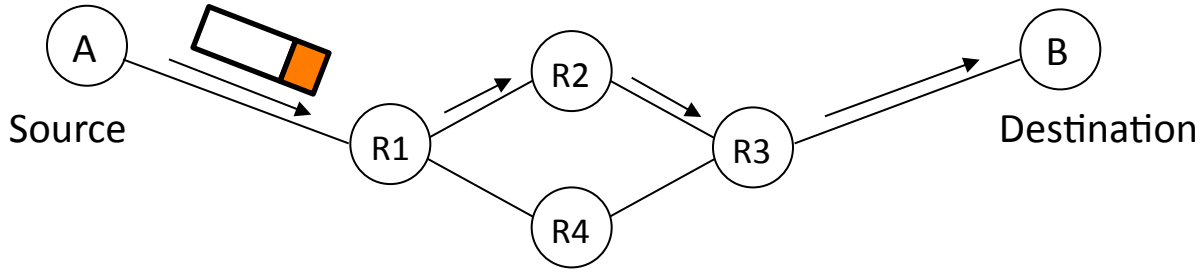
CS144
An Introduction to Computer Networks

Packet Switching

Philip Levis

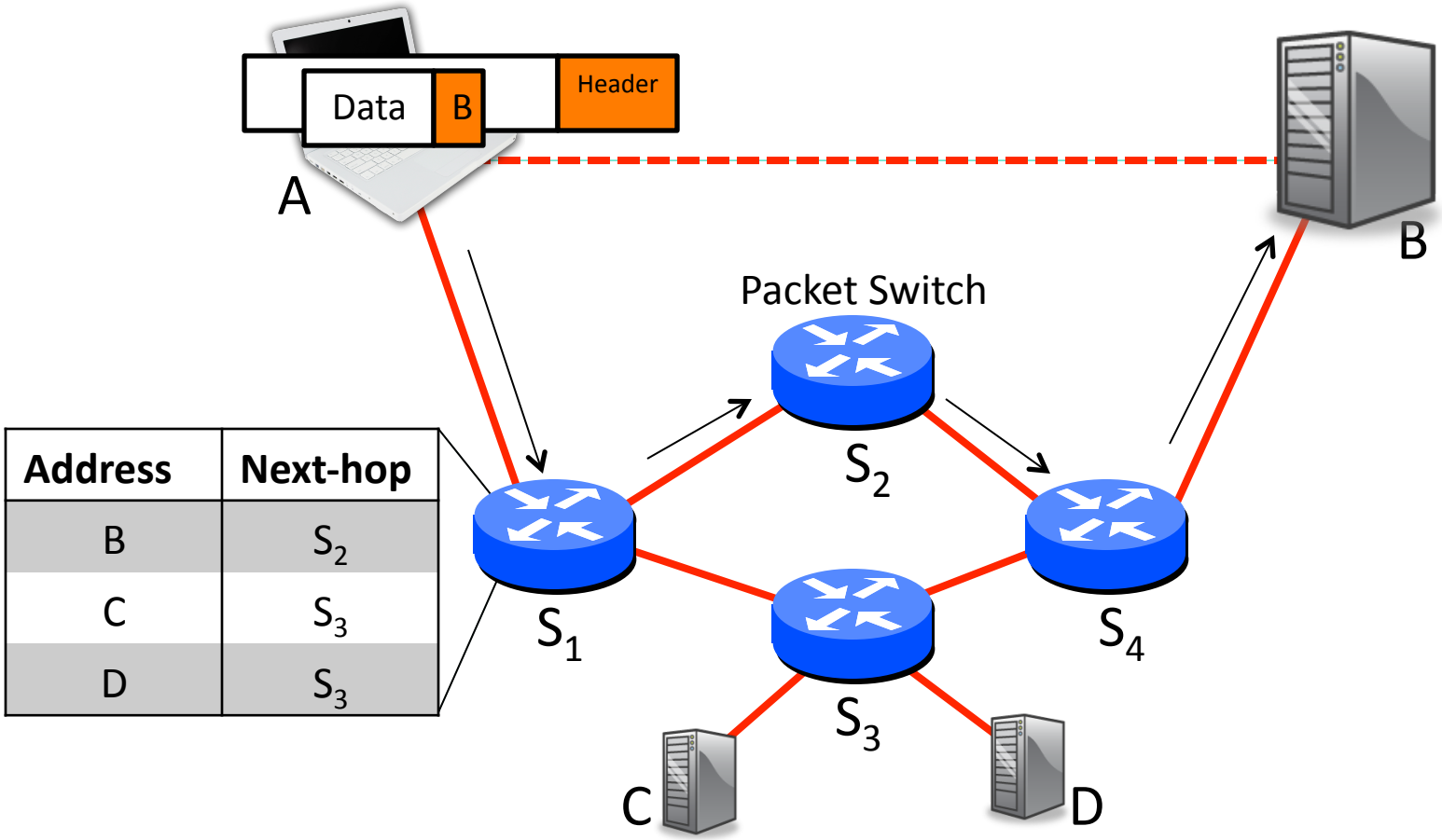
Oct 11, 2017

Packet Switching



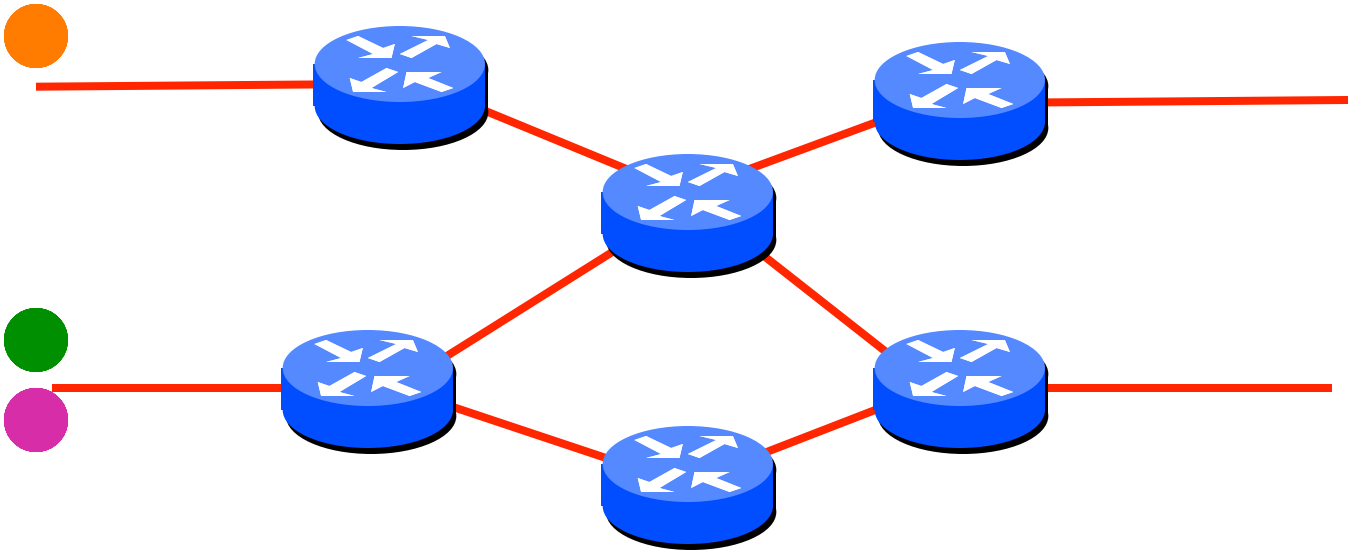
- Packets are routed individually, by looking up address in router's local table.
- All packets share the full capacity of a link.
- The routers maintain no per-communication state.

Packet Switching

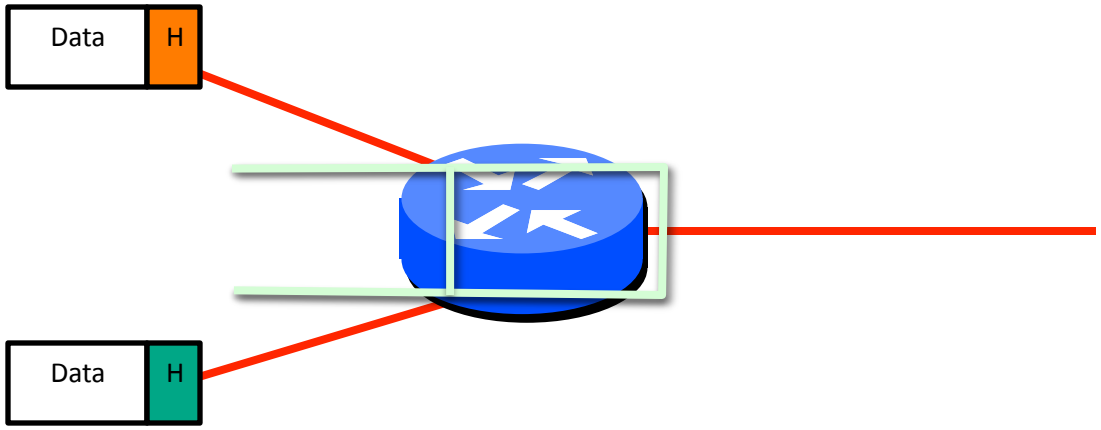


Address	Next-hop
B	S_2
C	S_3
D	S_3

Packet Switching



Packet switches have buffers



Buffers hold packets:

- When two or more packets arrive at the same time
- During periods of congestion

Efficient use of expensive links

- Links were assumed to be expensive and scarce.
- Packet switching allows many, bursty flows to share the same link efficiently.
- "Circuit switching is rarely used for data networks, ... because of very inefficient use of the links"
  – Bertsekas/*Gallager*

Resilience to failure of links & routers

- "For high reliability, ... [the Internet] was to be a datagram subnet, so if some lines and [routers] were destroyed, messages could be ... rerouted" - *Tanenbaum*

ping demo

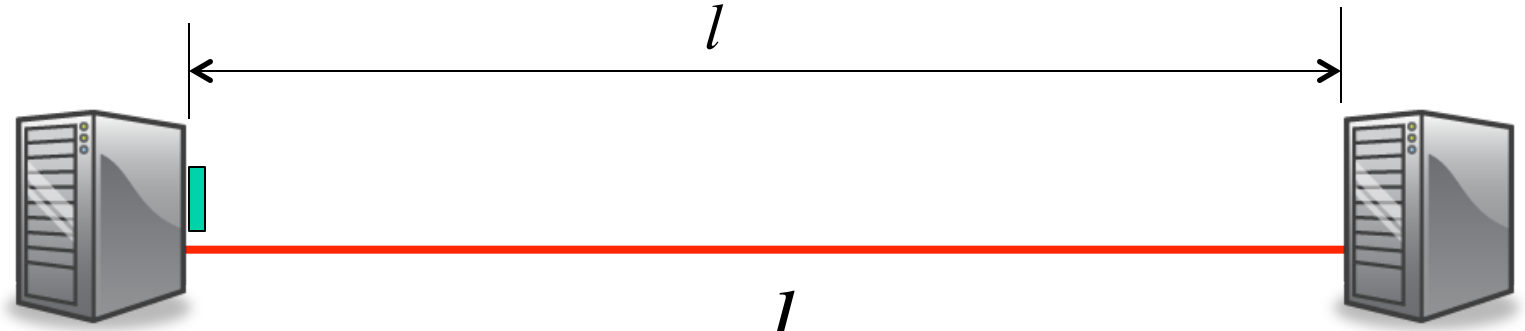
End-to-end Delay

Packets take time to travel between hosts

End-to-end delay: time from the start of first bit leaving the source and the end of the last bit arriving at the destination

End-to-end delay measures the travel time due to the *network*: endpoint software adds more

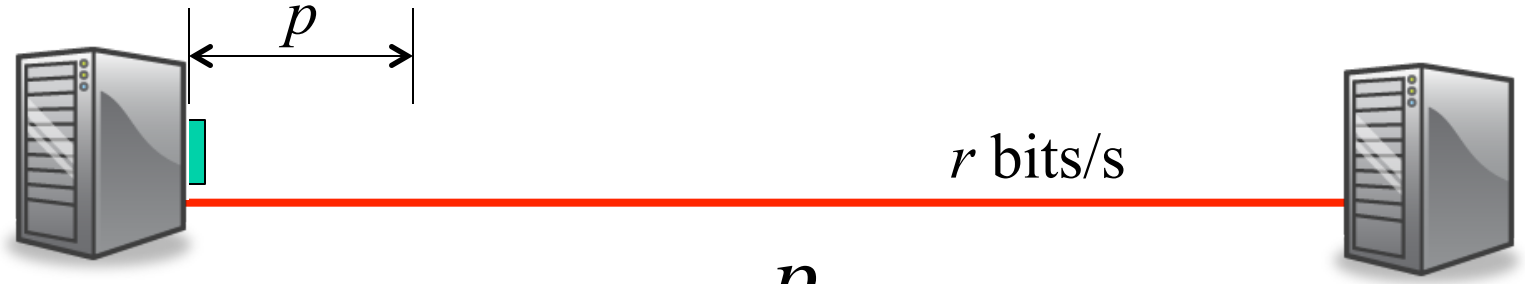
Propagation Delay, t_l: The time it takes a single bit to travel over a link at propagation speed c.



$$t_l = \frac{l}{c}$$

Example: A bit takes 5ms to travel 1,000km in an optical fiber with propagation speed 2 x 10^8 m/s.

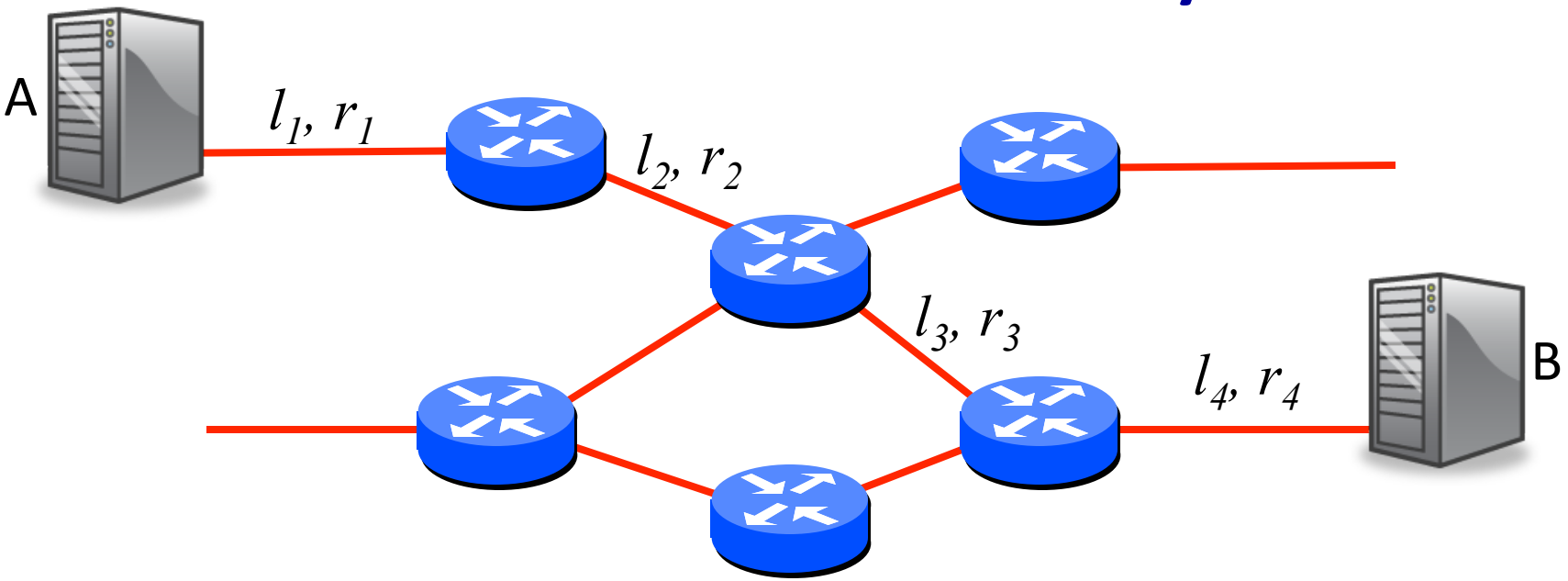
Packetization Delay, t_p: The time from when the first to the last bit of a packet is transmitted.



$$t_p = \frac{p}{r}$$

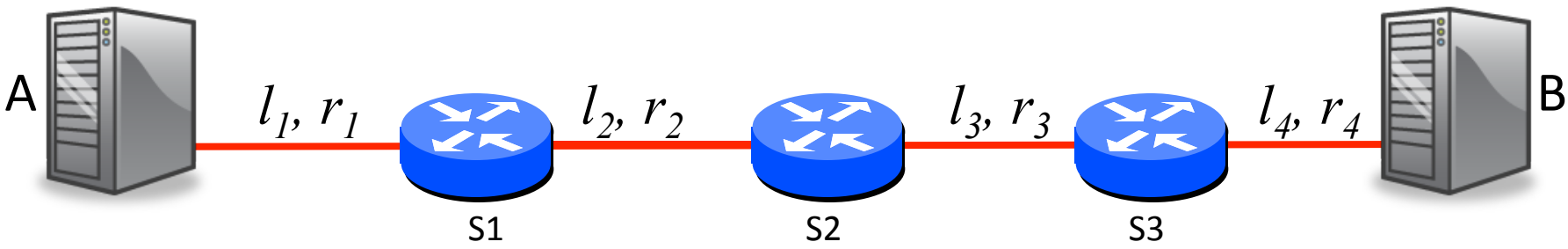
Example 1: A 64byte packet takes 5.12μs to be transmitted onto a 100Mb/s link.
Example 2: A 1kbit packet takes 1.024s to be transmitted onto a 1kb/s link.

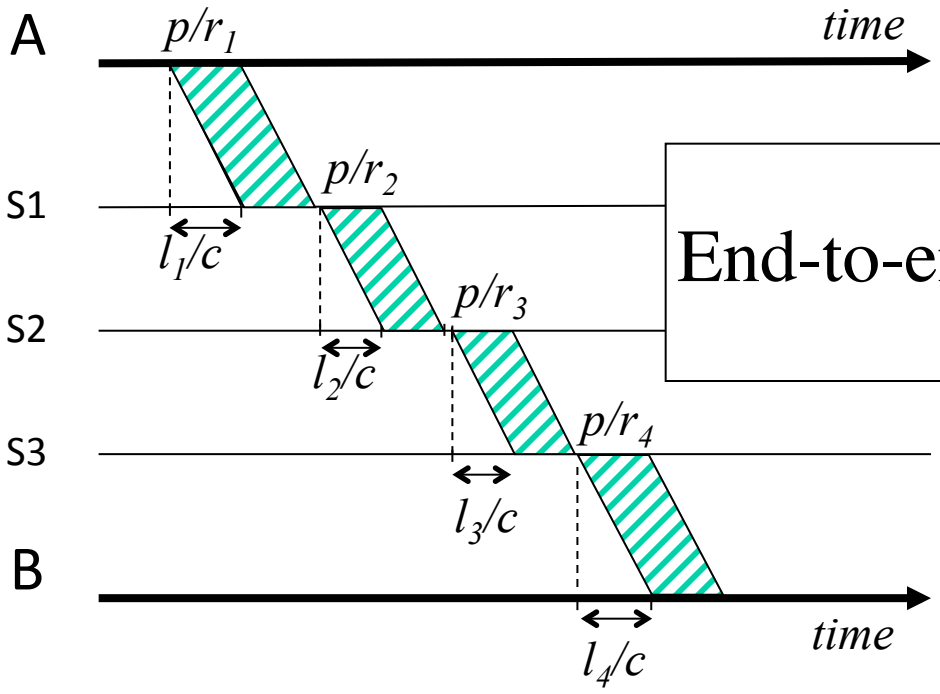
End-to-end delay



Example: How long will it take a packet of length p to travel from A to B, from when the 1st bit is sent, until the last bit arrives? Assume the switches *store-and-forward* packets along the path.

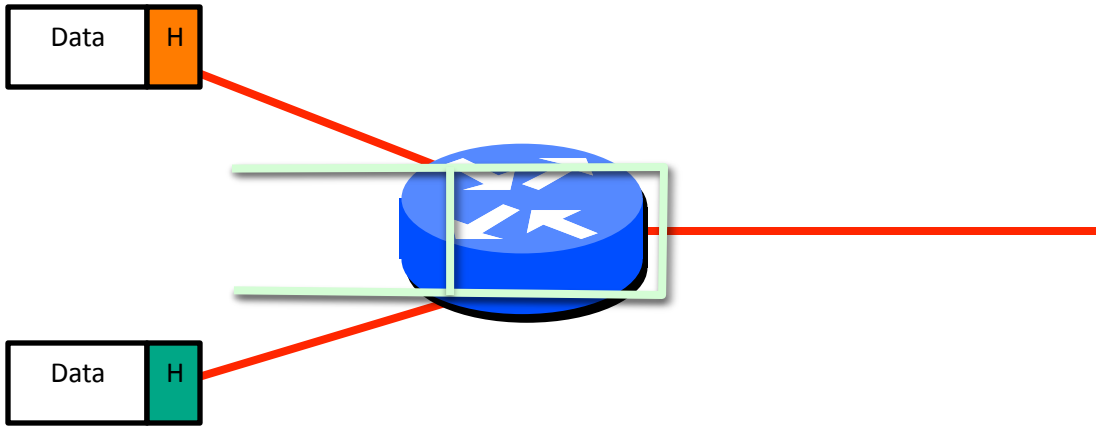
$$\text{End-to-end delay},\ t = \sum_i \left(\frac{p}{r_i} + \frac{l_i}{c} \right)$$





$$\text{End-to-end delay, } t = \sum_i \left(\frac{p}{r_i} + \frac{l_i}{c} \right)$$

What about buffers?



Buffers hold packets:

- When two or more packets arrive at the same time
- During periods of congestion
- Add additional delay, waiting to get to head of line

Queuing Delay

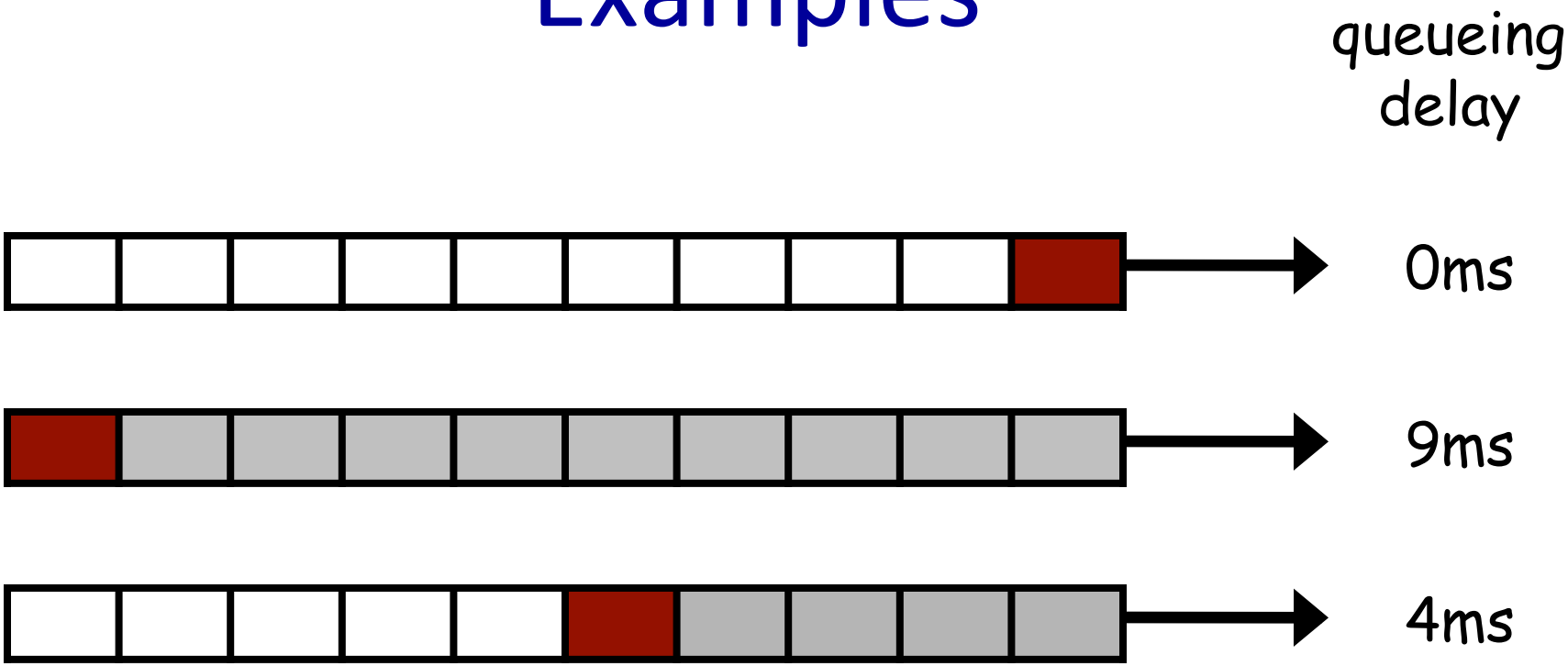
Queuing delay: time a packet waits in a queue

Queuing delay is a dynamic value: depends on depth of queue, so other packets

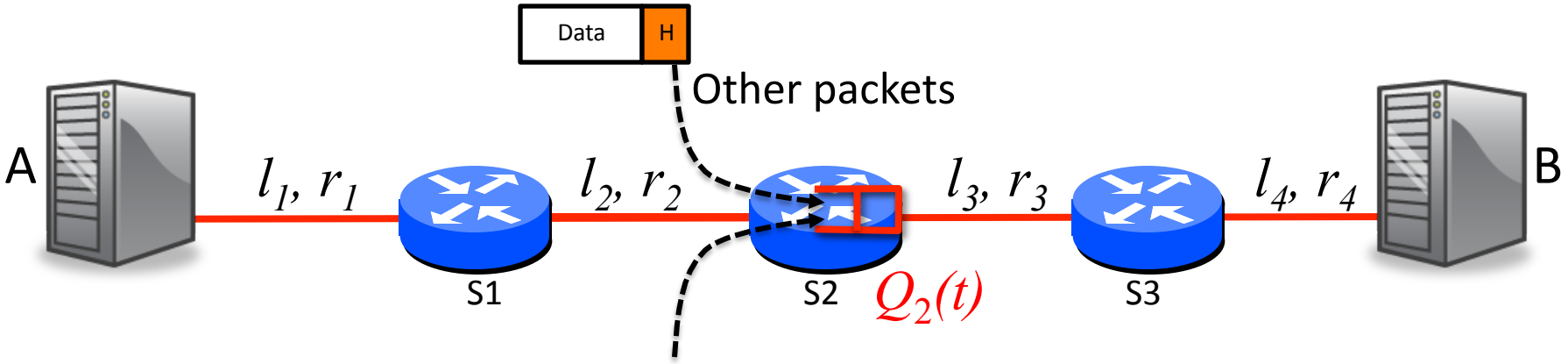
Queuing delay is the source of (almost) all end-to-end delay variation along a wired network path[1]

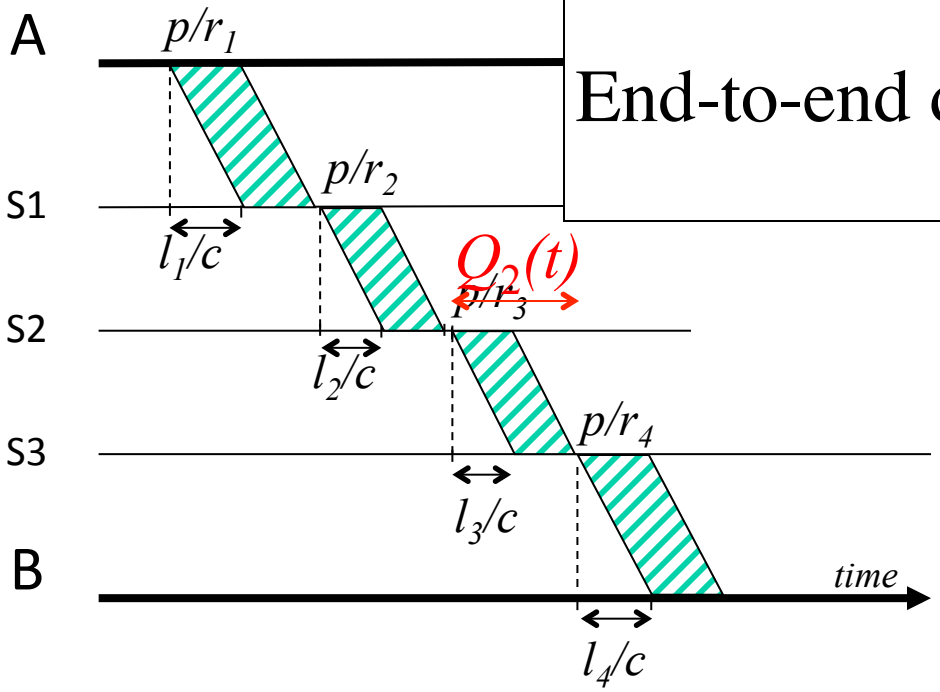
[1]there are some exceptions (e.g., some links have variable r), but we'll just assume r is constant

Examples



Assume all packets are same length, 10kbit.
Link speed is 10Mbps.





$$\text{End-to-end delay, } t = \sum_i \left(\frac{p}{r_i} + \frac{l_i}{c} + Q_i(t) \right)$$

*Queueing = UK spelling, adopted by Kleinrock at UCLA in 1960s. Queueing and queuing (US spelling) are both widely used.

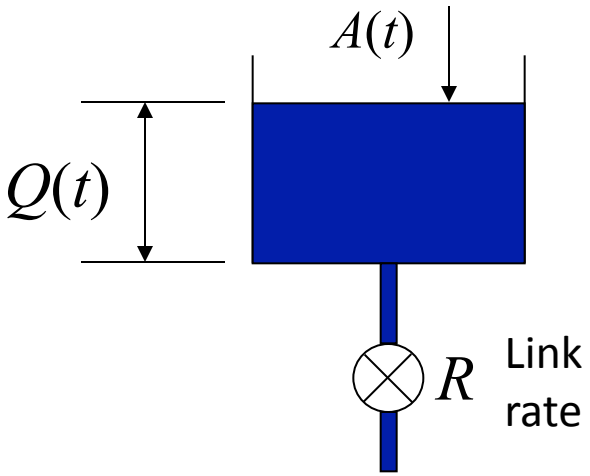
End-to-end Delay

End to end delay is made up of three main components:

- Propagation delay along the links (fixed)
- Packetization delay to place packets onto links (fixed)
- Queueing delay in the packet buffers of the routers (variable)

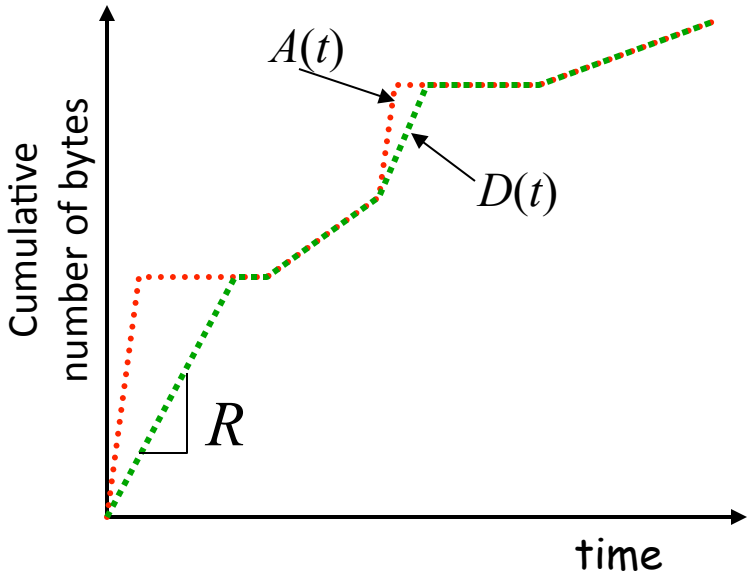
Simple model of a queue

Cumulative number of bytes arrived up until time t.



$D(t)$

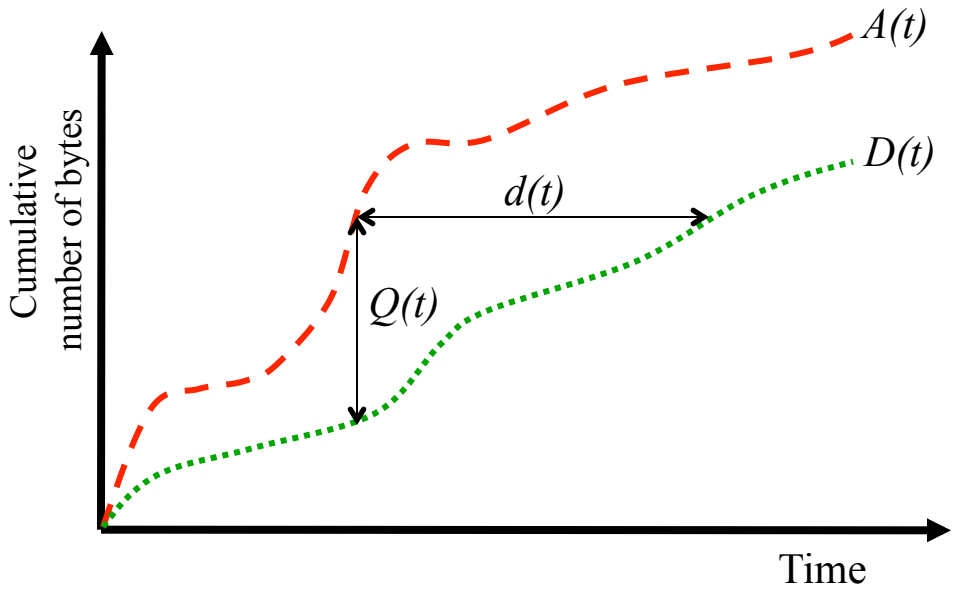
Cumulative number of bytes departed up until time t.



Properties of $A(t)$, $D(t)$:

- $A(t)$, $D(t)$ are non-decreasing
- $A(t) >= D(t)$

Simple model of a queue

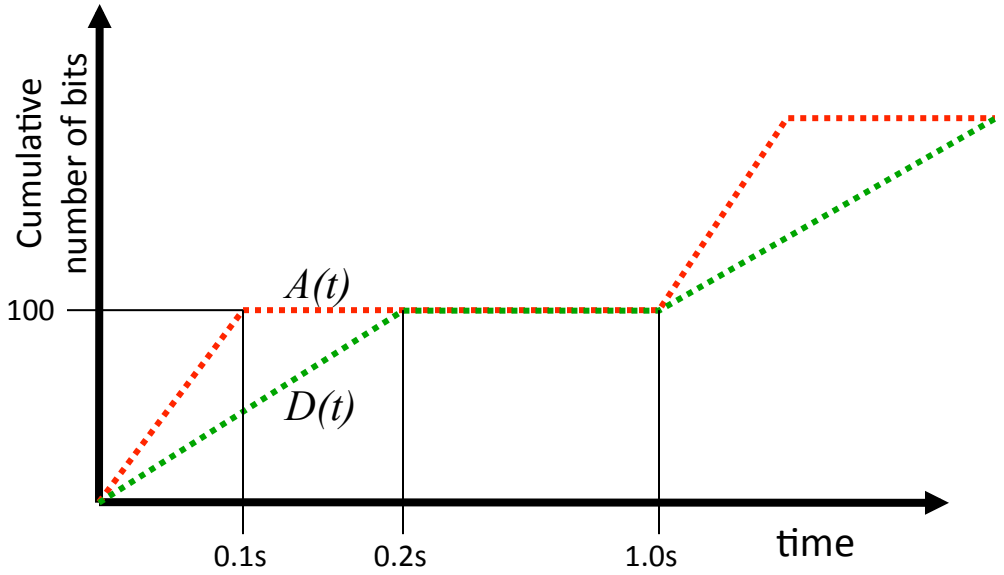


Queue occupancy: $Q(t) = A(t) - D(t)$.

Queueing delay, $d(t)$, is the time spent in the queue by a byte that arrived at time t, assuming the queue is served first-come-first-served (FCFS).

Example

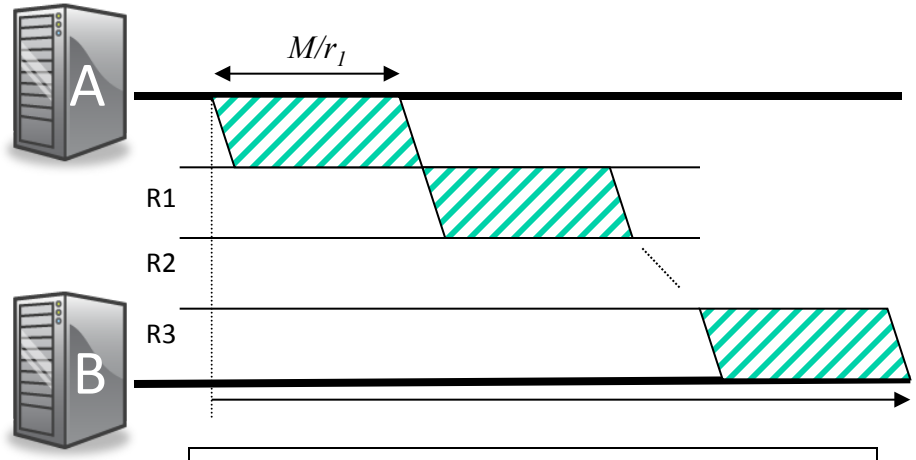
Every second, a 100 bit packet arrives to a queue at rate 1000b/s. The maximum departure rate is 500b/s. What is the average occupancy of the queue?

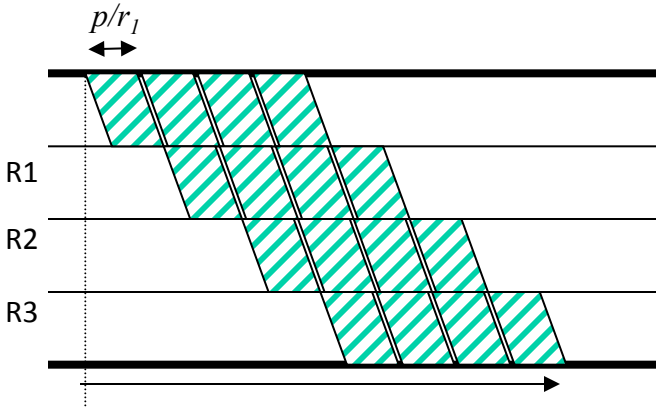


Solution: During each repeating 1s cycle, the queue fills at rate 500b/s for 0.1s, then drains at rate 500b/s for 0.1s. Over the first 0.2s, the average queue occupancy is therefore $0.5 \times (0.1 \times 500) = 25$ bits.
The queue is empty for 0.8s every cycle, and so average queue occupancy: $\overline{Q}(t) = (0.2 \times 25bits) + (0.8 \times 0) = 5bits$

Packet Switching

Why not send the entire message in one packet?





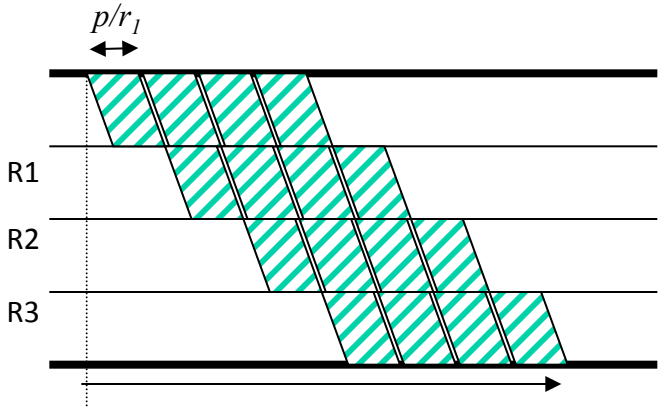
$$\text{End-to-end delay},\ t = \sum_i \left(\frac{M}{r_i} + \frac{l_i}{c} \right)$$

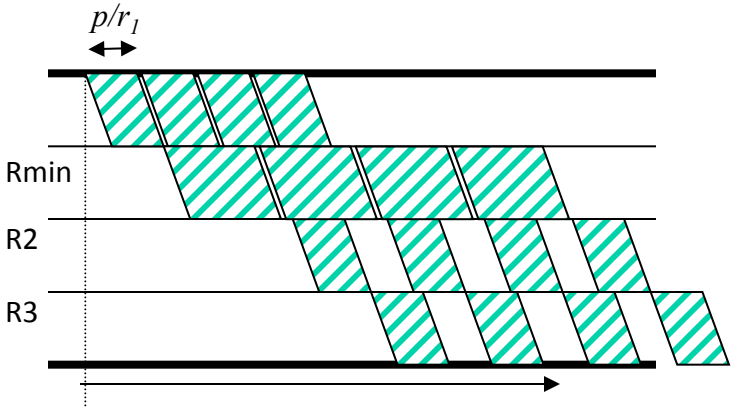
$$\text{End-to-end delay},\ t = \sum_i \left(\frac{p}{r_i} + \frac{l_i}{c} \right) + \left(\frac{M}{p} - 1 \right) \frac{p}{r_{\min}}$$

Breaking message into packets allows parallel transmission across links in a path, reducing end to end delay.

Packet Switching

Why not send the entire message in one packet?



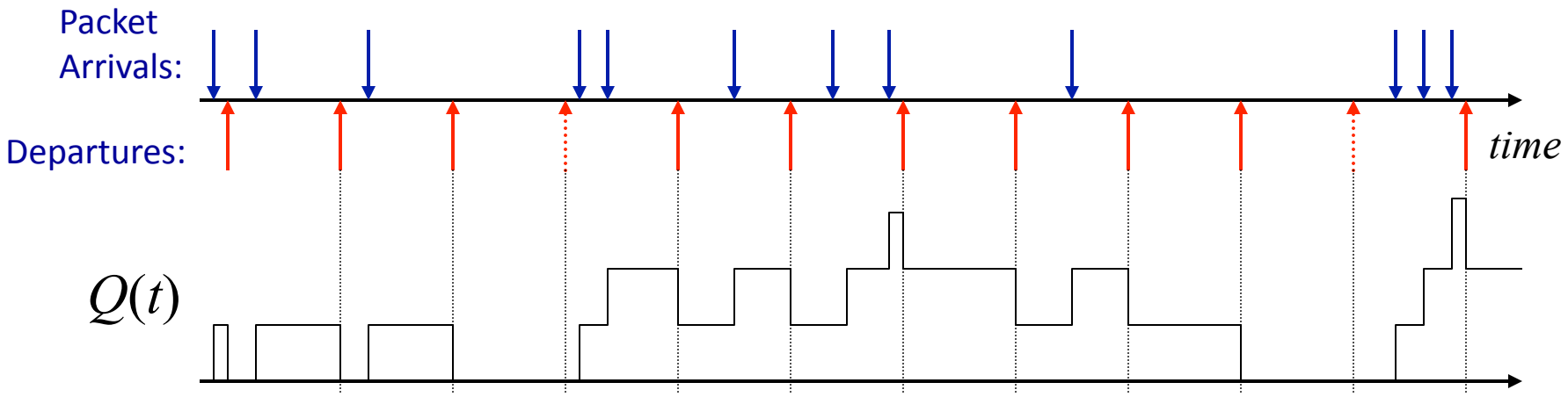


$$\text{End-to-end delay},\ t = \sum_i \left(\frac{p}{r_i} + \frac{l_i}{c}\right) + \left(\frac{M}{p} - 1\right)\frac{p}{r_{\min}}$$

This term describes the queuing delay at the bottleneck link.

This behavior is going to come up when we talk about TCP again next week!

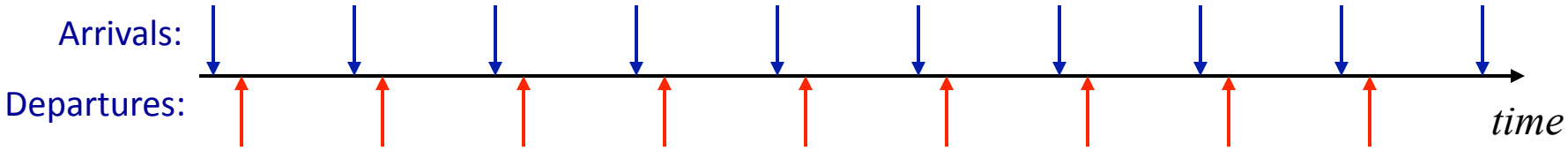
Time evolution of a queue



Queue Property #1

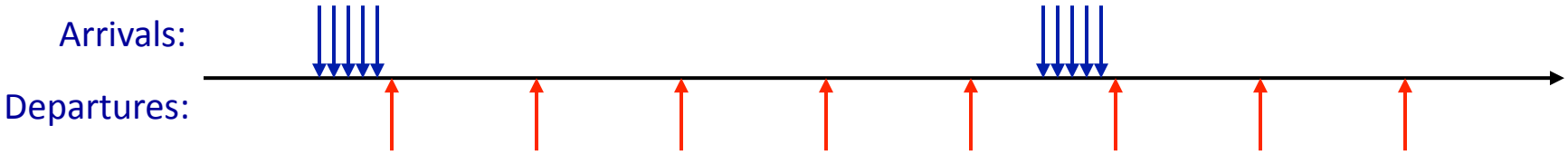
"Burstiness increases delay"

Example 1: Periodic single arrivals



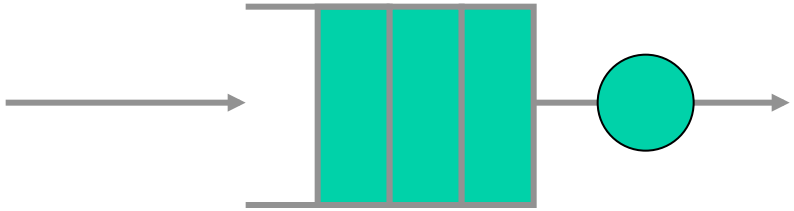
Queue Property #1
"Burstiness increases delay"

Example 2: Periodic burst arrivals



In general, burstiness increases delay

Queue Property #3
"Little's Result"



$$L = \lambda d$$

Where:

L is the average number of customers in the system (the number in the queue + the number in service),

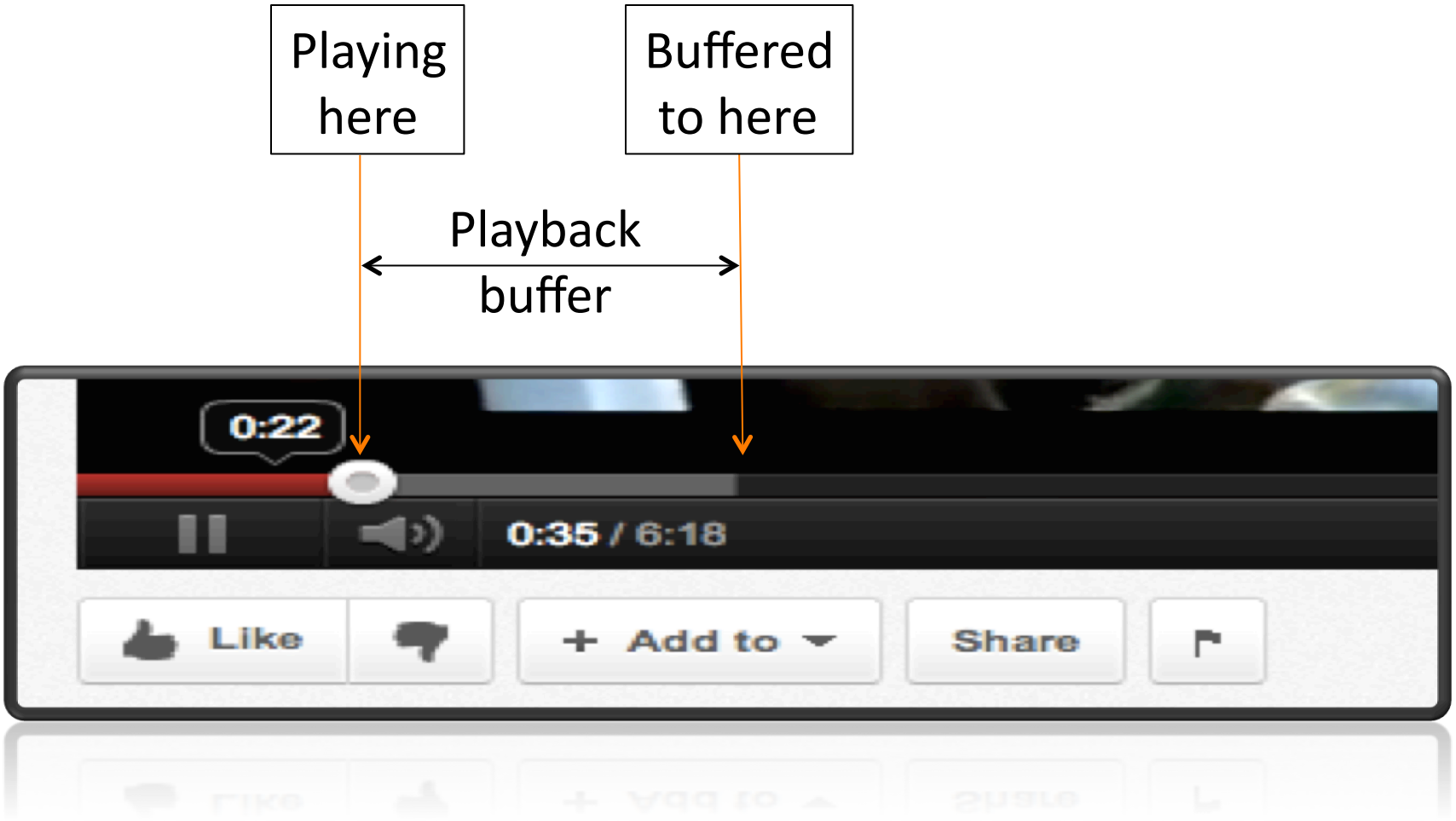
λ is the arrival rate, in customers per second, and

d is the average time that a customer waits in the system (time in queue + time in service).

Result holds so long as no customers are lost/dropped.

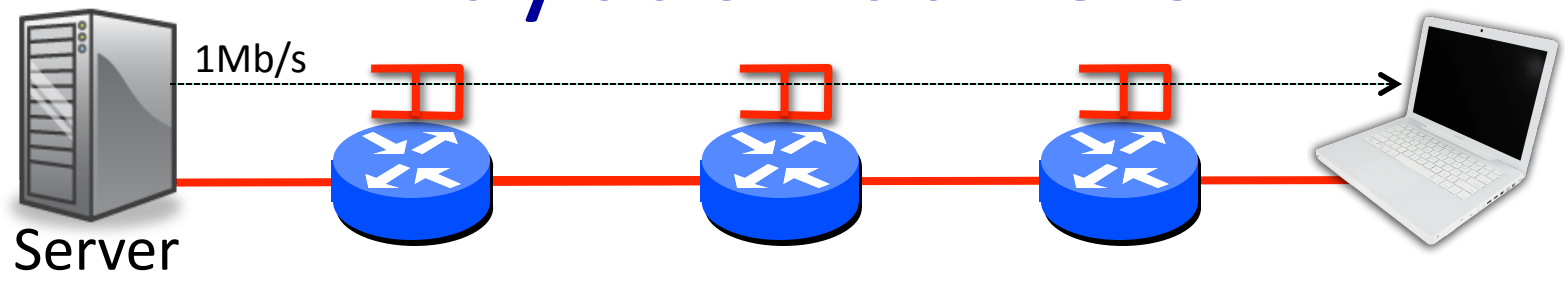
Real-time applications (e.g. YouTube and Skype) have to cope with variable queueing delay

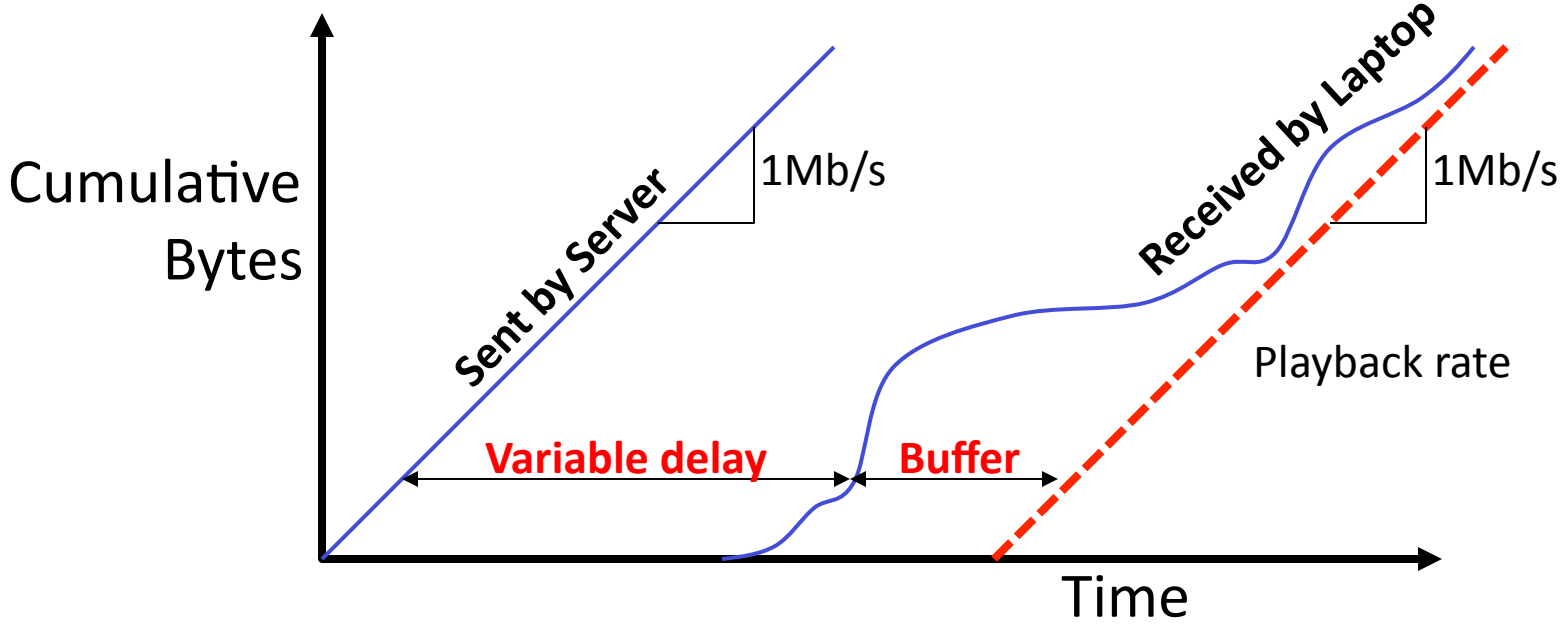
Playback buffers



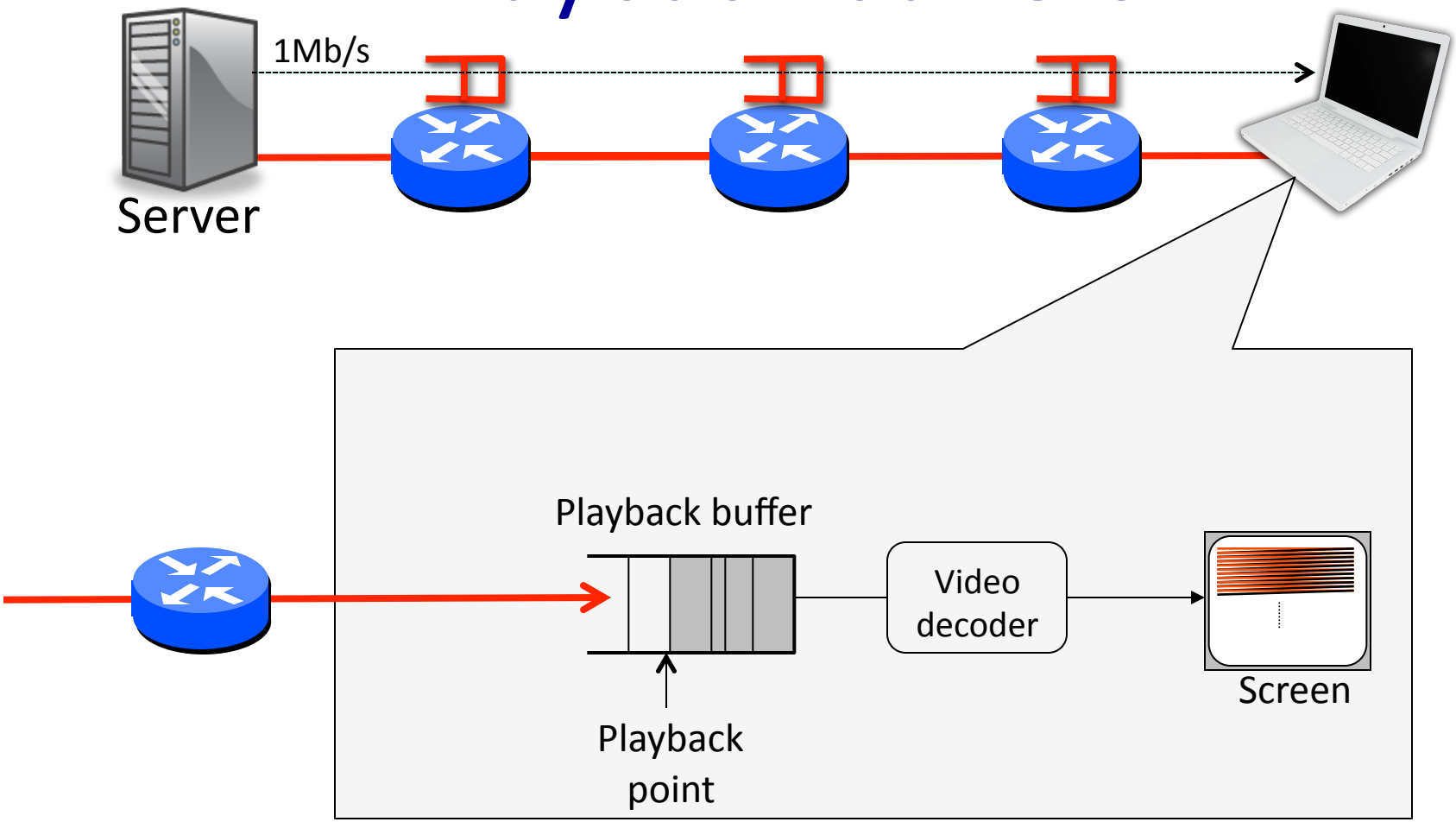
From: youtube.com

Playback buffers

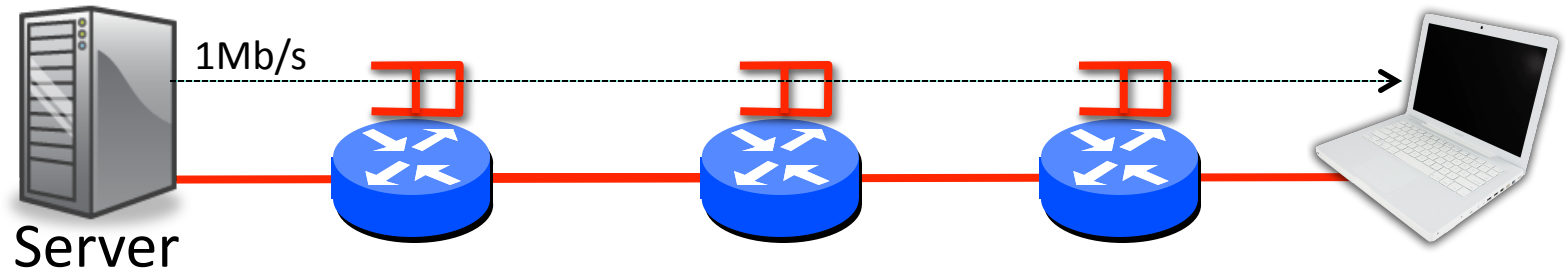


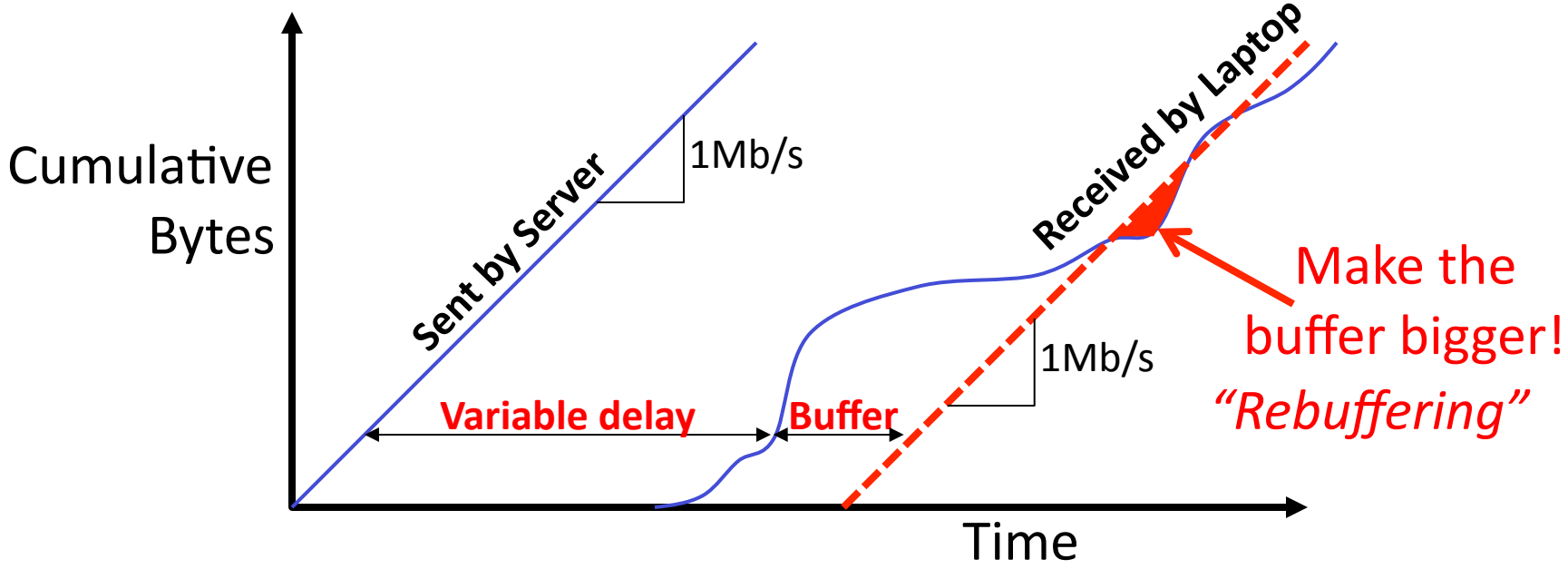


Playback buffers



If the buffer is too small



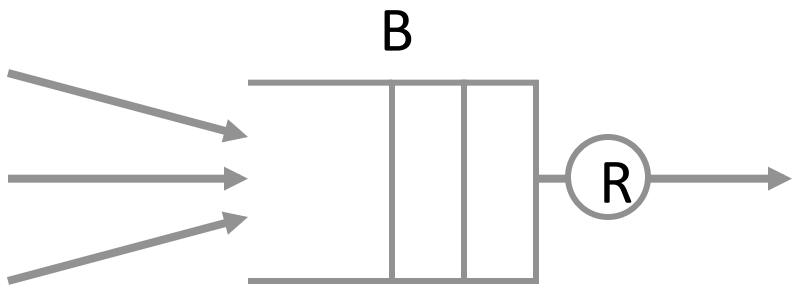


Summary

Real-time applications use playback buffers to absorb the variation in queuing delay.

Playback buffer must be at least as large as data rate times the queuing delay variation.

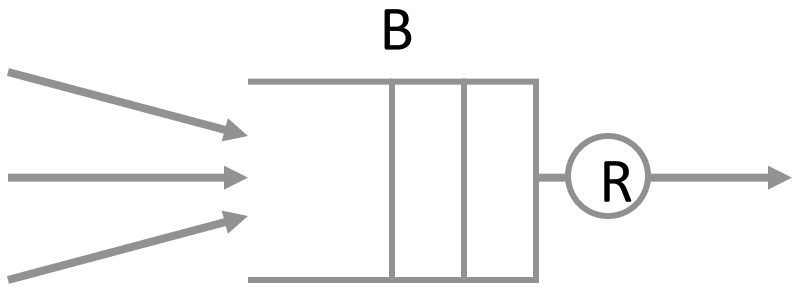
FIFO is a free for all



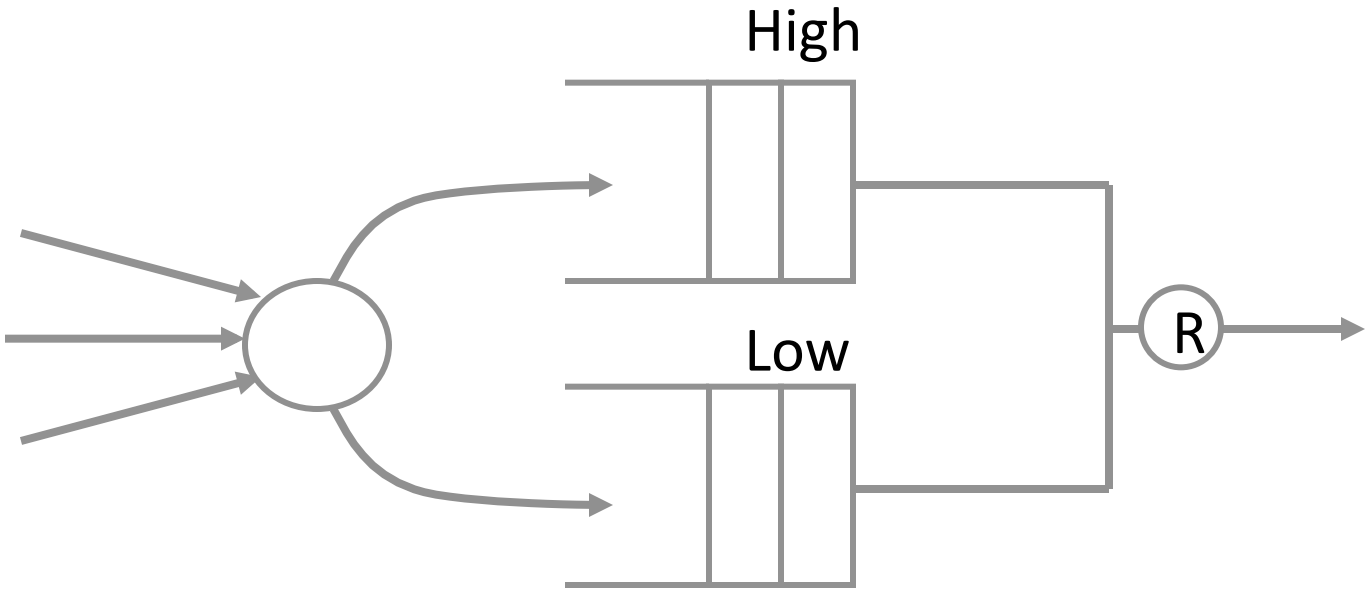
Outline

1. Strict priorities
2. Rate guarantees

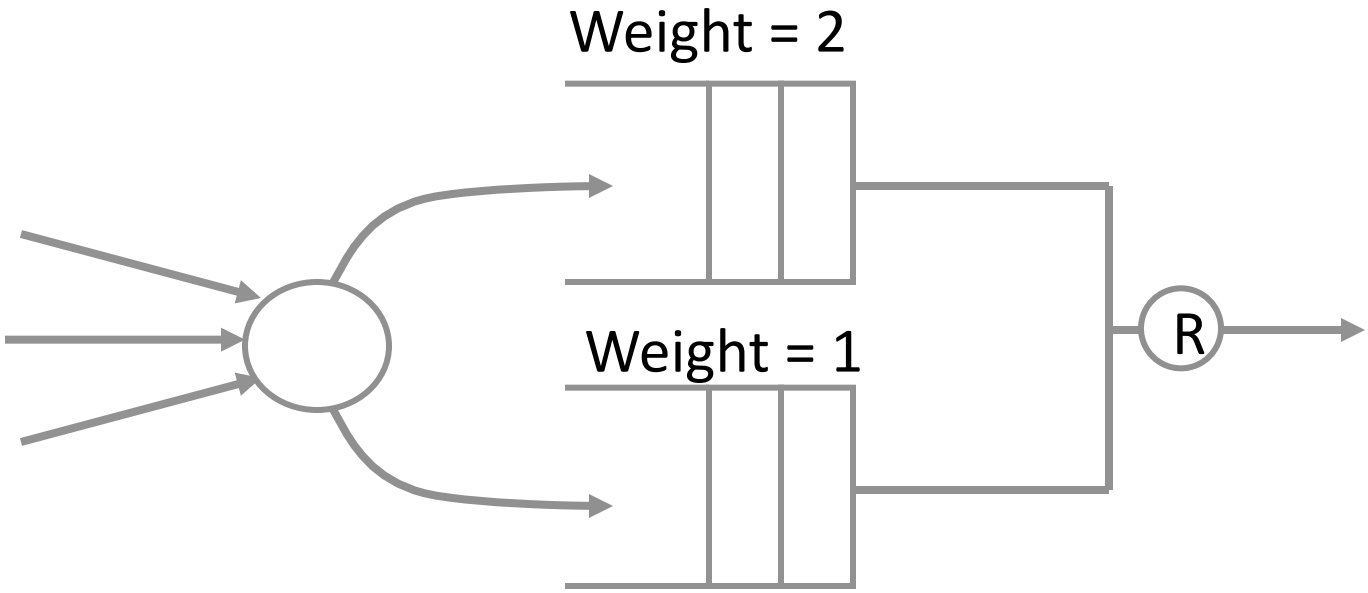
Single FIFO



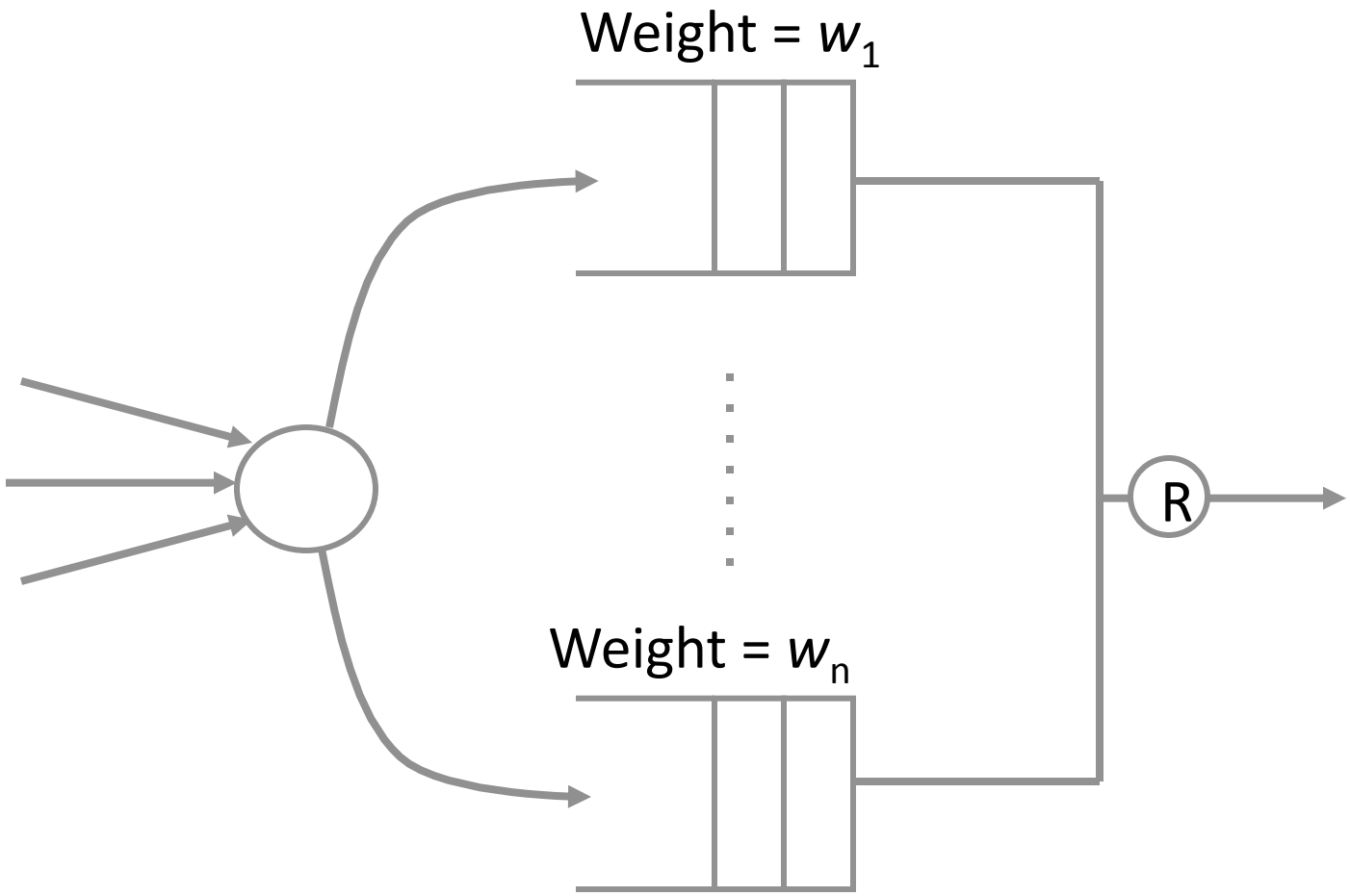
Strict Priorities



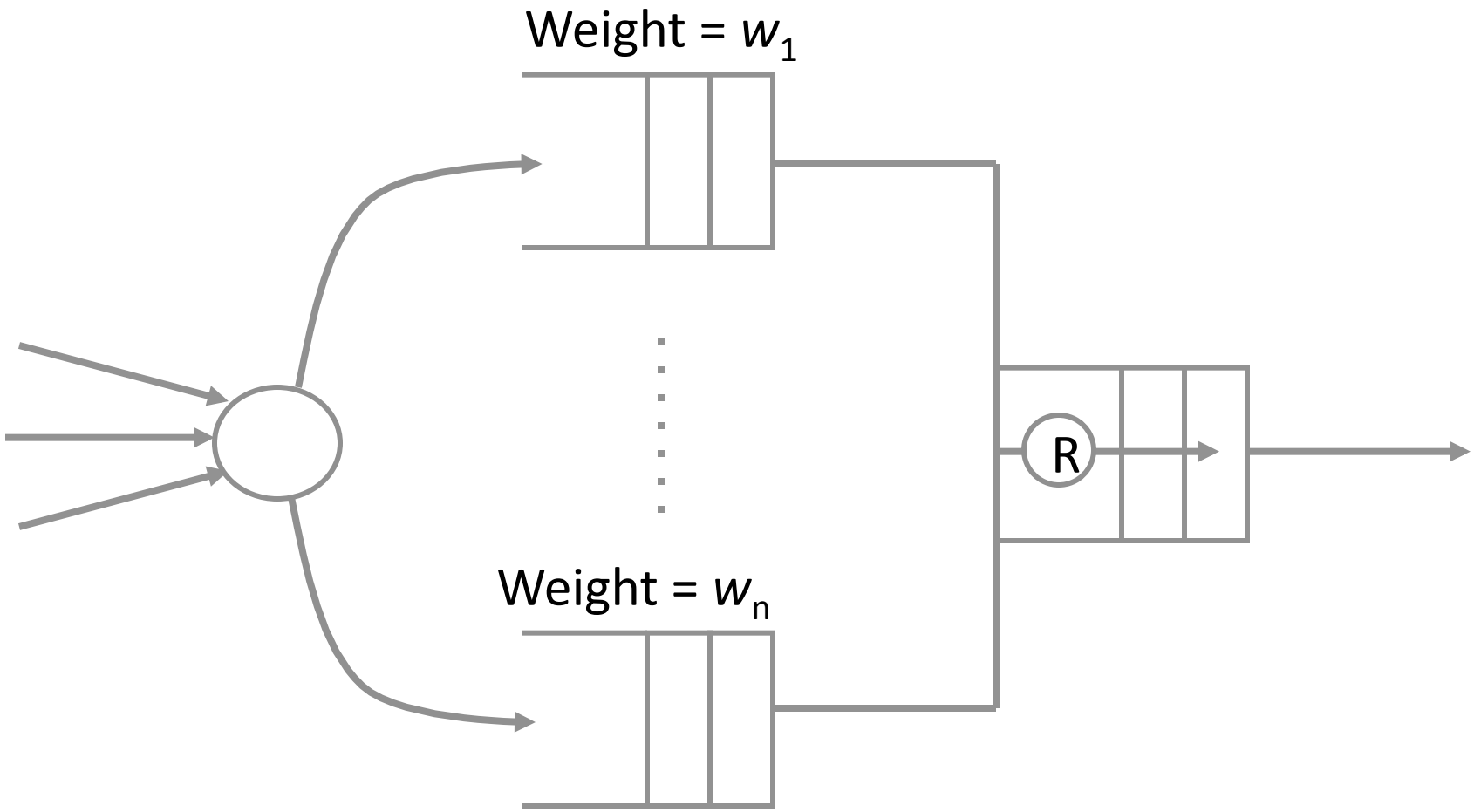
Weighted Priorities



Weighted Priorities



Weighted Fair Queueing



Summary

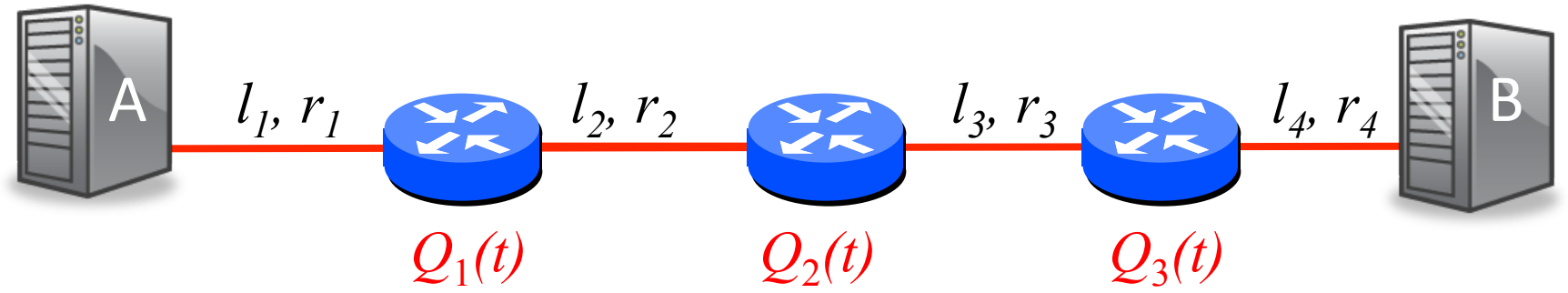
FIFO queues are a free for all: No priorities and no guaranteed rates.

Strict priorities: High priority traffic "sees" a network with no low priority traffic. Useful if we have limited amounts of high priority traffic.

Weighted Fair Queueing (WFQ) lets us give each flow a *rate guarantee*, by scheduling them in order of their bit-by-bit finishing times.

From Rate to Delay

Delay guarantees: Intuition



$$\text{End-to-end delay, } \tau = \sum_i \left(\frac{p}{r_i} + \frac{l_i}{c} + Q_i(t) \right)$$

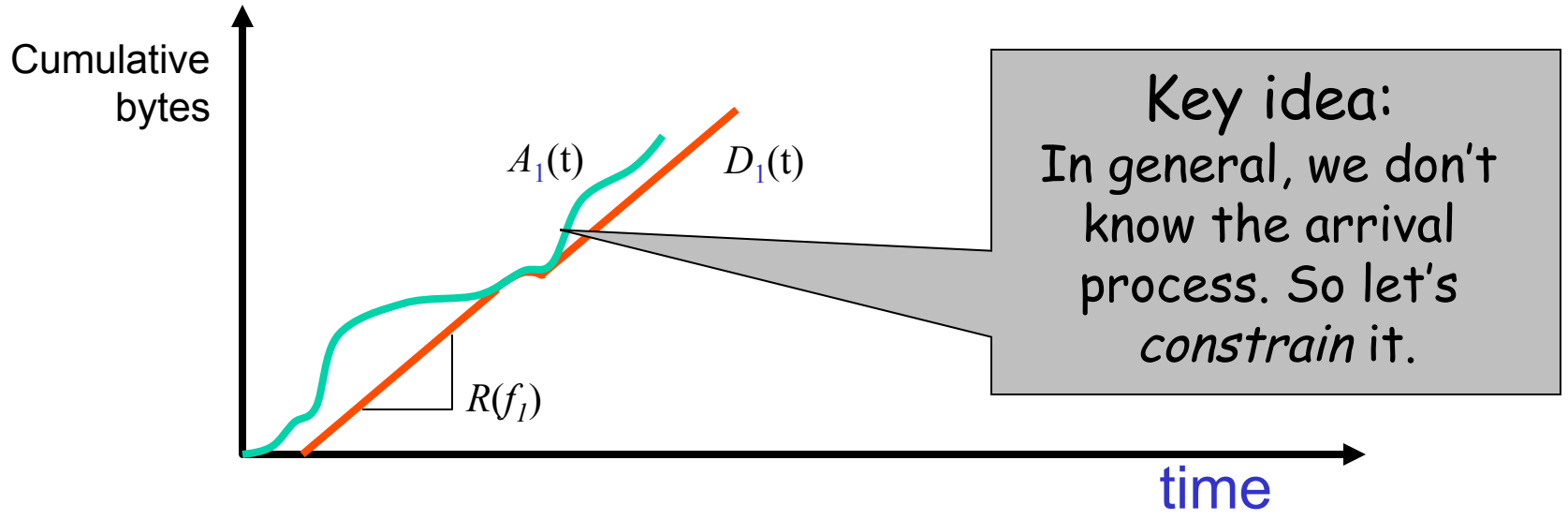
If we know the upper bound of $Q_1(t)$, $Q_2(t)$ and $Q_3(t)$, then we know the upper bound of the end-to-end delay.

So how can we control the delay of packets?

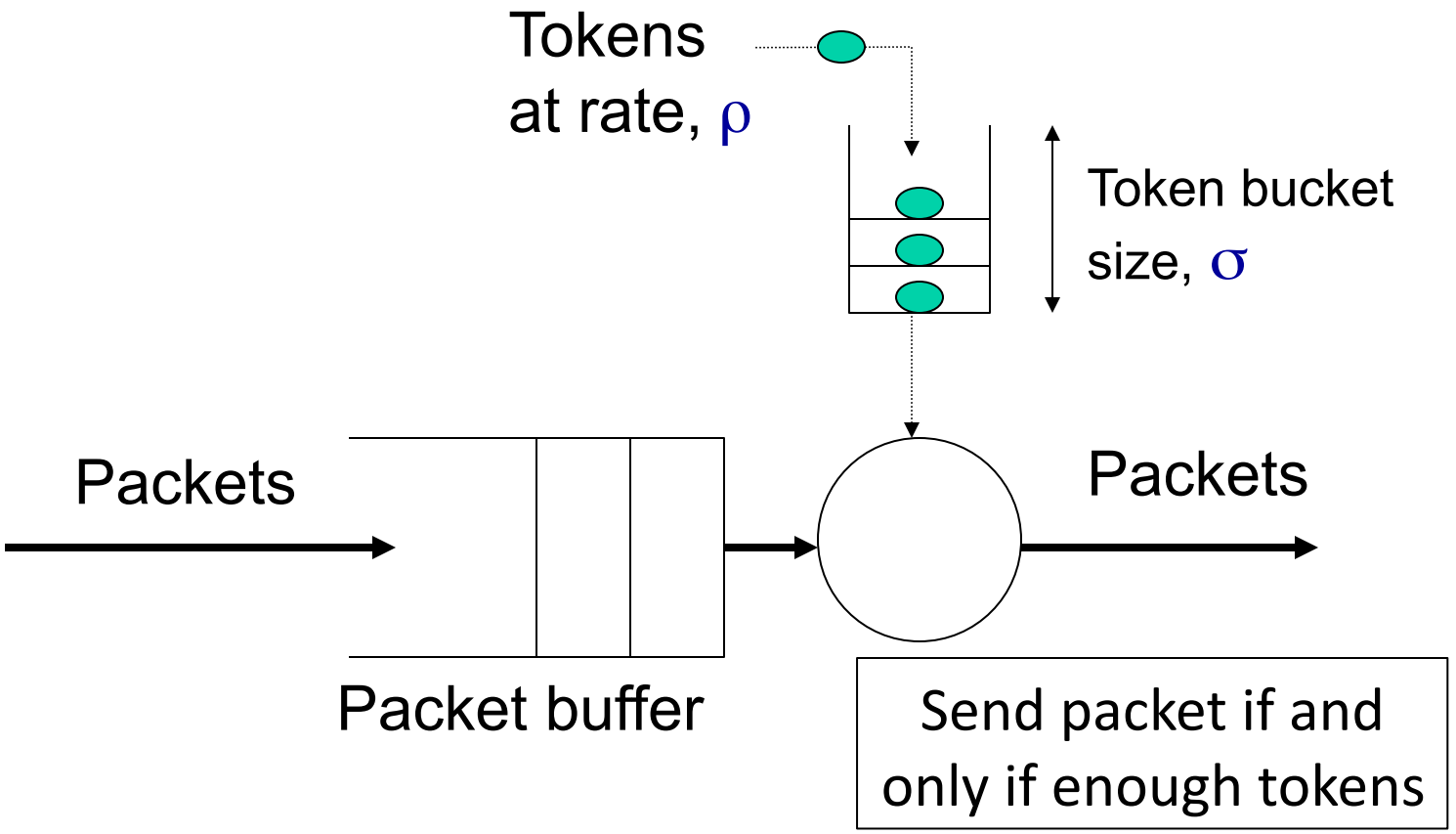
What we already know how to control:

1. The rate at which a queue is served (WFQ).
2. The size of each queue.

How do we make sure no packets are dropped?


Cumulative bytes
$A_1(t)$
$D_1(t)$
$R(f_1)$
time
Key idea:
In general, we don't know the arrival process. So let's *constrain* it.

The leaky bucket regulator



Delay Guarantees

If we know the size of a queue and the rate at which it is served, then we can bound the delay through it.

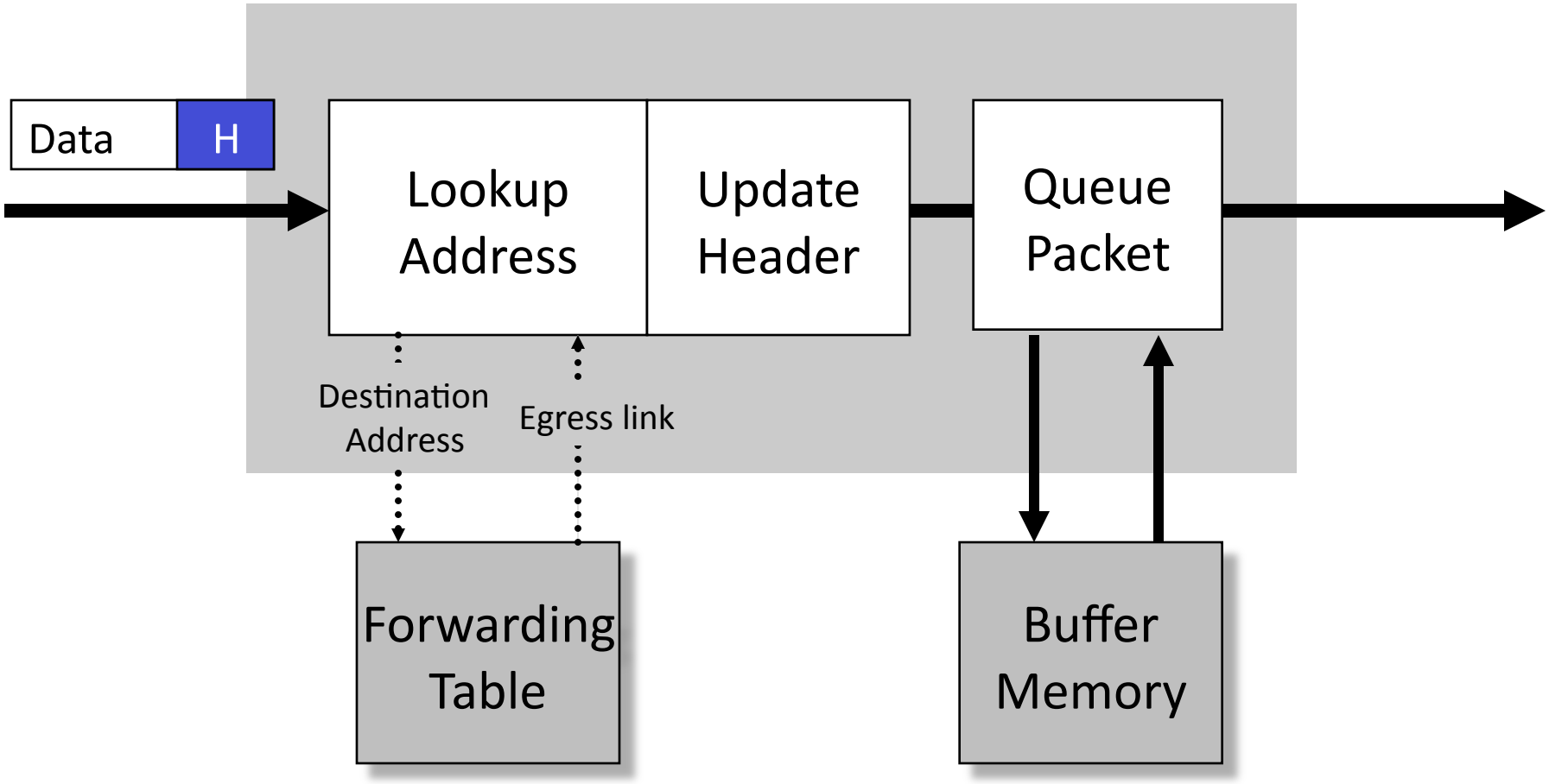
We can pick the size of the queue, and WFQ lets us pick the rate at which it is served.

Therefore, we just need a way to prevent packets being dropped along the way. For this, we use a leaky bucket regulator.

We can therefore bound the end to end delay.

Switching

Generic Packet Switch



Ethernet Switch

1. Examine the header of each arriving frame.
2. If the Ethernet DA is in the forwarding table, forward the frame to the correct output port(s).
3. If the Ethernet DA is not in the table, broadcast the frame to all ports (except the one through which the frame arrived).
4. Entries in the table are learned by examining the Ethernet SA of arriving packets.

Internet Router

1. If the Ethernet DA of the arriving frame belongs to the router, accept the frame. Else drop it.
2. Examine the IP version number and length of the datagram.
3. Decrement the TTL, update the IP header checksum.
4. Check to see if TTL == 0.
5. If the IP DA is in the forwarding table, forward to the correct egress port(s) for the next hop.
6. Find the Ethernet DA for the next hop router.
7. Create a new Ethernet frame and send it.

Basic Operations

1. Lookup Address: How is the address looked up in the forwarding table?
2. Switching: How is the packet sent to the correct output port?

Lookup Address: Ethernet

Ethernet addresses (in a switch)

Match	Action
Ethernet DA = 0xA8B72340E678	Forward to port 7
Ethernet DA = 0xB3D22571053B	Forward to port 3
…	…

Methods

- Store addresses in hash table (maybe 2-way hash)
- Look for exact match in hash table

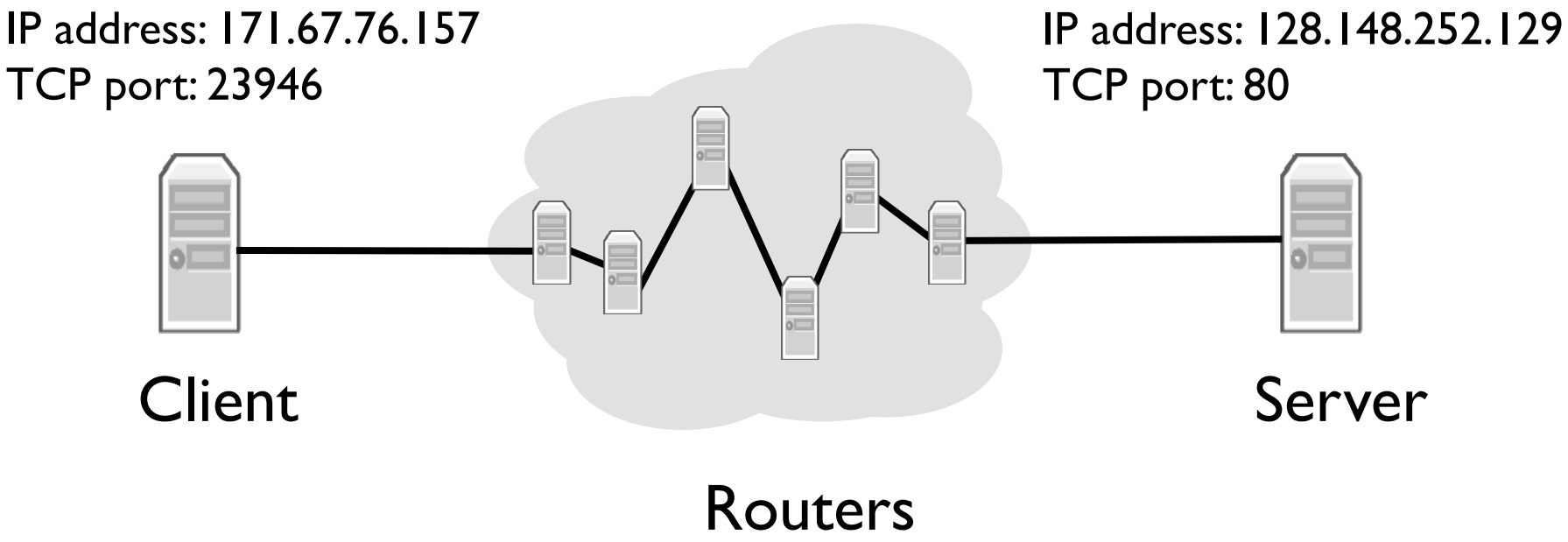
Lookup Address: IP

IP addresses (in a router)

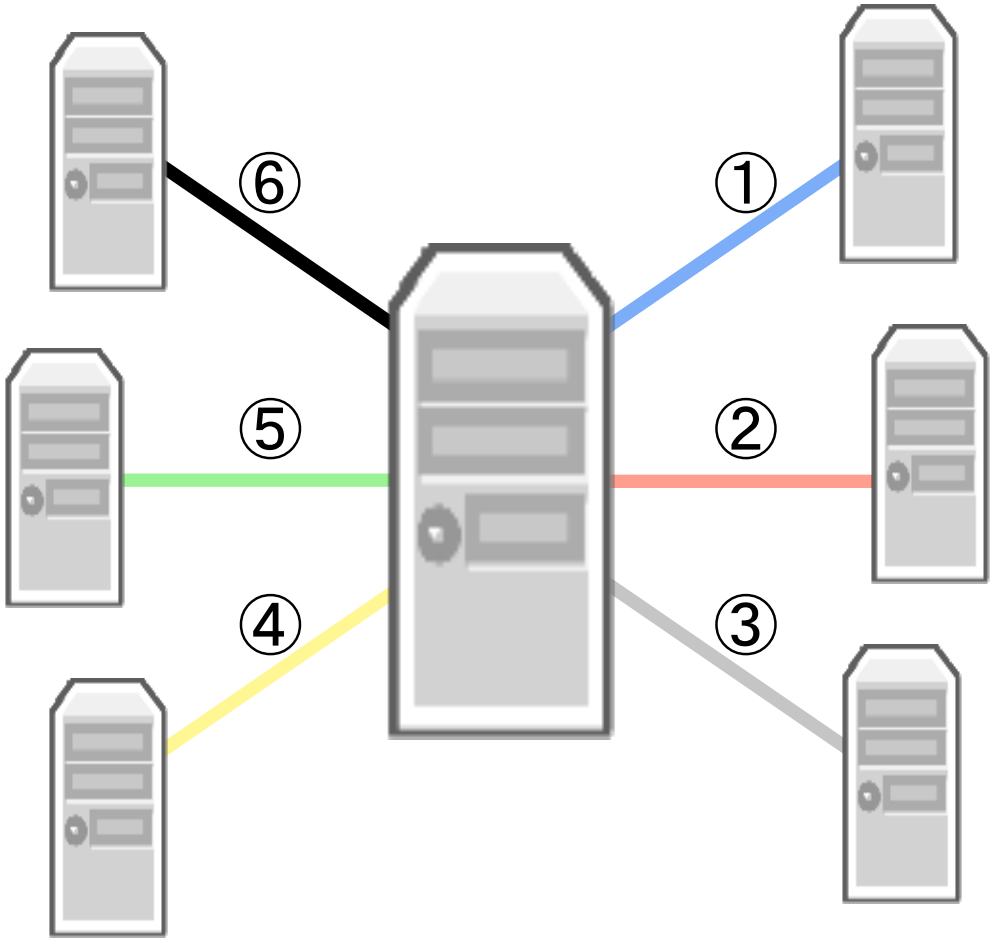
Match	Action
IP DA = 127.43.57.99	Forward to 56.99.32.16
IP DA = 123.66.44.X	Forward to 22.45.21.126
IP DA = 76.9.X.X	Forward to 56.99.32.16
…	…

Lookup is a longest prefix match, not an exact match

Inside the Stream



Inside Each Hop



dest	link
default	1
171.33.x.x	5
23.x.x.x	2
28.33.5.x	4
171.32.x.x	2
67.x.x.x	6
216.x.x.x	1

address: 216.239.47.186

Longest Prefix Match

Algorithm IP routers use to chose matching entry from forwarding table

dest	link
0.0.0.0/0	1
171.33.0.0/16	5

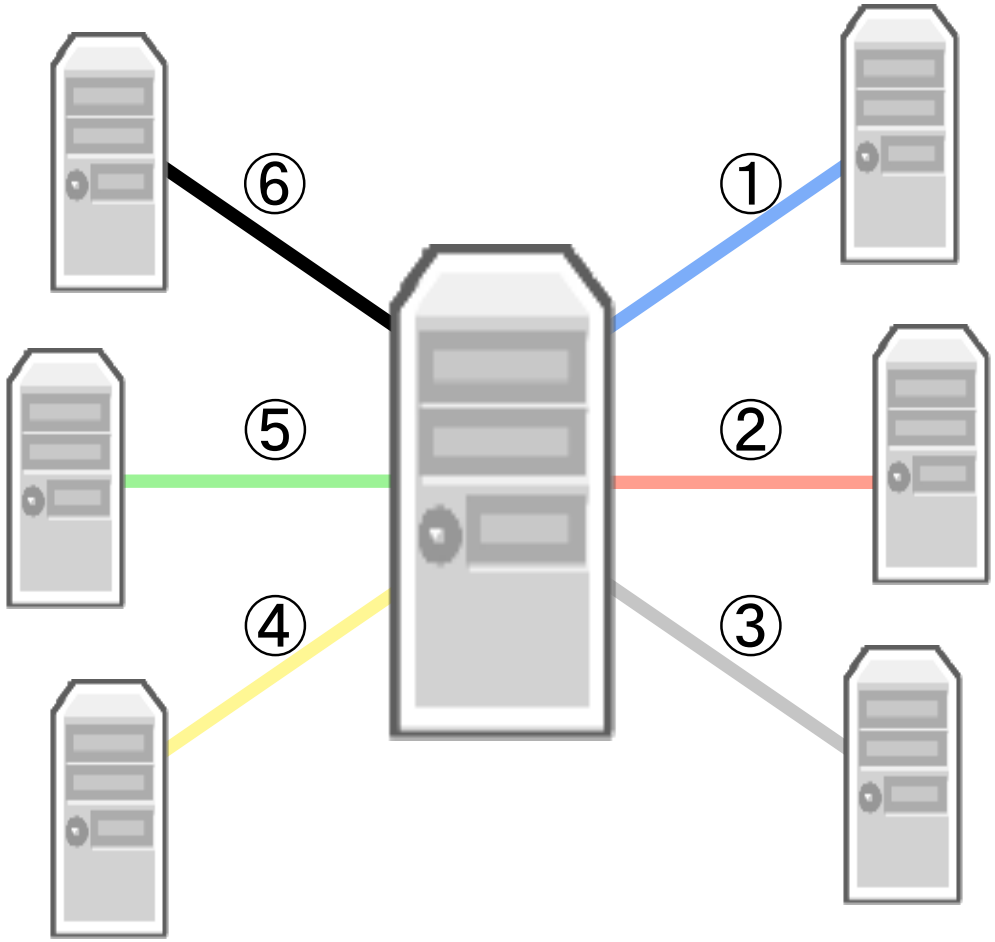
Forwarding table is a set of CIDR entries

- An address might match multiple entries
- E.g., 171.33.0.1 matches both entries on right

Algorithm: use forwarding entry with the longest matching prefix

- Longest prefix match will chose link 5 for 171.33.0.1

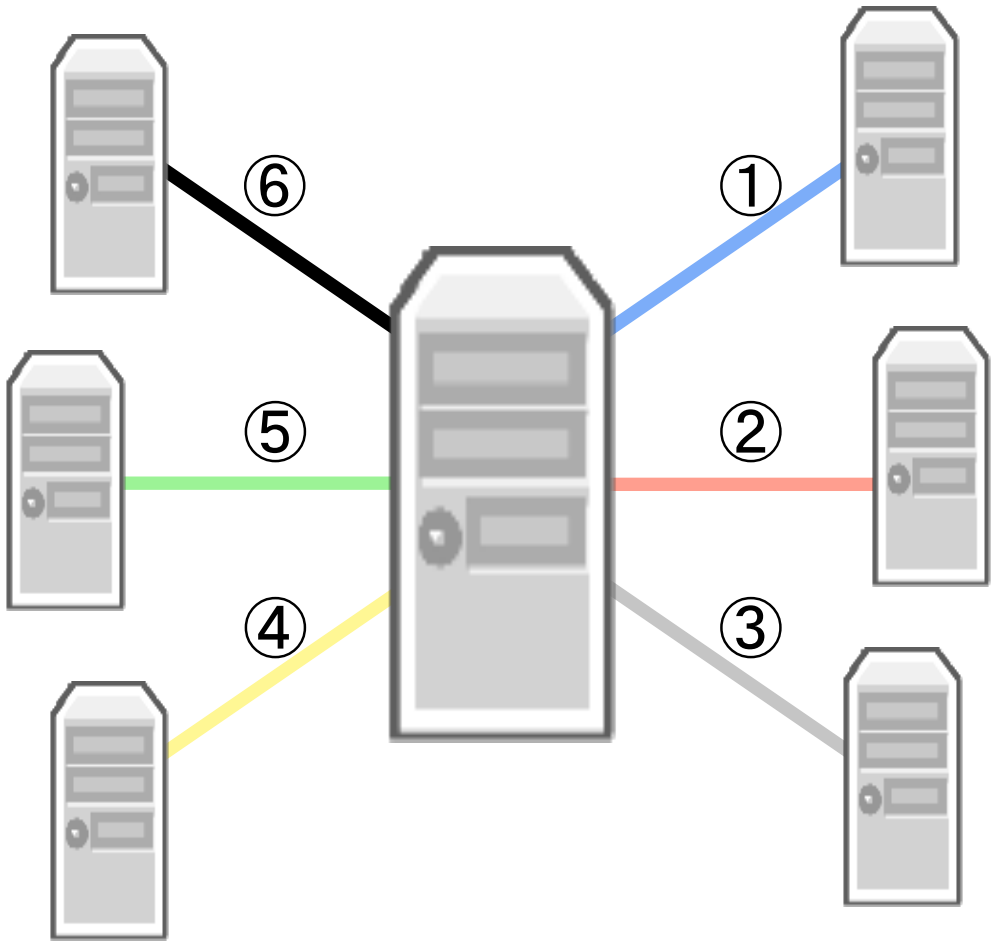
Inside Each Hop



dest	link
default	1
171.33.x.x	5
23.x.x.x	2
28.33.5.x	4
171.32.x.x	2
67.x.x.x	6
216.x.x.x	1

address: 216.239.47.186

Inside Each Hop (for real)



address: 216.239.47.186

dest	link
default	1
171.33.x.x	5
23.x.x.x	2
28.33.5.x	4
171.32.x.x	2
67.x.x.x	6
216.x.x.x	1

dest	link
0.0.0.0/0	1
171.33.0.0/16	5
23.0.0.0/8	2
28.33.5.0/24	4
171.32.0.0/16	2
67.0.0.0/8	6
216.0.0.0/8	1