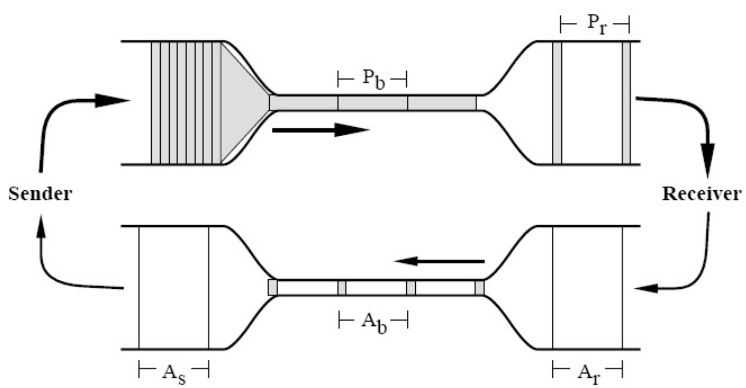


Class Meeting, Lectures 7 & 8: Congestion, Queues, & Middleboxes

Kyle Jamieson

COS 461: Computer Networks

www.cs.princeton.edu/courses/archive/fall21/cos461

[Parts adapted from material by M. Freedman (Princeton), B. Karp (UCL), D. Katabi, (MIT), S. Shenker (UCB)]

Today

- **Key concepts in Congestion Control**
  - **Retransmits and RTT estimator**
  - Slow Start and Self-clocking
  - AIMD Congestion control

- Queue Management

- Middleboxes

Mean and Variance: Jacobson's RTT Estimator

- Above link load of 30% at router, $\beta \times \text{RTT}_i$ will retransmit too early!
- Response to increasing load: waste bandwidth on duplicate packets
- Result: **congestion collapse!**

- [Jacobson]: estimate v_i, *mean deviation* (EWMA of $|m_i - \text{RTT}_i|$), stand-in for variance

$$v_i = v_{i-1} \times (1-\gamma) + \gamma \times |m_i - \text{RTT}_i|$$

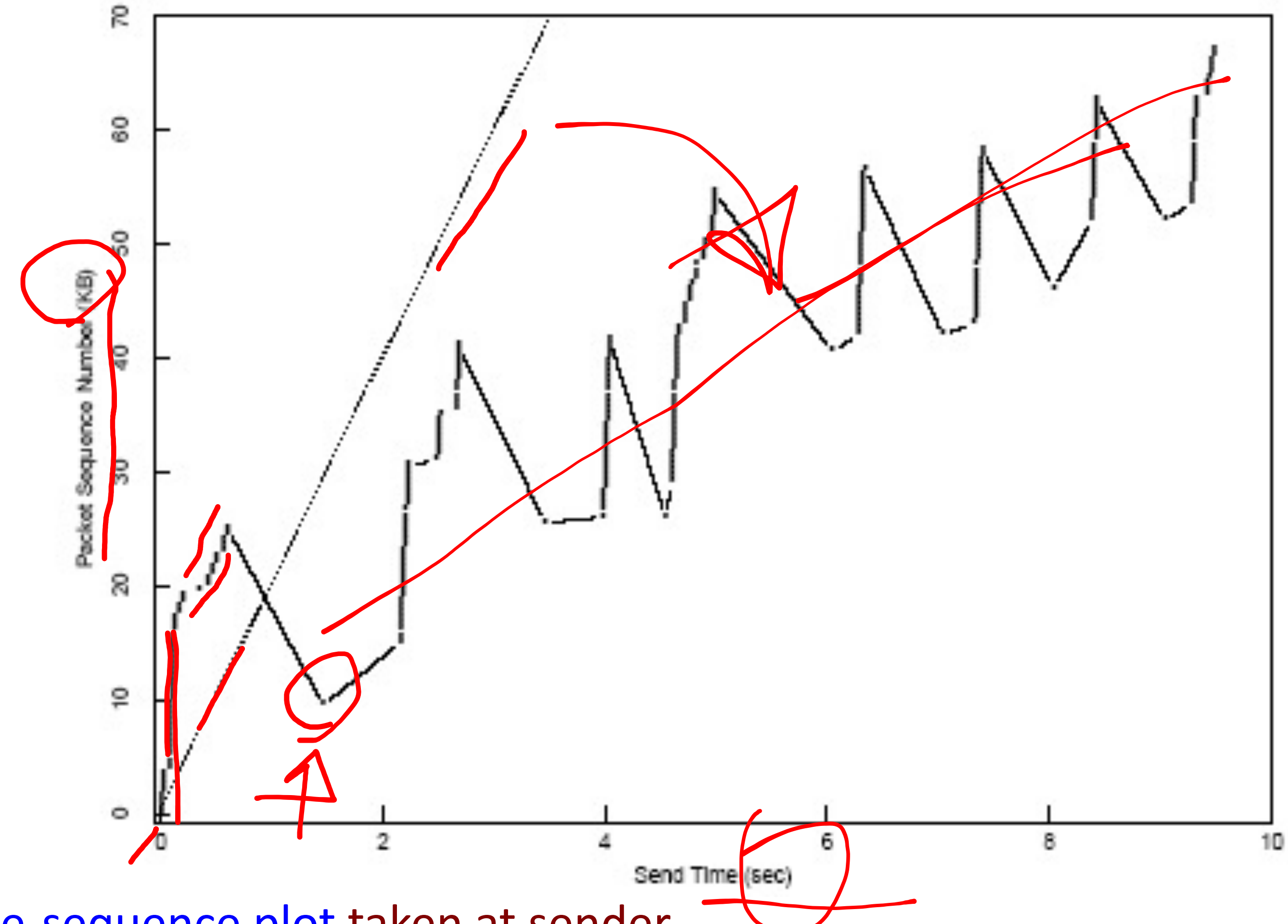
- **Modern TCPs use $\text{RTO}_i = \text{RTT}_i + 4v_i$**

Jacobson, V. and Karels, M., Congestion Avoidance and Control, *SIGCOMM 1988*.

Retransmit Behavior

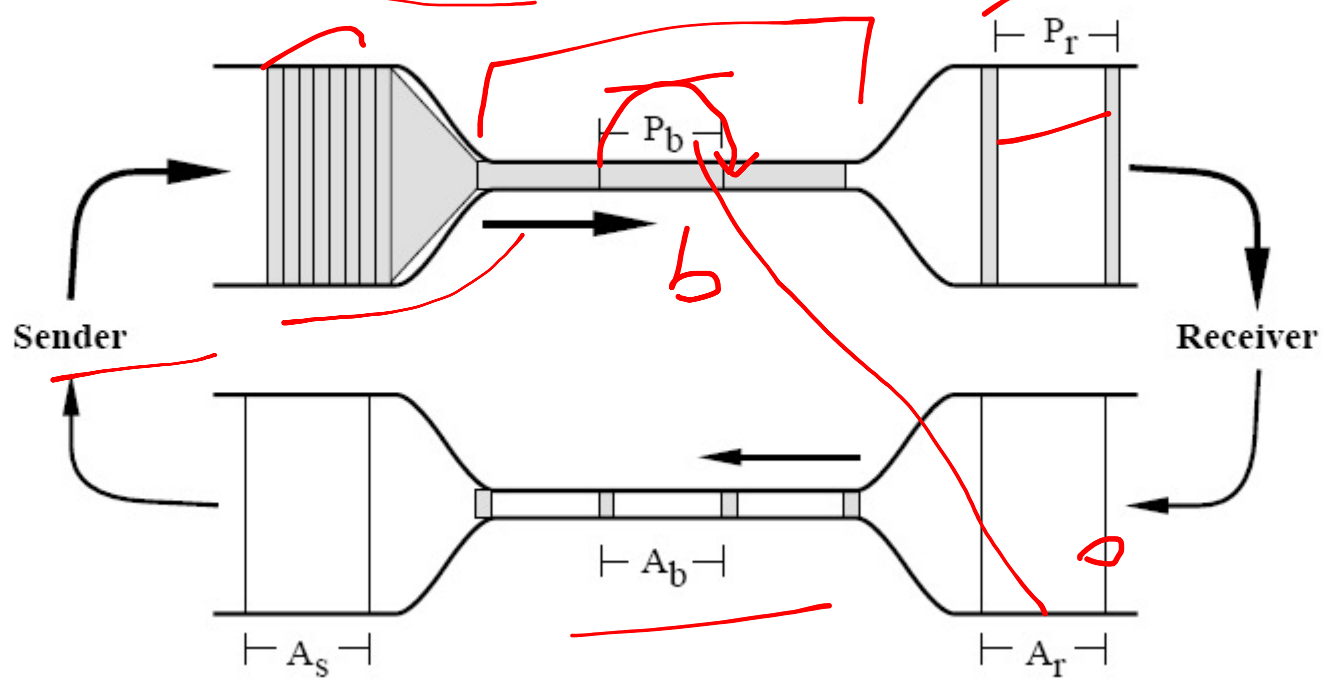
- Original TCP (pre-AIMD design), before [Jacobson 88]:
  - at start of connection, send full window of packets
  - retransmit each packet immediately after its timer expires
- Result: window-sized bursts of packets sent into network

Pre-Jacobson TCP (Obsolete!)



- Time-sequence plot taken at sender
- Bursts of packets: vertical lines
- Spurious retransmits: repeats at same y value
- Dashed line: available 20 Kbps capacity

Concept: “Self-Clocking” Conservation of Packets

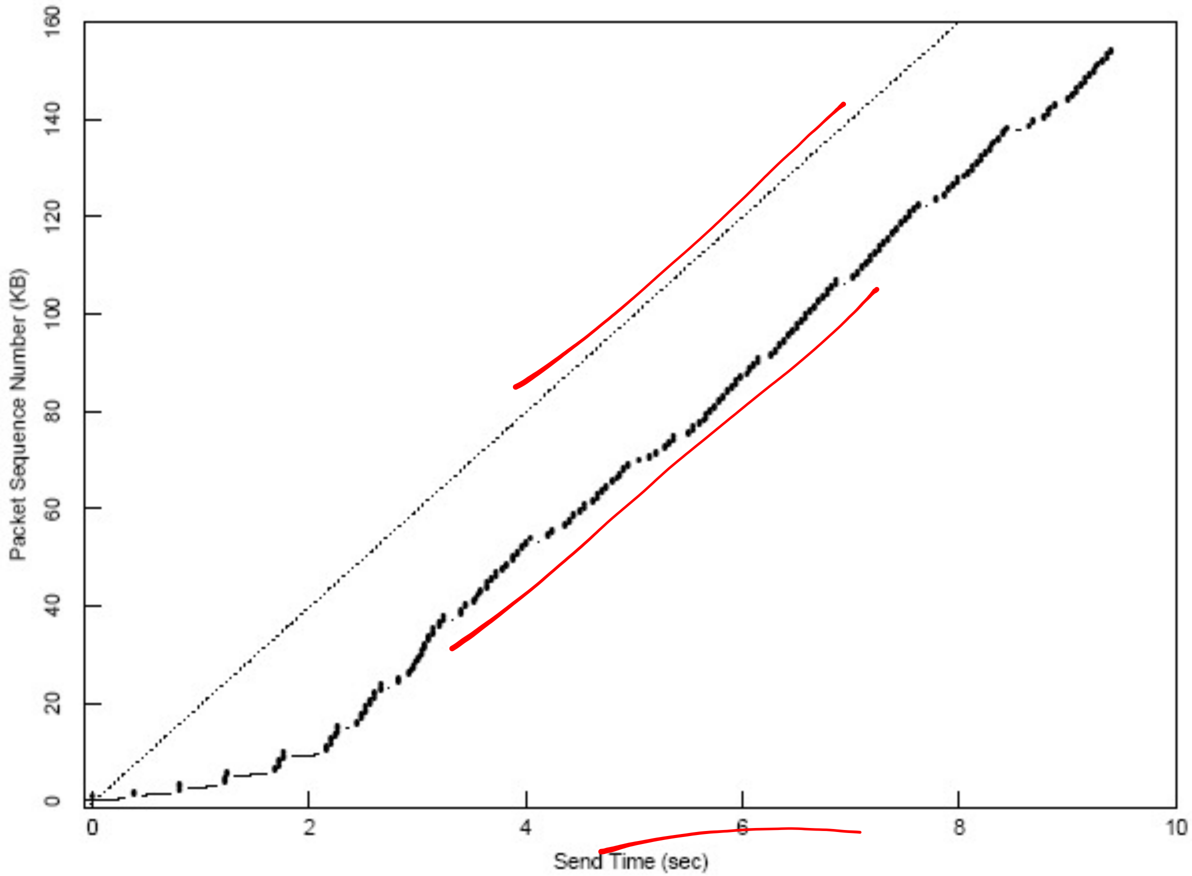


- Goal: “self-clocking” transmission paced by ACKs
  - each ACK returns, one data packet sent
  - spacing of returning ACKs: matches spacing of packets in time at slowest link on path P_b

Review: Reaching Equilibrium via Slow Start

- At connection start, sender sets congestion window size, cwnd, to pktSize (one packet's worth of bytes), not whole window
- Sender sends up to minimum of receiver's advertised window size W and cwnd
- Upon return of each ACK until receiver's advertised window size reached, increase cwnd by pktSize bytes
- "Slow" means exponential window increase!
- Takes $\log_2(W/pktSize)$ RTTs to reach receiver's advertised window size W

Post-Jacobson TCP: Slow Start and Mean+Variance RTT Estimator

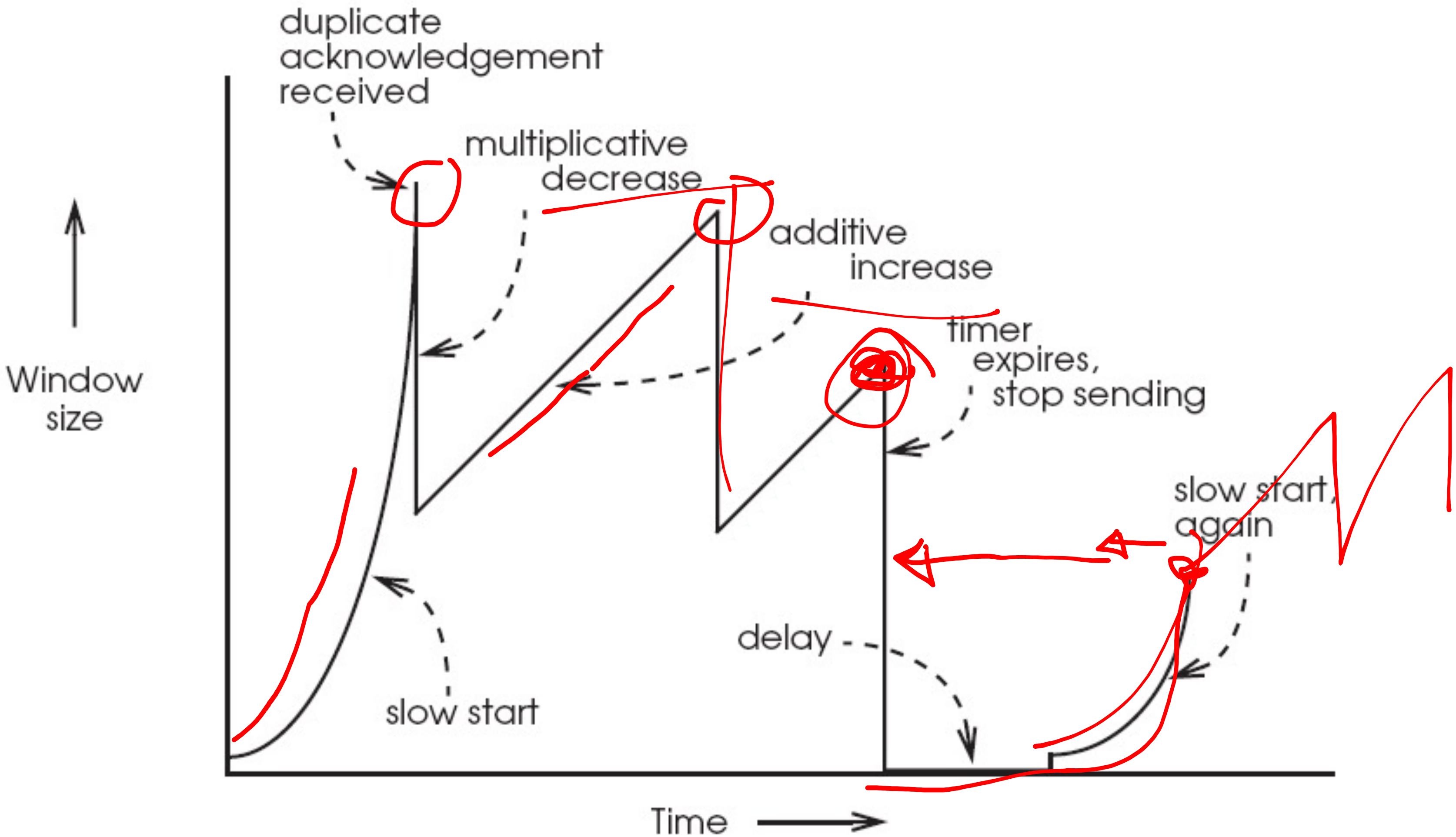


- Time-sequence plot at sender; dash line = available capacity
- “Slower” start
- No spurious retransmits

Congestion Requires Slowing Senders

- Recall: bigger buffers cannot prevent congestion
- Senders must slow to alleviate congestion
- Absence of ACKs implicitly indicates congestion
- TCP sender's window size determines sending rate
- Recall: correct window size is bottleneck bandwidth-delay product
- How can sender learn this value?
  - **Search** for it, by adapting window size
  - **Feedback** from network: ACKs return (window OK) or do not return (window too big)

AIMD in Action



- Sender **searches** for correct window size

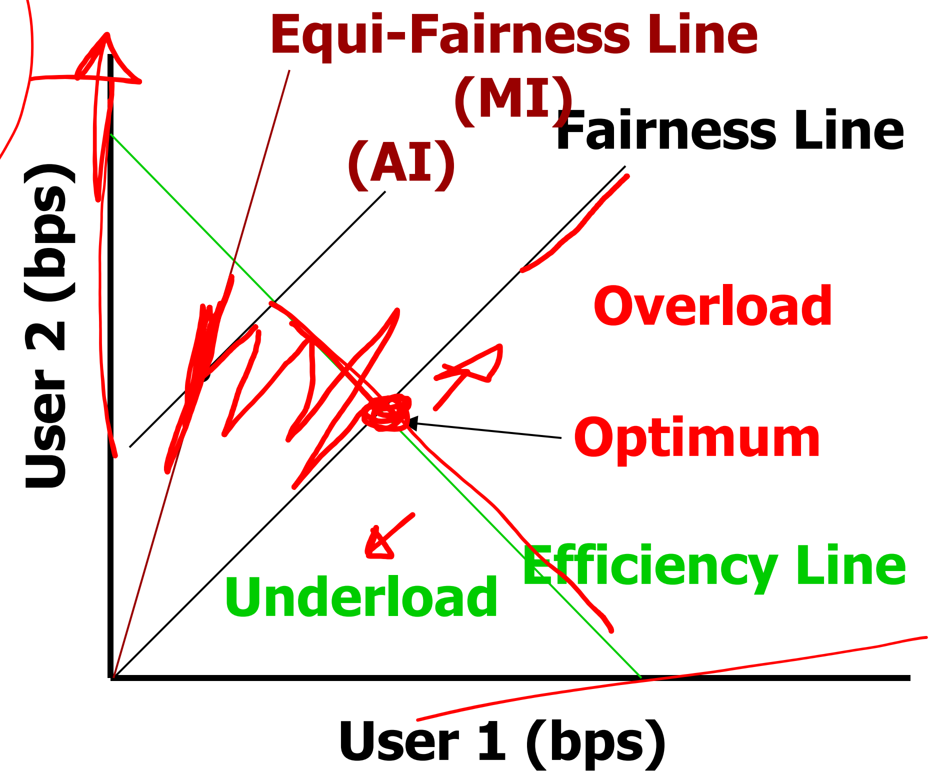
Why AIMD?

- Other control rules possible
  - E.g., MIMD, AIAD, …
- Recall goals:
  - Links fully utilized (efficient)
  - Users share resources fairly
- TCP adapts all flows' window sizes independently
- Must choose a control that will always converge to an efficient and fair allocation of windows

Chiu-Jain Phase Plots

- Consider two users sharing a bottleneck link
- Plot bandwidths allocated to each

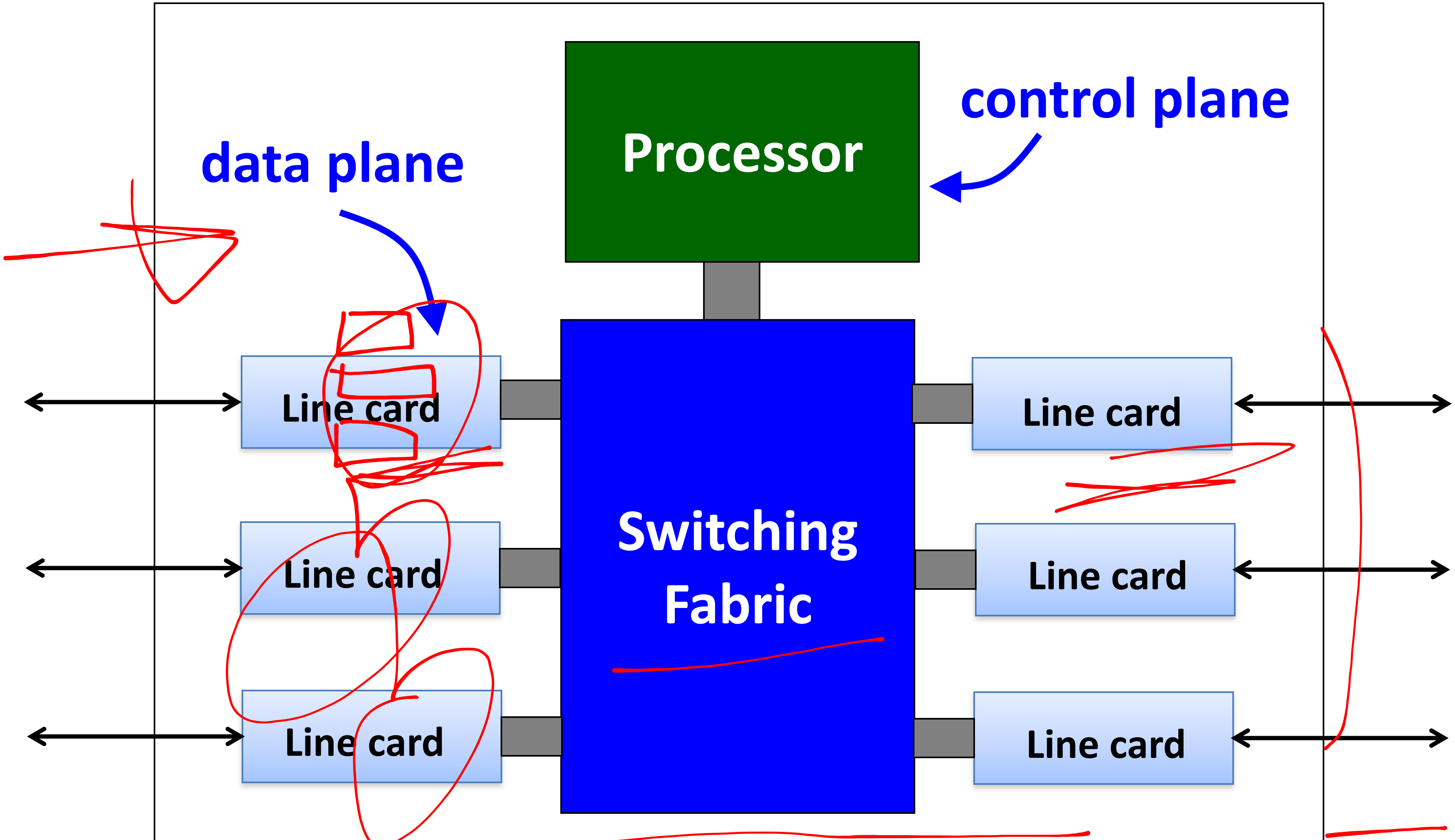
- Efficiency Line: sum of two users' rates = bottleneck capacity
- Fairness Line: two users' rates equal
- Equi-Fairness Line: ratio of two users' rates fixed



Today

- Key concepts in Congestion Control
  - Retransmits and RTT estimator
  - Slow Start and Self-clocking
  - AIMD Congestion control
- **Queue Management**
- Middleboxes

Context: Where are the queues?
The Routers!

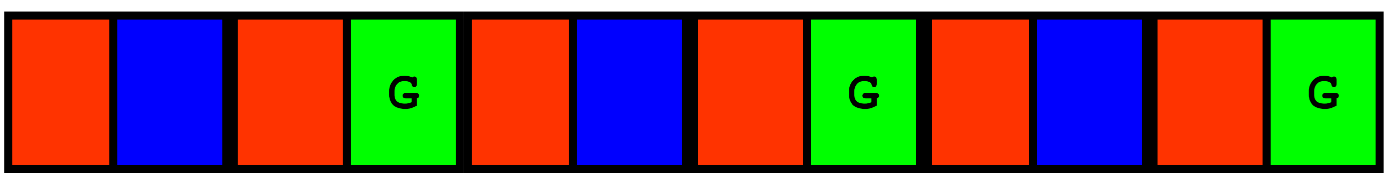


Problem: How to allocate egress router bandwidth to flows?

Weighted Fair Queuing (WFQ)

- Weighted fair queuing
  - Assign each queue a fraction of the link bandwidth
  - Rotate across queues on a small time scale

- WFQ results in max-min fairness
  - Maximizes the least rate that any flow gets



50% red, 25% blue, 25% green(G)

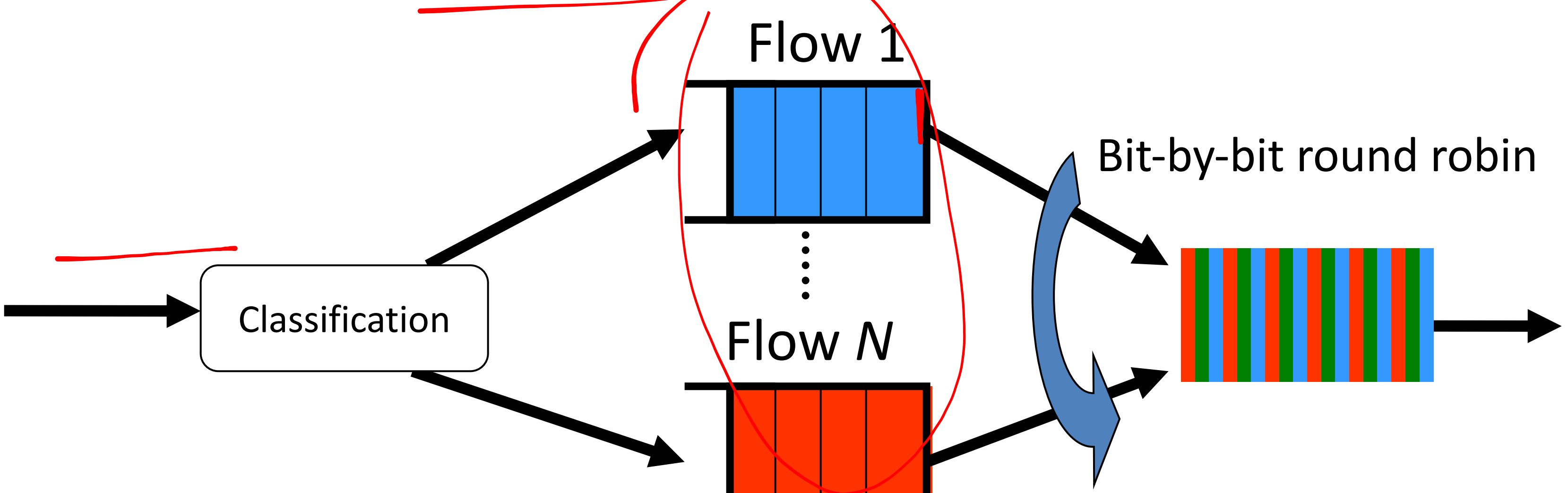
Bit-by-Bit "Fluid" Fair Queuing (FQ)

Question: What is a "flow"?

Flow 5-tuple: protocol, IP source/dest, port src/dest

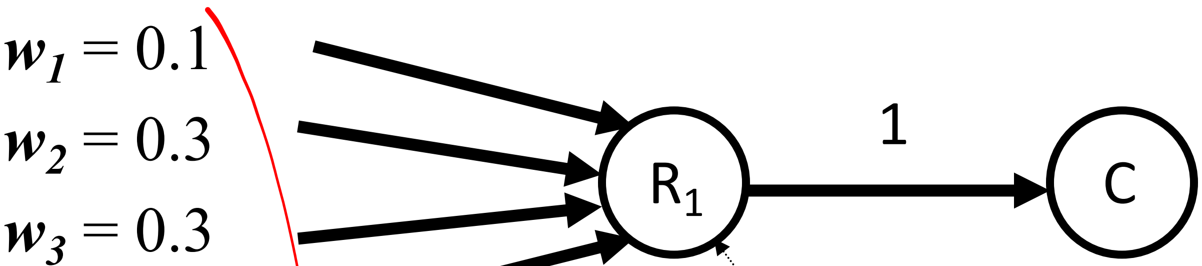
Question: How to choose weights?

Protocol class, bit markings, prefixes, etc.



Bit-by-Bit Weighted FQ

- Flows allocated different rates by servicing different number of bits for each flow during each round.



Order of service for the four queues:

$\ldots f_1, f_2, f_2, f_2, f_3, f_3, f_3, f_4, f_4, f_4, f_1, \ldots$

Packet vs. "Fluid" System

- Bit-by-bit FQ is not implementable:

  ...In real packet-based systems:
  - One queue is served at any given time
  - Packet transmission cannot be preempted

- **Goal:** A packet scheme close to fluid system
  - Bound performance w.r.t. fluid system

Packet-by-packet Fair Queuing (Weighted Fair Queuing)

Copes better with variable size packets & weights

Key Idea:

1. Determine the *finish time* of packets in bit-by-bit system, assuming no more arrivals
2. Serve packets in order of finish times

Implementing WFQ

Challenge: Determining finish time is hard

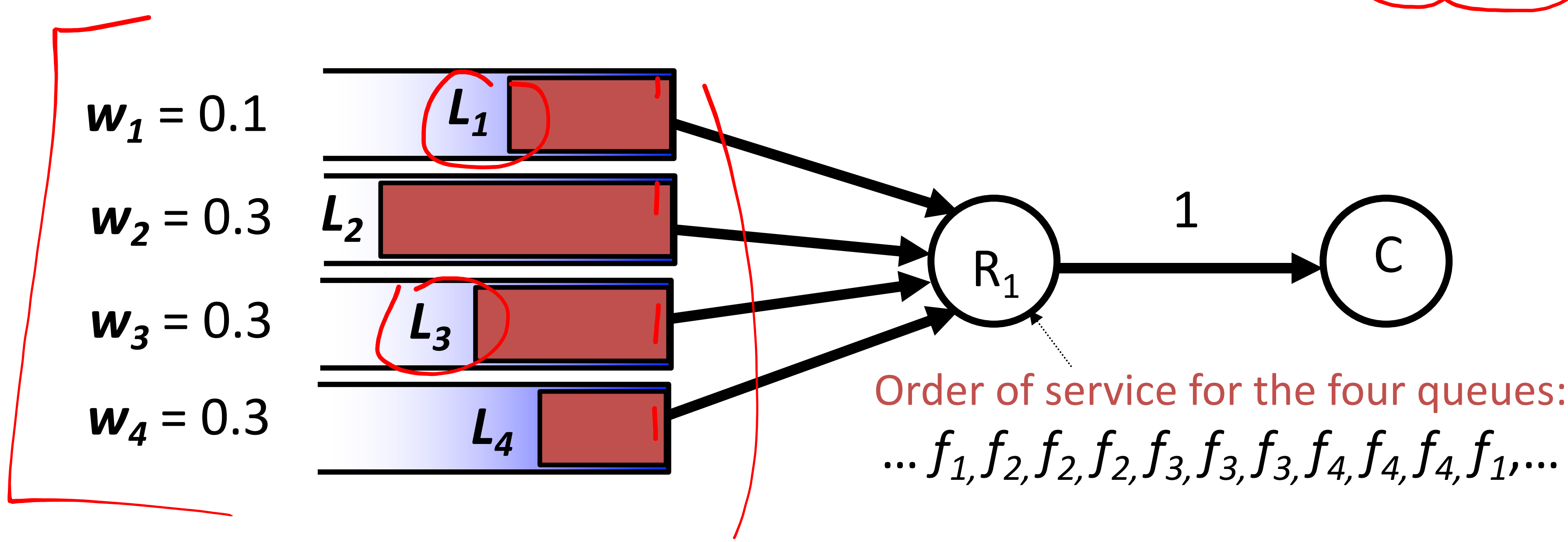
Idea: Don't need finish time. Need finish **order.**

The finish order is a lot easier to calculate.

Finish order

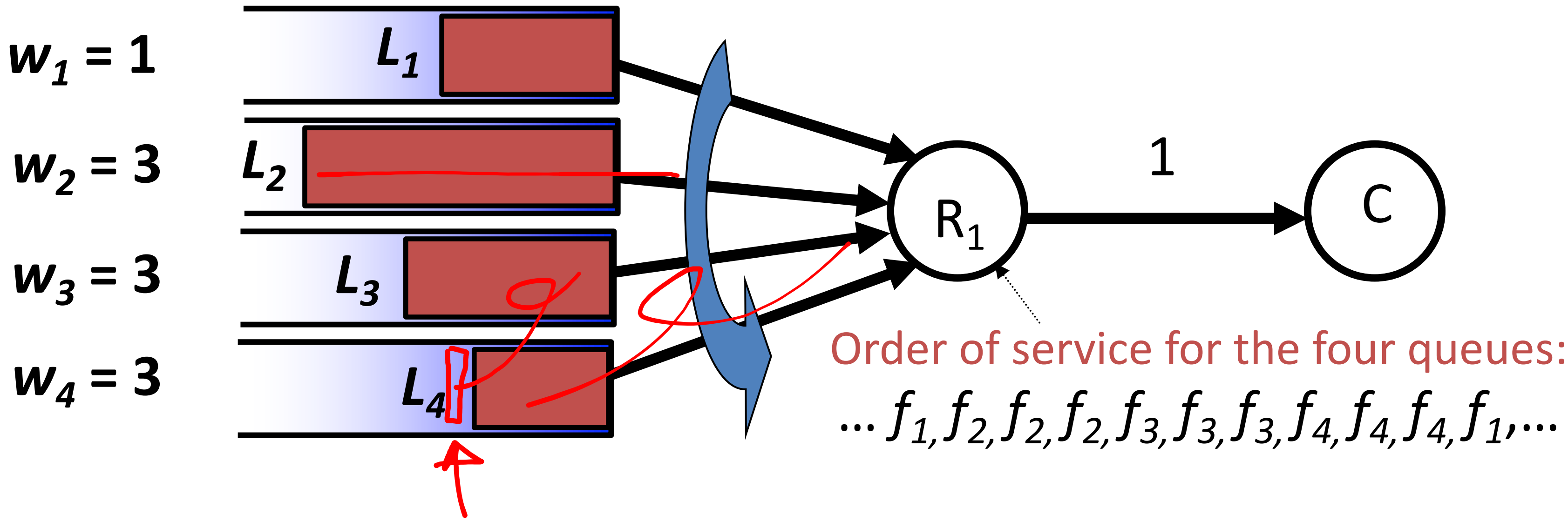
Let L_i be the length of the packet at the head of queue i

In what order do these packets finish? **Increasing L_i/w_i**



Does not change with future packet arrivals!

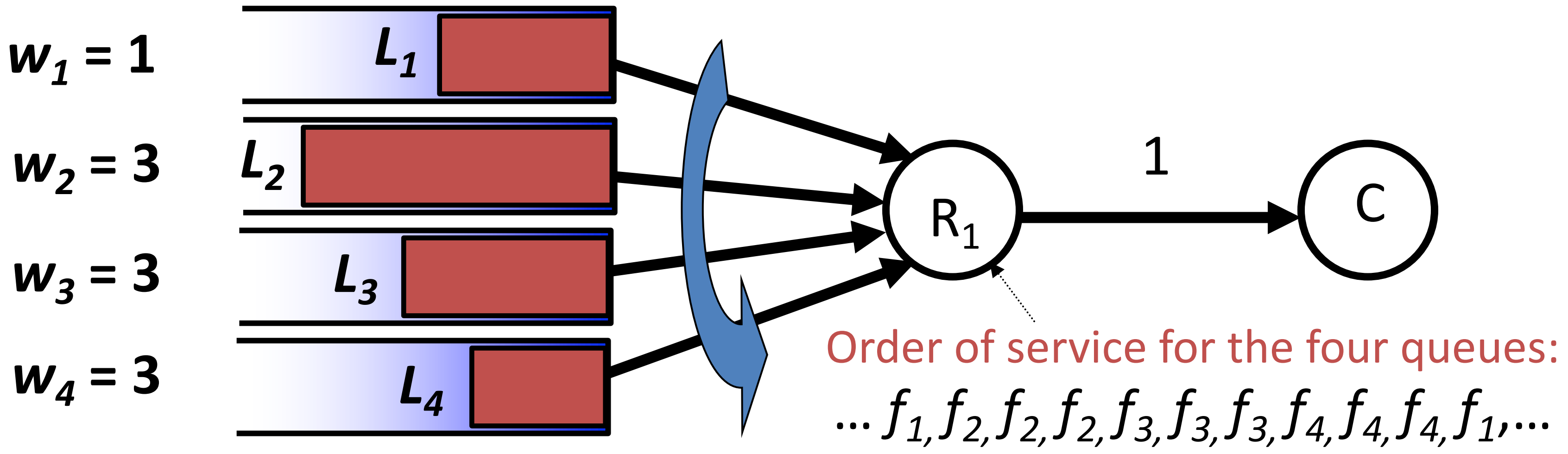
Bit-by-bit System Round



Round – One complete cycle through all the queues sending w_i bits per queue

Question: How many rounds does it take to serve a packet of length L from flow i?

Bit-by-bit System Round



Round – One complete cycle through all the queues sending w_i bits per queue

Packet of length L takes L/w_i rounds to serve

Round (aka. "Virtual Time") Implementation of WFQ

Question: What is *finish* round of k^{th} packet – F_i^k?

Round (aka "Virtual Time") Implementation of WFQ

Assign a start/finish round to each packet at arrival
→ serve packets in order of finish rounds

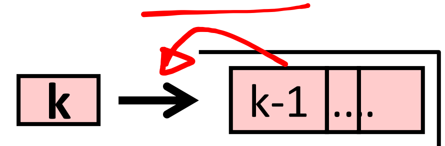
Suppose k^{th} packet of flow i arrives at time a

Finish round of *(k-1)*st *packet*

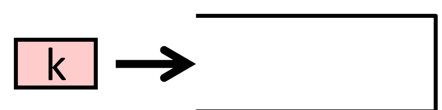
Start round of *k*th *packet*

$$S_i^k = \begin{cases} F_i^{k-1} \\ R(a) \end{cases}$$

Round number at time *a*



(Flow i backlogged)



(Flow i empty)

Putting it All Together

For k^{th} packet of flow i arriving at time a:

$$S_i^k = \max(F_i^{k-1}, R(a))$$

$$F_i^k = S_i^k + \frac{L_i^k}{w_i}$$

Simple approximation:

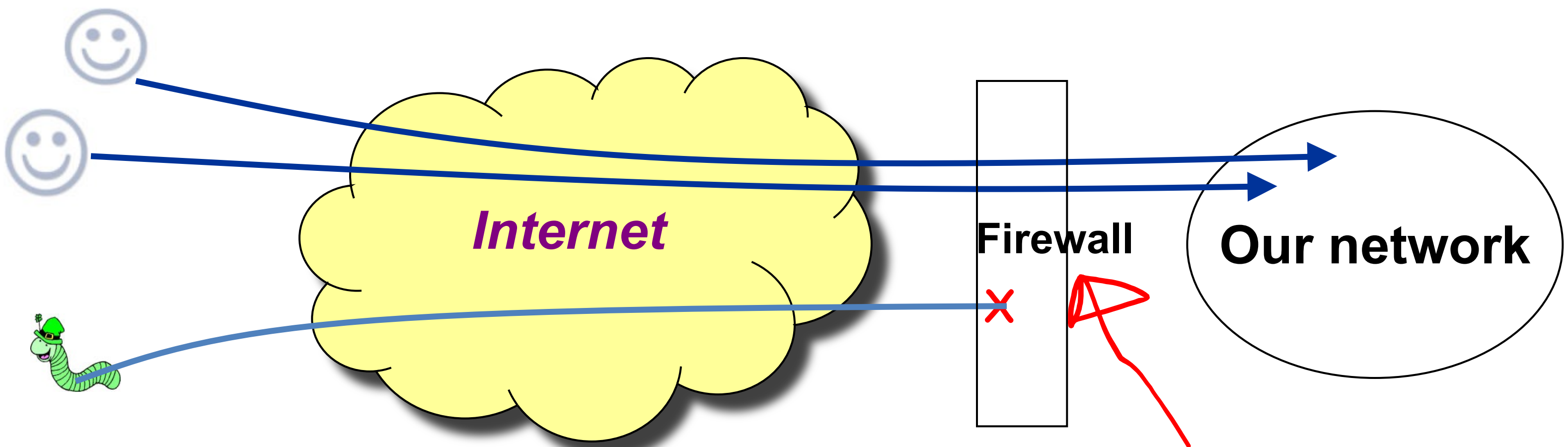
Set R(a) to start or finish round of packet currently in service

WFQ ::= Serve packets in order of finish round

Today

- Key concepts in Congestion Control
  - Retransmits and RTT estimator
  - Slow Start and Self-clocking
  - AIMD Congestion control
- Queue Management
- **Middleboxes**

End-to-end violation: Firewalls



- Box in middle of network that blocks "malicious" traffic
  - End-host software often vulnerable to remote-exploit malware
  - Users are **naive,** don't keep systems patched and up-to-date
- Firewalls clearly **violate the e2e principle**
  - **Endpoints** are capable of deciding what traffic to ignore
  - Firewall **entangled** with design of network and higher protocol layers and apps, and vice-versa
- **Yet, we probably do need firewalls**

Network Address Translation (NAT): Principled Objections

- Routers are not supposed to look at port #s
  - Network layer should care only about *IP* header
  - … and not be looking at the *port numbers* at all
- NAT violates the end-to-end argument
  - Network nodes should not modify the packets
- IPv6 is a cleaner solution
  - Better to migrate than to limp along with a hack

That's what happens when network puts power in hands of end users!

Middleboxes: Conclusions

- Middleboxes address important problems
  - Getting by with fewer IP addresses
  - Blocking unwanted traffic
  - Making fair use of network resources
  - Improving end-to-end performance
- Middleboxes cause problems of their own
  - No longer globally unique IP addresses
  - Cannot assume network simply delivers packets

Next Up in 461

Next Class Meeting

Lectures 9 (Routing Algorithms) and

10 (Routing Convergence)

Precepts this Thursday and Friday