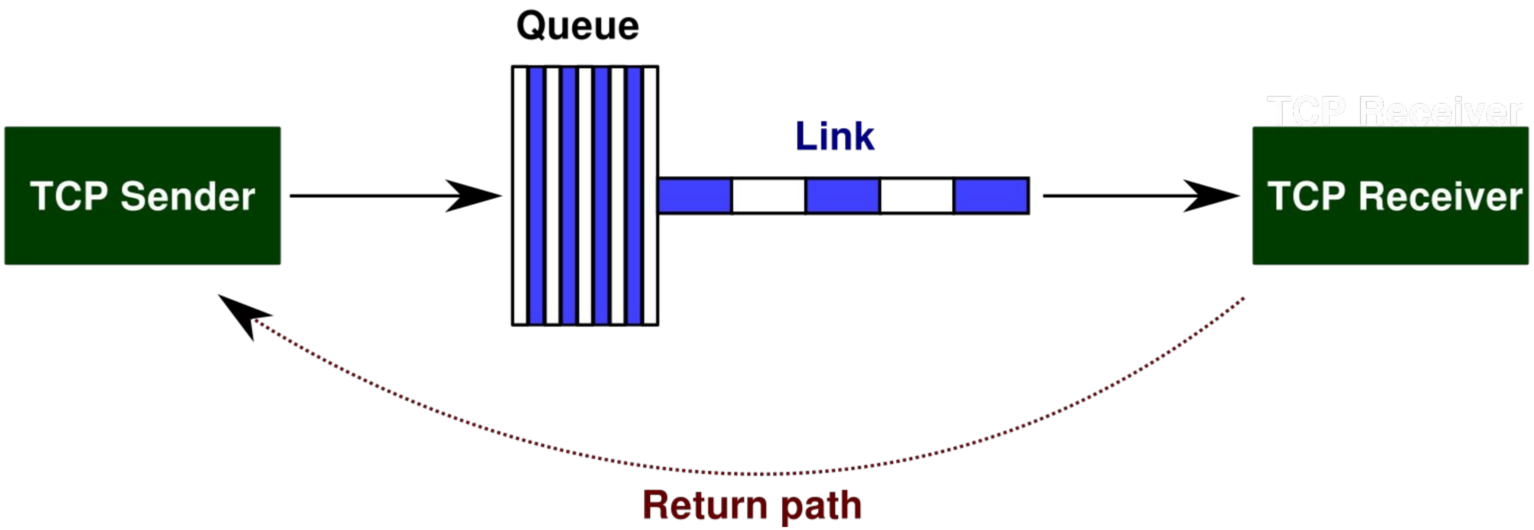
Why congestion control?

Part 1: model & collapse

TCP and flow control

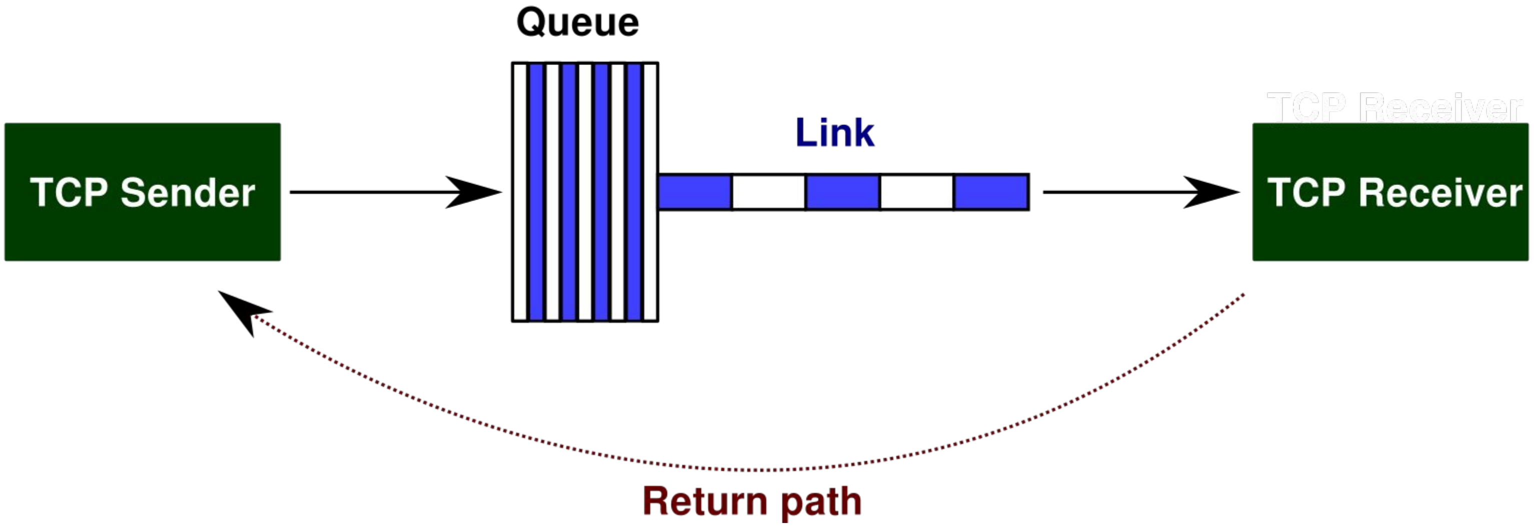
- TCP provides a **flow-controlled** bidirectional byte stream
- “**Flow-controlled**”: sender respects **receiver’s** capacity
- But… what about the **network’s** capacity?

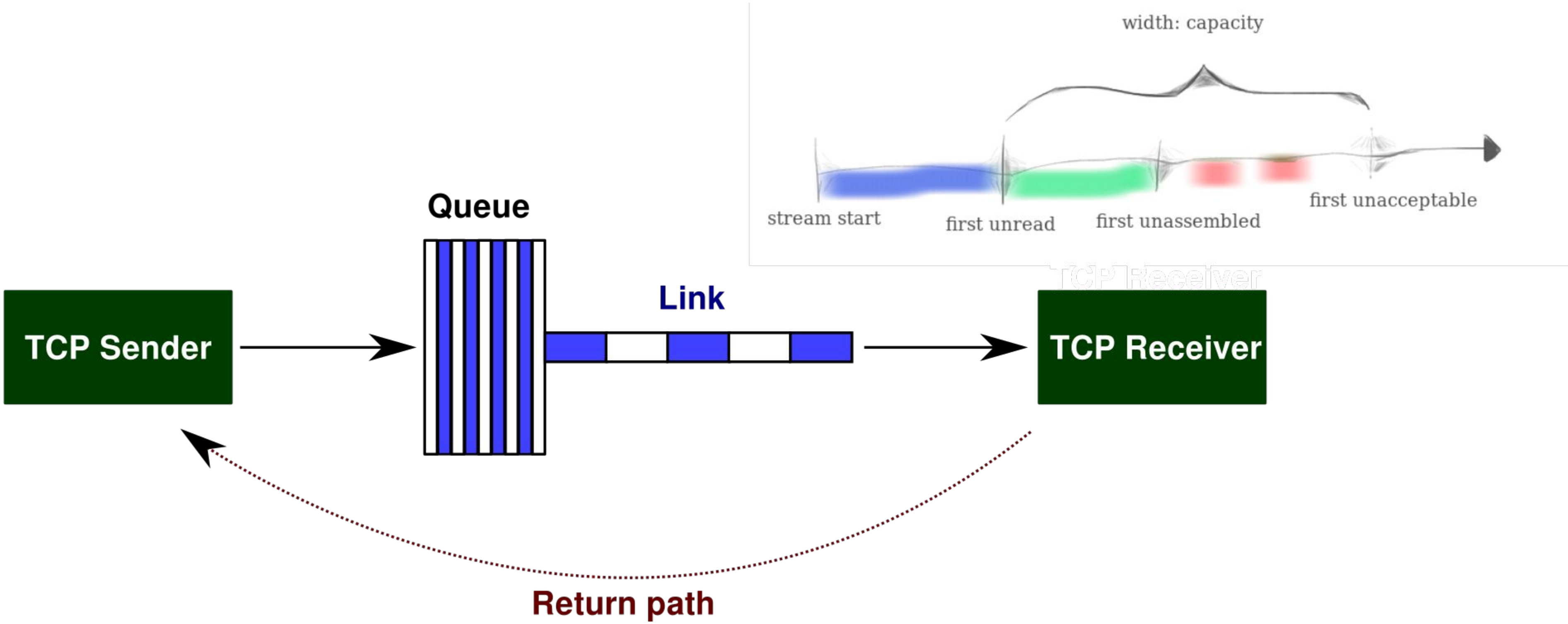
Single-flow, single-hop model

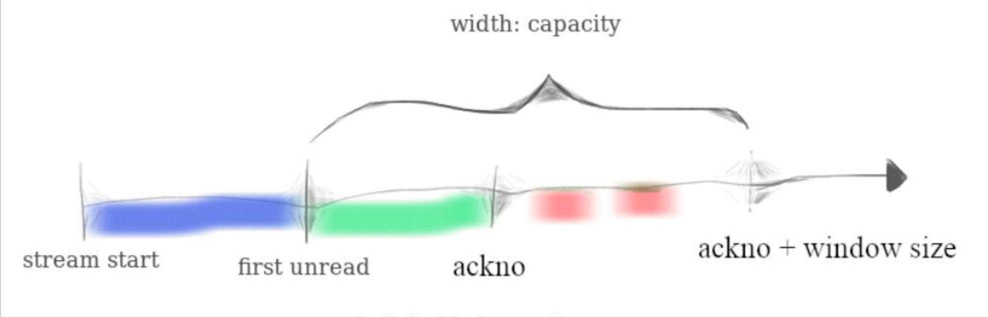


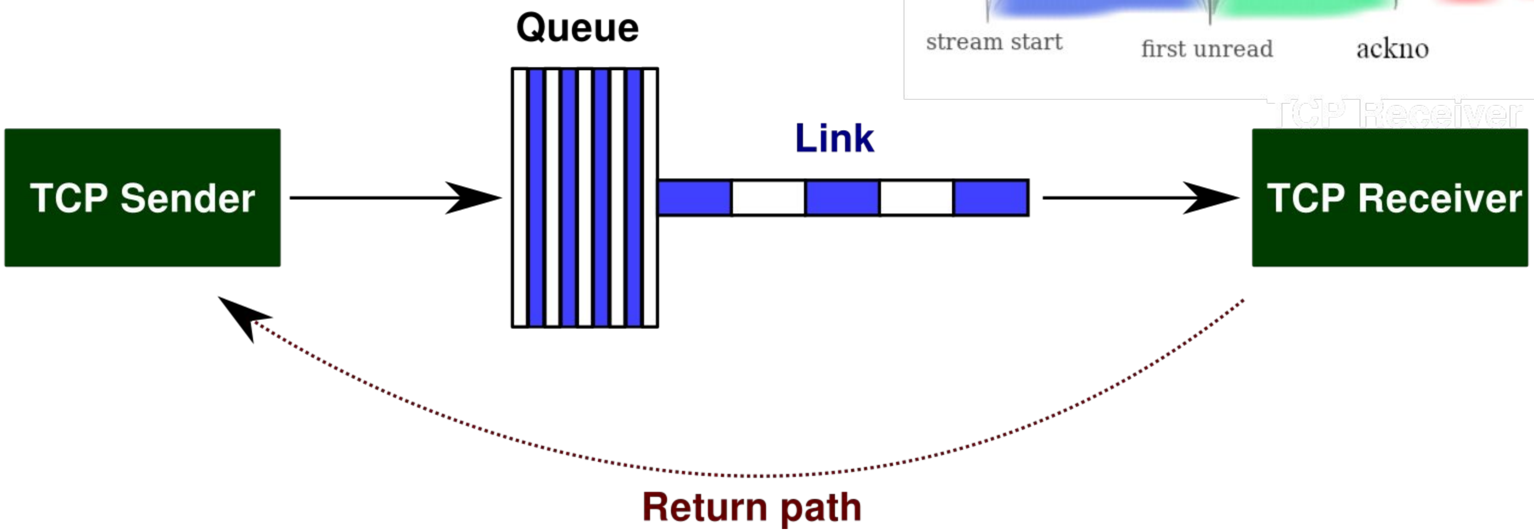
From sender’s perspective, three places packets can be

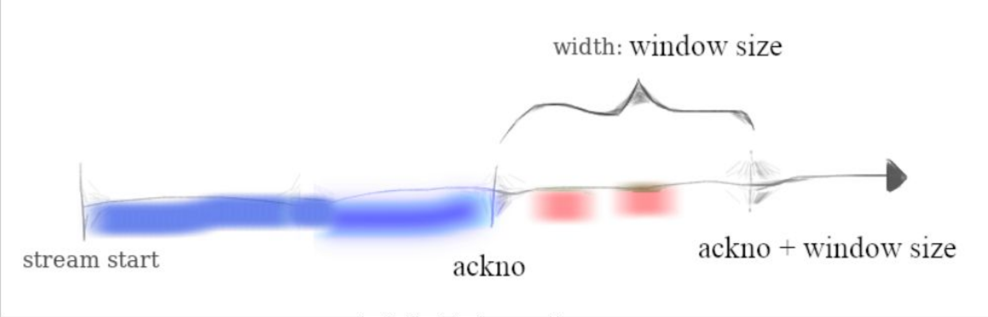
1. In the bottleneck queue
2. In transit on the link
3. At receiver, with acknowledgment in transmit back to sender

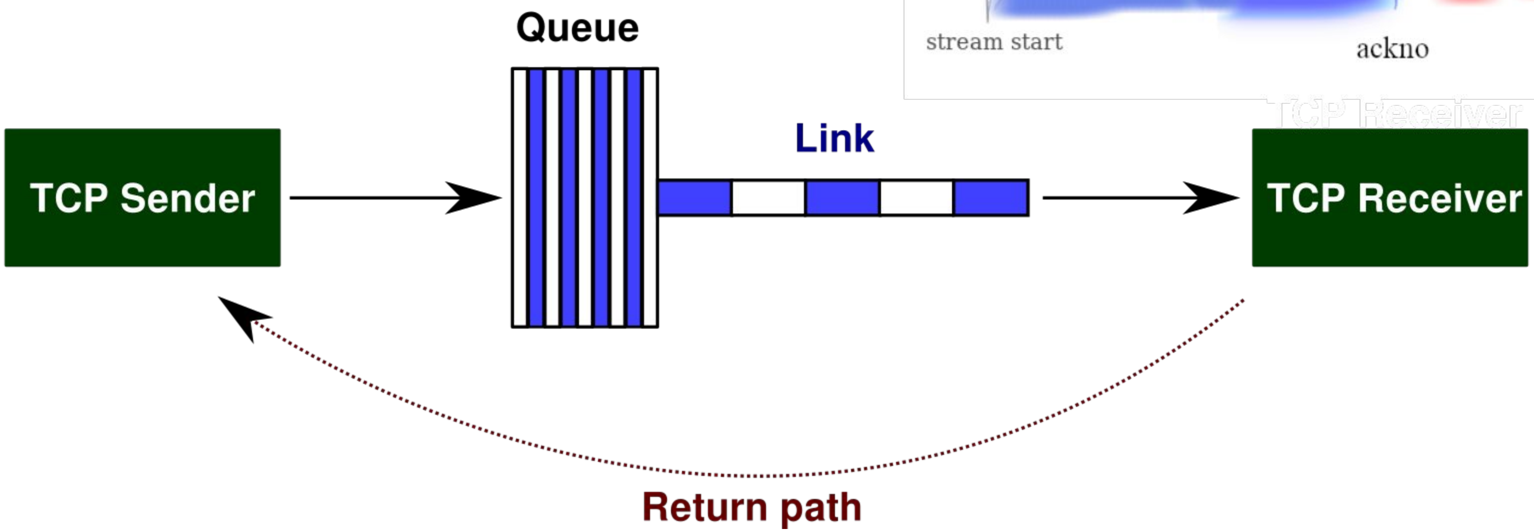



width: capacity
stream start
first unread
first unassembled
first unacceptable
Queue
Link
TCP Sender
TCP Receiver
Return path


width: capacity
stream start
first unread
ackno
ackno + window size


Queue
Link
TCP Sender
TCP Receiver
Return path


width: window size
stream start
ackno
ackno + window size


Queue
Link
TCP Sender
TCP Receiver
Return path

Windows: cap on number of bytes “outstanding”

- The receiver’s window size caps the number of bytes outstanding from sender’s perspective.

- “Outstanding” means **sent**, and not **acked** or judged **lost**.

- **Q:** *What if the window size is really small (e.g. 1 byte)?*

- **Q:** *What if the window size is really big?*

What if the receiver's window size is really big?

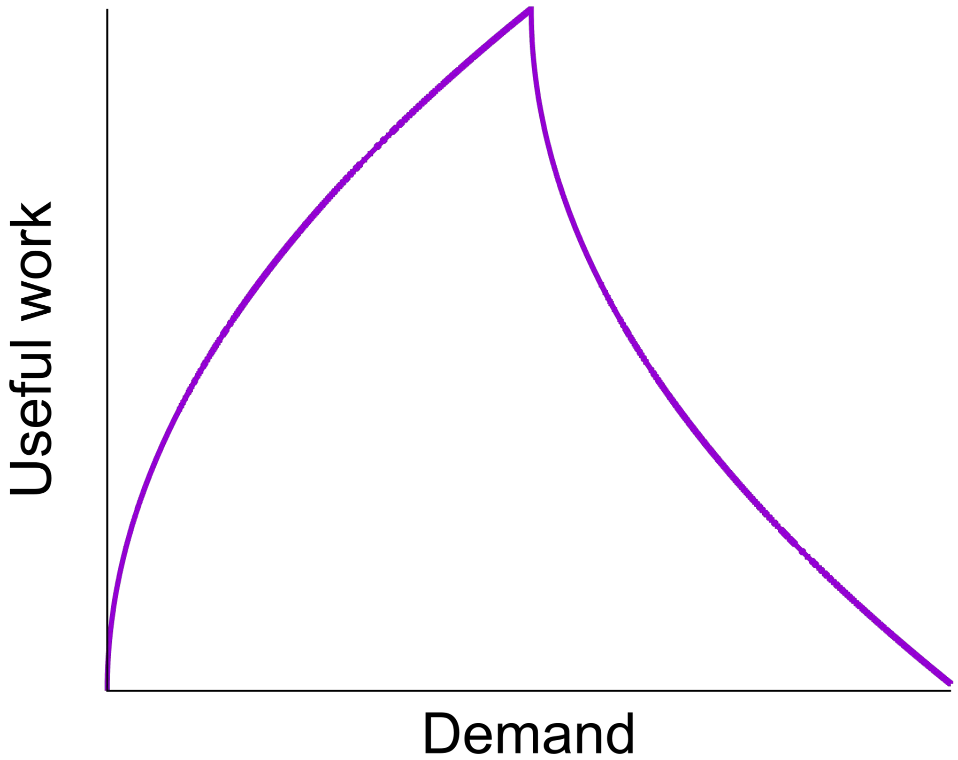
- Sender transmits **too many segments**. Most overflow router's queue and are dropped.
- We call this "**congestion**."
- Sender must resend the same bytes again and again. Eventually, stream comes out of receiver's TCP correctly.
- Q: ***Why is this bad?***

The problem with unlimited sending: collapse and fairness

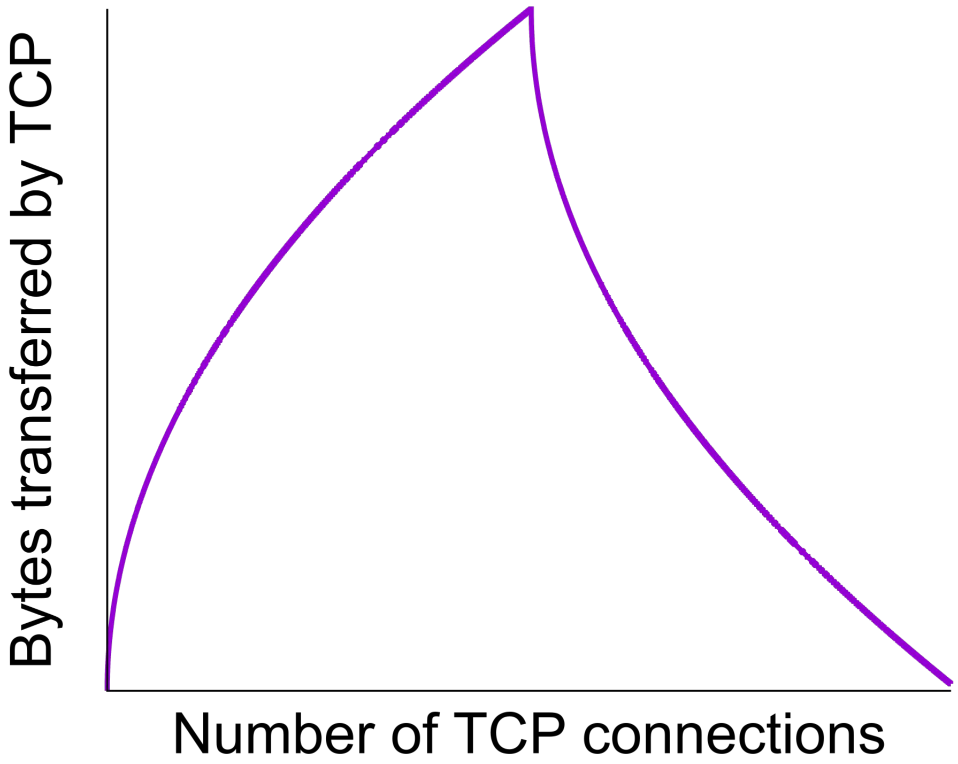
1. Forcing routers to drop lots of packets can lead to "**congestion collapse.**"
    - Lots of demand, but network not doing useful work.

2. When some flows send too much, others are starved.
    - Network exhibits bad "**fairness.**"

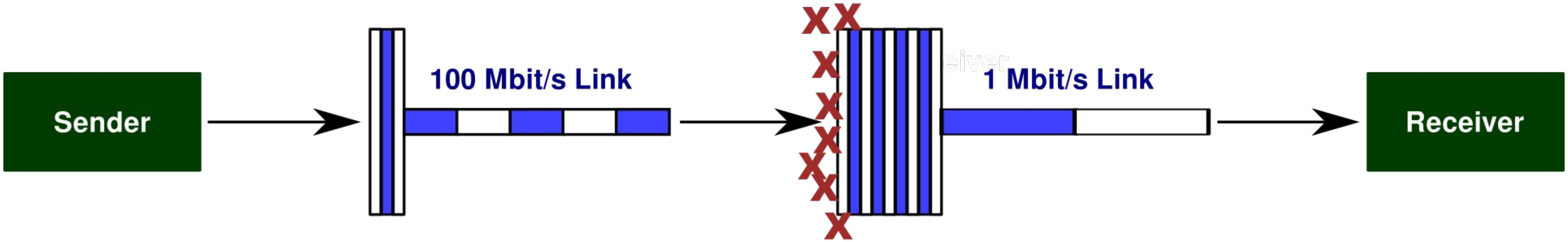
Collapse!



Collapse!



An easy way to get collapse



Why congestion control?

Part 2: fairness & objectives

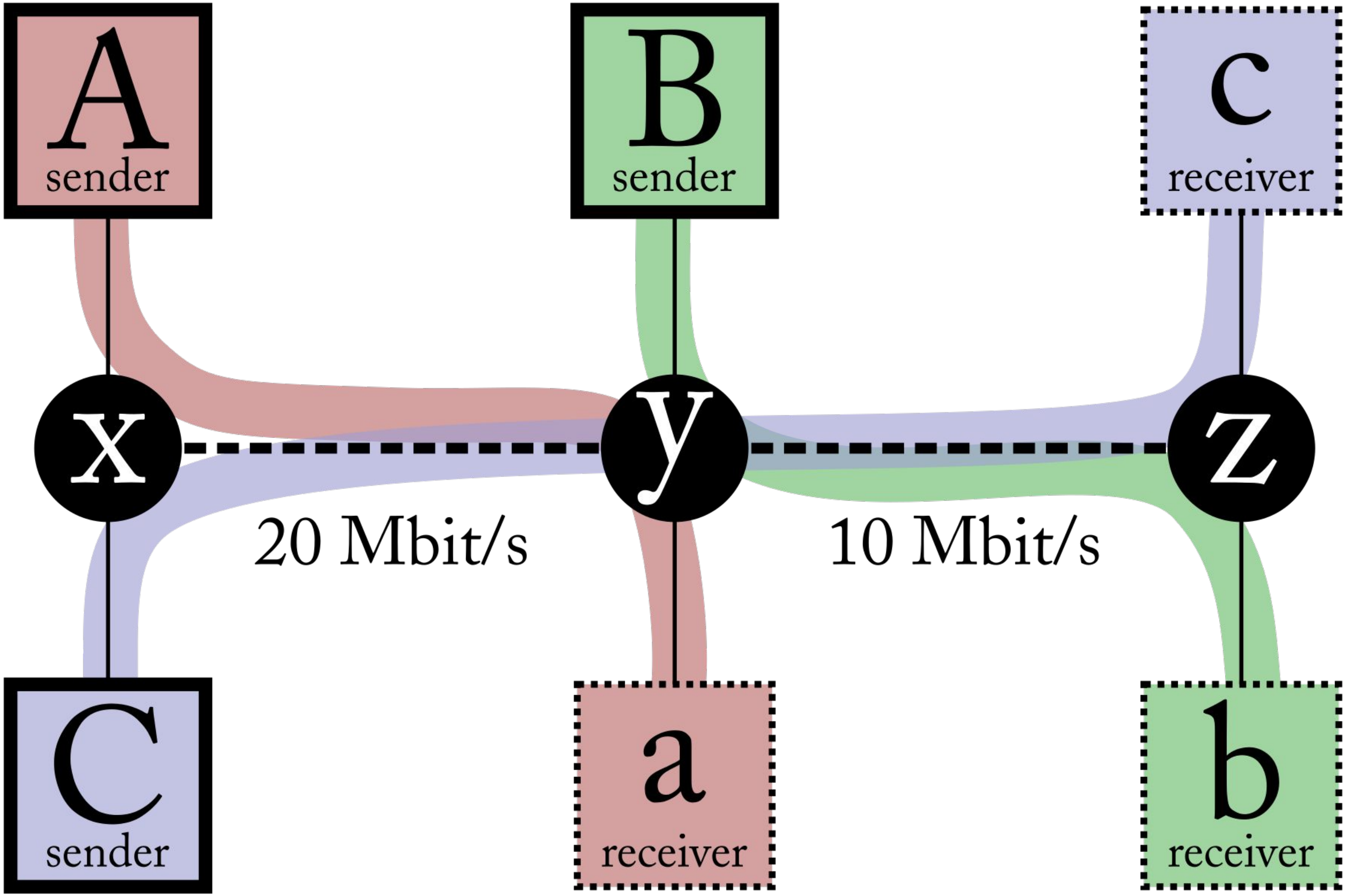
The problem with unlimited sending: collapse and fairness

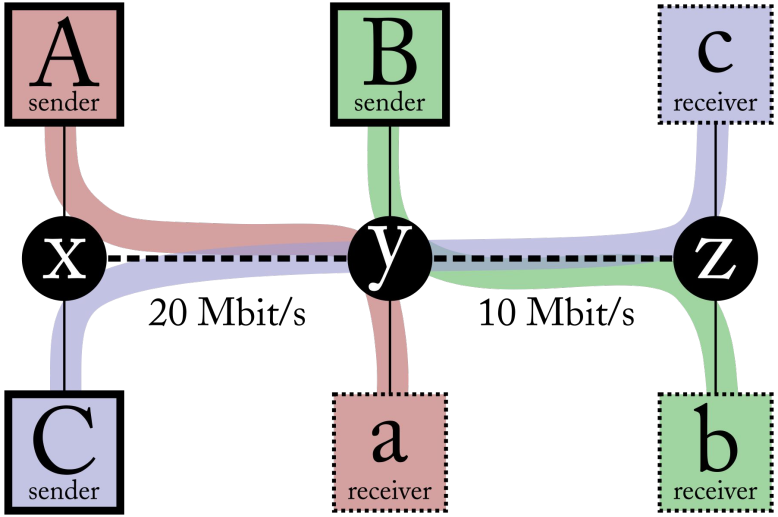
1. Forcing routers to drop lots of packets can lead to "**congestion collapse.**"
    - Lots of demand, but network not doing useful work.

2. When some flows send too much, others are starved.
    - Network exhibits bad "**fairness.**"

Fairness: what’s the right way to divide the network?

- Network resources are limited.
- Flows share the same network. They can’t all get all of it.
- What’s the best way to divide up the pie?


A
sender
B
sender
c
receiver
x
y
z
20 Mbit/s
10 Mbit/s
C
sender
a
receiver
b
receiver



A→a	B→b	C→c	Total
20	10	0	30 *(max utilization)*
10	0	10	20 *(best for C)*
0	0	~~20~~ 10	10 *(collapse!)*
15	5	5	25 *(**max-min fair**: worst outcome is as good as possible)*
16	6	4	26 *(**proportionally fair**: improvement by x requires harm of >1/x)*

The mathematics of resource allocation

$$\max_{\{x_r\} \in S} \sum_r U_r(x_r)$$

$$\text{subject to} \sum_{r:l \in r} x_r \leq c_l, l \in \mathcal{L}$$

$$x_r \geq 0, r \in \mathcal{S}$$

If user r receives throughput x_r, that produces utility $U_r(x_r)$. $\mathcal{L}$ = all links. $\mathcal{S}$ = all users.

Alpha-fairness

$$U(x) = \frac{x^{1-\alpha}}{1-\alpha}$$

$$\begin{array}{ll} \alpha = 0 & \textit{max utilization} \\ \alpha \to 1 & \textit{proportional fairness} \\ \alpha = 2 & \textit{min-potential-delay fairness} \\ \alpha \to \inf & \textit{max-min fairness} \end{array}$$

Other “group” objectives

- Minimize **flow completion time** (of average download)
- Minimize **page load time** (of website with many downloads)
- Maximize “power” (= throughput / delay)
- ...

Rest of this unit

- The algorithms that let flows share the network and prevent collapse are called "**congestion control**."

- In networking, almost any problem that involves decentralized resource allocation = congestion control.

- Big questions to come:
  - **what should be** the window size?
  - how should flows **learn** the right window size?