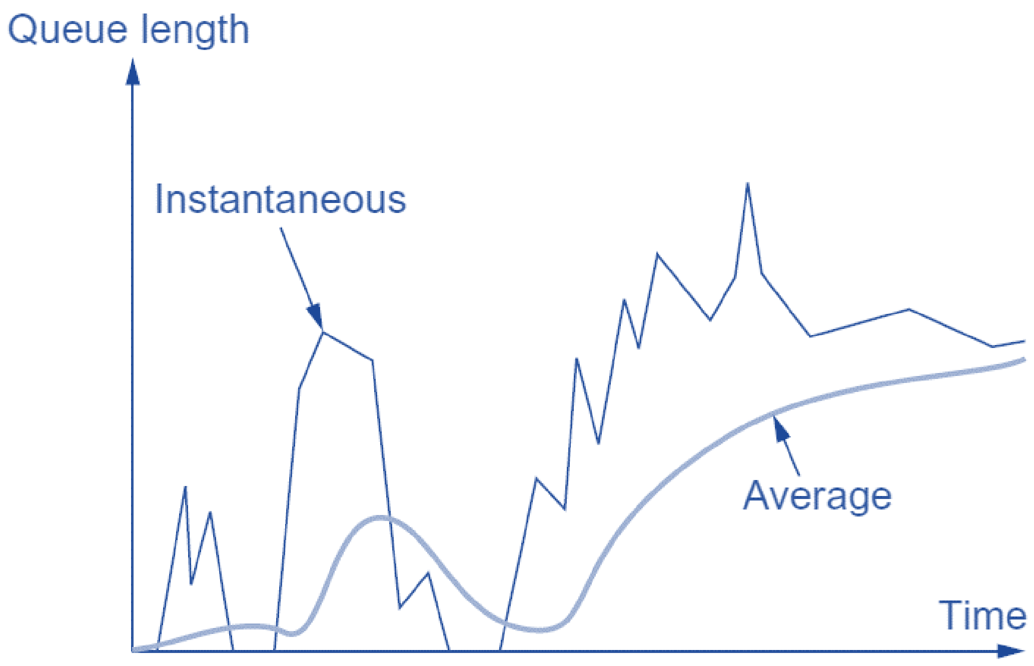
Random Early Detection (RED)

- Developed for TCP/IP, offload a part of congestion control function to routers
- Idea: let routers implicitly indicate congestion
  - Router notices that the queue is getting backlogged
  - Router randomly drops packets to signal congestion
  - Source adjust cwnd



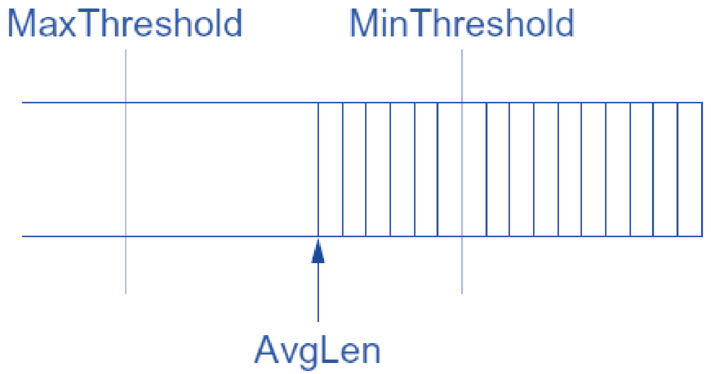
RED Algorithm

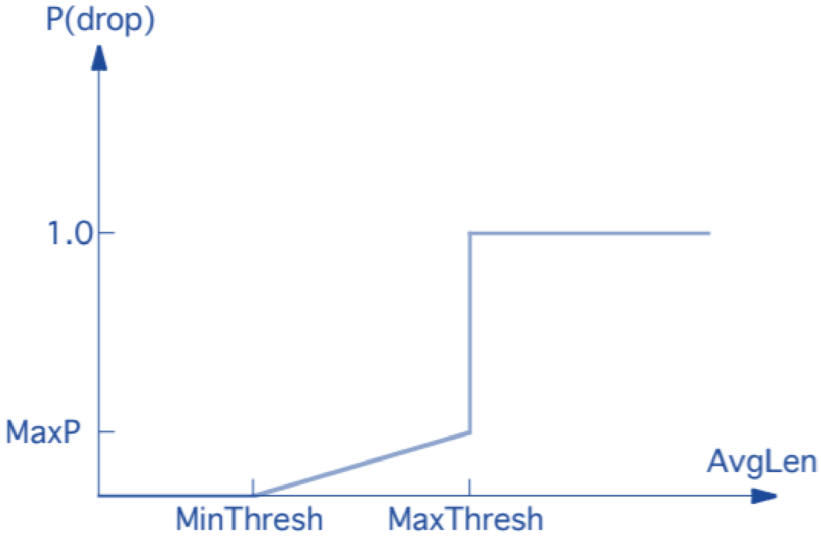
- Compute Average Queue Length
  - Moving average
    - AvgLen = (1 - Weight) * AvgLen + Weight * SampleLen



RED Algorithm

- Two queue length thresholds
  - if AvgLen <= MinThreshold
    - Enqueue the packet
  - if MinThreshold < AvgLen < MaxThreshold
    - Drop arriving packet with probability P
  - if ManThreshold <= AvgLen then drop arriving packet





RED Algorithm

- Computing probability P
  - TempP = MaxP * (AvgLen - MinThreshold)/(MaxThreshold - MinThreshold)
  - P = TempP/(1 - count * TempP)
    - Count: number of continuously queued packets without drop
- Why?
  - Drops are spaced out in time
    - Count == 0 => P=TempP
    - Count is large => P=1

Reference

- Textbook 6.4