Project 2: Transport

- You will implement the core parts of a TCP socket (Discussion#3)
- Use a network simulator (by Murphy McCauley & others at NetSys) to test, validate, and interact with your socket implementation
- The project is split and scored by (9) stages
- The goal is to guide you through the basic procedures of the TCP protocol, e.g., three-way handshake, reassembly of out-of-order packets, packet retransmission, and passive/active close
- Due: 11:59pm, Nov. 11th. Logistics & OH will be announced on Ed

Announcement#1: Lectures 18-21

- Will release lecture recordings by Murphy
- Topics: DNS, HTTP, Ethernet, discovery protocols
- No in-person lectures: 10/27, 11/1, 11/3
- Flipped lecture on 11/08
- Reminder and details will be posted on Ed

Congestion Control: Advanced Topics

CS 168

http://cs168.io

Sylvia Ratnasamy

Last Time

- The gory details of TCP CC

Today

- Modeling TCP
- Critiquing TCP
- Router-assisted CC

- We'll cover a broad range of design ideas
- Focus on the *why* and key insight behind the *how*
- Don't worry about the details

TCP Throughput Equation

TCP Throughput

- Given a path, what TCP throughput can we expect?
- We'll derive a simple model that expresses TCP throughput in terms of path properties:
  - RTT
  - Loss rate, p

A Simple Model for TCP Throughput

- Assume loss occurs whenever CWND reaches W_{max}
- And is detected by duplicate ACKs (i.e., no timeouts)

- Hence, evolution of window size:
  - $\frac{1}{2}W_{max}$ (after detecting loss)
  - $\frac{1}{2}W_{max}$ +1 (one RTT later)
  - $\frac{1}{2}W_{max}$ +2 (two RTTs later)
  - $\frac{1}{2}W_{max}$ +3 (three RTTs later)
  - ...
  - W_{max} [drop]
  - $\frac{1}{2}W_{max}$
  - $\frac{1}{2}W_{max}$ +1

A Simple Model for TCP Throughput

- Assume loss occurs whenever CWND reaches W_{max}
- And is detected by duplicate ACKs (i.e., no timeouts)

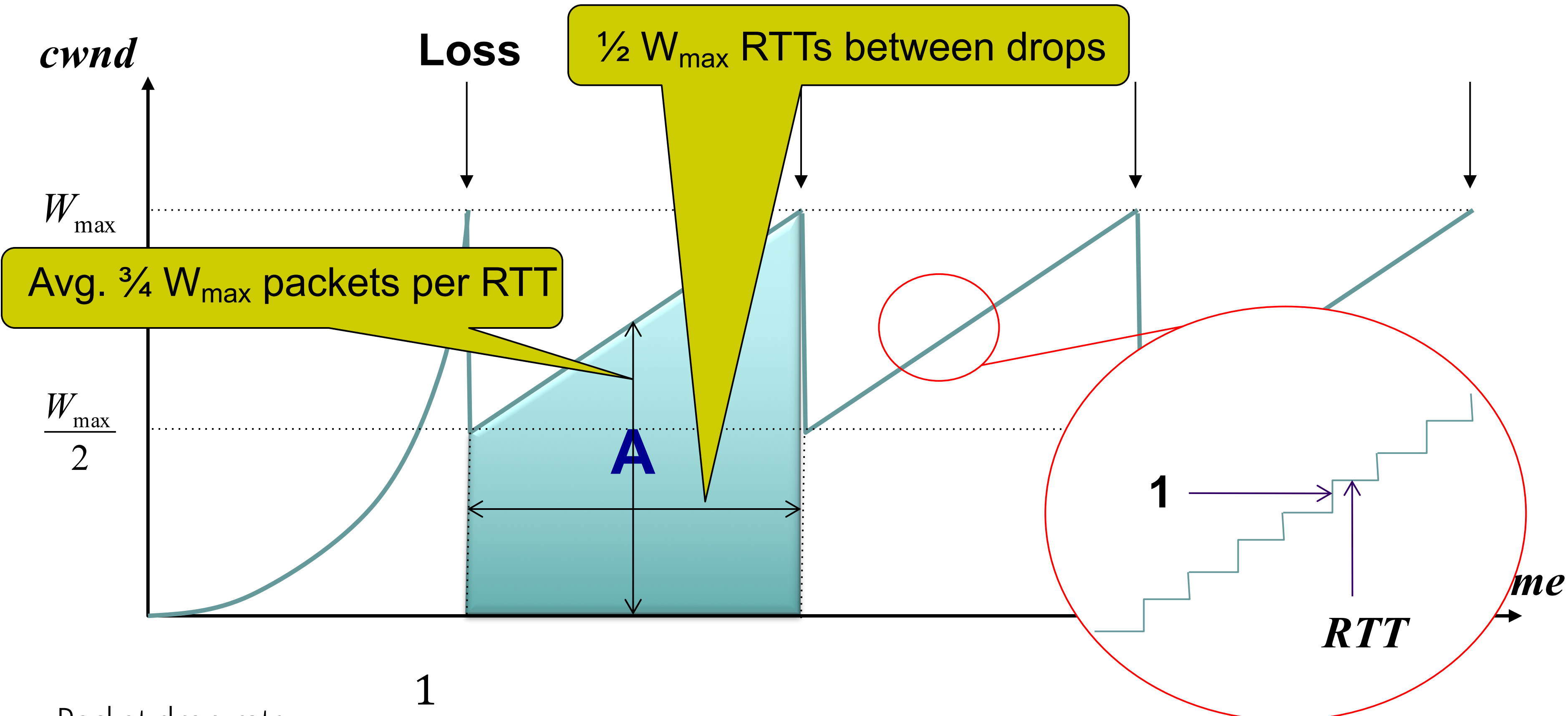
- Hence, evolution of window size:
  - $\frac{1}{2}W_{max}$, $\frac{1}{2}W_{max}$ +1, $\frac{1}{2}W_{max}$ +2 , ... , W_{max} [drop] , $\frac{1}{2}W_{max}$, $\frac{1}{2}W_{max}$ +1 , ...
  - Increase by 1 for $\frac{1}{2}W_{max}$ RTTs, then drop, then repeat

- Average window size per RTT = $\frac{3}{4}W_{max}$

- Average throughput = $\frac{3}{4}W_{max}$ x $\frac{MSS}{RTT}$

- Remaining step: express W_{max} in terms of loss rate p

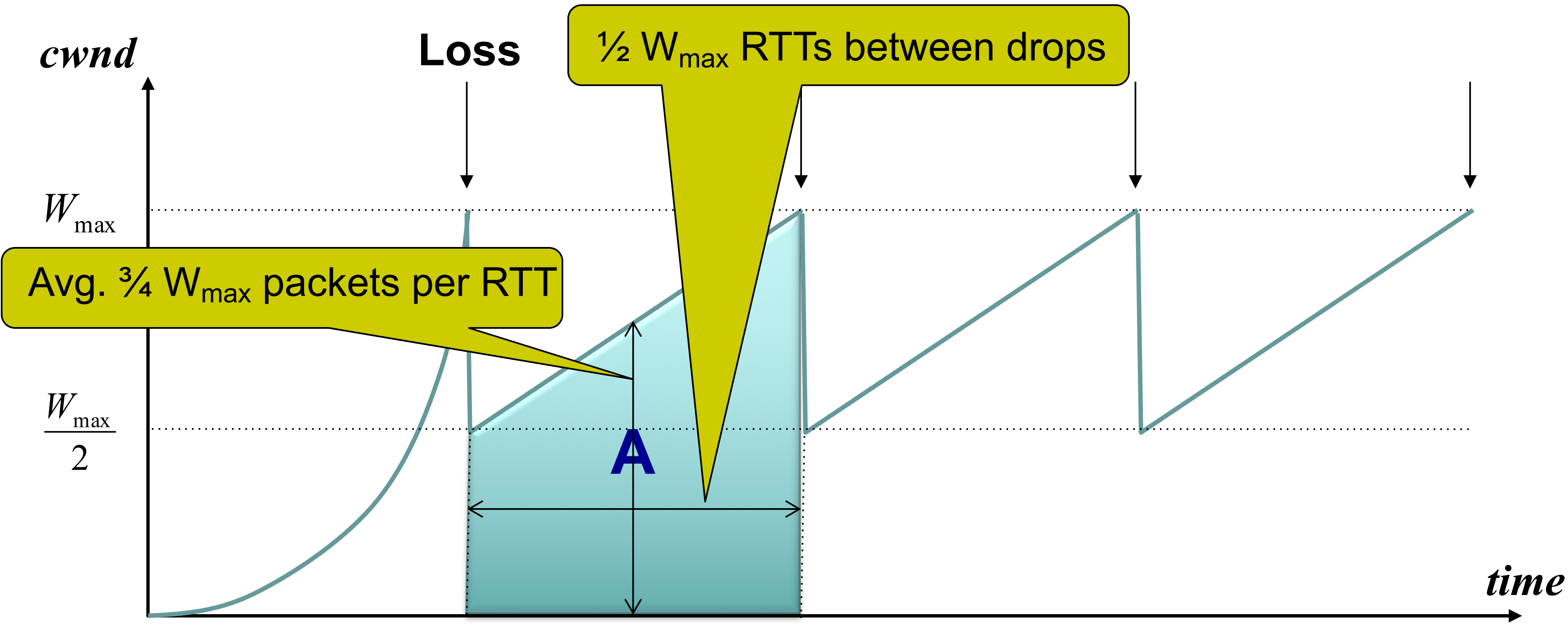
A Simple Model for TCP Throughput



Packet drop rate, $p = \frac{1}{A}$

On average, one of all packets in shaded region is lost (i.e., loss rate is 1/A, where A is #packets in shaded region)

A Simple Model for TCP Throughput



Packet drop rate, $p = \frac{1}{A}$

A = $\frac{3}{8} W_{max}^2$ $\rightarrow W_{max} = \frac{2\sqrt{2}}{\sqrt{3p}}$

Average Throughput = $\frac{\frac{3}{4} W_{max} \times MSS}{RTT} = \sqrt{\frac{3}{2}} \frac{\text{MSS}}{\text{RTT}\sqrt{p}}$

TCP Throughput

- Given a path, what TCP throughput can we expect?

- TCP throughput is proportional to $\frac{1}{\mathrm{RTT}}$ and $\frac{1}{\sqrt{p}}$
  - RTT is path round-trip time and p is the packet loss rate

- Model makes many simplifying assumptions
  - Ignores slow-start, assumes fixed RTT, isolated loss, *etc.*

- But leads to some insights (coming up)

Taking Stock: TCP CC

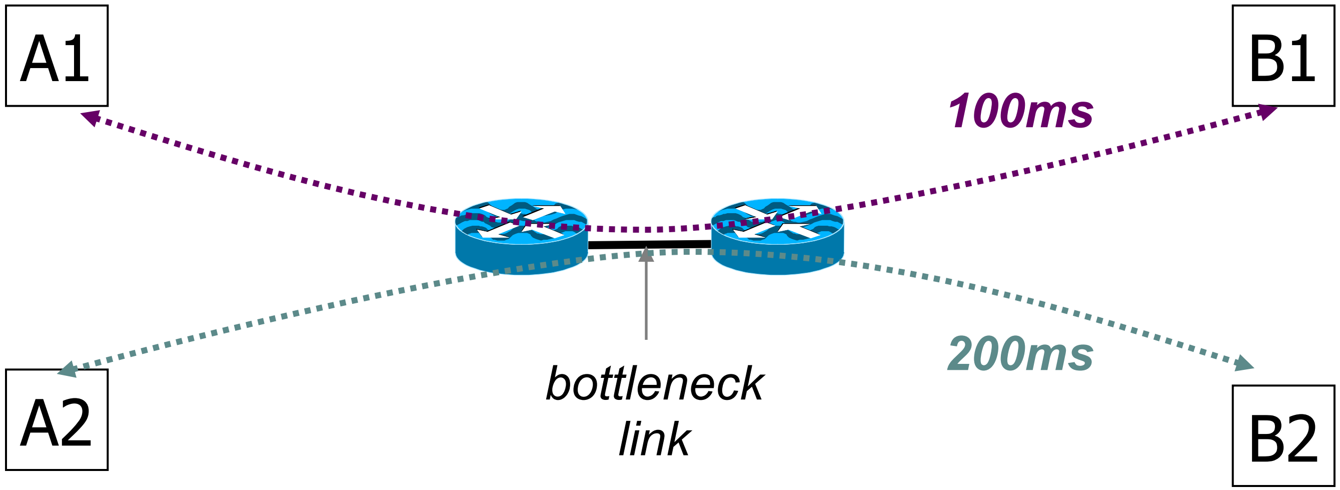
- (Sender) host based
- Loss based
- Adapts every RTT
- Starts out in slow start (start small, double every RTT)
- Adapts based on AIMD (gentle increase, rapid decrease)
- TCP throughput depends on path RTT and loss rate

$$\text{Throughput} = \sqrt{\frac{3}{2}}\,\frac{\mathrm{MSS}}{\mathrm{RTT}\sqrt{p}}$$

Implications (1): Different RTTs

$$\text{Throughput} = \sqrt{\frac{3}{2}}\,\frac{\text{MSS}}{\text{RTT}\sqrt{p}}$$

- Flows get throughput inversely proportional to RTT
- TCP unfair in the face of heterogeneous RTTs!



Implications (2): High Speed TCP

$$\text{Throughput} = \sqrt{\frac{3}{2}} \frac{\text{MSS}}{\text{RTT}\sqrt{p}}$$

- Assume **BW=100Gbps**, RTT = 100ms, MSS=1500B
- Value of *p* required to reach 100Gbps throughput: 2×10^{-12}
  - Requires dropping only one out of 50 billion packets!
  - Going ~16.6 hours between drops
- These are not practical numbers
- Problem: scaling a single flow to high throughput is **very** slow with additive increase

HighSpeed TCP [RFC 3649]

- Once past a threshold speed, increase CWND faster
  - Make the increase rule a function of CWND

- Other approaches?
  - Multiple simultaneous connections (workaround)
  - Router-assisted approaches (will see shortly)

Implications (3): ***Rate***-based CC [RFC 5348]

$$\text{Throughput} = \sqrt{\frac{3}{2}}\,\frac{1}{\mathrm{RTT}\sqrt{p}}$$

- TCP throughput is "choppy"
  - repeated swings between W/2 to W
- Some apps would prefer sending at a steady rate
  - e.g., streaming apps
- A solution: Equation-based Congestion Control
  - ditch TCP's increase/decrease rules and just follow the equation
  - measure RTT and drop percentage p, and set rate accordingly
- Following the TCP equation ensures we're "TCP friendly"
  - i.e., use no more than TCP does in similar setting

Other Limitations of TCP Congestion Control

(4) Loss not due to congestion?

- TCP will confuse corruption with congestion
- Flow will cut its rate
  - Throughput ~ $\frac{1}{\sqrt{p}}$ even for non-congestion losses!

(5) How do short flows fare?

- 50% of flows have < 1500B to send; 80% < 100KB

- Implication (1): many flows never leave slow start!
  - Short flows never attain their fair share
  - In fact, short flows are likely to suffer unduly long transfer times

- Implication (2): too few packets to trigger dupACKs
  - Isolated loss may lead to timeouts
  - At typical timeout values of ~500ms, might severely impact flow completion time

- A partial fix: use a higher initial CWND [Google IW10]

(6) TCP fills up queues → long delays

- A flow deliberately overshoots capacity, until it experiences a drop
- Recall: loss follows delay (i.e,. queue *must* fill up)
- Means that delays are large, for *everyone*
  - Consider a flow transferring a 10GB file sharing a bottleneck link with 10 flows transferring 100B
- Problem exacerbated by the trend towards adding large amounts of memory on routers (a.k.a. "bufferbloat")

(6) TCP fills up queues → long delays

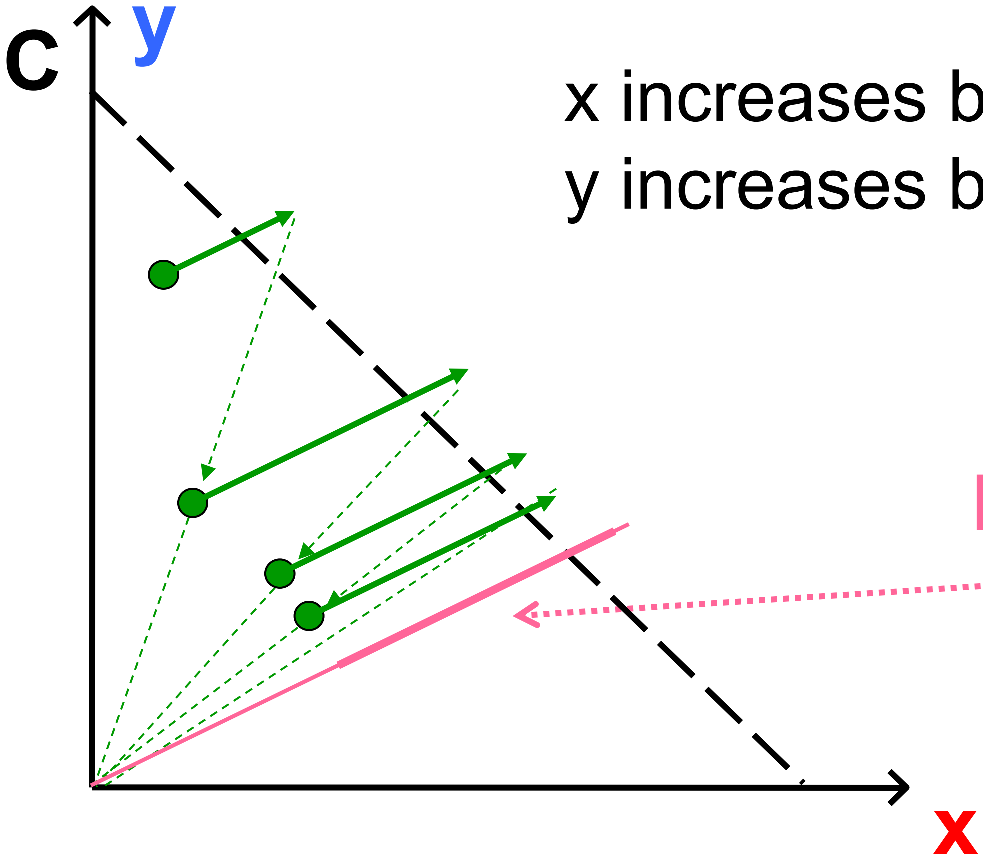
- Focus of Google's BBR algorithm[1]
- Basic idea (simplified):
  - Sender learns its minimum RTT (~ propagation RTT)
  - Decreases its rate when the observed RTT exceeds the minimum RTT

[1] *BBR: Congestion-Based Congestion Control; Cardwell et al, ACM Queue 2016*

(7) Cheating

- Three easy ways to cheat
  - Increasing CWND faster than +1 MSS per RTT

Increasing CWND Faster



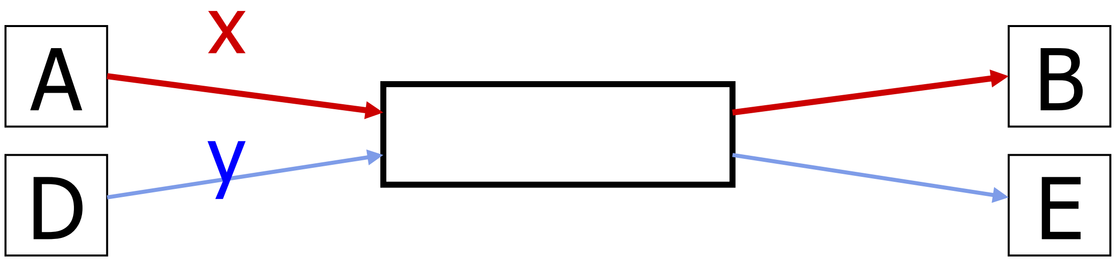
x increases by 2 per RTT
y increases by 1 per RTT

Limit rates:
x = 2y

(7) Cheating

- Three easy ways to cheat
  - Increasing CWND faster than +1 MSS per RTT
  - Opening many connections

Open Many Connections



Assume

- A starts 10 connections to B
- D starts 1 connection to E
- Each connection gets about the same throughput

Then A gets 10 times more throughput than D

(7) Cheating

- Three easy ways to cheat
  - Increasing CWND faster than +1 MSS per RTT
  - Opening many connections
  - Using large initial CWND

Why hasn't the Internet suffered another congestion collapse?

- Even "cheaters" do back off!
  - Leads to unfairness, not necessarily collapse
- Hard to say whether unfair behavior is common

MOTHERBOARD
TECH BY VICE

Google's Network Congestion Algorithm Isn't Fair, Researchers Say

(8) CC intertwined with reliability

- Mechanisms for CC and reliability are tightly coupled
  - CWND adjusted based on ACKs and timeouts
  - Cumulative ACKs and fast retransmit/recovery rules
- Complicates evolution
  - Consider changing from cumulative to selective ACKs
  - A failure of modularity, not layering
- Sometimes we want CC but not reliability
  - e.g., real-time applications
- Sometimes we want reliability but not CC (?)

Recap: TCP problems

- Misled by non-congestion losses
- Fills up queues leading to high delays
- Short flows complete before discovering available capacity
- AIMD impractical for high speed links
- Sawtooth discovery too choppy for some apps
- Unfair under heterogeneous RTTs
- Tight coupling with reliability mechanisms
- Endhosts can cheat

Routers tell endpoints if they're congested

Routers tell endpoints what rate to send at

Routers enforce fair sharing

Could fix many of these with some help from routers!

Router-Assisted Congestion Control

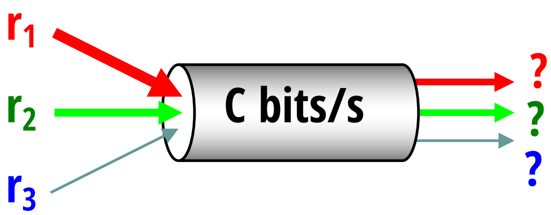
- Three ways routers can help
  - Enforce fairness
  - More precise rate adaptation
  - Detecting congestion

How can routers ensure each flow gets its “fair share”?

Fairness: General Approach

- Consider a single router's actions
- Router classifies incoming packets into "flows"
  - (For now) let's assume flows are TCP connections
- Each flow has its own FIFO queue in router
- Router picks a queue (i.e., flow) in a fair order; transmits packet from the front of the queue
- What does "fair" mean exactly?

Max-Min Fairness



- Total available bandwidth $\mathbf{C}$
- Each flow $\mathbf{i}$ has bandwidth demand $\mathbf{r_i}$
- What is a fair allocation $\mathbf{a_i}$ of bandwidth to each flow $\mathbf{i}$?
- Max-min bandwidth allocations are:

$$\mathbf{a_i = \min(f, r_i)}$$

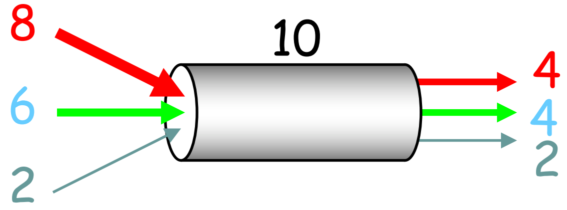
where $\mathbf{f}$ is the unique value such that $\mathbf{Sum(a_i) = C}$

Example

- C = 10; N = 3; r_1 = 8, r_2 = 6, r_3 = 2

- C/N = 10/3 = 3.33 →
  - But r_3's need is only 2
  - Can service all of r_3
  - Allocate 2 to r_3 and remove it from accounting: $C = C - r_3 = 8$; $N = 2$

- C/2 = 4 →
  - Can't service all of r_1 or r_2
  - So hold them to the remaining fair share: $f = 4$



f = 4:
min(8, 4) = 4
min(6, 4) = 4
min(2, 4) = 2

Max-Min Fairness

- Property:
  - If you don’t get full demand, no one gets more than you
- This is what round-robin service gives if all packets are the same size

How do we deal with packets of different sizes?

- Mental model: Bit-by-bit round robin (“fluid flow”)
- Cannot do this in practice!
- But we can approximate it
  - This is what **“fair queuing”** routers do

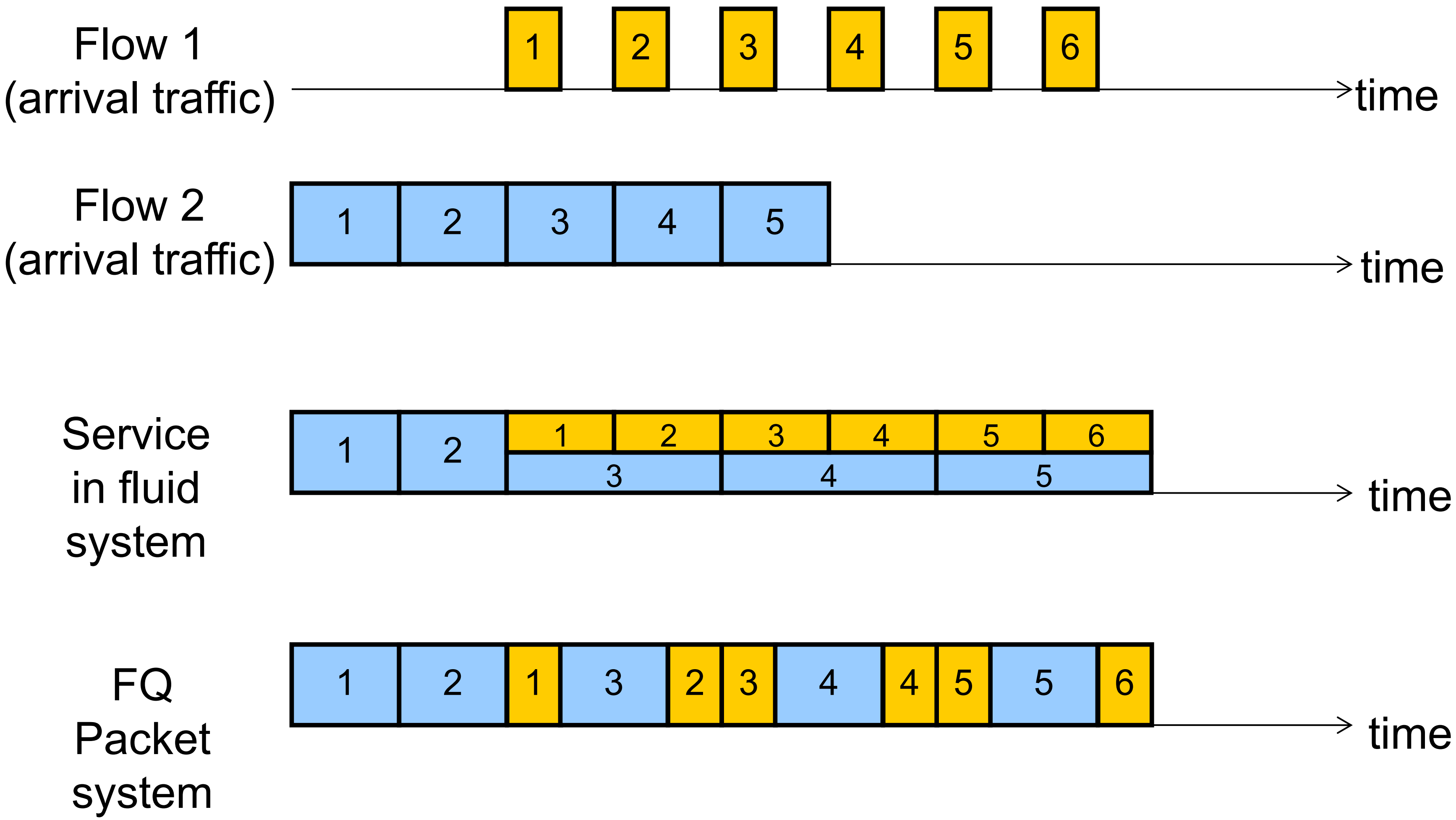
Fair Queuing (FQ)

- For each packet, compute the time at which the last bit of a packet would have left the router *if* flows are served bit-by-bit (called "deadlines")
- Then serve packets in increasing order of their deadlines
- Think of it as an implementation of round-robin extended to the case where not all packets are equal sized

Analysis and Simulation of a Fair Queueing Algorithm

Alan Demers
Srinivasan Keshav†
Scott Shenker

Example


Flow 1
(arrival traffic)
1 2 3 4 5 6
time
Flow 2
(arrival traffic)
1 2 3 4 5
time
Service
in fluid
system
1 2
1 2 3 4 5 6
3 4 5
time
FQ
Packet
system
1 2 1 3 2 3 4 4 5 5 6
time

FQ vs. FIFO

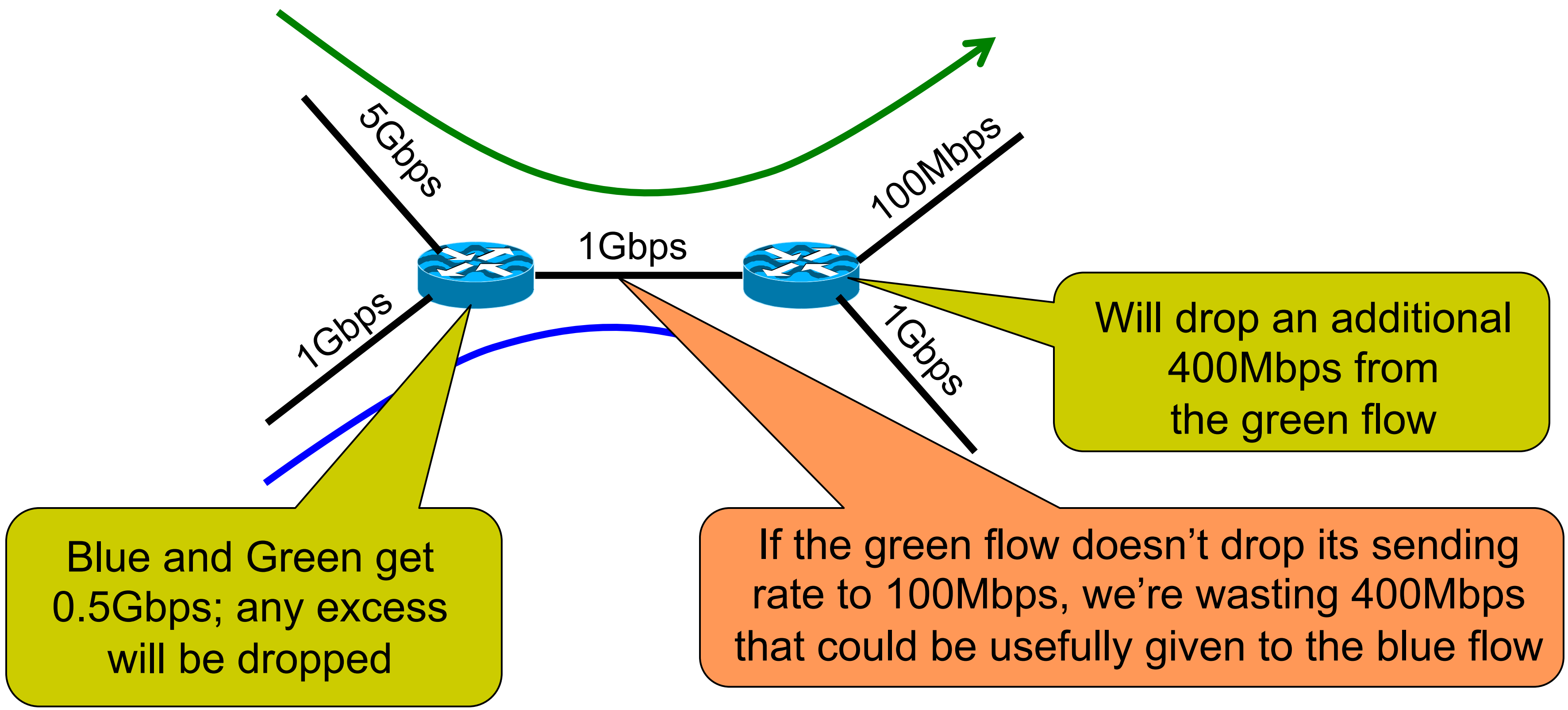
- FQ advantages:
  - Isolation: cheating flows don't benefit
  - Bandwidth share does not depend on RTT
  - Flows can pick any rate adjustment scheme they want
- Disadvantages:
  - More complex than FIFO: per flow queue/state, additional per-packet book-keeping
  - Still only a partial solution (coming up)

Fair Queuing In Practice

- "Pure" FQ too complex to implement at high speeds
- But several approximations exist
  - E.g., Deficit Round Robin (DRR)
- Today:
  - Routers typically implement approximate FQ (e.g., DRR)
  - For a small number of queues
  - Commonly used for coarser-grained isolation (e.g., for select customer prefixes) rather than per-flow isolation

FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion



FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion
- FQ's benefit is its resilience (to cheating, variations in RTT, details of delay, reordering, *etc.*)
- But congestion and packet drops still occur
- And we still want end-hosts to discover/adapt to their fair share!

Per-flow fairness is a controversial goal

- What if you have 8 flows, and I have 4?
  - Why should you get twice the bandwidth
- What if your flow goes over 4 congested hops, and mine only goes over 1?
  - Shouldn't you be penalized for using more of scarce bandwidth?
- And at what granularity do we really want fairness?
  - TCP connection? Source-Destination pair? Source?
- Nonetheless, FQ/DRR is a great way to ensure **isolation**
  - Avoiding starvation even in the worst cases

Router-Assisted Congestion Control

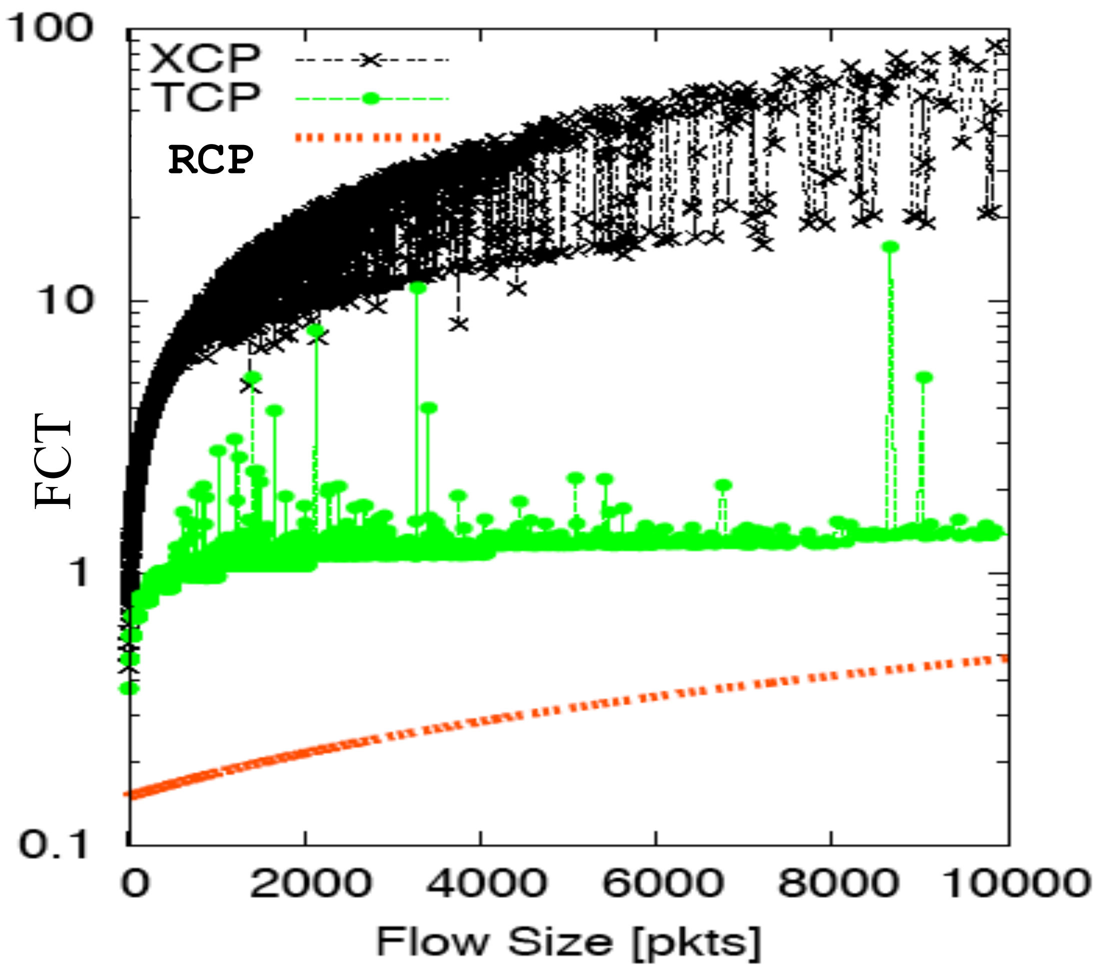
- Three ways routers can help
  - Enforce fairness
  - More precise rate adaptation
  - Detecting congestion

Why not just let routers tell endhosts what rate they should use?

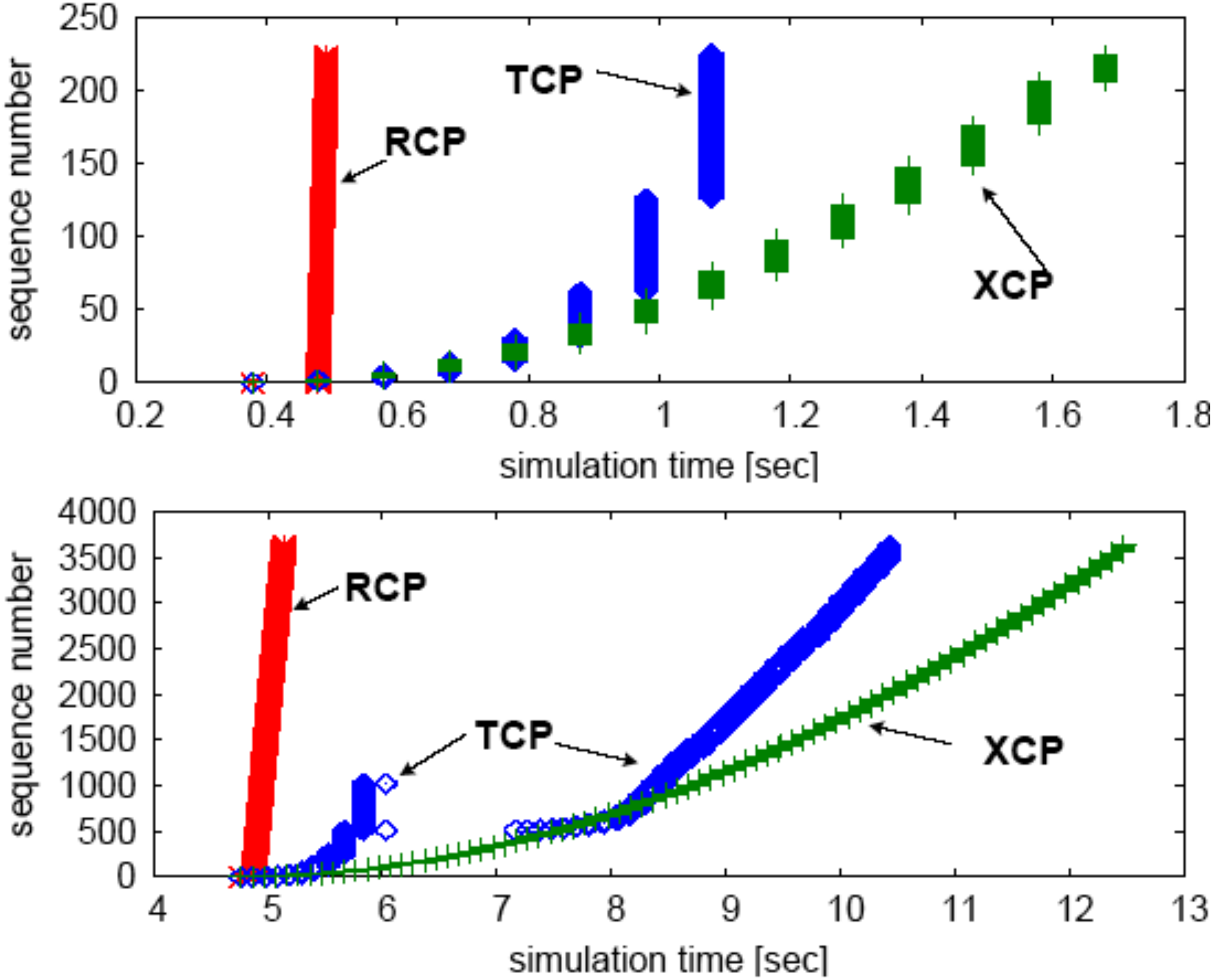
- Packets carry “rate field”
- Routers insert a flow’s fair share *f* in packet header
- End-hosts set sending rate (or window size) to *f*
- This is the basic idea behind the “Rate Control Protocol” (RCP) from Dukkipati *et al.* ’07

Flow Completion Time: TCP vs. RCP (Ignore XCP)

Flow Completion Time (secs) vs. Flow Size



Why the improvement?



Router-Assisted Congestion Control

- Three ways routers can help
  - Enforce fairness
  - More precise rate adaptation
  - Detecting congestion

Explicit Congestion Notification (ECN)

- Single bit in packet header; set by congested routers
  - If data packet has bit set, then ACK has ECN bit set
- Many options for *when* routers set the bit
  - Tradeoff between link utilization and packet delay
- Host can react as though it was a drop

- Advantages:
  - Don't confuse corruption with congestion
  - Early indicator of congestion → avoid delays
  - Lightweight to implement

- Today:
  - Widely implemented in routers
  - Some use in datacenters (e.g., Azure)

Final idea: Congestion-Based Charging

- Use ECN as congestion markers
- Whenever I get an ECN bit set, I have to pay $$
  - The more congested the network, the more I pay
- No debate over what a flow is, or what fair is…
- Idea started by Frank Kelly at Cambridge
  - "optimal" solution, backed by much math
  - Great idea: simple, elegant, effective
  - But requires an entirely new charging model!

Recap: Router-Assisted CC

- FQ: routers *enforce* per-flow fairness
- RCP: routers *inform* endhosts of their fair share
- ECN: routers set “I’m congested” bit in packets
- Congestion pricing: users pay based on congestion

Perspective: Router-Assisted CC

- Can be highly effective, approaching optimal perf.
- But deployment is more challenging
  - Need support at hosts and routers
  - Some require more complex book-keeping at routers
  - Some require deployment at *every* router
- Though worth revisiting in datacenter contexts

Perspective: TCP CC

- Not perfect, a little ad-hoc
- But deeply practical/deployable
- Good enough to have raised the bar for the deployment of new, more optimal, approaches
- Though datacenters are reshaping the CC agenda
  - different needs and constraints (future lecture)

Next Topics

- The Domain Name System (DNS) and resolving names to addresses
- Remember: no in-person lecture on Thursday