Administrivia

- Review 1
  - Specific instructions to structure review in lecture 1 slides 26/27
- Office hours moving to Thursday 10—12
- MUD: Send me your top 1—3 questions on this lecture
- Think about use cases and directions for your project!

Congestion Control

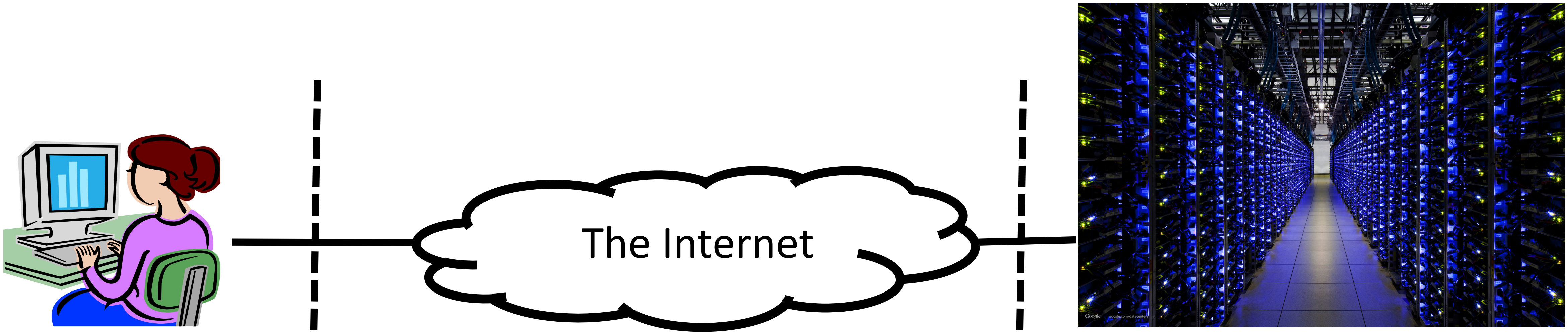
Lecture 4, Computer Networks (198:552)

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Edge vs. core division of labor

- Core: best-effort packet delivery

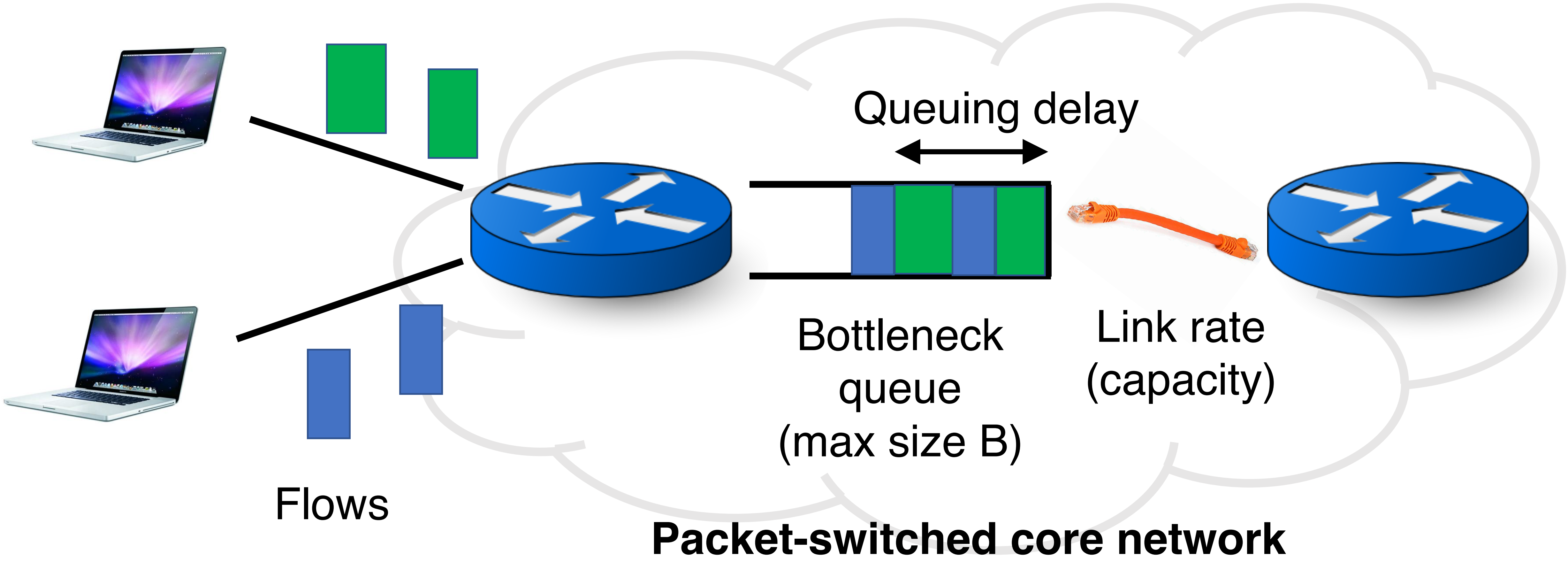
- Edge: end-to-end guarantees to applications
  - *Transport* should figure out how to use the core effectively



The role of the endpoint

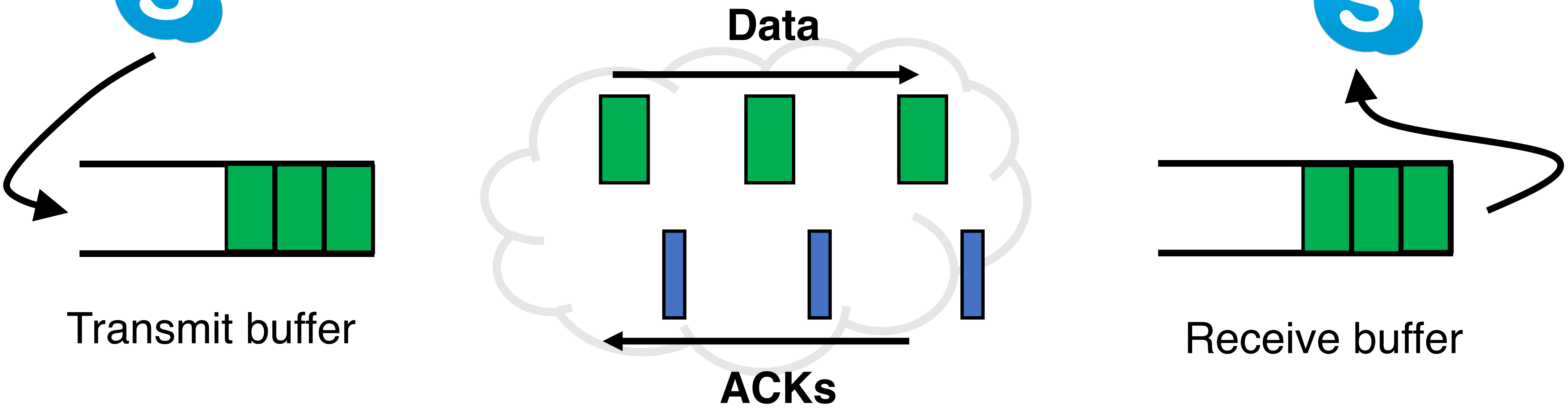
- Network discovery and bootstrapping
  - How does the endpoint join the network?
  - How does the endpoint get an address?
- Interface to networked applications
  - What interface to higher-level applications?
  - How does the host realize that abstraction?
- Distributed resource sharing
  - What roles does the host play in network resource allocation decisions?

Network model



TCP: Reliable & ordered transport

Reliable delivery using ACKs
Data ordered using sequence numbers
Receiver assembles data in order
before sending to the application

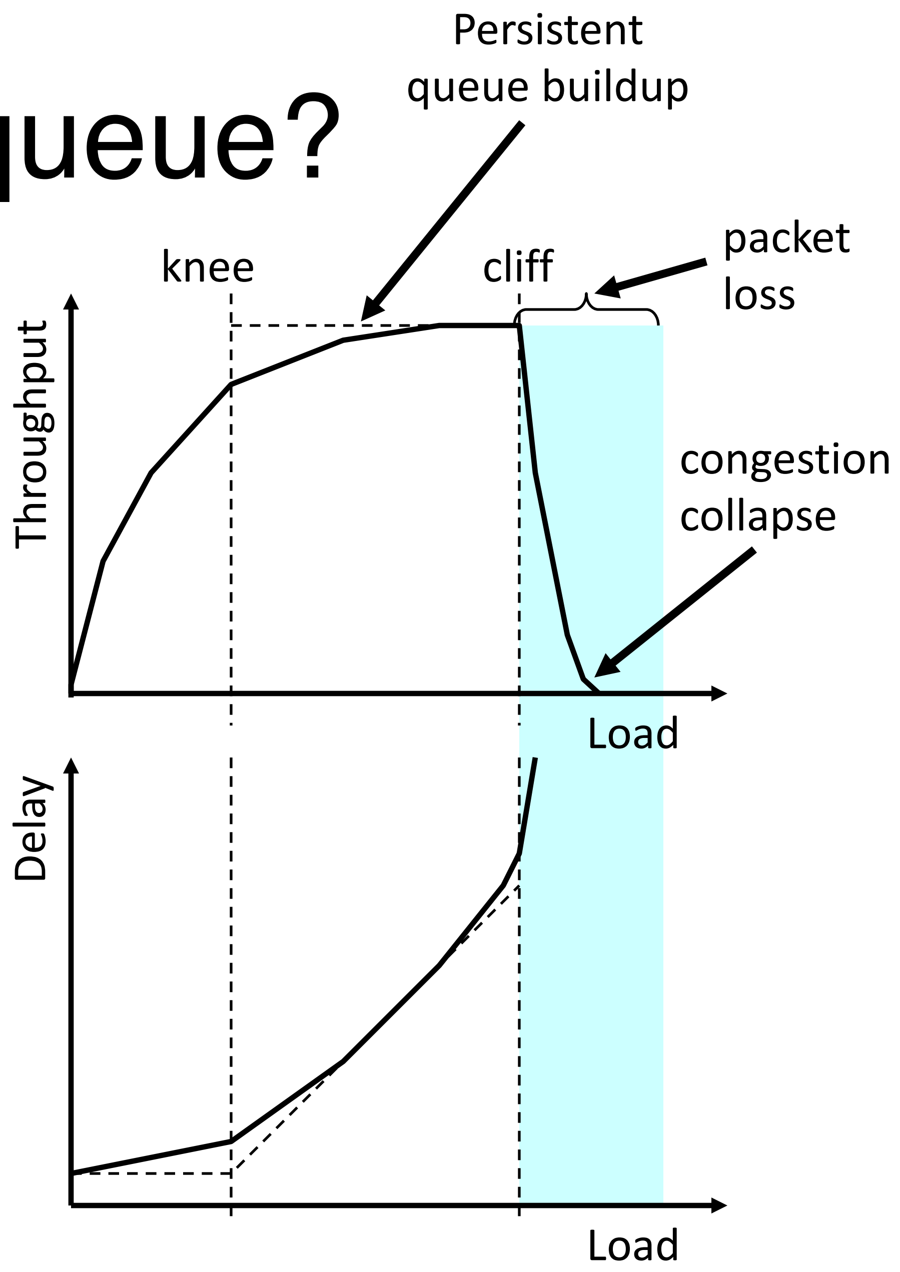


But what about performance?

- Throughput: the width of the pipe (MBytes/sec)
- Delay: the length of the pipe (milliseconds)
- Packet drop: leaks from the pipe (percentage of packets)

What happens at a queue?

- Knee – point after which
  - Throughput increases very slowly
  - Delay increases fast
- Cliff – point after which
  - Throughput starts to decrease very fast to zero
  - Delay approaches infinity
- For an M/M/1 queue
  - Delay = 1/(1 – utilization)



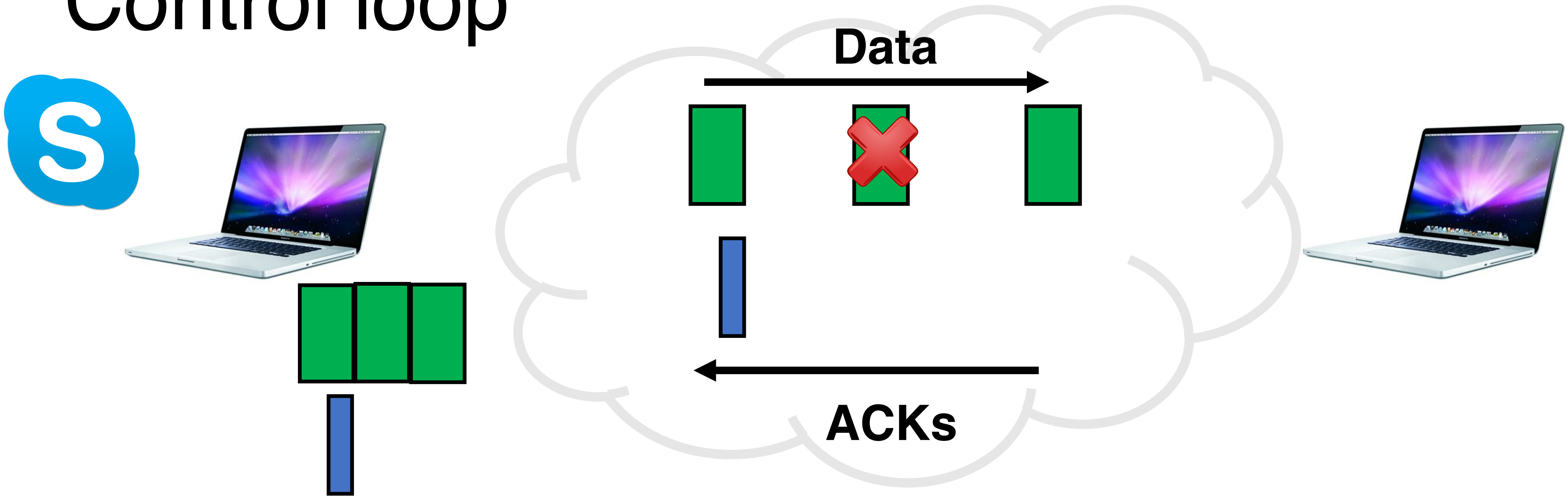
Flow ctrl vs. Cong ctrl vs. Cong avoidance

- Flow control:
  - Receiver tells sender how much it can accept in its buffer
- Congestion avoidance:
  - Trying to stay to the left of the knee
- Congestion control:
  - Trying to stay to the left of the cliff
- Final window == min (receiver window, *congestion window*)

How should an endpoint transmit data

… and not overwhelm queues?

Control loop



- Congestion window: transmitted yet unACKed data
- How should an endpoint adapt its congestion window
  - … based on *congestion signals* provided by the network?

Congestion signals

- Explicit network signals
  - Send packet back to source (e.g., ICMP source quench)
  - Set bit in header (e.g., ECN)
  - Send direct feedback (e.g., sending rate)
  - Unless on every router, still need end-to-end signal

- Implicit network signals
  - Loss (e.g. TCP New Reno, SACK)
  - Delay (e.g. TCP Vegas)
  - Easily deployable
  - Robustness?
    - Wireless?

Exercise

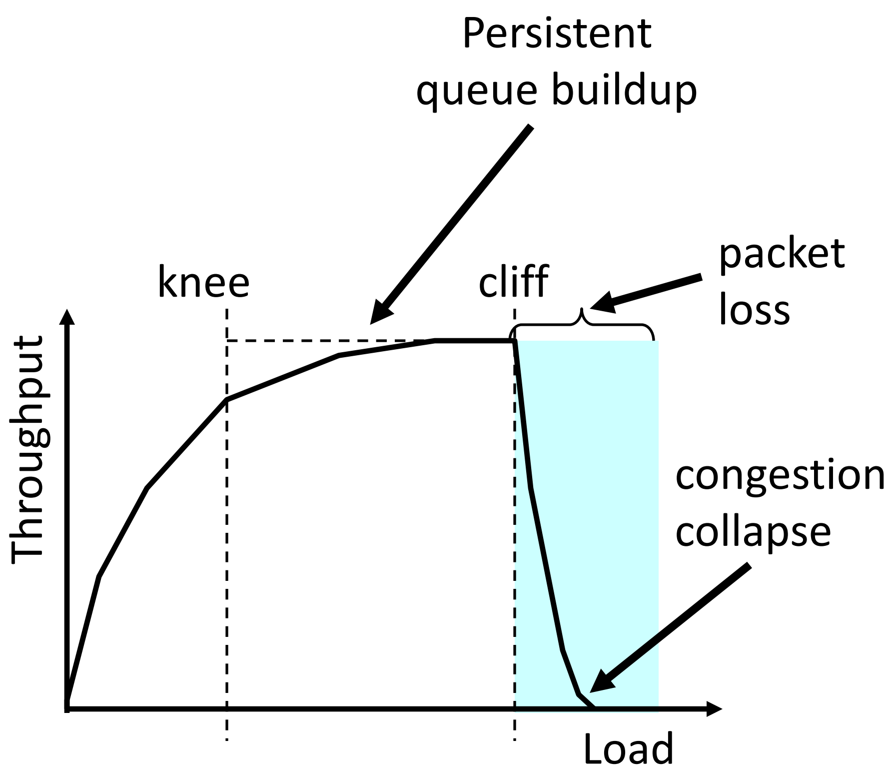
- Buffer size B at the bottleneck queue
- Link capacity C
- Propagation delay T
- What’s the congestion window for a flow at the knee? Cliff?
- Is there a network buffer size that’s always sufficient?
- Is there a receiver buffer size that’s always sufficient?

Congestion Avoidance and Control

ACM SIGCOMM ‘88

Van Jacobson and Michael Karels

One possible set of goals



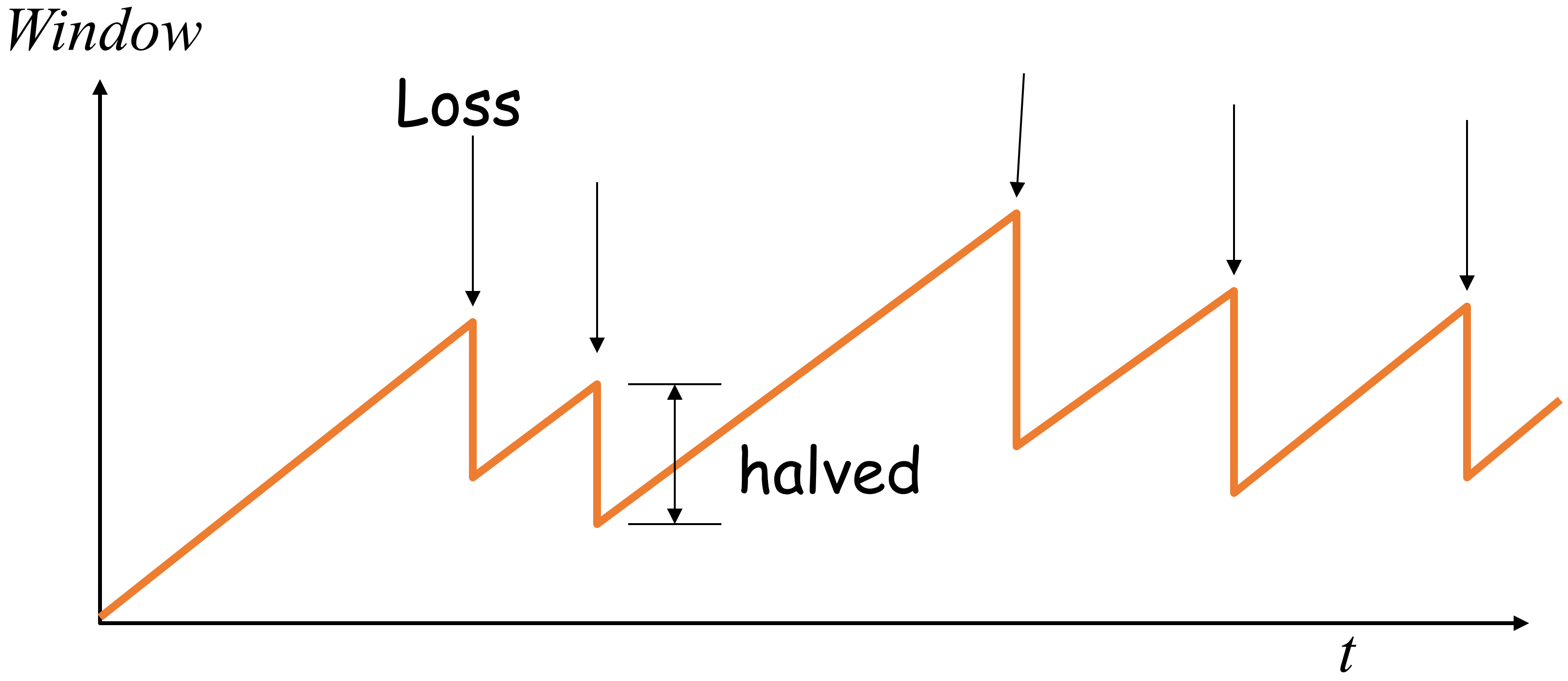
- Equilibrium: stay close to the “cliff”
- At equilibrium:
  - Don’t send a new packet
  - ... until current one leaves the network
- Return to equilibrium if you go “over” the cliff
  - Try to approach the cliff if below it
- Share bandwidth ‘fairly’ across flows

TCP New Reno: Mechanisms

- Packet conservation
- Reacting to loss: multiplicative decrease
- Probing for more throughput: additive increase
- Getting started: slow start
- A better retransmission timeout
- Fast retransmit
- Fast recovery

TCP congestion control

- Additive increase, multiplicative decrease (AIMD)
  - On packet loss, divide congestion window in half
  - On success for last window, increase window linearly



Mechanisms not shown: slow start, fast retransmit, recovery, timeout loss, etc.

Discussion of Jacobson’s paper

- Is loss always congestive?
  - Wireless networks
  - Errors and corruption?
- Is loss the best congestion signal?
  - How about delays?
  - Explicit congestion notifications from the network?
- Is packet conservation always a good thing?
  - Differences in flow propagation delays?
  - Detect incipient congestion?
- Why AIMD?

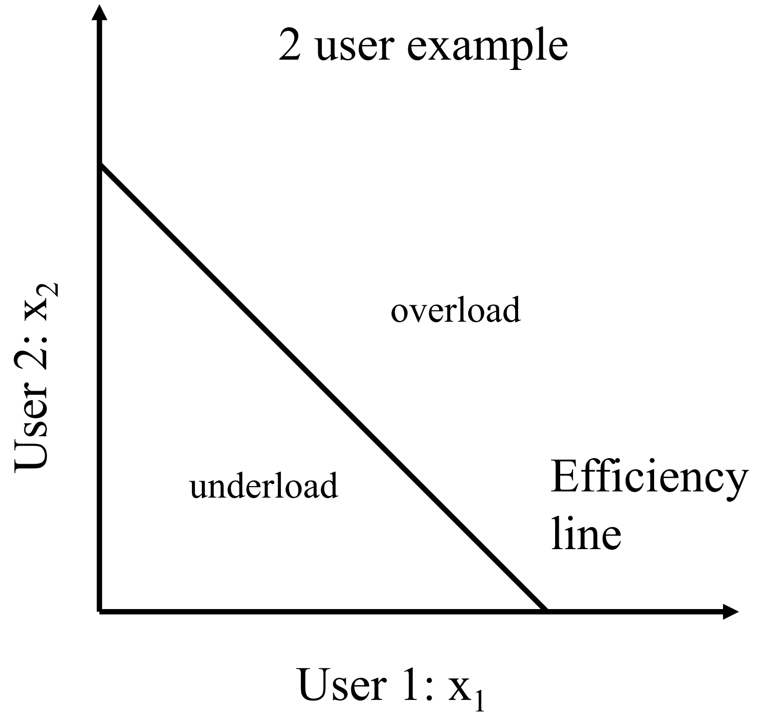
Why AIMD?

Some thoughts from Chiu and Jain's '89 paper,

[CJ89]: "An Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks"

Efficient allocation

- Too slow?
  - Underutilize the network
- Too fast?
  - High delays, lose packets
- Every endpoint is doing it
  - May *all* under/over shoot
  - Large oscillations
- Optimal:
  - $\Sigma x_i = X_{goal}$, e.g., link capacity
- Efficiency = 1 - distance from efficiency line

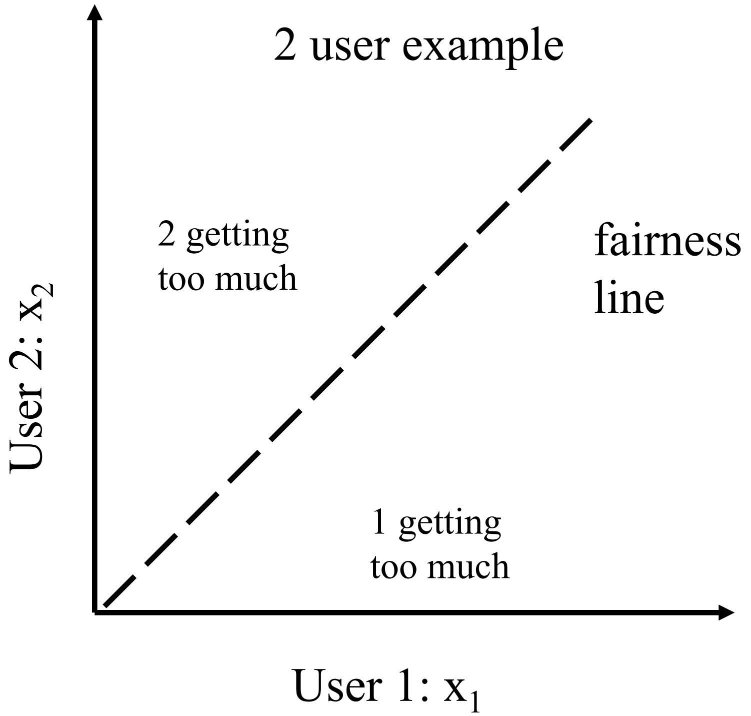


Fair allocation

- Max-min fairness
  - Flows which share the same bottleneck get the same amount of bandwidth

$$F(x) = \frac{\left(\sum x_i\right)^2}{n\left(\sum x_i^2\right)}$$

- Assumes no knowledge of priorities
- Fairness = 1 - distance from fairness line



Linear window adaptation rules

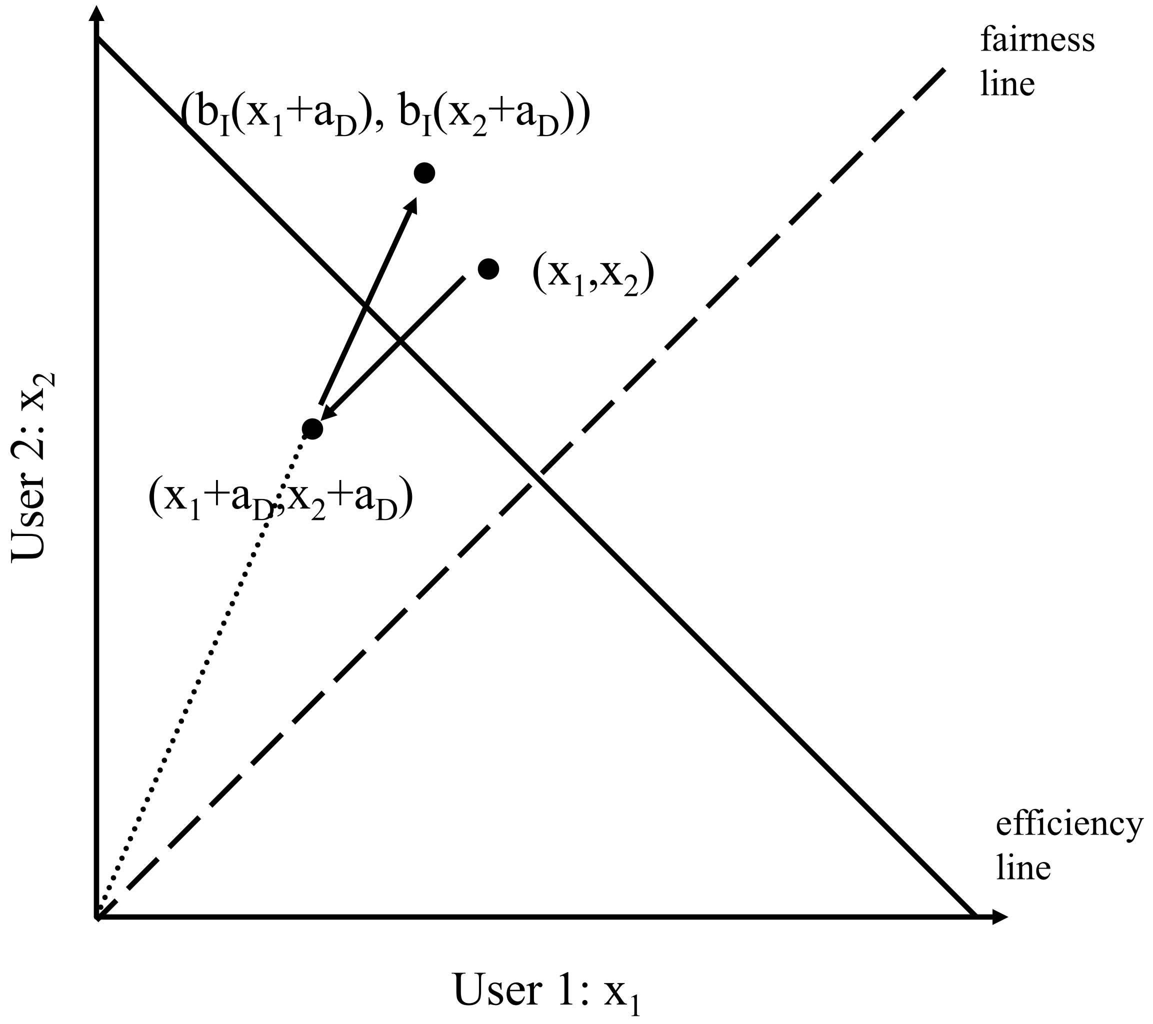
$$x_i(t+1) = \begin{Bmatrix} a_I + b_I x_i(t) \;\; increase \\ a_D + b_D x_i(t) \, decrease \end{Bmatrix}$$

- $x_i(t)$: window or rate of the i^{th} user at time t
- a_I, a_D, b_I, b_D: constant increase/decrease coefficients
- All users receive same network feedback
  - *Binary* feedback: sense congestion or available capacity
- All users increase or decrease simultaneously

Multiplicative increase, additive decrease

- Does not converge to fairness
  - Not stable at all
- Does not converge to efficiency
  - Stable iff

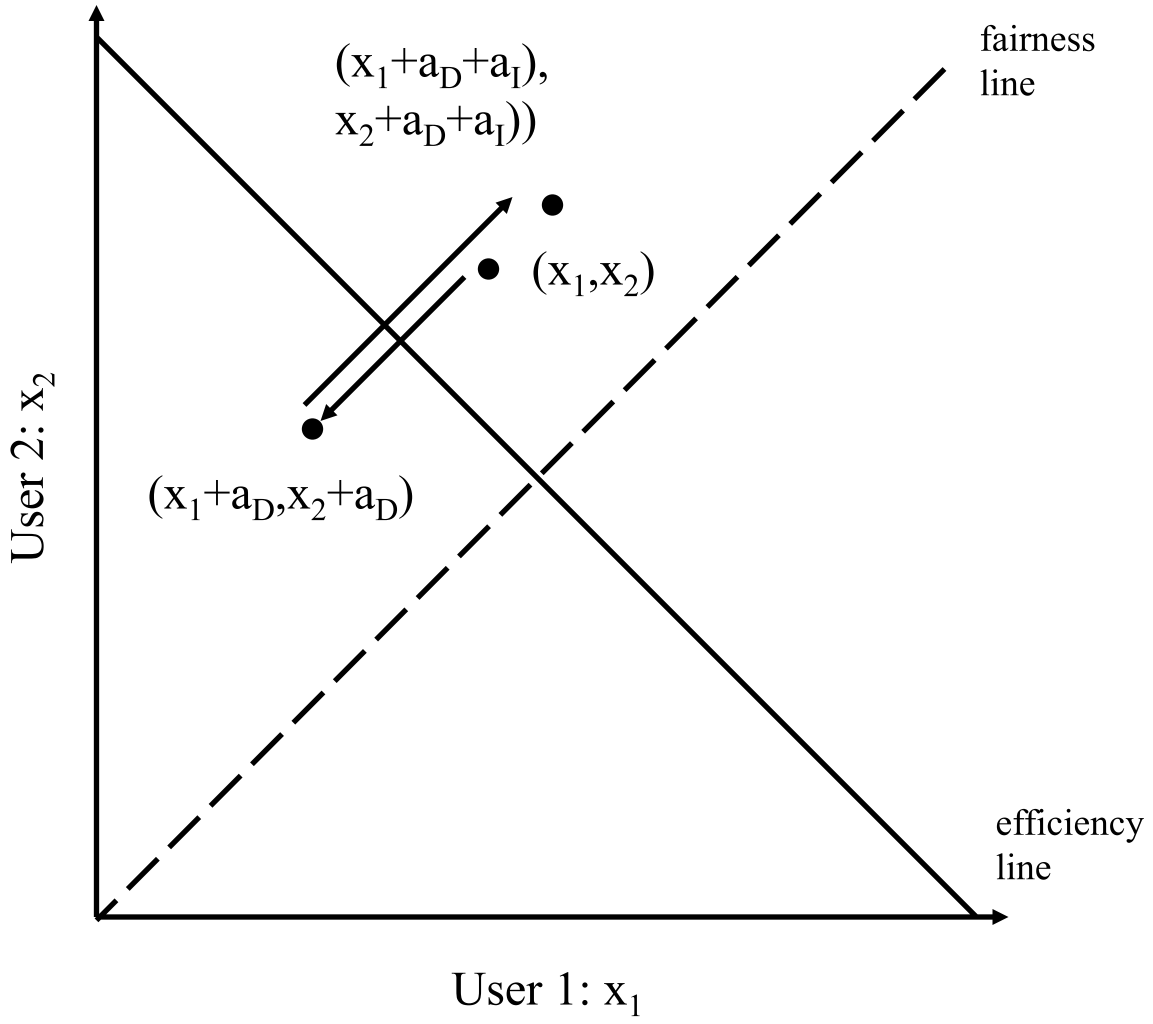
$$x_{1h} = x_{2h} = \frac{b_I a_D}{1 - b_I}$$



Additive increase, additive decrease

- Does not converge to fairness
  - Stable
- Does not converge to efficiency
  - Stable iff

$$a_D = a_I$$



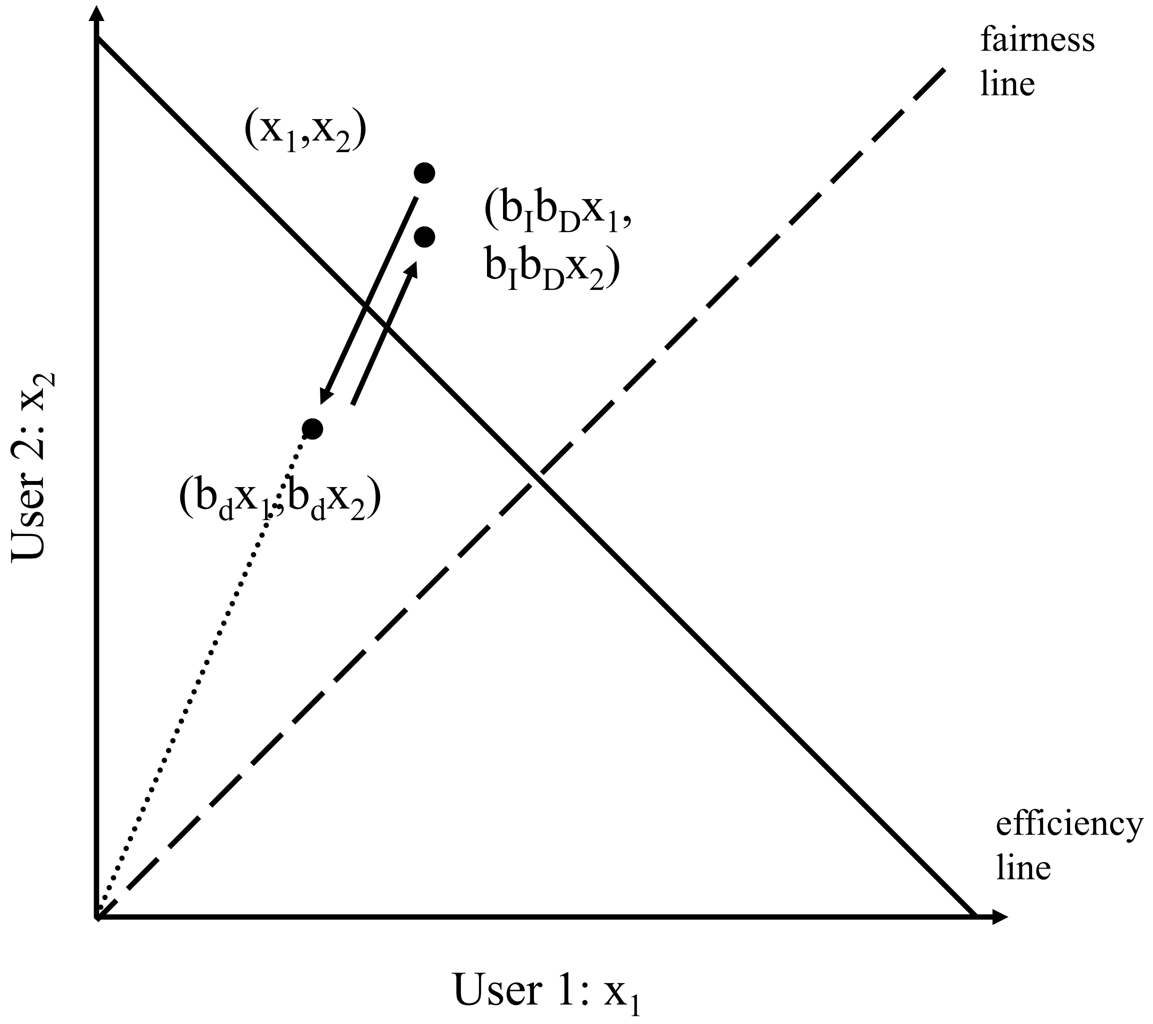
Multiplicative increase, mult. decrease

- Does not converge to fairness
  - Stable

- Converges to efficiency iff

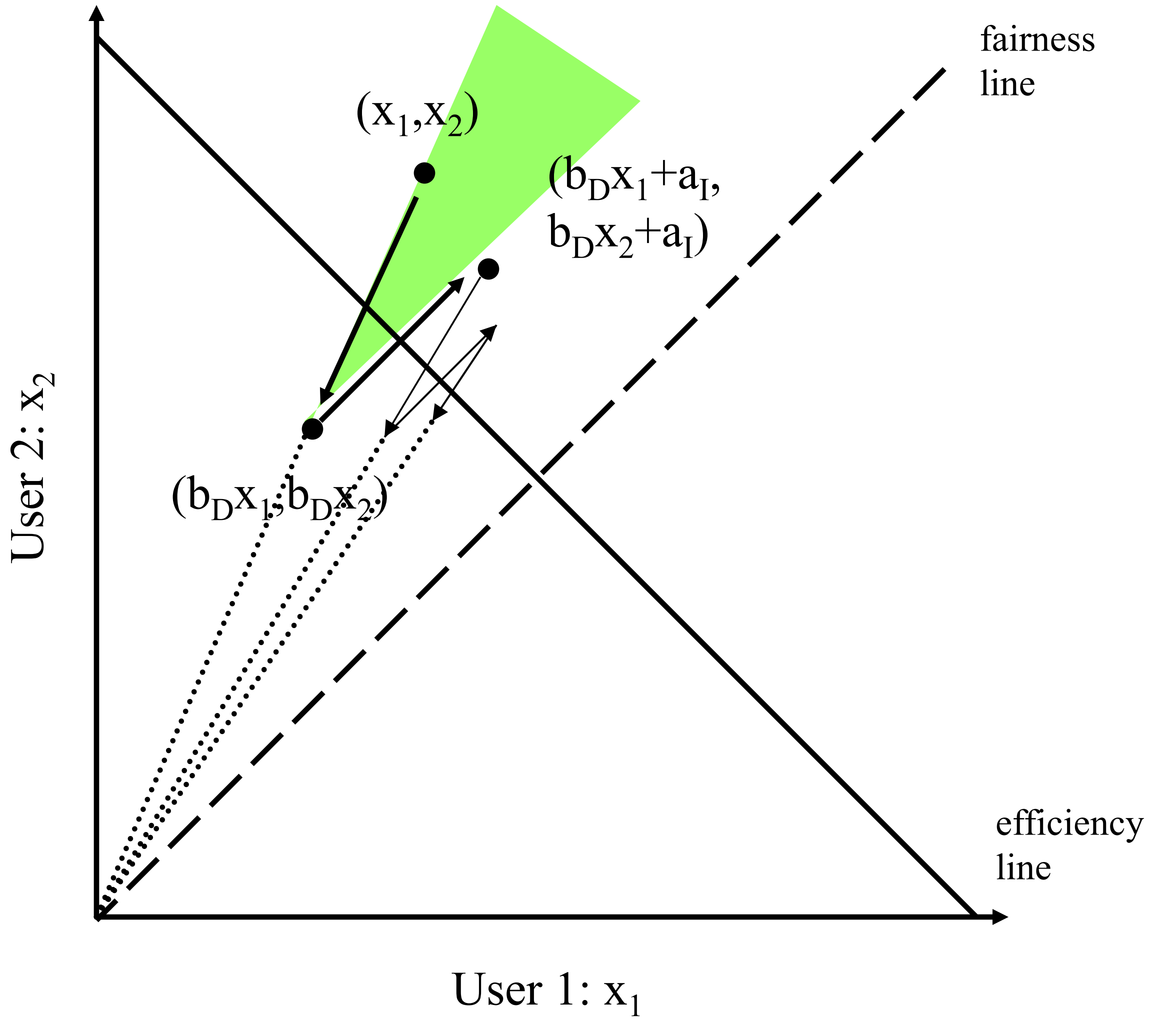
$$b_I \geq 1$$

$$0 \leq b_D < 1$$



Additive increase, multiplicative decrease

- Converges to fairness
- Converges to efficiency
- Increments smaller as fairness increases
  - Effect on metrics?



Significance of AIMD/CJ89

- Characteristics
  - Converges to efficiency, fairness
  - Easily deployable: feedback is binary
  - Fully distributed
  - No need to know the full state of the system (e.g., # users, link rates)

- [CJ89] result that enabled the Internet to grow beyond the ‘80s
  - A scalable Internet required a fully distributed congestion control
  - Formed the basis of TCP as we know it

Modeling

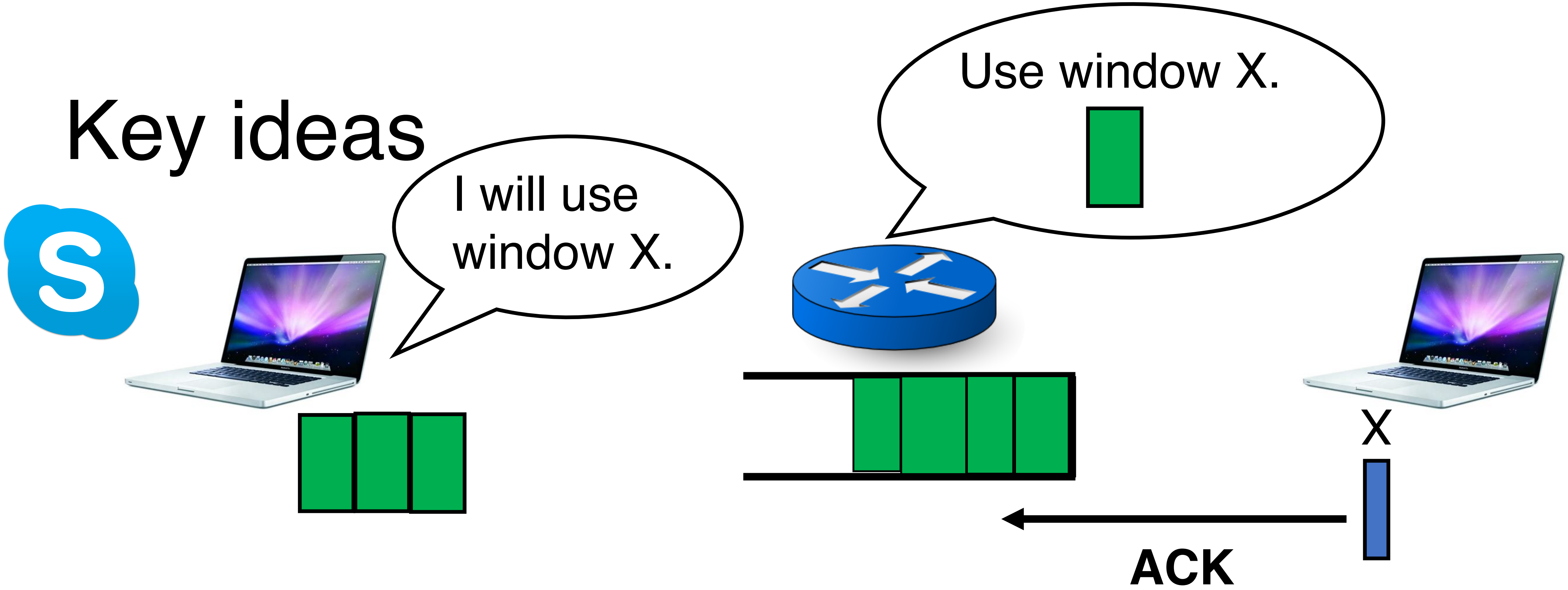
- Critical to understanding complex systems
  - [CJ89] model relevant even today (since '89!)
  - 10^6 increase of bandwidth, 1000x increase in number of users
- Criteria for good models
  - Realistic
  - Simple
    - Easy to work with
    - Easy for others to understand
  - Realistic, complex model $\rightarrow$ useless
  - Unrealistic, simple model $\rightarrow$ may teach something!

Congestion Control for High Bandwidth-Delay Product Networks

ACM SIGCOMM ‘02

Dina Katabi, Mark Handley, and Charlie Rohrs

Key ideas



- An explicit control system running in the network
  - Give direct, *multi-bit* feedback on transmission windows/rates to endpoints
- Separate efficiency control from fairness

XCP efficiency control loop

$$\phi = \alpha \cdot d \cdot \underbrace{S}_{\text{Spare bandwidth}} - \beta \cdot \underbrace{Q}_{\text{Persistent queue}}$$

- Sender (roughly): cwnd = cwnd + ϕ
- Why do you need both terms of this equation?

XCP fairness rules

- If $\phi > 0$, increase in throughput of all flows must be the same
- If $\phi < 0$, decrease in throughput must be proportional to a flow's current throughput
- Bandwidth shuffling: if too close to convergence, perturb by adding a small window to over-utilize the link

Discussion of XCP

- What window increase/decrease rules are used by XCP?
  - Is it fair?
- Why couldn’t you separate efficiency & fairness in TCP New Reno?
  - Is generating multi-bit feedback harder than binary?
  - What about the end-to-end argument?
- Deployment concerns?
  - Ensuring control loop runs at every bottleneck router?
  - Selfish or rogue endpoints?
- Should you necessarily push queue size to zero?
  - Is it always possible to keep the pipe full?
  - How to accommodate noise in measurements or window sizes?

Some questions to think about…

- What should networks provide as feedback for endpoints?
- Should the network operate at the cliff, the knee, or elsewhere?
- What is a good definition of fairness? Take into account:
  - Demands? Usage of multiple resources? App goals?
- What about hosts who cheat to hog resources?
  - How to detect cheating? How to prevent/punish?
- What about wireless networks?
  - Loss caused by interference, not just congestion
  - Difficulty of detecting collisions (due to fading)

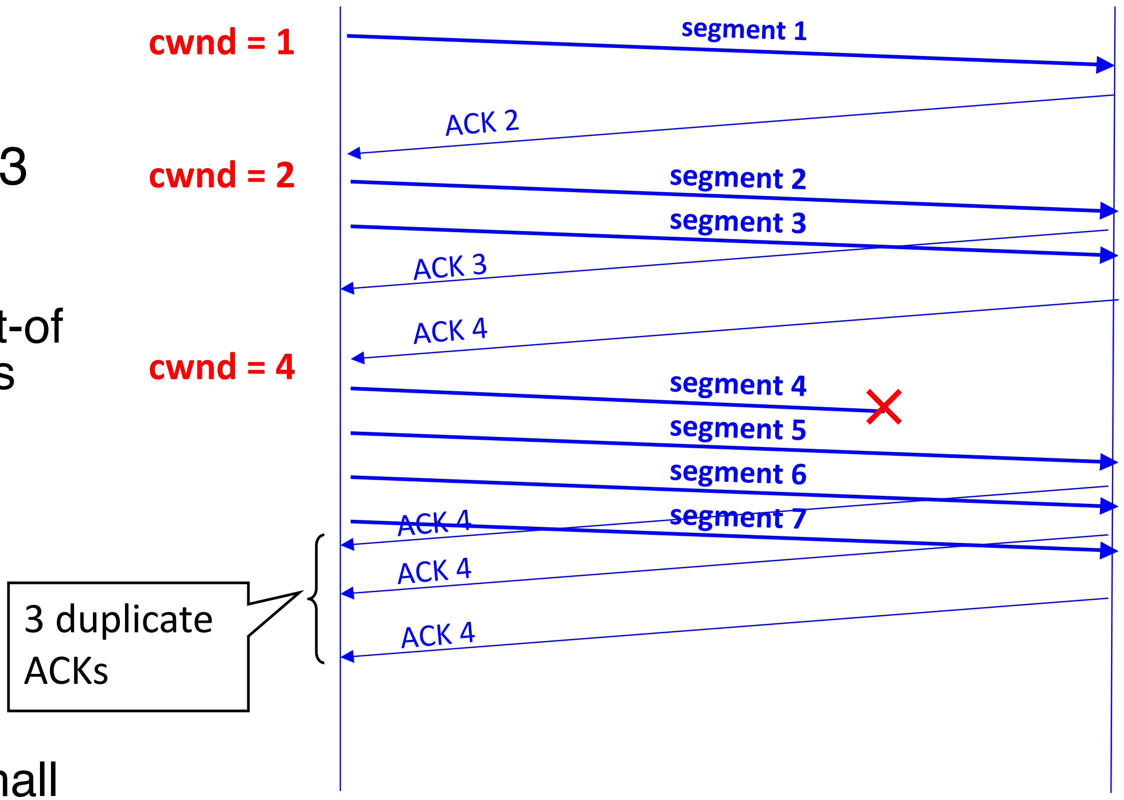
Backup slides

Proposition 1 from [Chiu and Jain '89]

- In order to satisfy the requirements of distributed convergence to efficiency and fairness without truncation,
- the linear decrease policy should be multiplicative, and
- the linear increase policy should always have an additive component, and
- optionally it may have a multiplicative component with the coefficient no less than one.

Fast Retransmit

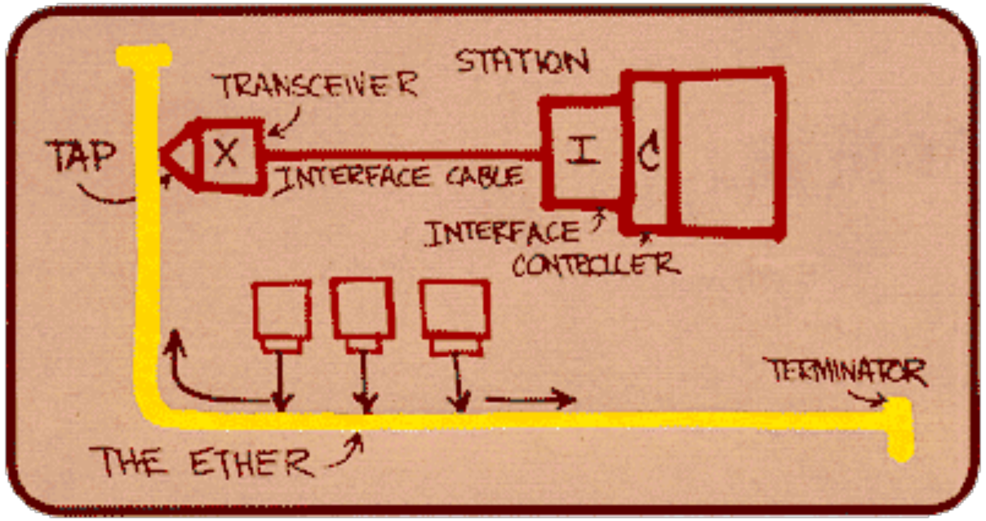
- Don’t wait for window to drain
- Resend a segment after 3 duplicate ACKs
    - remember a duplicate ACK means that an out-of sequence segment was received
- Notes:
    - duplicate ACKs due to packet reordering
        - why reordering?
    - iwindow may be too small to get duplicate ACKs



Fast Recovery

- After a fast-retransmit set *cwnd* to *ssthresh/2*
  - i.e., don’t reset *cwnd* to 1
- But when RTO expires still do *cwnd* = 1
- Fast Retransmit and Fast Recovery → implemented by TCP Reno; most widely used version of TCP today

Ethernet back-off mechanism



- Carrier sense: wait for link to be idle
  - If idle, start sending; if not, wait until idle
- Collision detection: listen while transmitting
  - If collision: abort transmission, and send jam signal
- Exponential back-off: wait before retransmitting
  - Wait random time, exponentially larger on each retry