




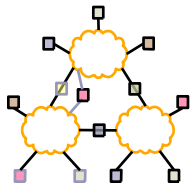
# CE693: Adv. Computer Networking

## L-4 TCP

*Acknowledgments: Lecture slides are from the graduate level Computer Networks course taught by Srinivasan Seshan at CMU. When slides are obtained from other sources, a reference will be noted on the bottom of that slide. A full list of references is provided on the last slide.*

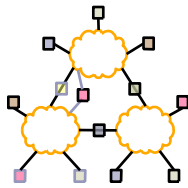


# TCP Congestion Control



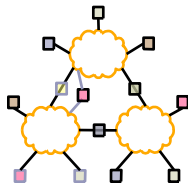
- Congestion Control
- RED
- Assigned Reading
  - [FJ93] Random Early Detection Gateways for Congestion Avoidance
  - [TFRC] Equation-Based Congestion Control for Unicast Applications

# Introduction to TCP



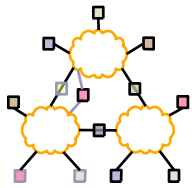
- Communication abstraction:
  - Reliable
  - Ordered
  - Point-to-point
  - Byte-stream
  - Full duplex
  - Flow and congestion controlled
- Protocol implemented entirely at the ends
  - Fate sharing
- Sliding window with cumulative acks
  - Ack field contains last in-order packet received
  - Duplicate acks sent when out-of-order packet received

# Key Things You Should Know Already



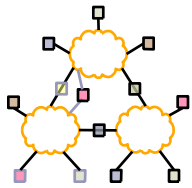
- Port numbers
- TCP/UDP checksum
- Sliding window flow control
  - Sequence numbers
- TCP connection setup
- TCP reliability
  - Timeout
  - Data-driven

# Overview



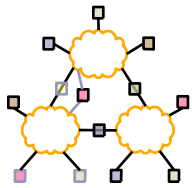
- TCP congestion control
- TFRC
- TCP and queues
- Queuing disciplines
- RED

# TCP Congestion Control



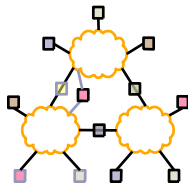
- Motivated by ARPANET congestion collapse
- Underlying design principle: packet conservation
  - At equilibrium, inject packet into network only when one is removed
  - Basis for stability of physical systems
- Why was this not working?
  - Connection doesn't reach equilibrium
  - Spurious retransmissions
  - Resource limitations prevent equilibrium

# TCP Congestion Control - Solutions



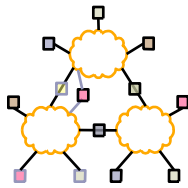
- Reaching equilibrium
  - Slow start
- Eliminates spurious retransmissions
  - Accurate RTO estimation
  - Fast retransmit
- Adapting to resource availability
  - Congestion avoidance

# TCP Congestion Control

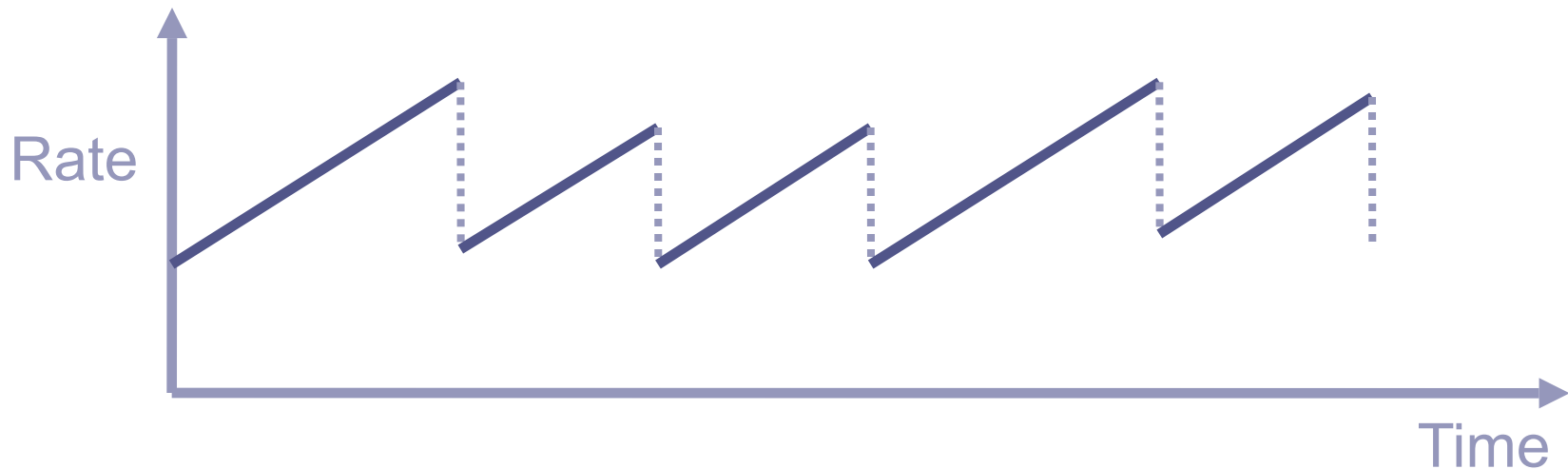


- Changes to TCP motivated by ARPANET congestion collapse
- Basic principles
  - AIMD
  - Packet conservation
  - Reaching steady state quickly
  - ACK clocking

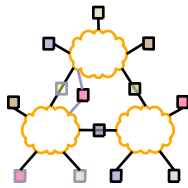
# AIMD



- Distributed, fair and efficient
- Packet loss is seen as sign of congestion and results in a multiplicative rate decrease
  - Factor of 2
- TCP periodically probes for available bandwidth by increasing its rate

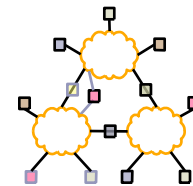


# Implementation Issue



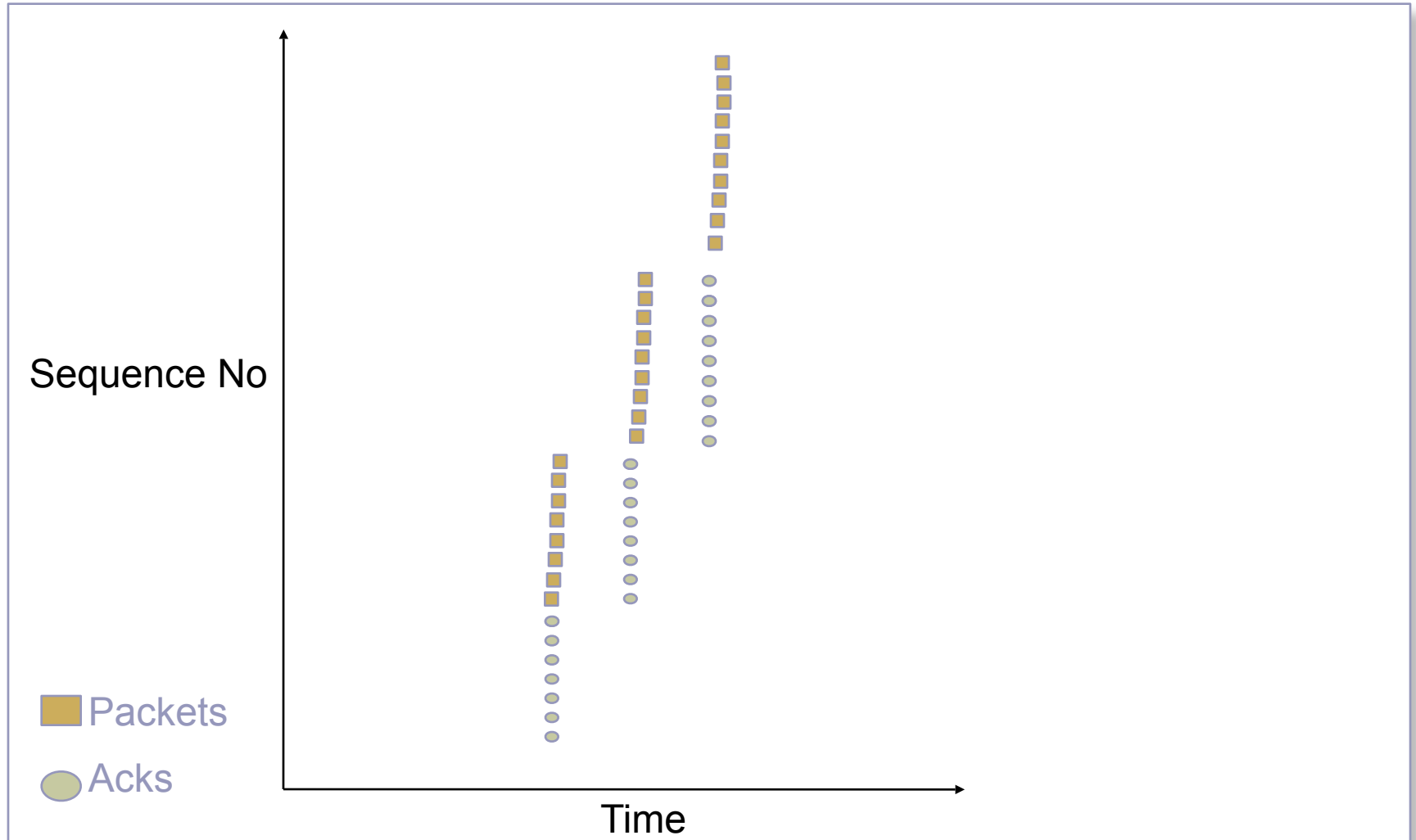
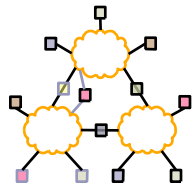
- Operating system timers are very coarse – how to pace packets out smoothly?
- Implemented using a congestion window that limits how much data can be in the network.
  - TCP also keeps track of how much data is in transit
- Data can only be sent when the amount of outstanding data is less than the congestion window.
  - The amount of outstanding data is increased on a “send” and decreased on “ack”
  - $(\text{last sent} - \text{last acked}) < \text{congestion window}$
- Window limited by both congestion and buffering
  - Sender's maximum window =  $\text{Min}(\text{advertised window}, \text{cwnd})$

# Congestion Avoidance

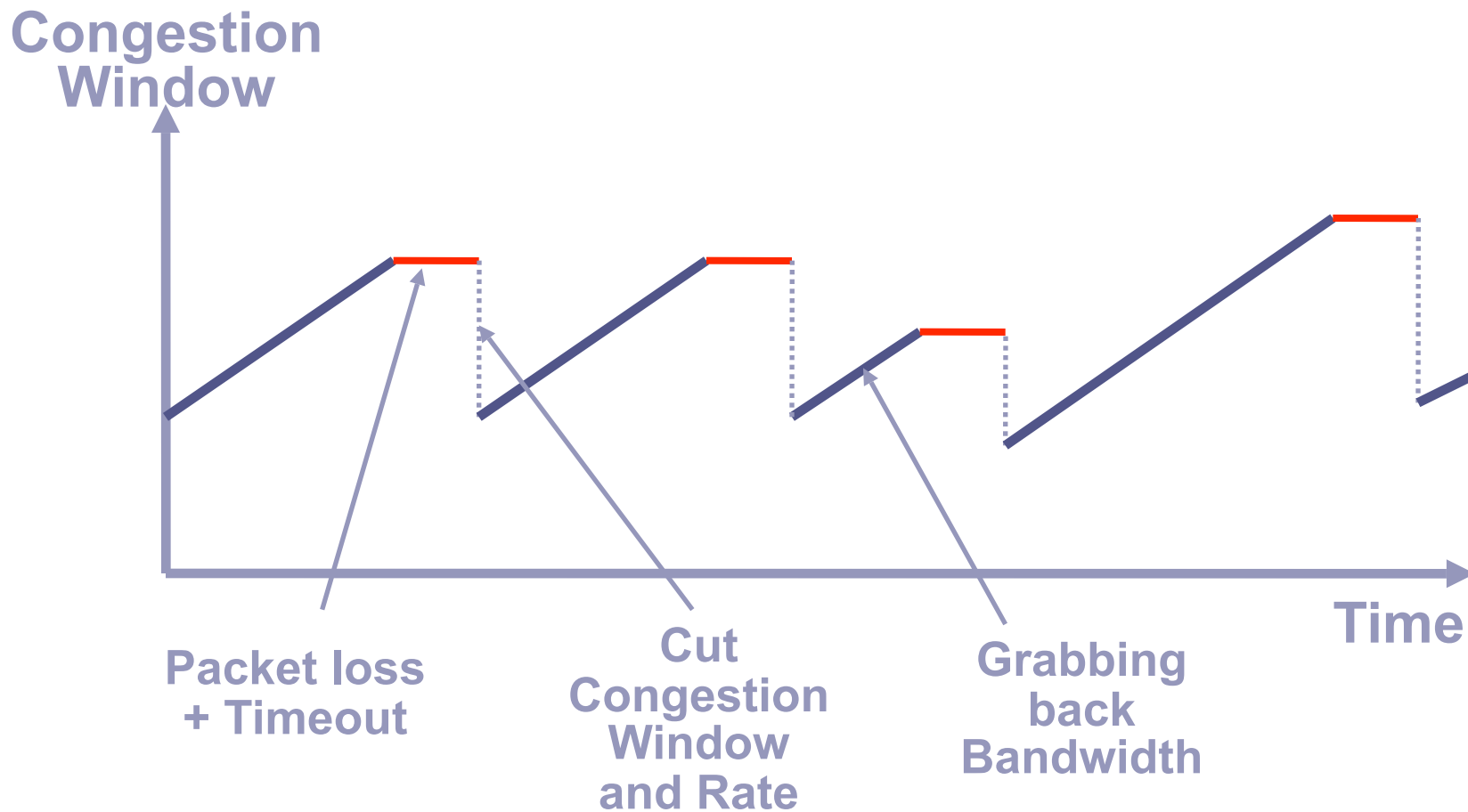
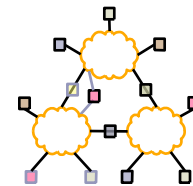


- If loss occurs when  $cwnd = W$ 
  - Network can handle  $0.5W \sim W$  segments
  - Set  $cwnd$  to  $0.5W$  (multiplicative decrease)
- Upon receiving ACK
  - Increase  $cwnd$  by  $(1 \text{ packet})/cwnd$ 
    - What is 1 packet?  $\rightarrow$  1 MSS worth of bytes
    - After  $cwnd$  packets have passed by  $\rightarrow$  approximately increase of 1 MSS
- Implements AIMD

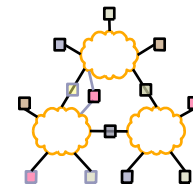
# Congestion Avoidance Sequence Plot



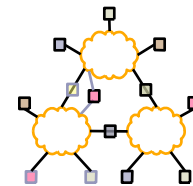
# Congestion Avoidance Behavior



# Packet Conservation

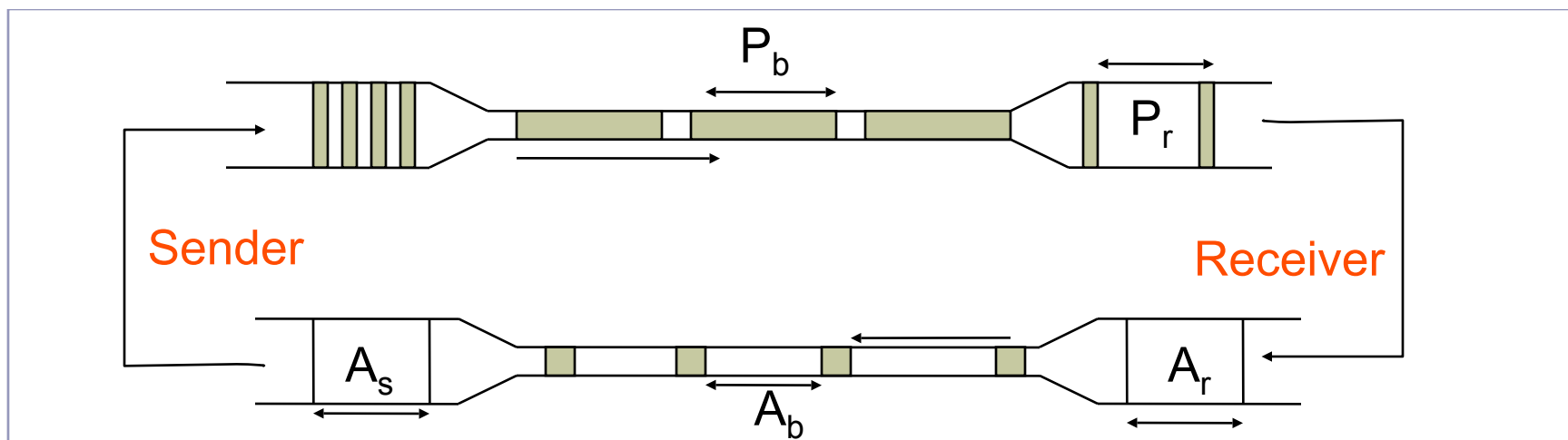


- At equilibrium, inject packet into network only when one is removed
  - Sliding window and not rate controlled
  - But still need to avoid sending burst of packets
    - would overflow links
      - Need to carefully pace out packets
      - Helps provide stability
- Need to eliminate spurious retransmissions
  - Accurate RTO estimation
  - Better loss recovery techniques (e.g. fast retransmit)

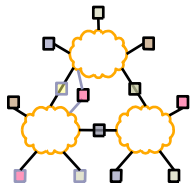


# TCP Packet Pacing

- Congestion window helps to “pace” the transmission of data packets
- In steady state, a packet is sent when an ack is received
  - Data transmission remains smooth, once it is smooth
  - Self-clocking behavior

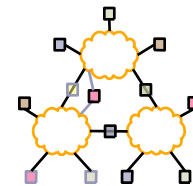


# Aside: Packet Pair



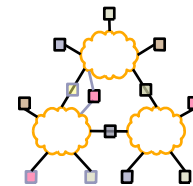
- What would happen if a source transmitted a pair of packets back-to-back?
- FIFO scheduling
  - Unlikely that another flows packet will get inserted in-between
  - Packets sent back-to-back are likely to be queued/forwarded back-to-back
  - Spacing will reflect link bandwidth
- Fair queuing
  - Router alternates between different flows
  - Bottleneck router will separate packet pair at exactly fair share rate
- Basis for many measurement techniques

# Reaching Steady State

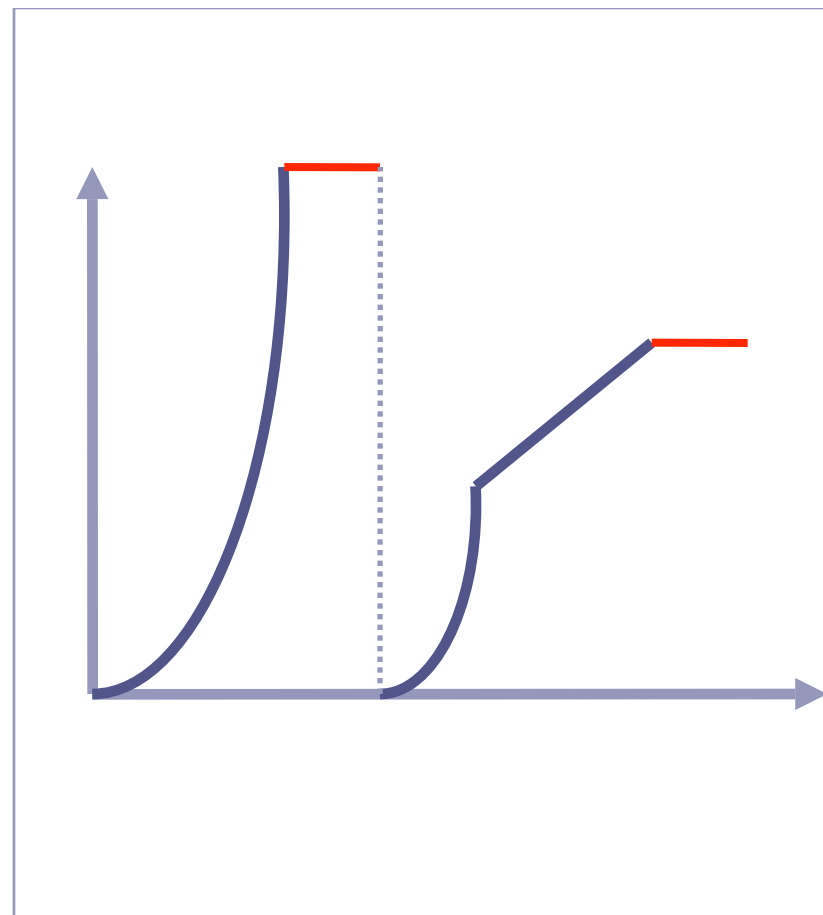


- Doing AIMD is fine in steady state but slow...
- How does TCP know what is a good initial rate to start with?
  - Should work both for a Modem (10s of Kbps or less) and for supercomputer links (10 Gbps and growing)
- Quick initial phase to help get up to speed (slow start)

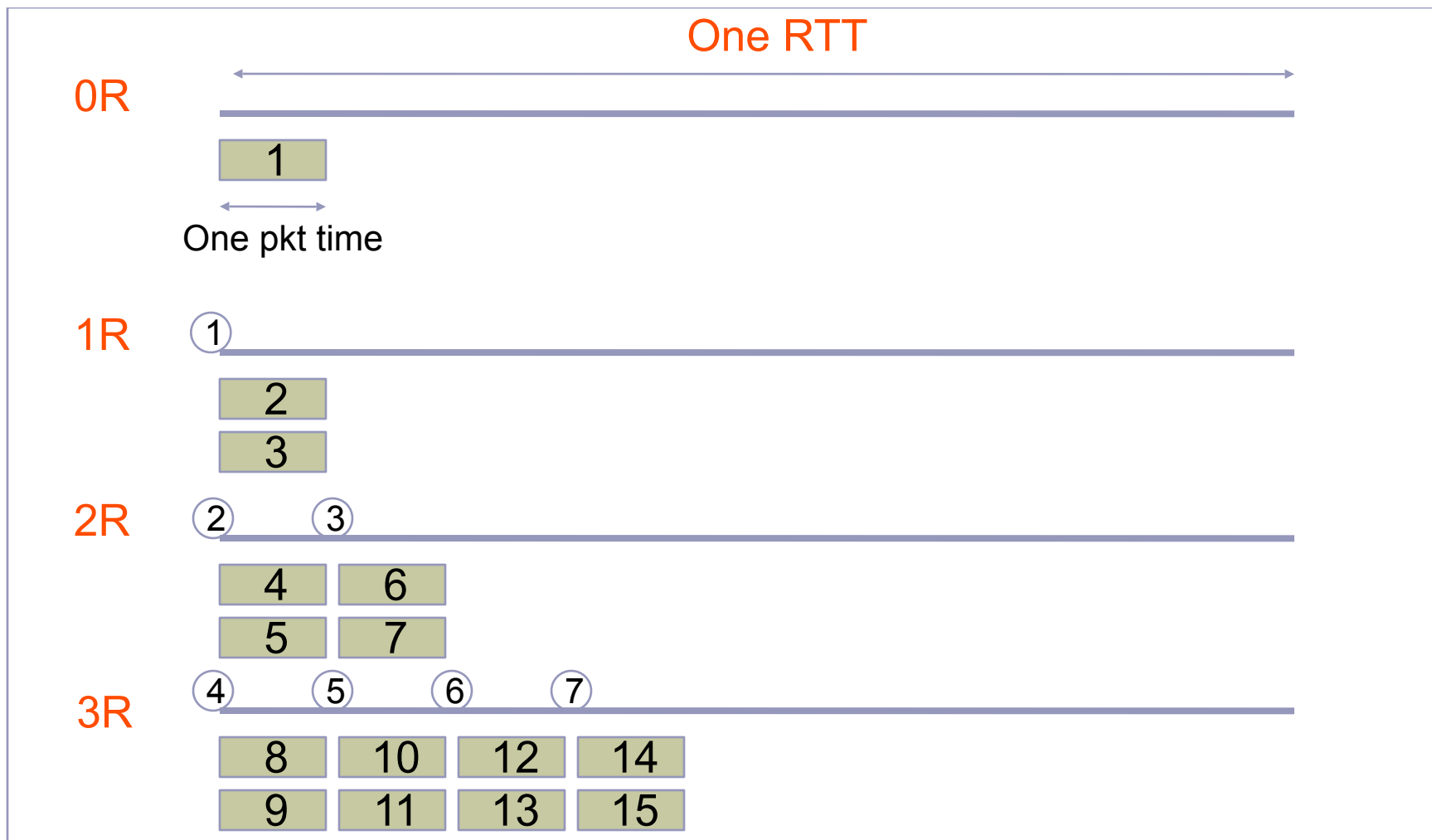
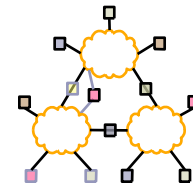
# Slow Start Packet Pacing



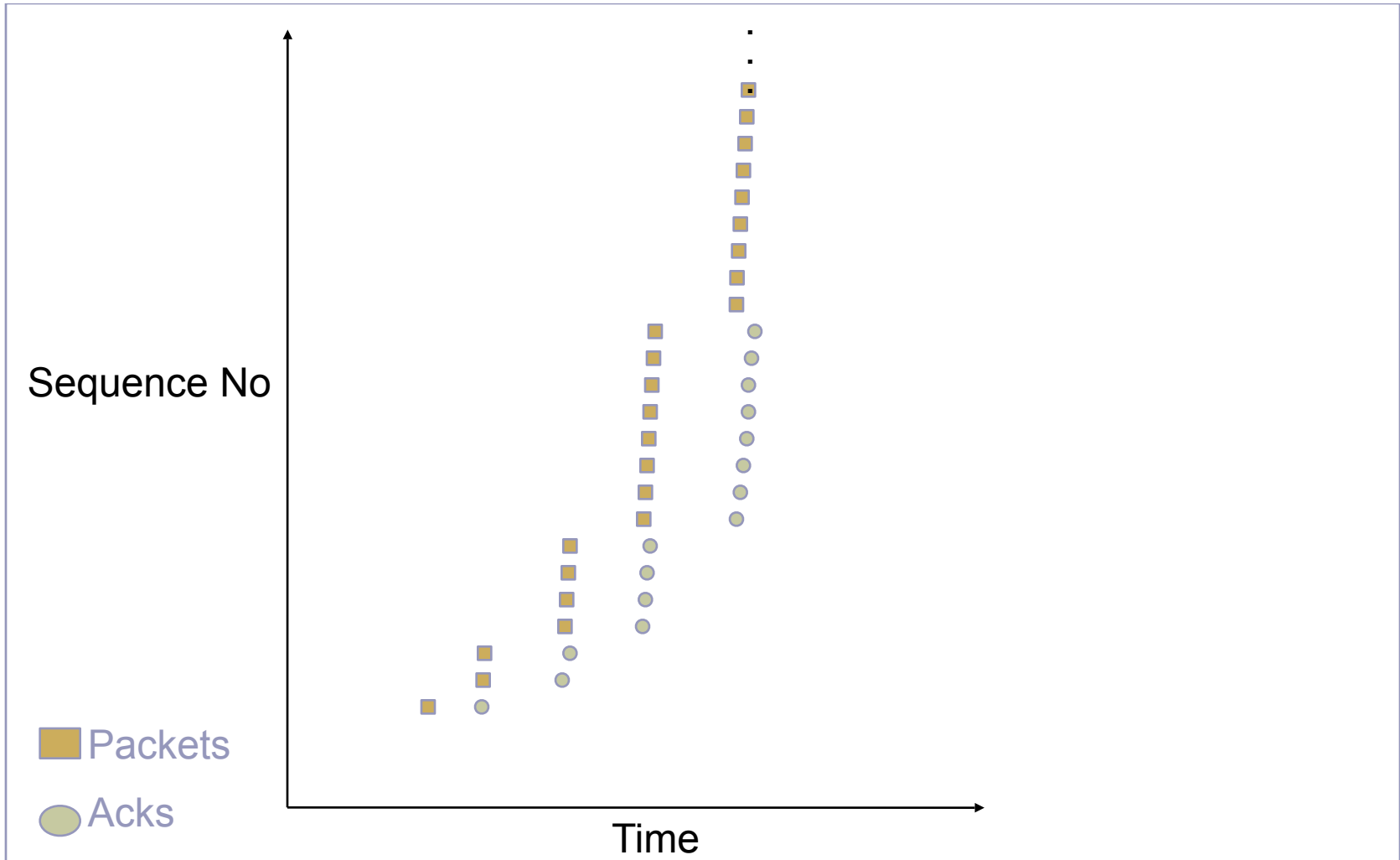
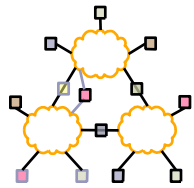
- How do we get this clocking behavior to start?
  - Initialize  $cwnd = 1$
  - Upon receipt of every ack,  $cwnd = cwnd + 1$
- Implications
  - Window actually increases to  $W$  in  $RTT * \log_2(W)$
  - Can overshoot window and cause packet loss



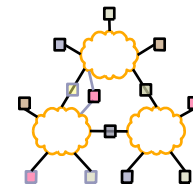
# Slow Start Example



# Slow Start Sequence Plot

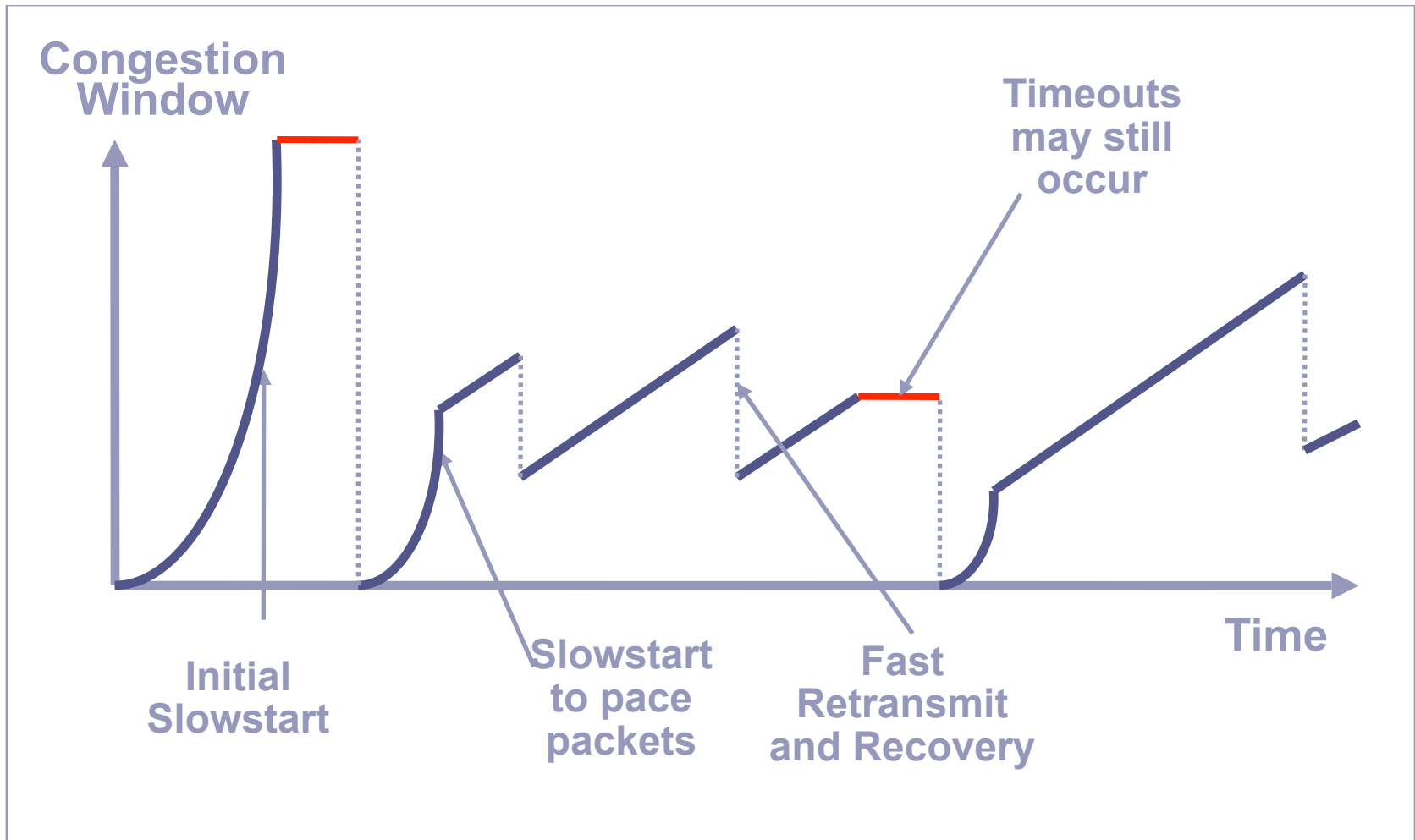
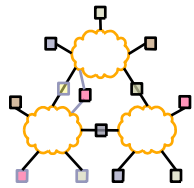


# Return to Slow Start

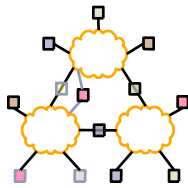


- If packet is lost we lose our self clocking as well
  - Need to implement slow-start and congestion avoidance together
- When timeout occurs set  $ssthresh$  to  $0.5w$ 
  - If  $cwnd < ssthresh$ , use slow start
  - Else use congestion avoidance

# TCP Saw Tooth Behavior

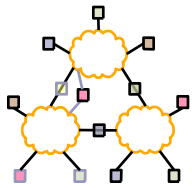


# Questions



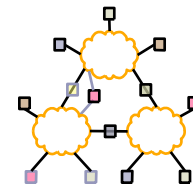
- Current loss rates – 10% in paper
- Uniform reaction to congestion – can different nodes do different things?
  - TCP friendliness, GAIMD, etc.
- Can we use queuing delay as an indicator?
  - TCP Vegas
- What about non-linear controls?

# Overview



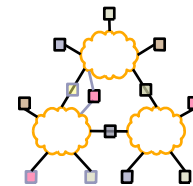
- TCP congestion control
- TFRC
- TCP and queues
- Queuing disciplines
- RED

# Changing Workloads



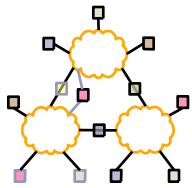
- New applications are changing the way TCP is used
- 1980's Internet
  - Telnet & FTP → long lived flows
  - Well behaved end hosts
  - Homogenous end host capabilities
  - Simple symmetric routing
- 2000's Internet
  - Web & more Web → large number of short xfers
  - Wild west – everyone is playing games to get bandwidth
  - Cell phones and toasters on the Internet
  - Policy routing
- How to accommodate new applications?

# TCP Friendliness



- What does it mean to be TCP friendly?
  - TCP is not going away
  - Any new congestion control must compete with TCP flows
    - Should not clobber TCP flows and grab bulk of link
    - Should also be able to hold its own, i.e. grab its fair share, or it will never become popular
- How is this quantified/shown?
  - Has evolved into evaluating loss/throughput behavior
  - But is this really true?

# TCP Friendly Rate Control (TFRC)

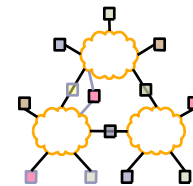


- Equation 1 – real TCP response

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO} \left(3\sqrt{\frac{3p}{8}}\right)p(1 + 32p^2)}$$

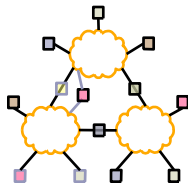
- 1<sup>st</sup> term corresponds to simple derivation
- 2<sup>nd</sup> term corresponds to more complicated timeout behavior
  - Is critical in situations with > 5% loss rates → where timeouts occur frequently
- Key parameters
  - RTO
  - RTT
  - Loss rate

# Loss Estimation



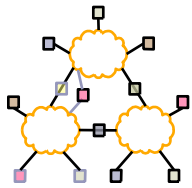
- Loss event rate vs. loss rate
- Characteristics
  - Should work well in steady loss rate
  - Should weight recent samples more
  - Should increase only with a new loss
  - Should decrease only with long period without loss
- Possible choices
  - Dynamic window – loss rate over last  $X$  packets
  - EWMA of interval between losses
  - Weighted average of last  $n$  intervals
    - Last  $n/2$  have equal weight

# Congestion Avoidance



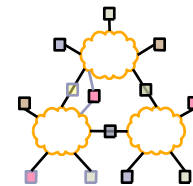
- Loss interval increases in order to increase rate
  - Primarily due to the transmission of new packets in current interval
  - History discounting increases interval by removing old intervals
  - .14 packets per RTT without history discounting
  - .22 packets per RTT with discounting
- Much slower increase than TCP
- Decrease is also slower
  - 4 – 8 RTTs to halve speed

# Overview



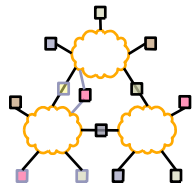
- TCP congestion control
- TFRC
- TCP and queues
- Queuing disciplines
- RED

# TCP Performance



- Can TCP saturate a link?
- Congestion control
  - Increase utilization until... link becomes congested
  - React by decreasing window by 50%
  - Window is proportional to rate \* RTT
- Doesn't this mean that the network oscillates between 50 and 100% utilization?
  - Average utilization = 75%??
  - **No...this is \*not\* right!**

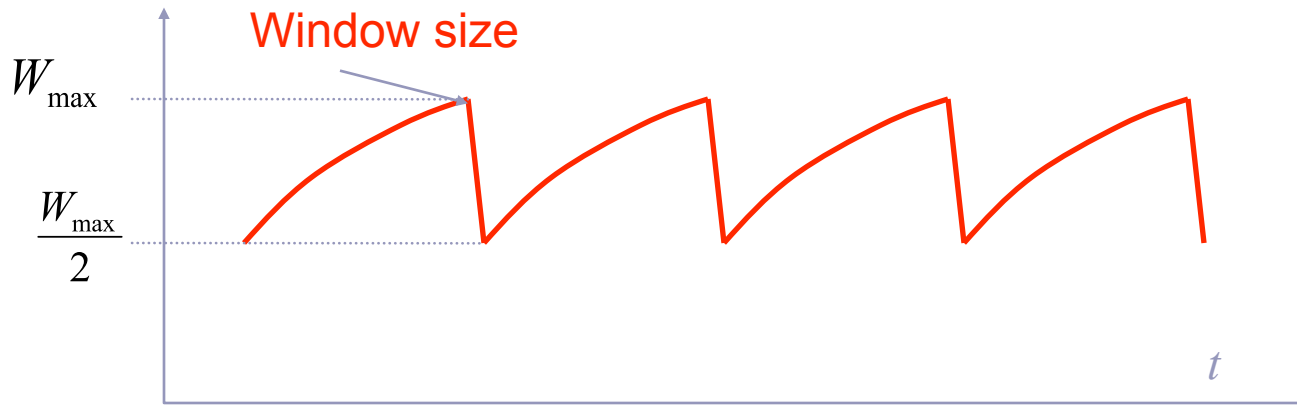
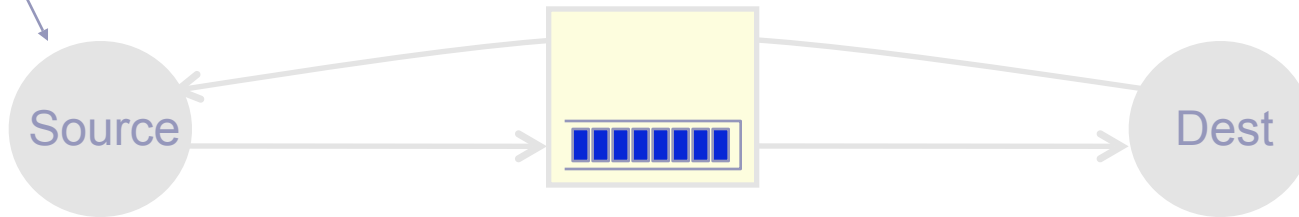
# TCP Congestion Control



## Rule for adjusting $W$

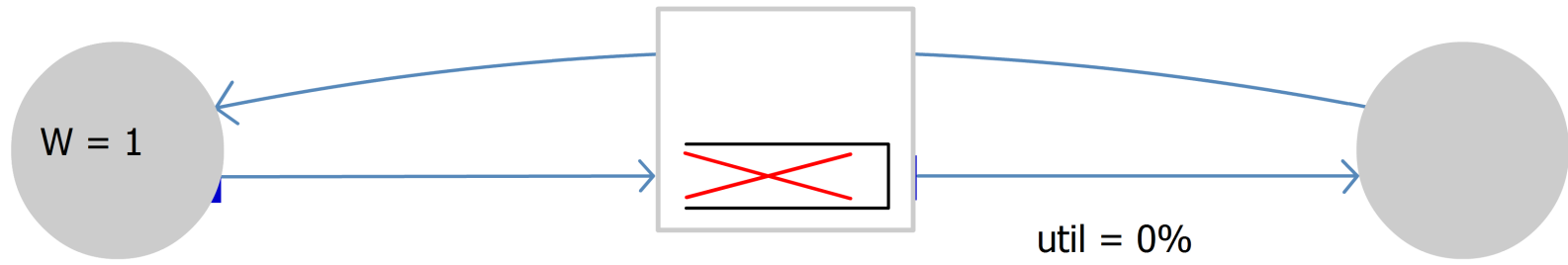
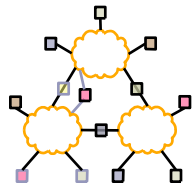
- If an ACK is received:  $W \leftarrow W + 1/W$
- If a packet is lost:  $W \leftarrow W/2$

Only  $W$  packets may be outstanding

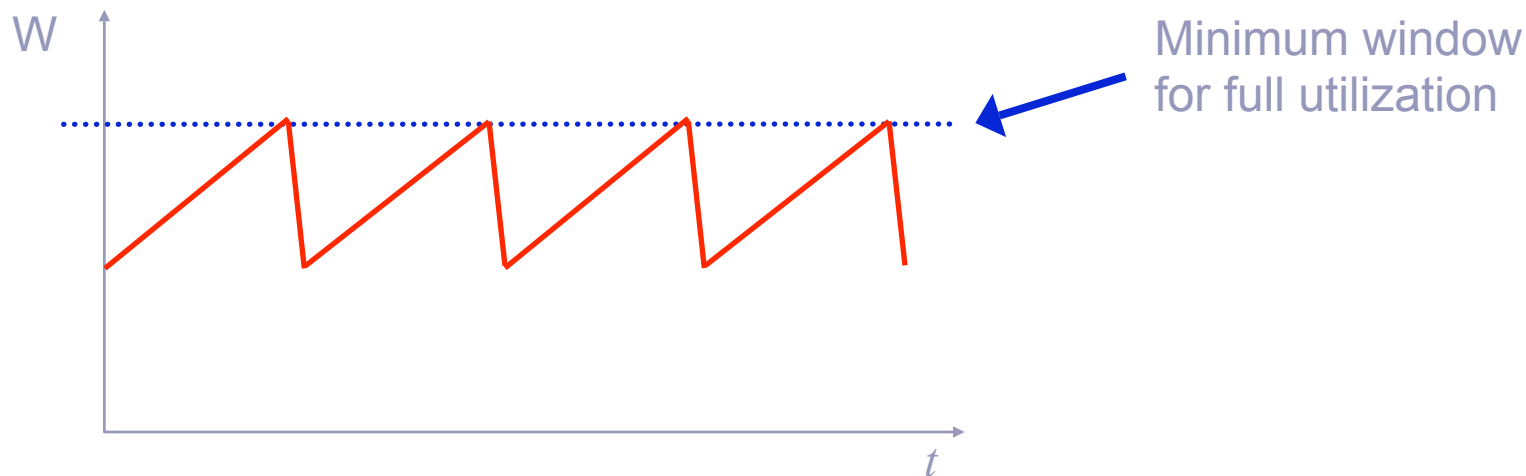
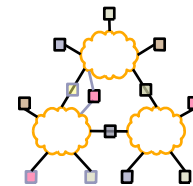


# Single TCP Flow

Router *without* buffers

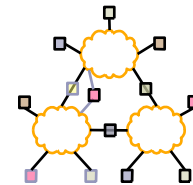


# Summary Unbuffered Link



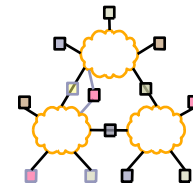
- The router can't fully utilize the link
  - If the window is too small, link is not full
  - If the link is full, next window increase causes drop
  - With no buffer it still achieves 75% utilization

# TCP Performance



- In the real world, router queues play important role
  - Window is proportional to  $\text{rate} * \text{RTT}$ 
    - But, RTT changes as well the window
  - Window to fill links =  $\text{propagation RTT} * \text{bottleneck bandwidth}$ 
    - If window is larger, packets sit in queue on bottleneck link

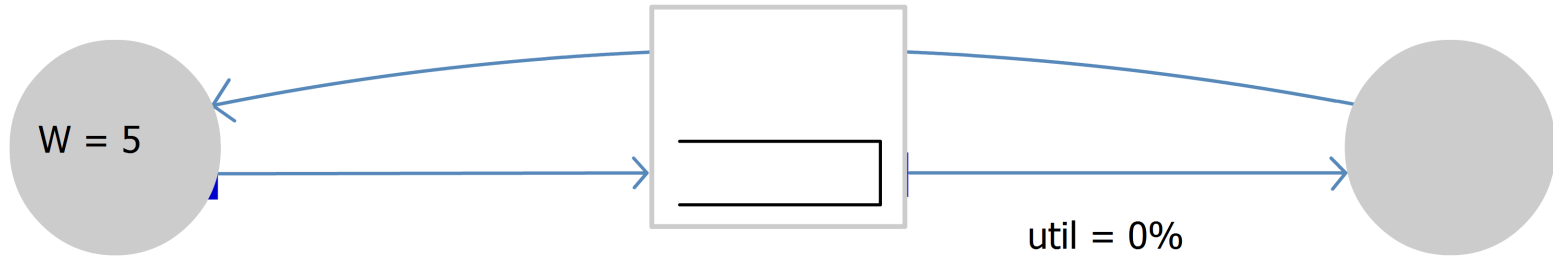
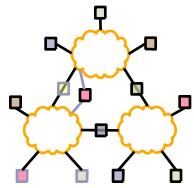
# TCP Performance



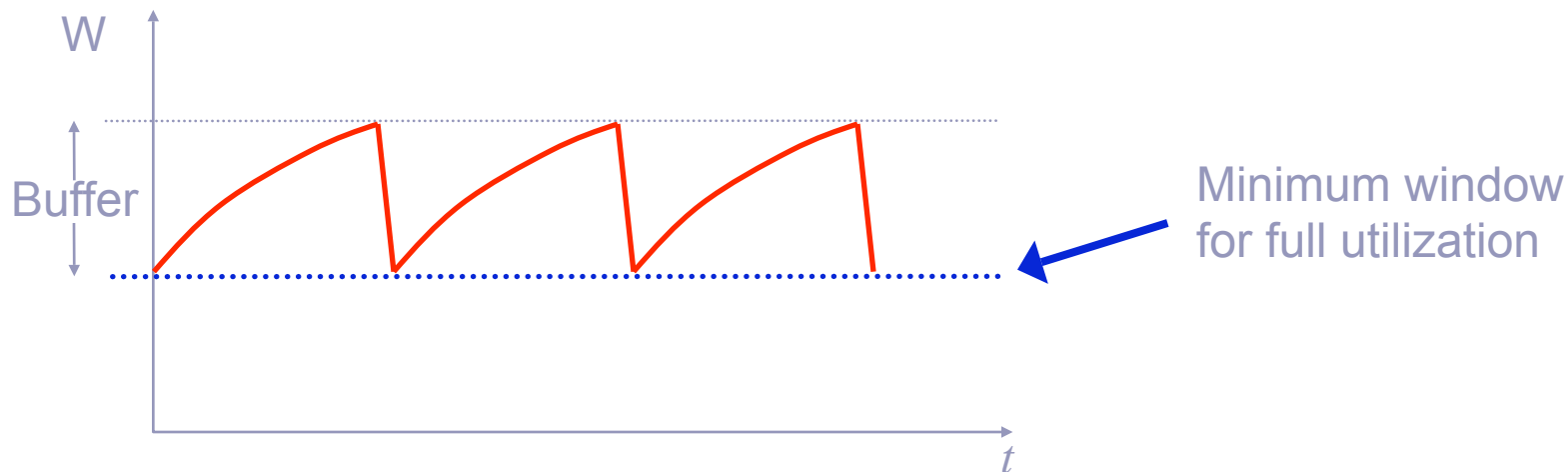
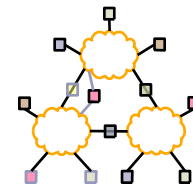
- If we have a large router queue  $\rightarrow$  can get 100% utilization
  - But, router queues can cause large delays
- How big does the queue need to be?
  - Windows vary from  $W \rightarrow W/2$ 
    - Must make sure that link is always full
    - $W/2 > RTT * BW$
    - $W = RTT * BW + Qsize$
    - Therefore,  $Qsize > RTT * BW$
  - Ensures 100% utilization
  - Delay?
    - Varies between  $RTT$  and  $2 * RTT$

# Single TCP Flow

Router with large enough buffers for full link utilization

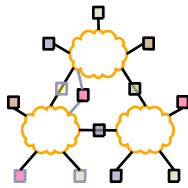


# Summary Buffered Link



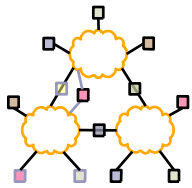
- With sufficient buffering we achieve full link utilization
  - The window is always above the critical threshold
  - Buffer absorbs changes in window size
    - Buffer Size = Height of TCP Sawtooth
    - Minimum buffer size needed is  $2T \cdot C$
  - This is the origin of the rule-of-thumb

# Example



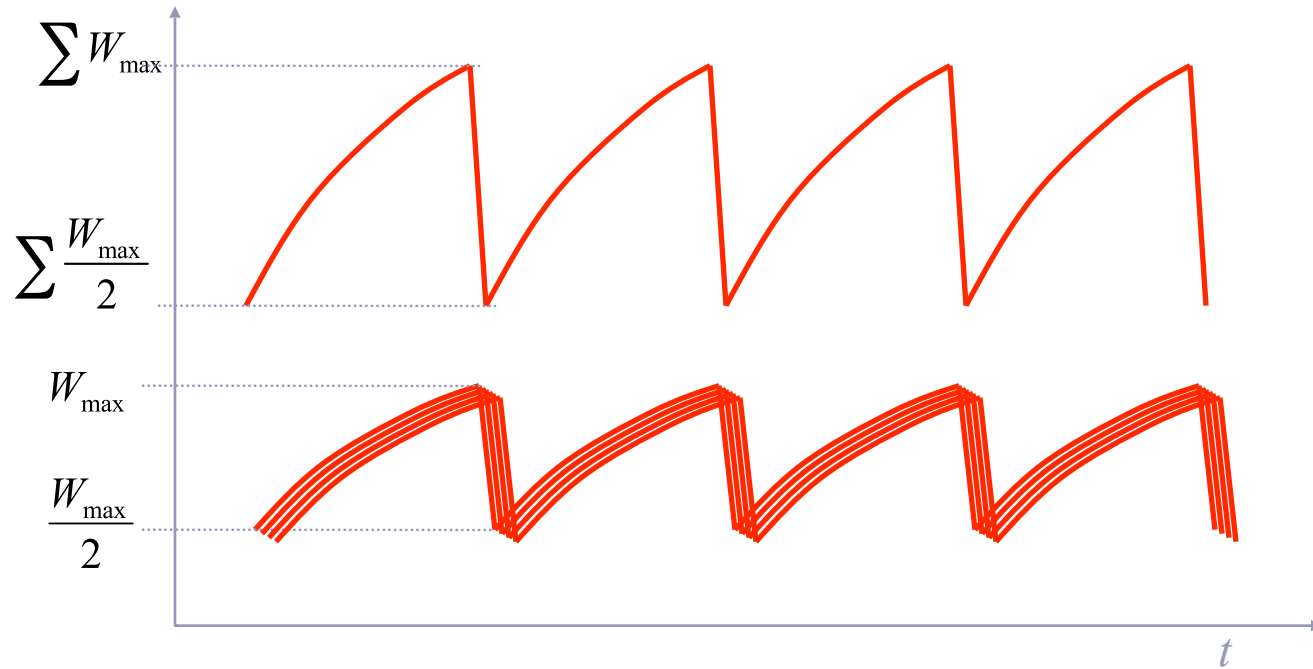
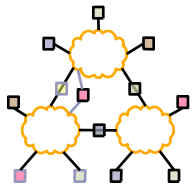
- 10Gb/s linecard
  - Requires 300Mbytes of buffering.
  - Read and write 40 byte packet every 32ns.
- Memory technologies
  - DRAM: require 4 devices, but too slow.
  - SRAM: require 80 devices, density/power issues, 1kW, \$2000
- Problem gets harder at 40Gb/s
  - Hence RLDRAM, FCRAM, etc.

# Rule-of-thumb



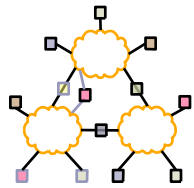
- Rule-of-thumb makes sense for one flow
- Typical backbone link has  $> 20,000$  flows
- Does the rule-of-thumb still hold?

# If flows are synchronized

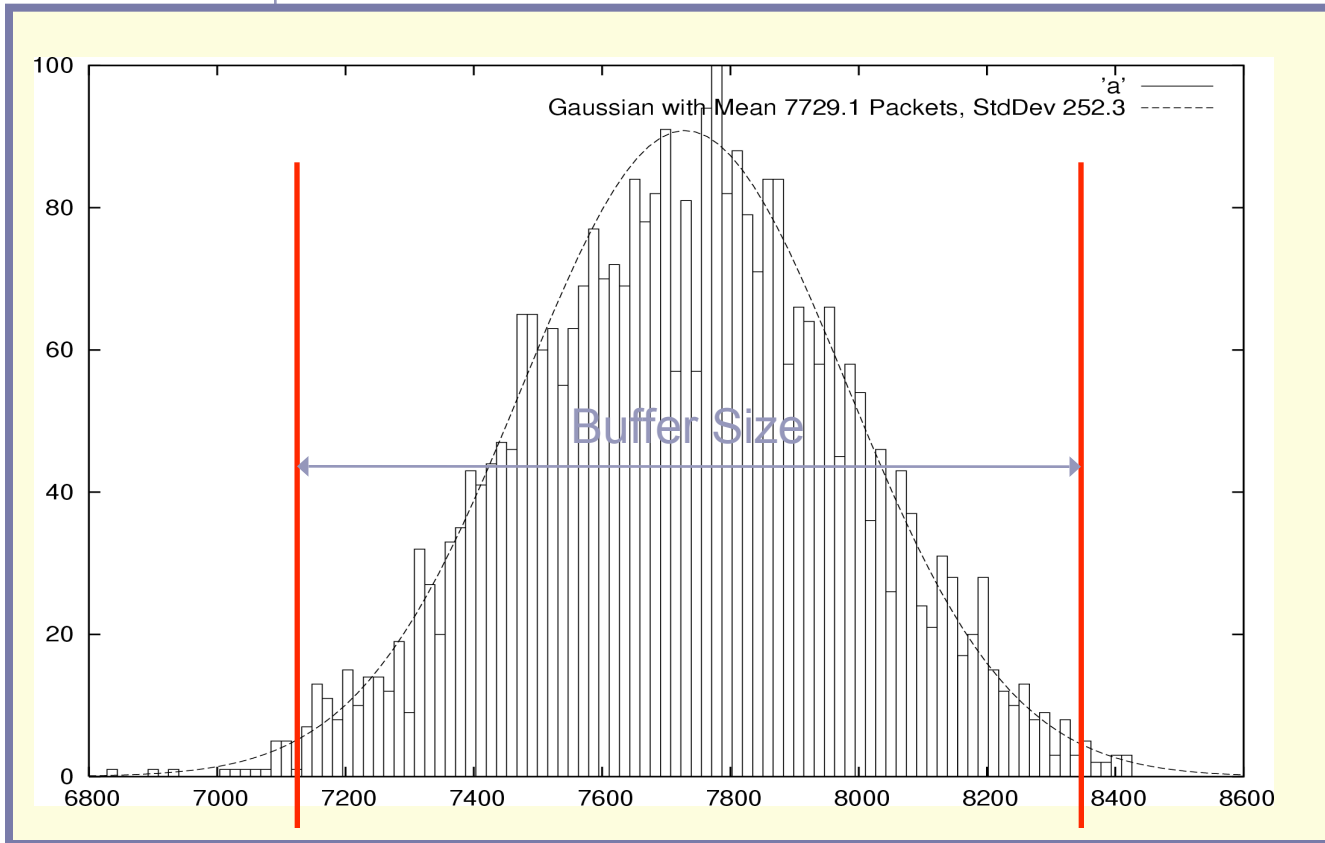
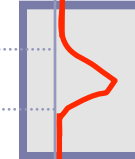
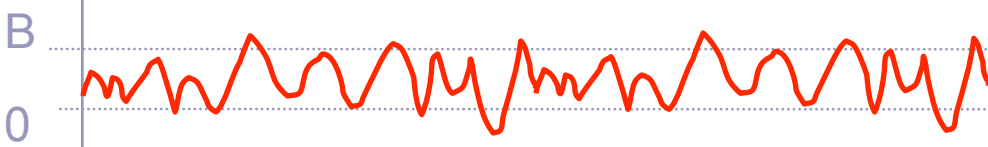


- Aggregate window has same dynamics
- Therefore buffer occupancy has same dynamics
- Rule-of-thumb still holds.

# If flows are not synchronized

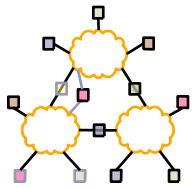


$\sum w$   
B  
0



Probability Distribution

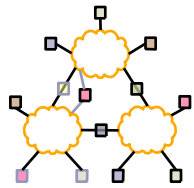
# Central Limit Theorem



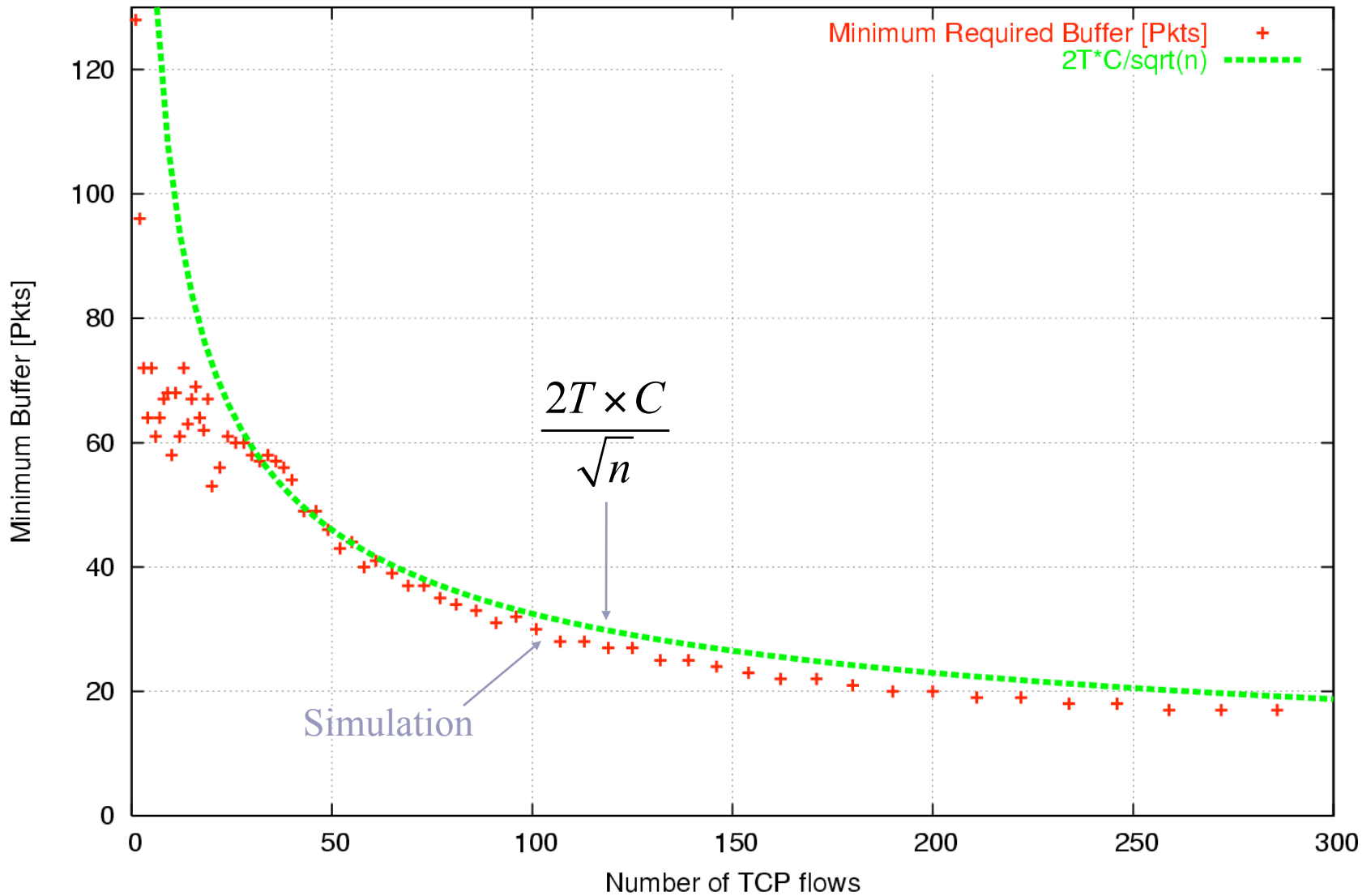
- CLT tells us that the more variables (Congestion Windows of Flows) we have, the narrower the Gaussian (Fluctuation of sum of windows)
  - Width of Gaussian decreases with  $\frac{1}{\sqrt{n}}$
  - Buffer size should also decrease with  $\frac{1}{\sqrt{n}}$

$$B = \frac{2T \times C}{\sqrt{n}}$$

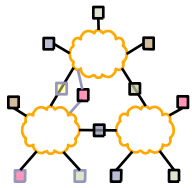
# Required buffer size



Minimum Required Buffer to Achieve 95% Goodput

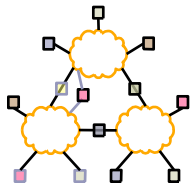


# Overview



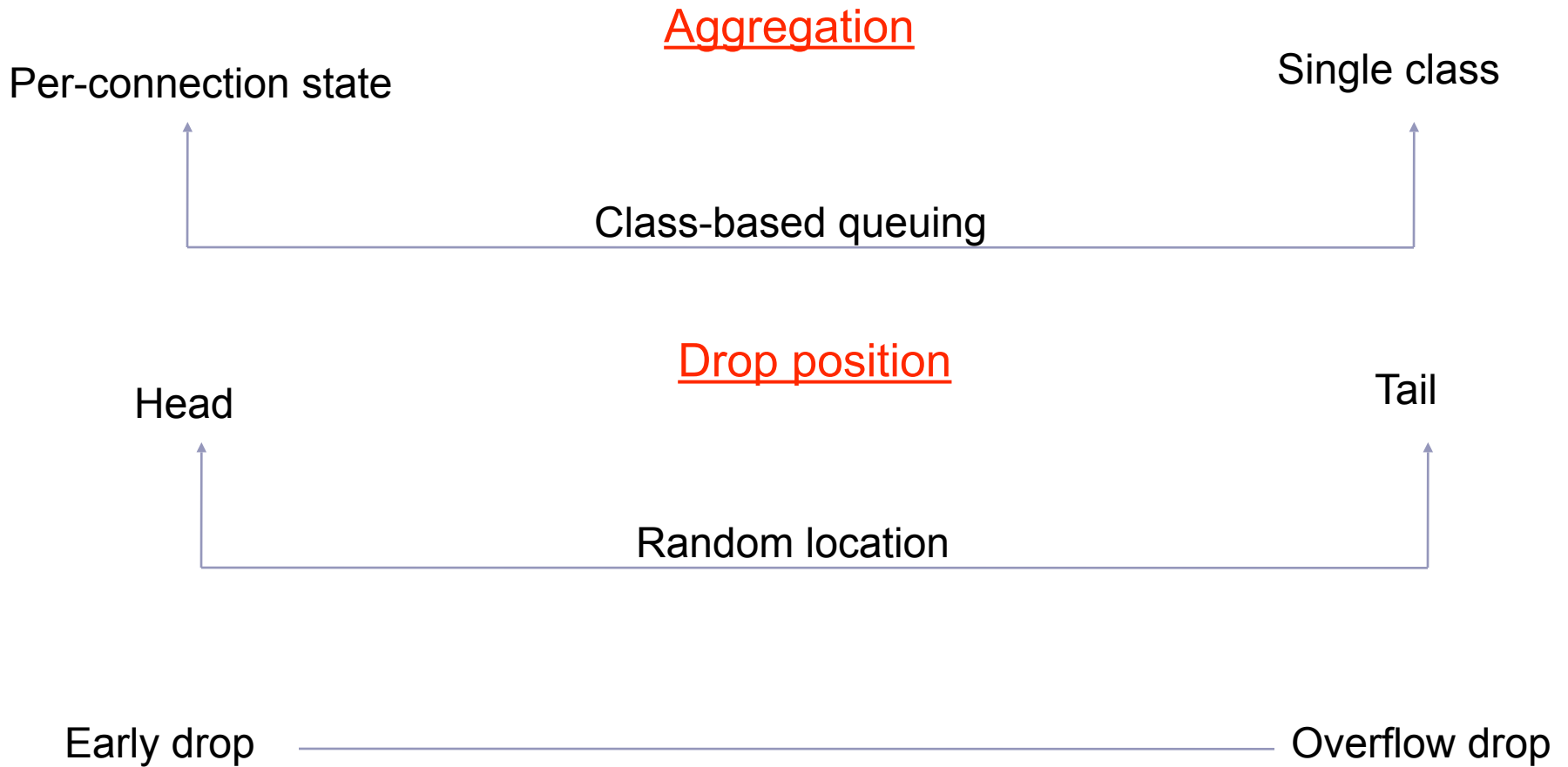
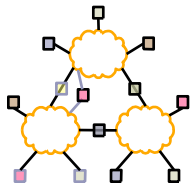
- TCP congestion control
- TFRC
- TCP and queues
- Queuing disciplines
- RED

# Queuing Disciplines

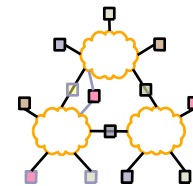


- Each router **must** implement some queuing discipline
- Queuing allocates both bandwidth and buffer space:
  - Bandwidth: which packet to serve (transmit) next
  - Buffer space: which packet to drop next (when required)
- Queuing also affects latency

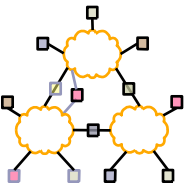
# Packet Drop Dimensions



# Typical Internet Queuing



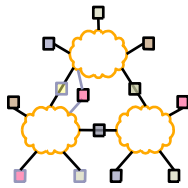
- FIFO + drop-tail
  - Simplest choice
  - Used widely in the Internet
- FIFO (first-in-first-out)
  - Implies single class of traffic
- Drop-tail
  - Arriving packets get dropped when queue is full regardless of flow or importance
- Important distinction:
  - FIFO: scheduling discipline
  - Drop-tail: drop policy



# FIFO + Drop-tail Problems

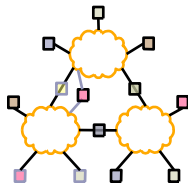
- Leaves responsibility of congestion control to edges (e.g., TCP)
- Does not separate between different flows
- No policing: send more packets → get more service
- Synchronization: end hosts react to same events

# Active Queue Management



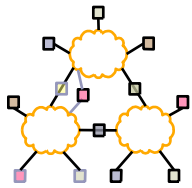
- Design active router queue management to aid congestion control
- Why?
  - Routers can distinguish between propagation and persistent queuing delays
  - Routers can decide on transient congestion, based on workload

# Active Queue Designs



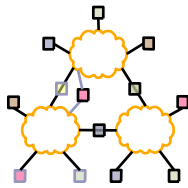
- Modify both router and hosts
  - DECbit – congestion bit in packet header
- Modify router, hosts use TCP
  - Fair queuing
    - Per-connection buffer allocation
  - RED (Random Early Detection)
    - Drop packet or set bit in packet header as soon as congestion is starting

# Overview



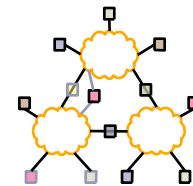
- TCP congestion control
- TFRC
- TCP and queues
- Queuing disciplines
- RED

# Internet Problems



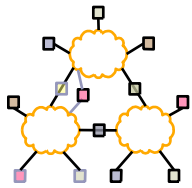
- Full queues
  - Routers are forced to have large queues to maintain high utilizations
  - TCP detects congestion from loss
    - Forces network to have long standing queues in steady-state
- Lock-out problem
  - Drop-tail routers treat bursty traffic poorly
  - Traffic gets synchronized easily → allows a few flows to monopolize the queue space

# Design Objectives

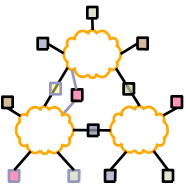


- Keep throughput high and delay low
- Accommodate bursts
- Queue size should reflect ability to accept bursts rather than steady-state queuing
- Improve TCP performance with minimal hardware changes

# Lock-out Problem



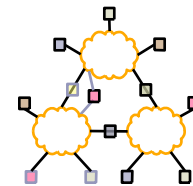
- Random drop
  - Packet arriving when queue is full causes some random packet to be dropped
- Drop front
  - On full queue, drop packet at head of queue
- Random drop and drop front solve the lock-out problem but not the full-queues problem



# Full Queues Problem

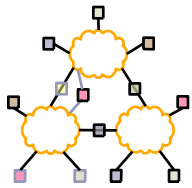
- Drop packets before queue becomes full (early drop)
- Intuition: notify senders of incipient congestion
  - Example: early random drop (ERD):
    - If  $q_{len} > \text{drop level}$ , drop each new packet with fixed probability  $p$
    - Does not control misbehaving users

# Random Early Detection (RED)



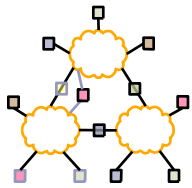
- Detect incipient congestion, allow bursts
- Keep power (throughput/delay) high
  - Keep average queue size low
  - Assume hosts respond to lost packets
- Avoid window synchronization
  - Randomly mark packets
- Avoid bias against bursty traffic
- Some protection against ill-behaved users

# RED Algorithm



- Maintain running average of queue length
- If  $avgq < min_{th}$  do nothing
  - Low queuing, send packets through
- If  $avgq > max_{th}$ , drop packet
  - Protection from misbehaving sources
- Else mark packet in a manner proportional to queue length
  - Notify sources of incipient congestion

# RED Operation



Max thresh

Min thresh



Average Queue Length

$P(\text{drop})$

1.0

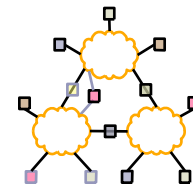
$\max_p$

$\min_{th}$

$\max_{th}$

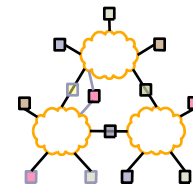
Avg queue length

# RED Algorithm



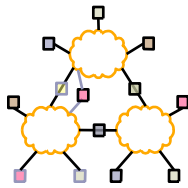
- Maintain running average of queue length
  - Byte mode vs. packet mode – why?
- For each packet arrival
  - Calculate average queue size (avg)
  - If  $\min_{th} \leq avgq < \max_{th}$ 
    - Calculate probability  $P_a$
    - With probability  $P_a$ 
      - Mark the arriving packet
  - Else if  $\max_{th} \leq avg$ 
    - Mark the arriving packet

# Queue Estimation



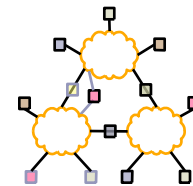
- Standard EWMA:  $avgq = (1-w_q) avgq + w_q qlen$
- Upper bound on  $w_q$  depends on  $min_{th}$ 
  - Want to ignore transient congestion
  - Can calculate the queue average if a burst arrives
    - Set  $w_q$  such that certain burst size does not exceed  $min_{th}$
- Lower bound on  $w_q$  to detect congestion relatively quickly
- Typical  $w_q = 0.002$

# Thresholds



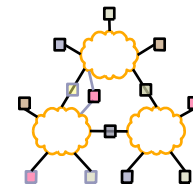
- $\min_{th}$  determined by the utilization requirement
  - Tradeoff between queuing delay and utilization
- Relationship between  $\max_{th}$  and  $\min_{th}$ 
  - Want to ensure that feedback has enough time to make difference in load
  - Depends on average queue increase in one RTT
  - Paper suggest ratio of 2
    - Current rule of thumb is factor of 3

# Packet Marking



- $\max_p$  is reflective of typical loss rates
- Paper uses 0.02
  - 0.1 is more realistic value
- If network needs marking of 20-30% then need to buy a better link!
- Gentle variant of RED (recommended)
  - Vary drop rate from  $\max_p$  to 1 as the avgq varies from  $\max_{th}$  to  $2^* \max_{th}$

# Extending RED for Flow Isolation



- Problem: what to do with non-cooperative flows?
- Fair queuing achieves isolation using per-flow state – expensive at backbone routers
  - How can we isolate unresponsive flows without per-flow state?
- RED penalty box
  - Monitor history for packet drops, identify flows that use disproportionate bandwidth
  - Isolate and punish those flows