

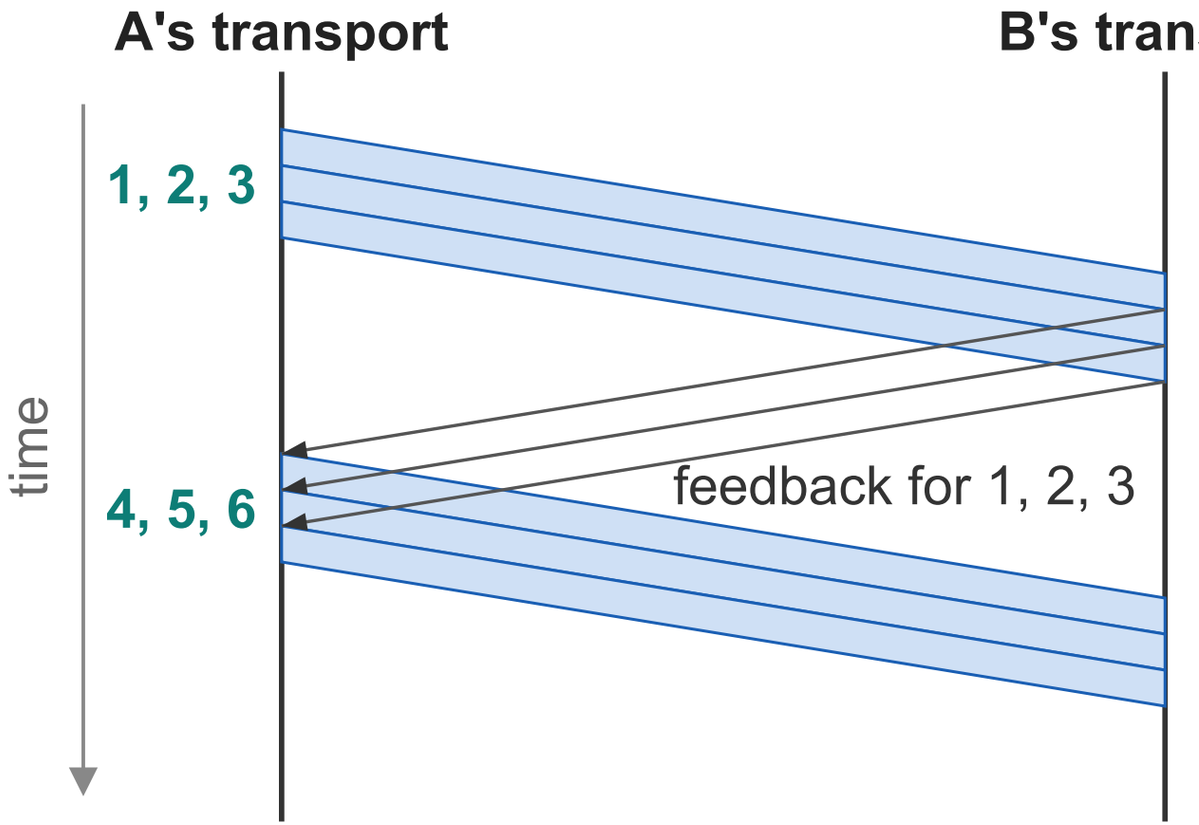
UNIVERSITY OF WATERLOO

Congestion control - Part I

CS 456/656, Fall 2026

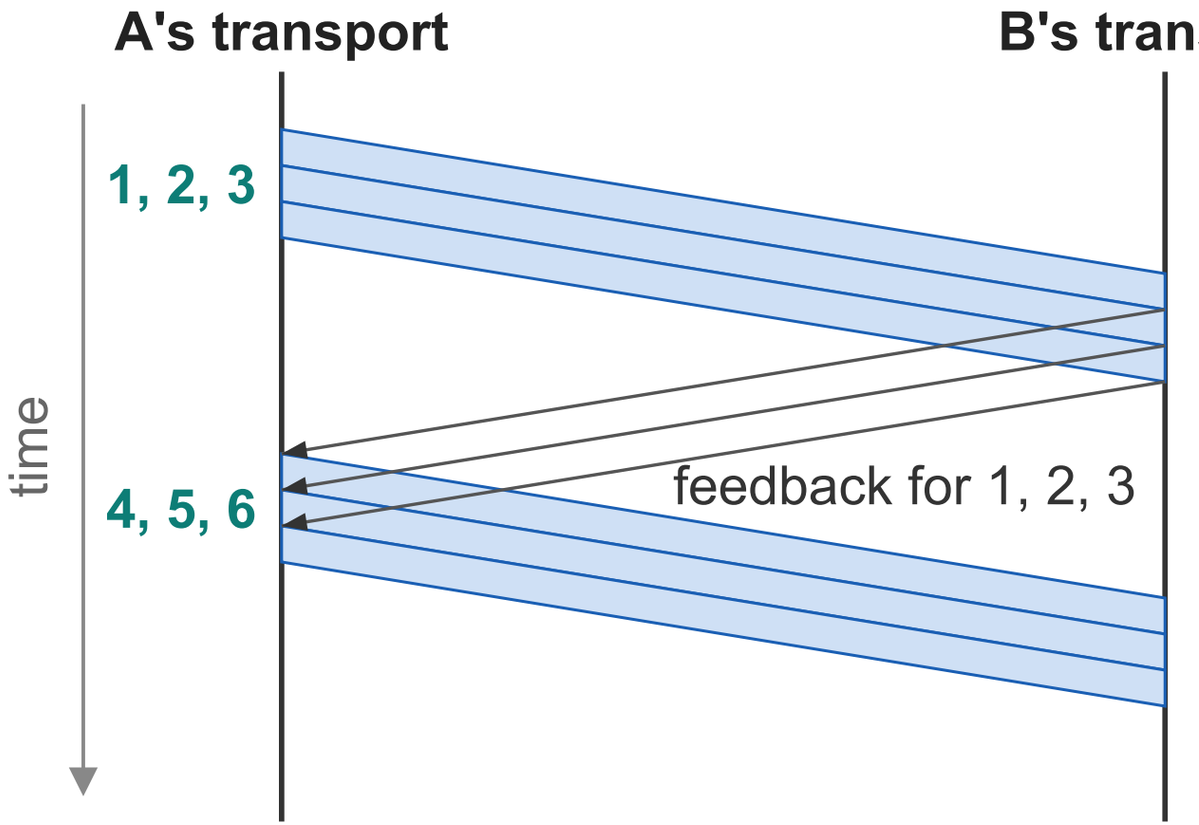
Mina Tahmasbi Arashloo

Recap: a batch of three packets at a time



A sends 3 packets back to back,
without waiting for feedback
in between.

Recap: a batch of three packets at a time



A sends 3 packets back to back, without waiting for feedback in between.

3 packets every round trip:
3 times the utilization

0.024 / (30 + 0.024) = 0.0008

Recap: what is a good batch size?

- **Too small:** we are underutilizing the network, and it takes longer to get the data delivered
- **Too big:**
  - The network is shared. If everyone sends everything they have at full speed, it will create congestion that can result in increasing drops and delays
  - The sender may “overwhelm” the receiver

Recap: what is a good batch size?

- **Too small:** we are underutilizing the network, and it takes longer to get the data delivered
- **Too big:**
  - The network is shared. If everyone sends everything they have at full speed, it will create congestion that can result in increasing drops and delays
  - The sender may “overwhelm” the receiver

✓ **flow control with a sliding window**

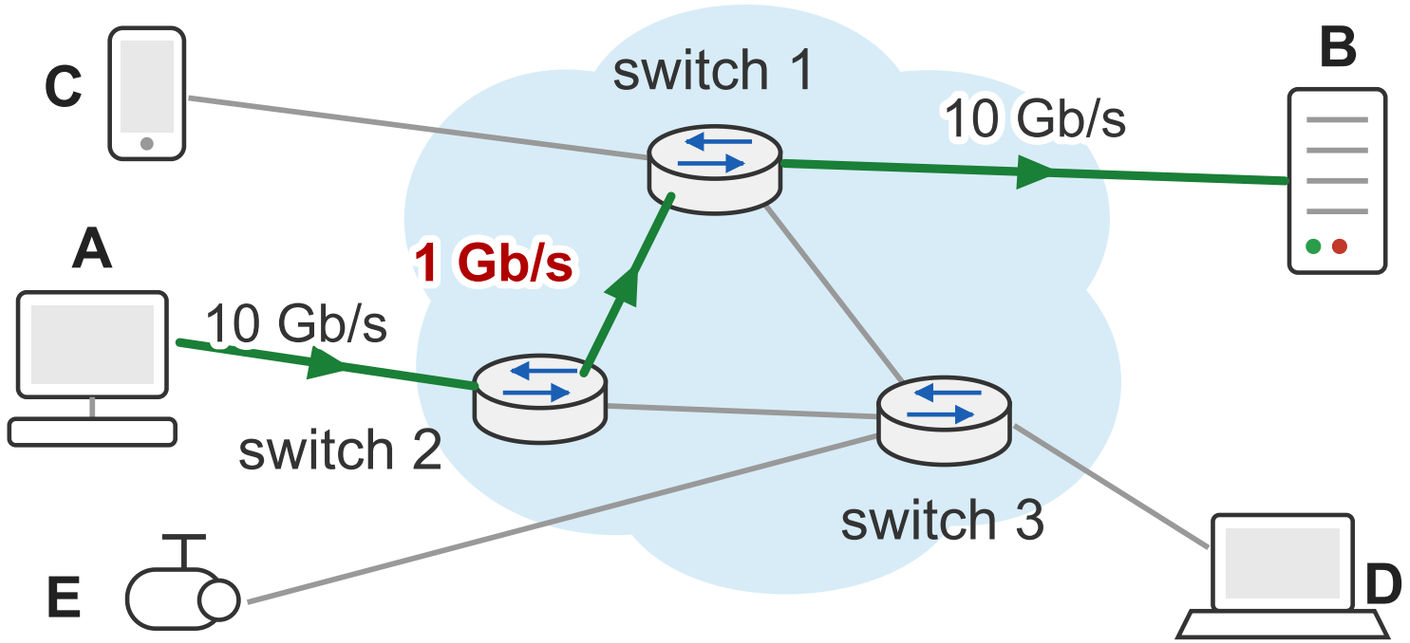
Recap: what is a good batch size?

Part 2

- **Too small:** we are underutilizing the network, and it takes longer to get the data delivered
- **Too big:**
  - The network is shared. If everyone sends everything they have at full speed, it will create congestion that can result in increasing drops and delays
  - The sender may “overwhelm” the receiver

✓ flow control with a sliding window

How fast should the sender send?



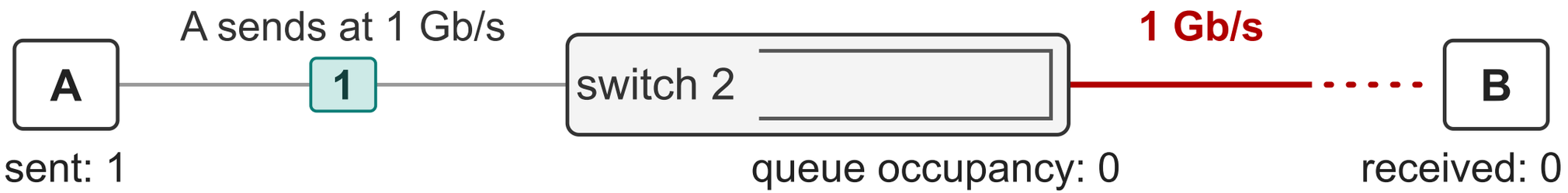
For intuition, we first talk about **sending rate**, not batch size (rate ≈ batch size / RTT).

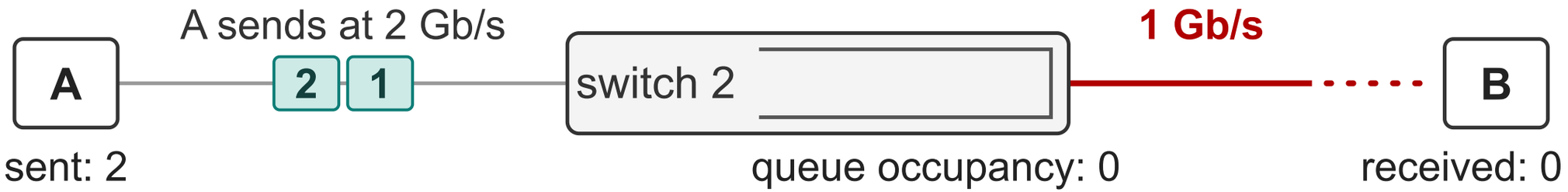
Consider a sender and a receiver:

- The sender knows exactly what path its traffic will take to the destination, and the properties (rate, propagation delay, etc.) of each link
- The sender always has more data to send

What if A sends faster than the slowest link?

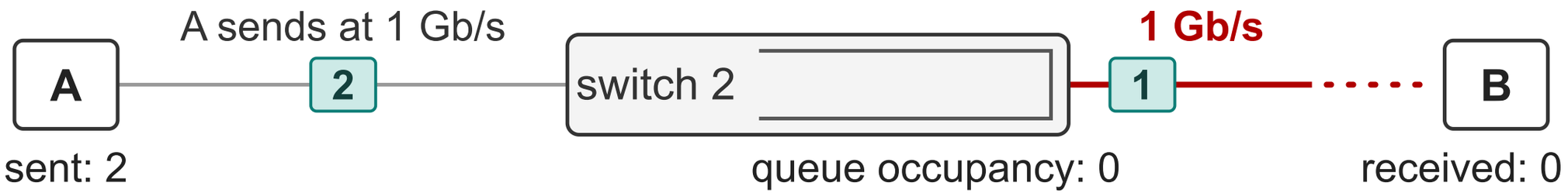
When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).

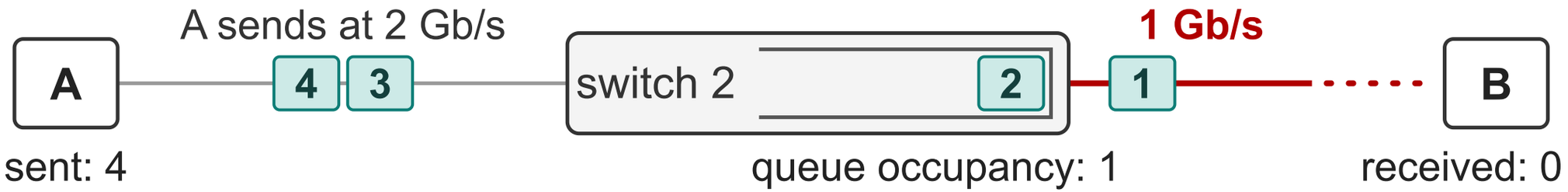




What if A sends faster than the slowest link?

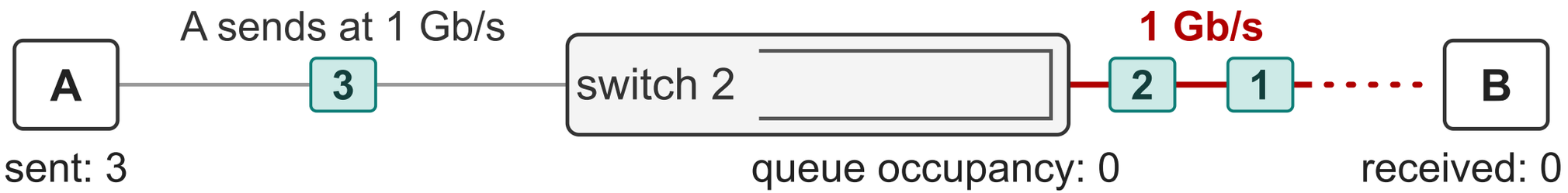
When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).

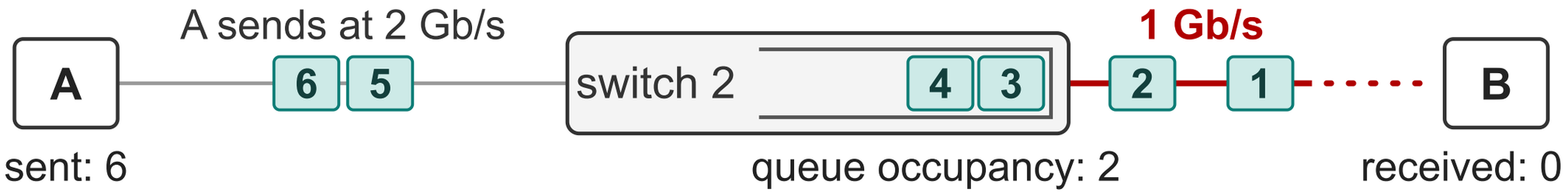




What if A sends faster than the slowest link?

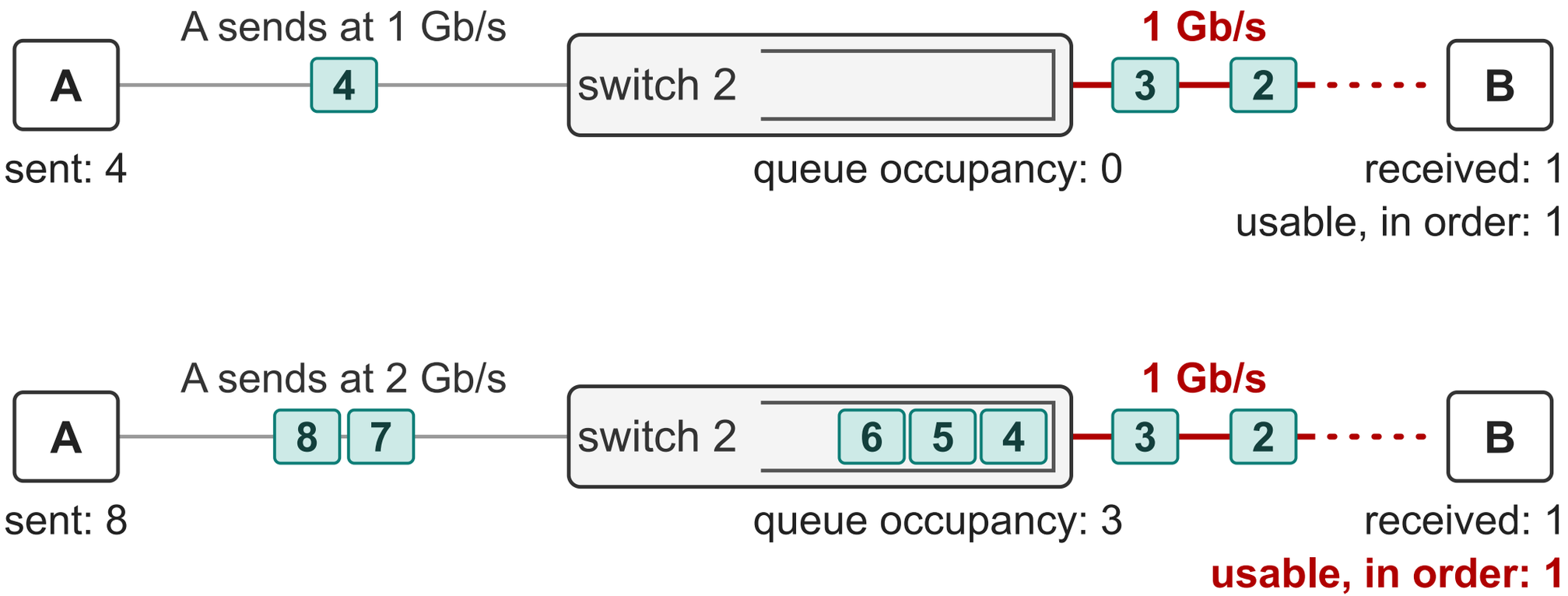
When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).





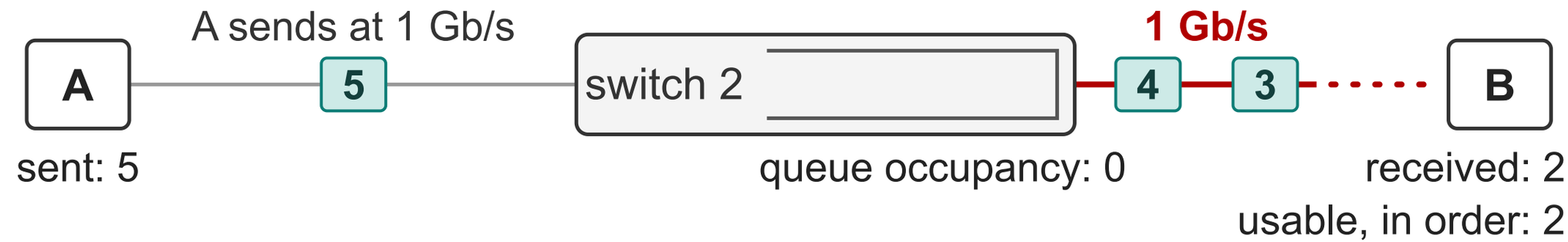
What if A sends faster than the slowest link?

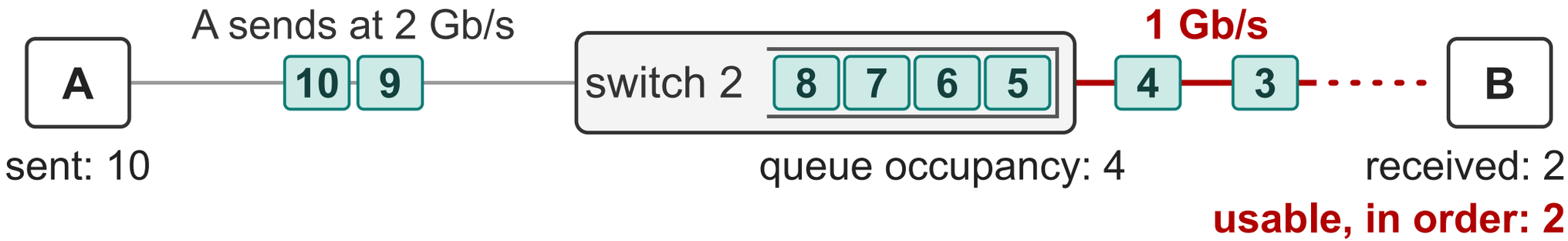
When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



What if A sends faster than the slowest link?

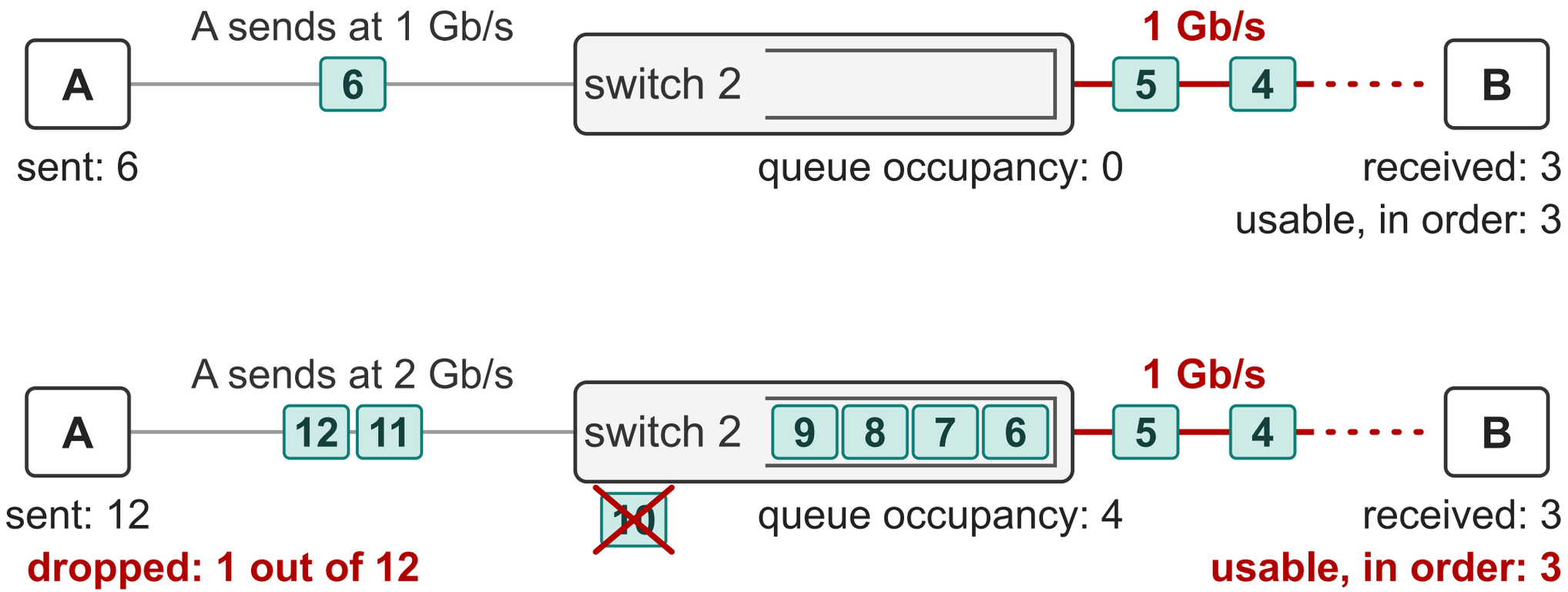
When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).





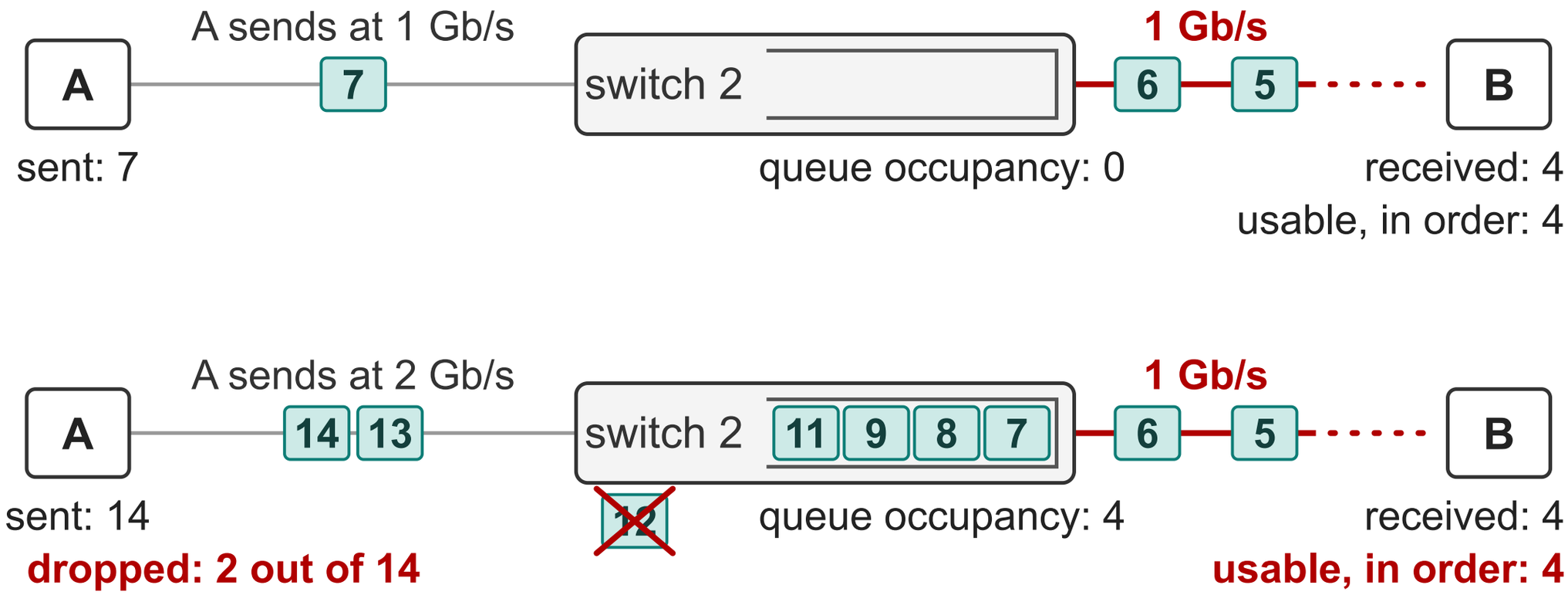
What if A sends faster than the slowest link?

When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



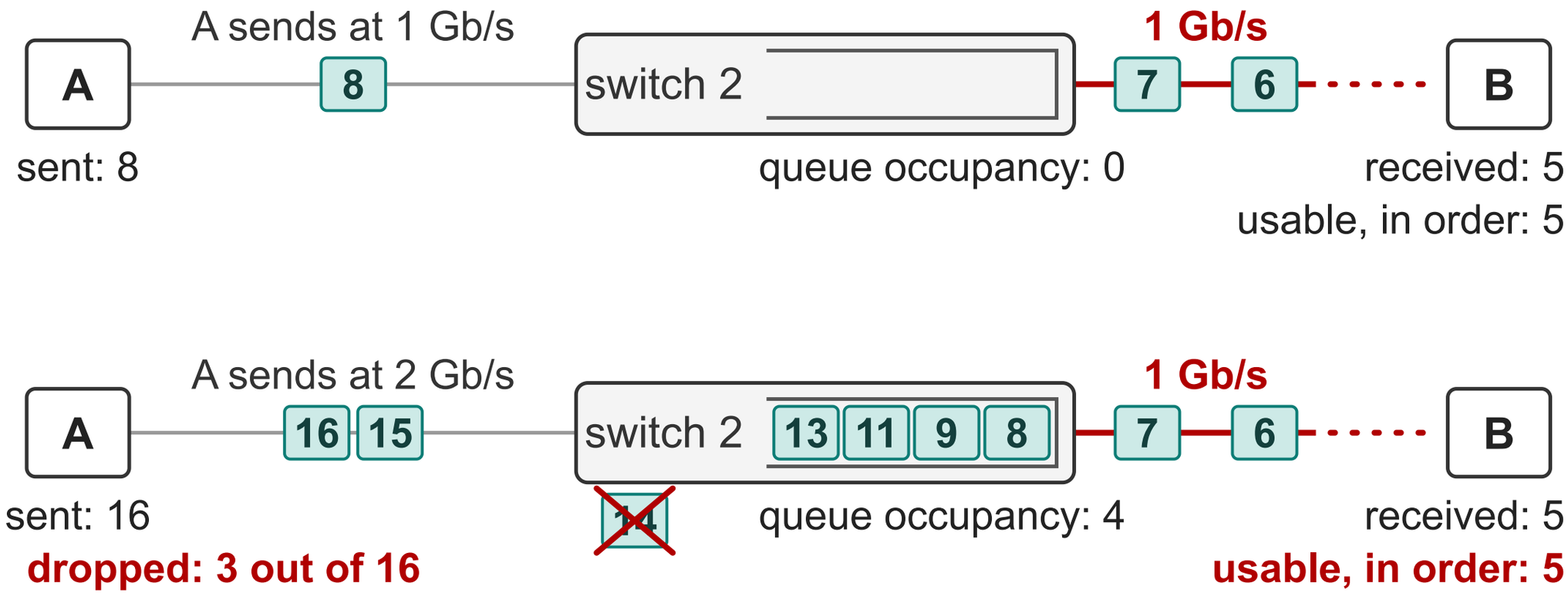
What if A sends faster than the slowest link?

When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



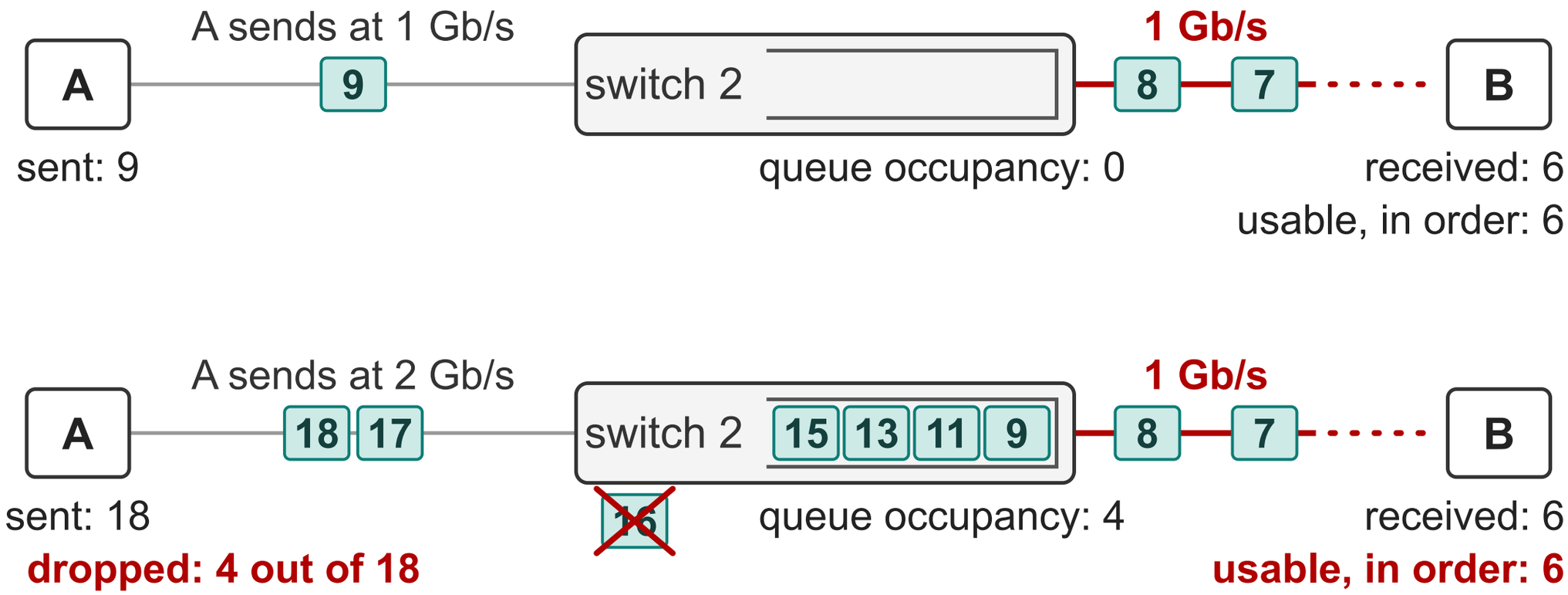
What if A sends faster than the slowest link?

When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



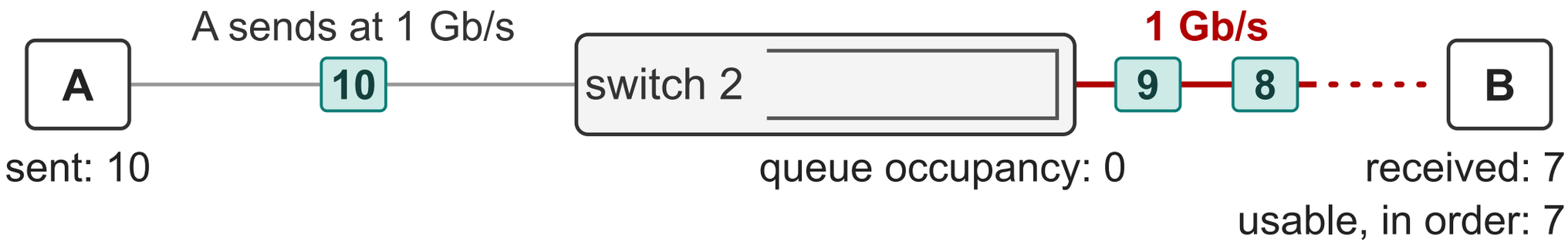
What if A sends faster than the slowest link?

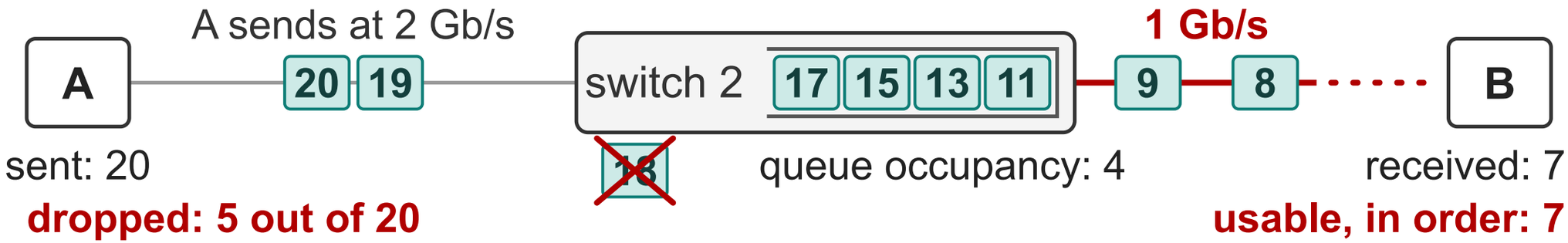
When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



What if A sends faster than the slowest link?

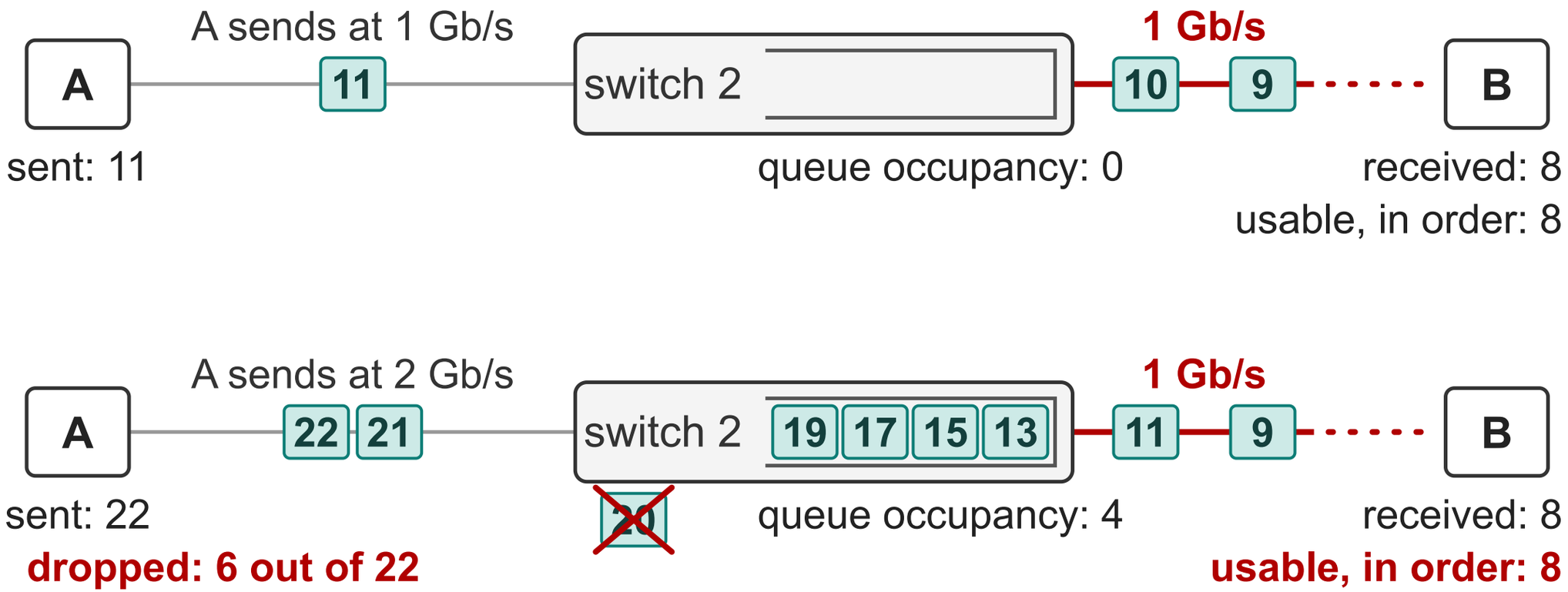
When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).





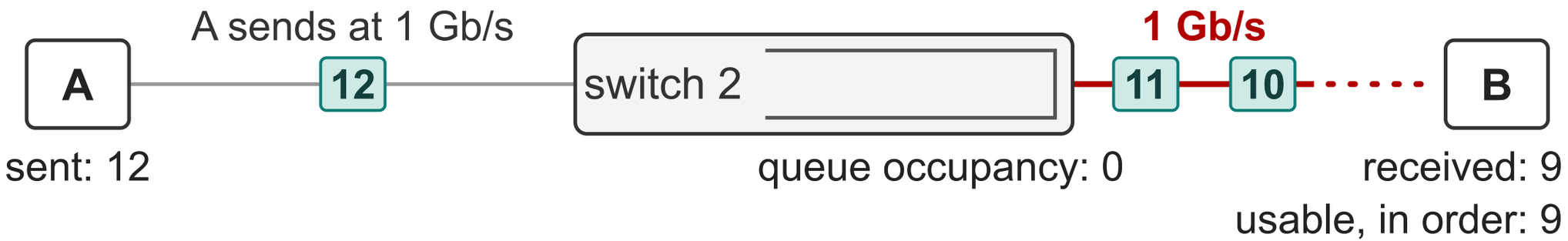
What if A sends faster than the slowest link?

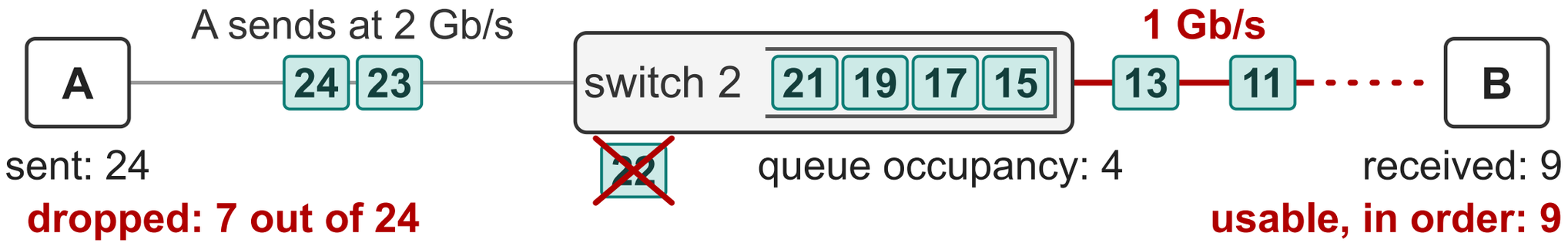
When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



What if A sends faster than the slowest link?

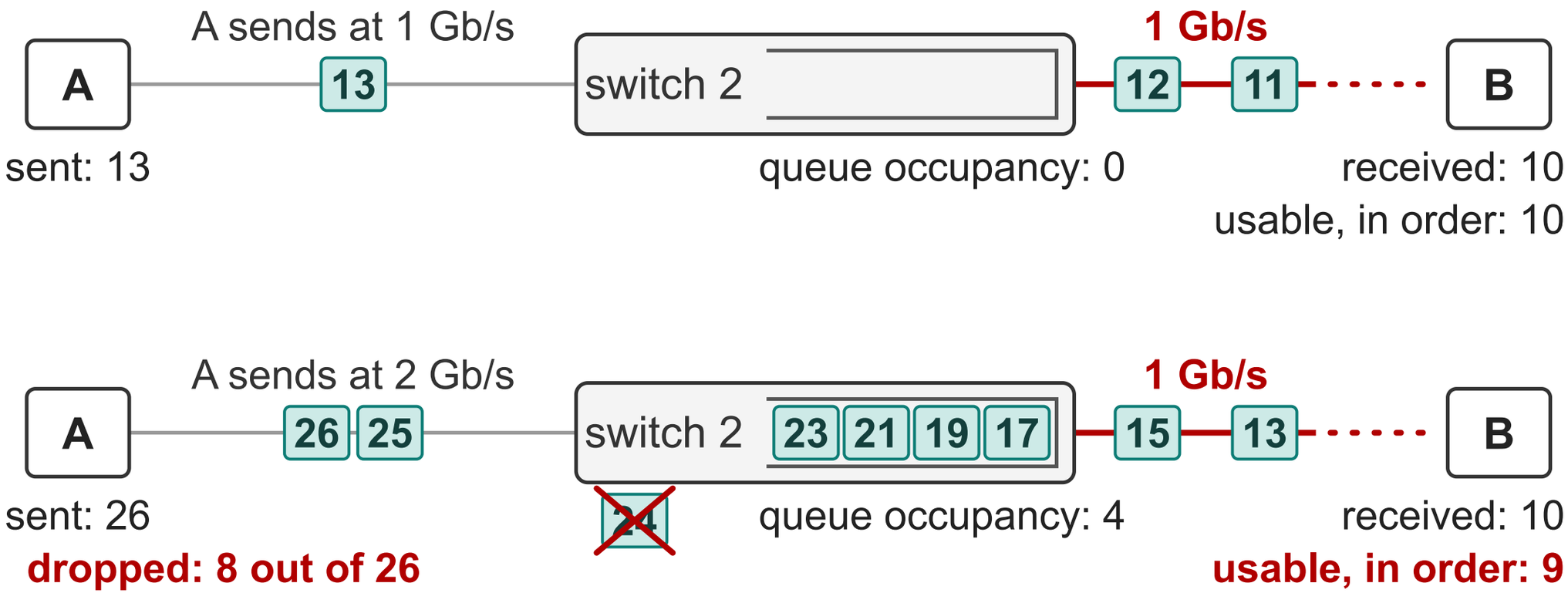
When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).





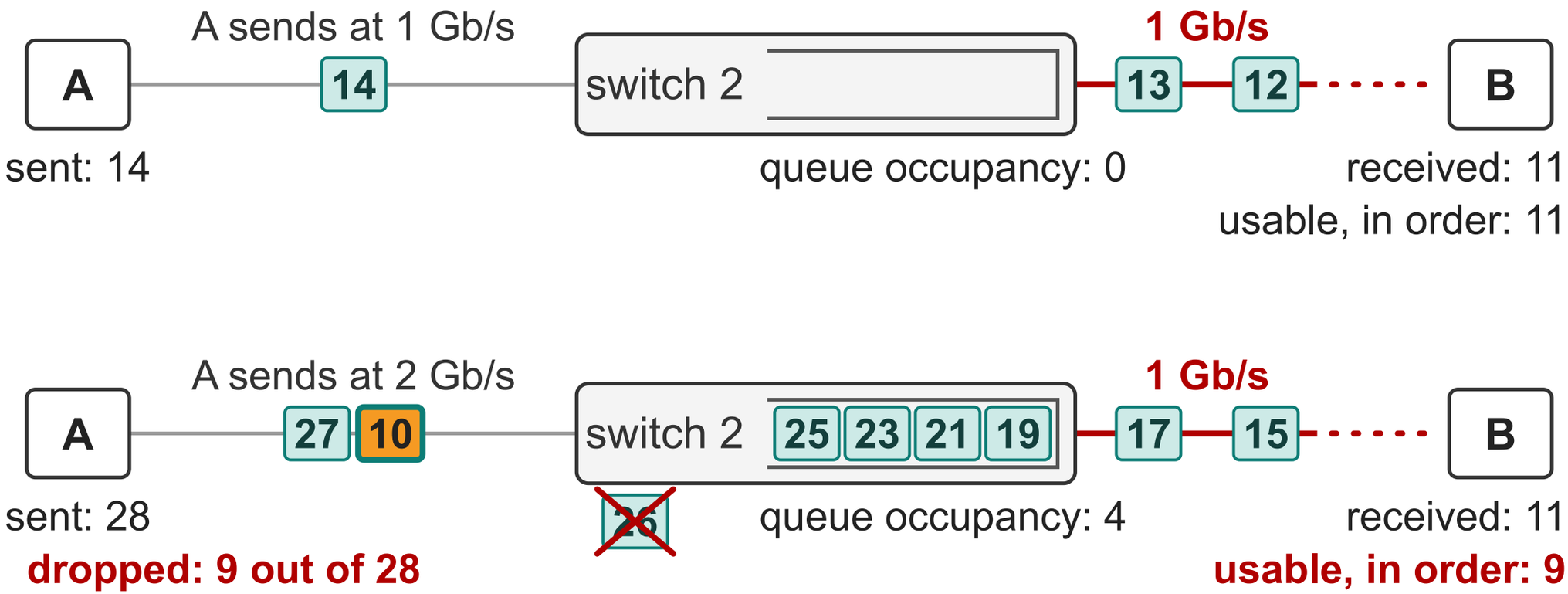
What if A sends faster than the slowest link?

When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



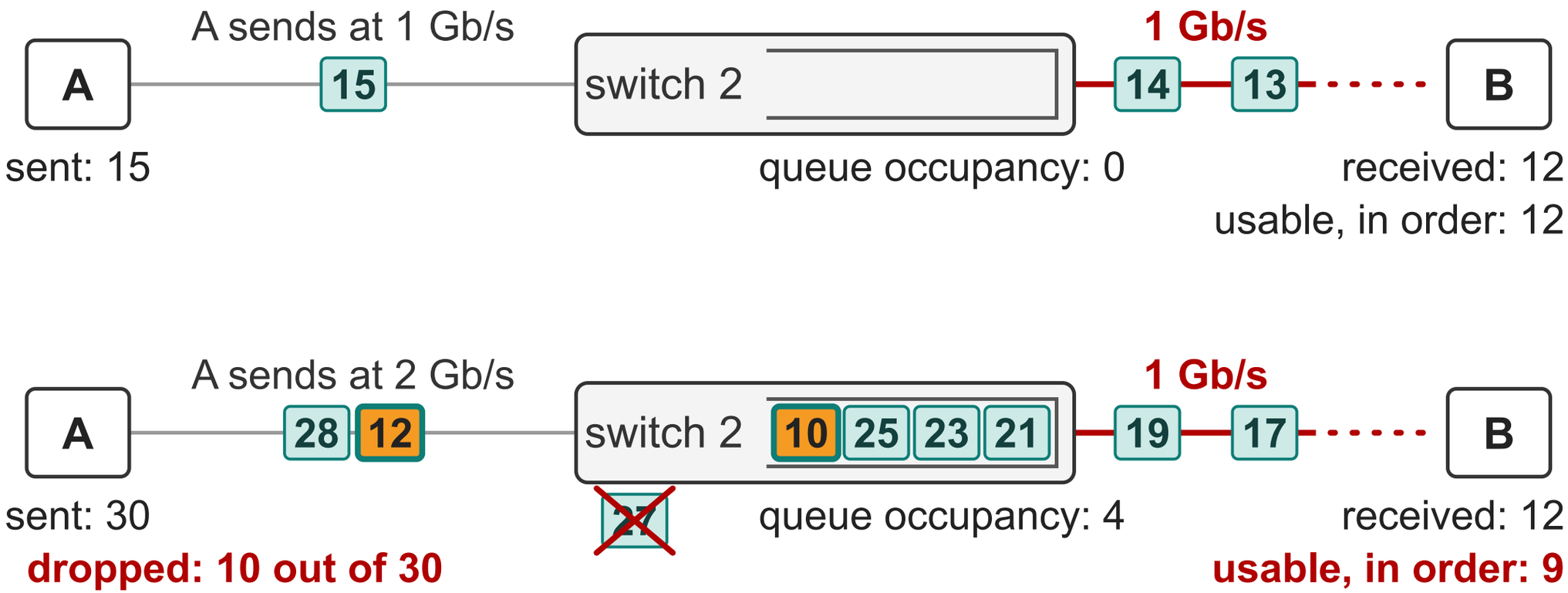
What if A sends faster than the slowest link?

When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



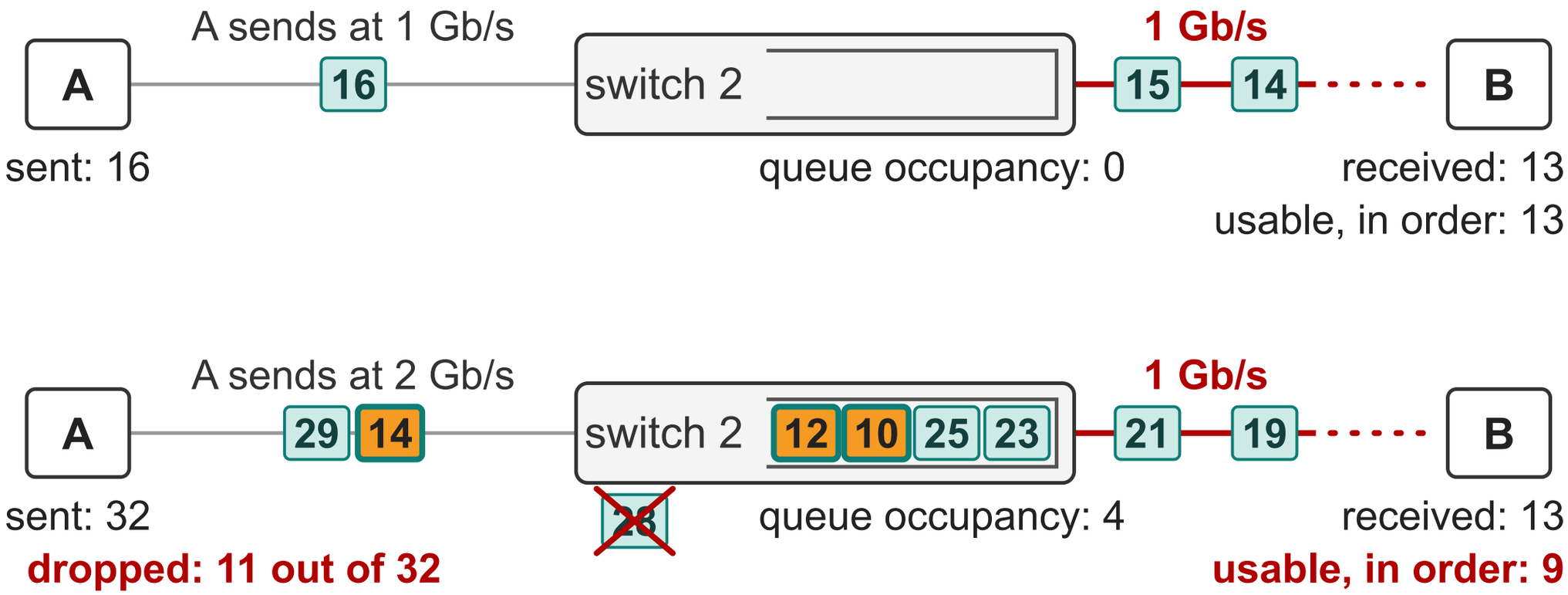
What if A sends faster than the slowest link?

When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



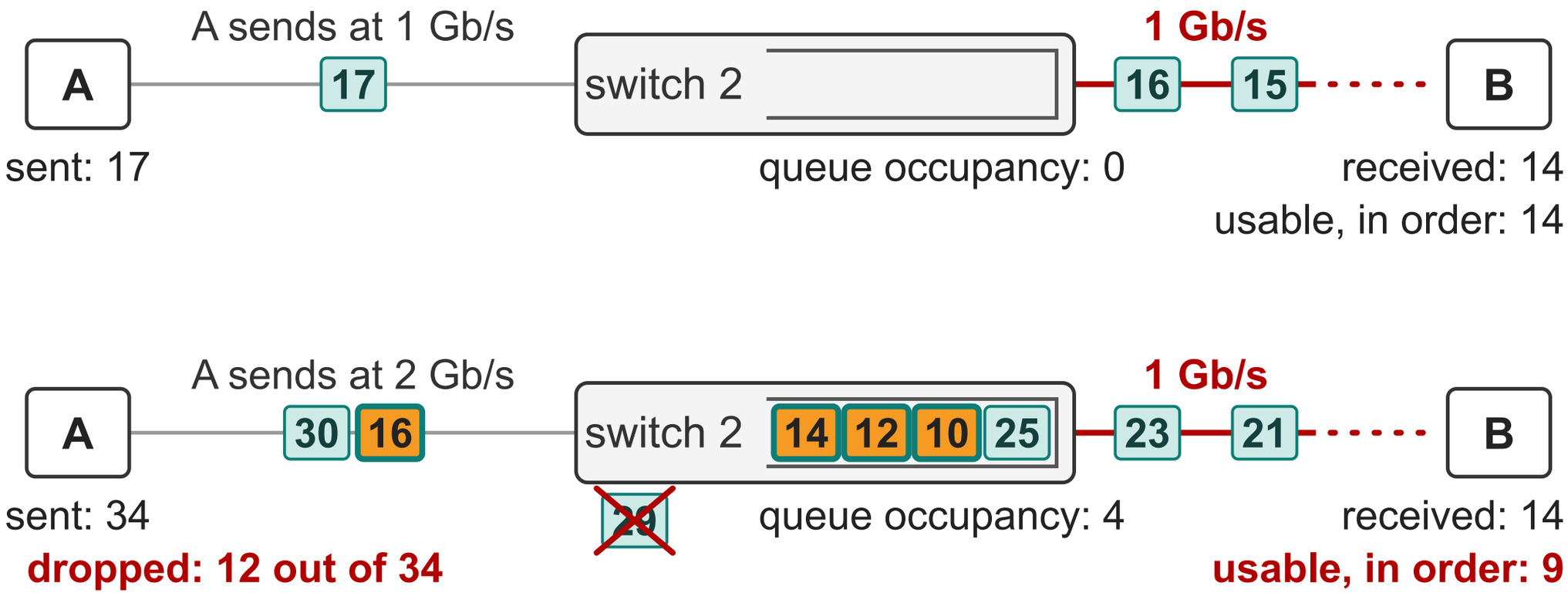
What if A sends faster than the slowest link?

When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



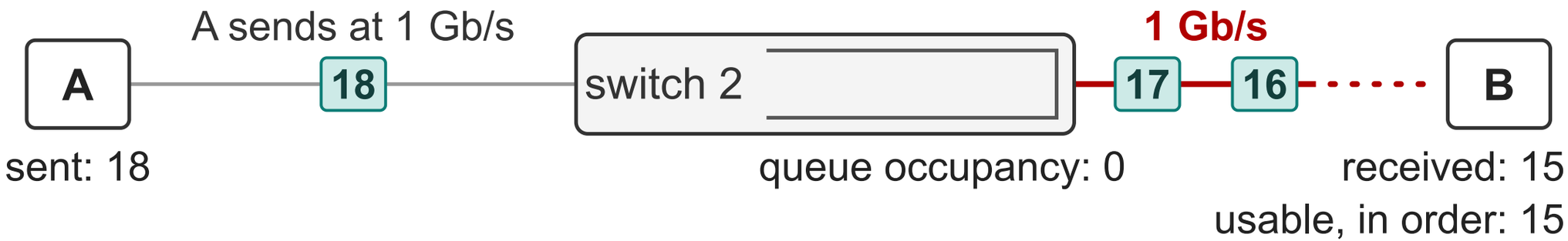
What if A sends faster than the slowest link?

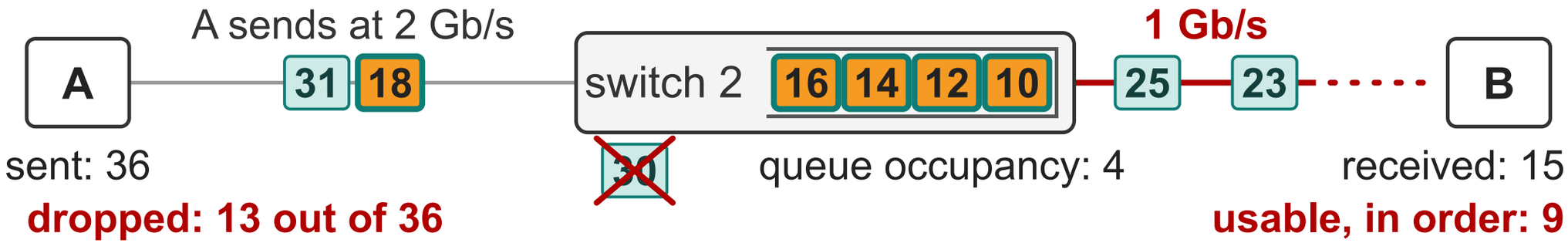
When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



What if A sends faster than the slowest link?

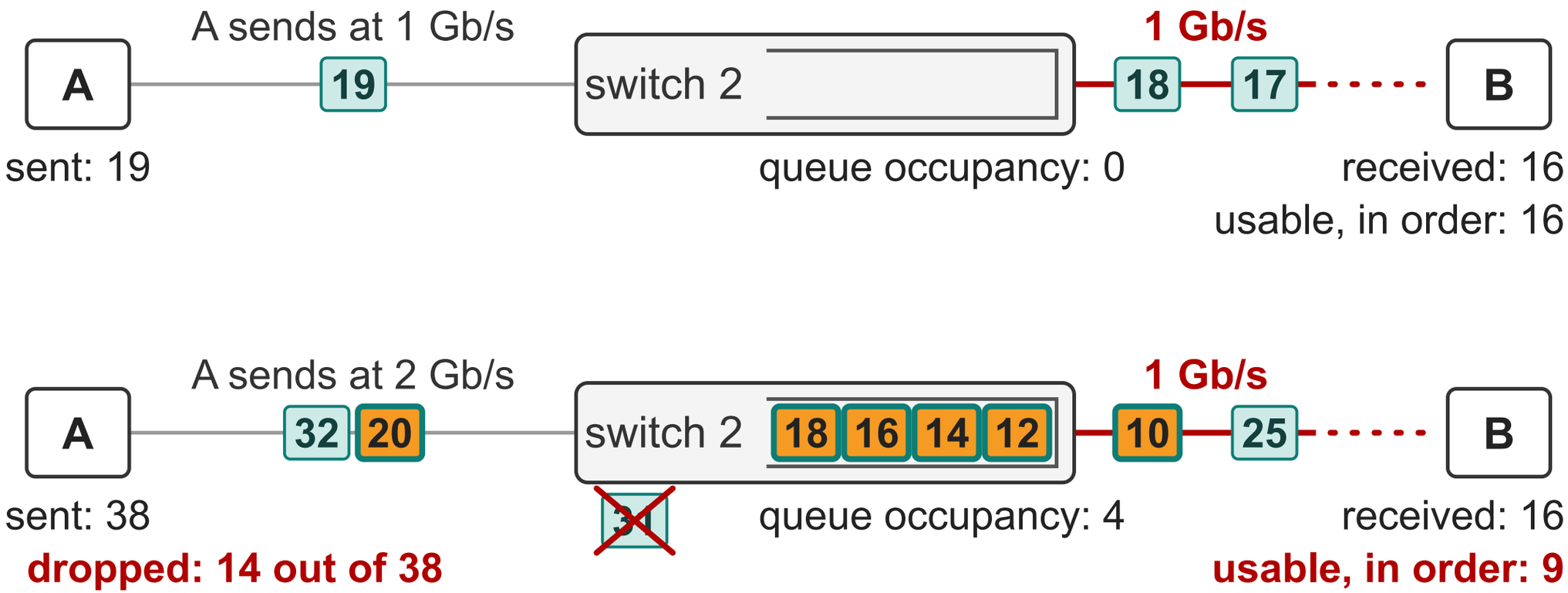
When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).





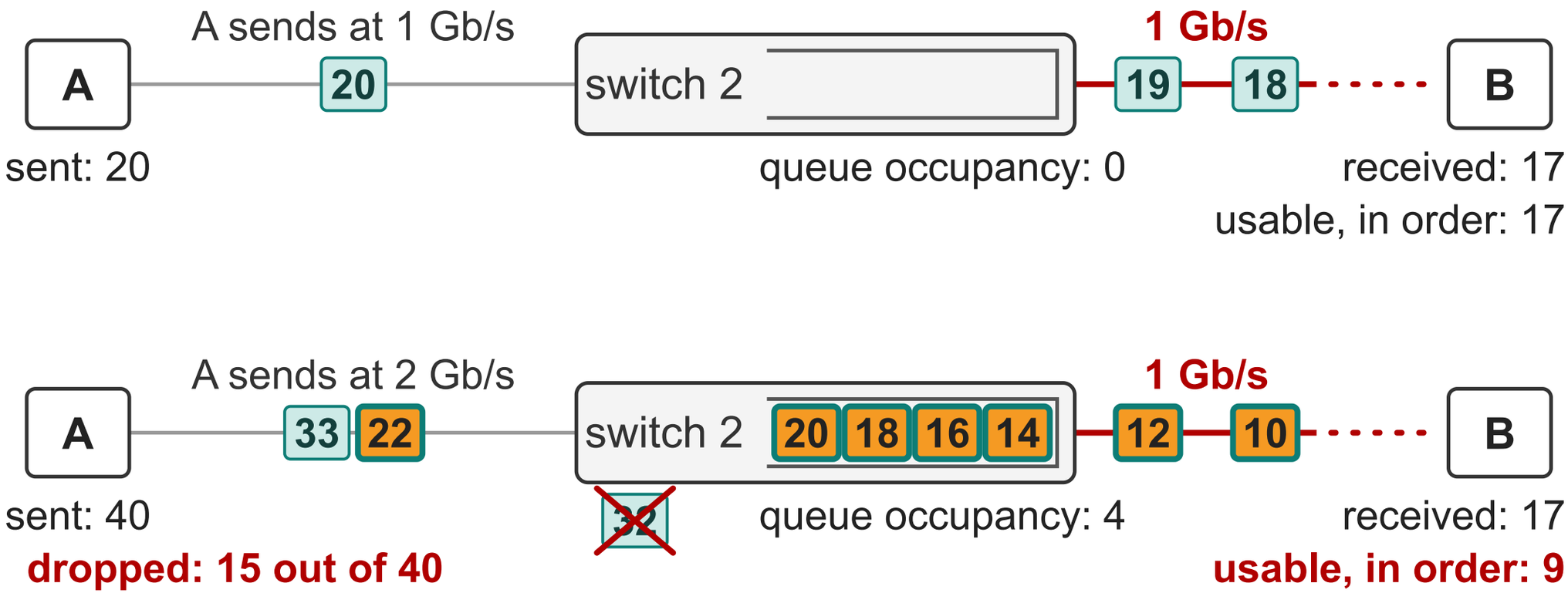
What if A sends faster than the slowest link?

When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



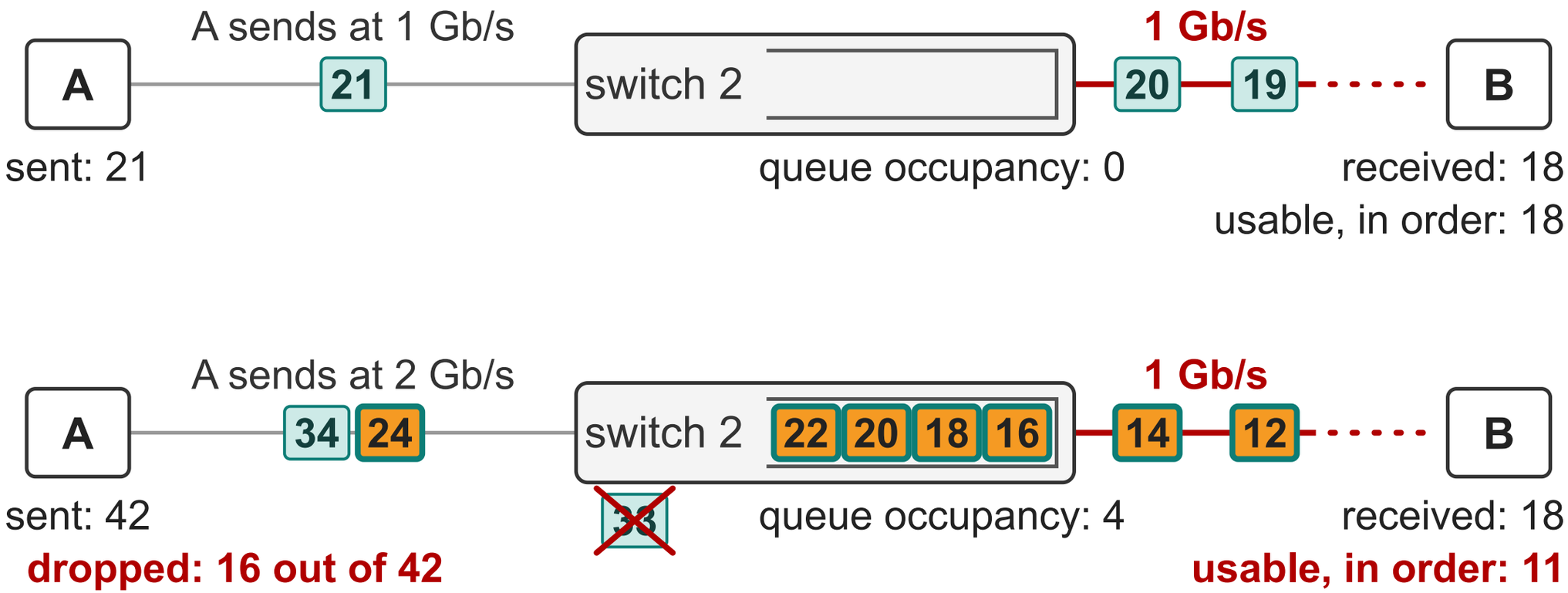
What if A sends faster than the slowest link?

When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).



What if A sends faster than the slowest link?

When A finds out a packet was lost (over an RTT later, using info in the acks), A sends it again (orange).

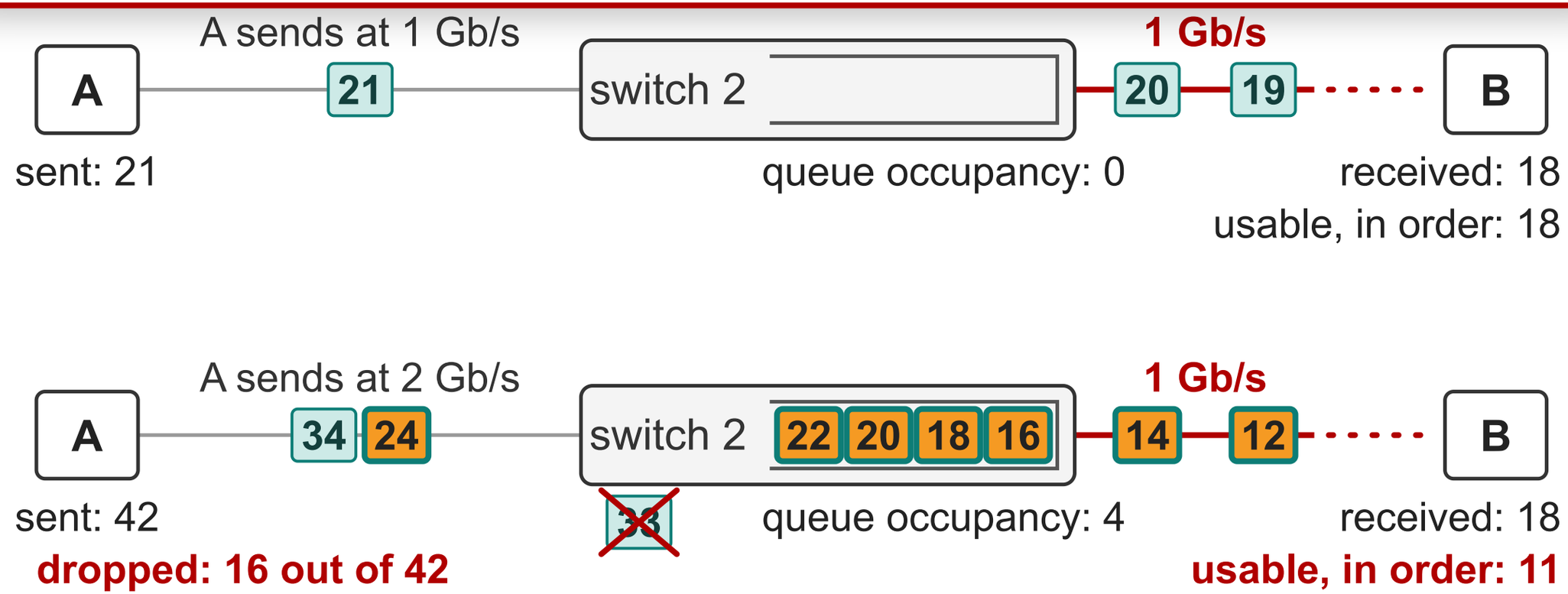


What if A sends faster than the slowest link?

When
(orar

Sending faster does not get more data through. If anything, it adds losses and wasted sends.

gain

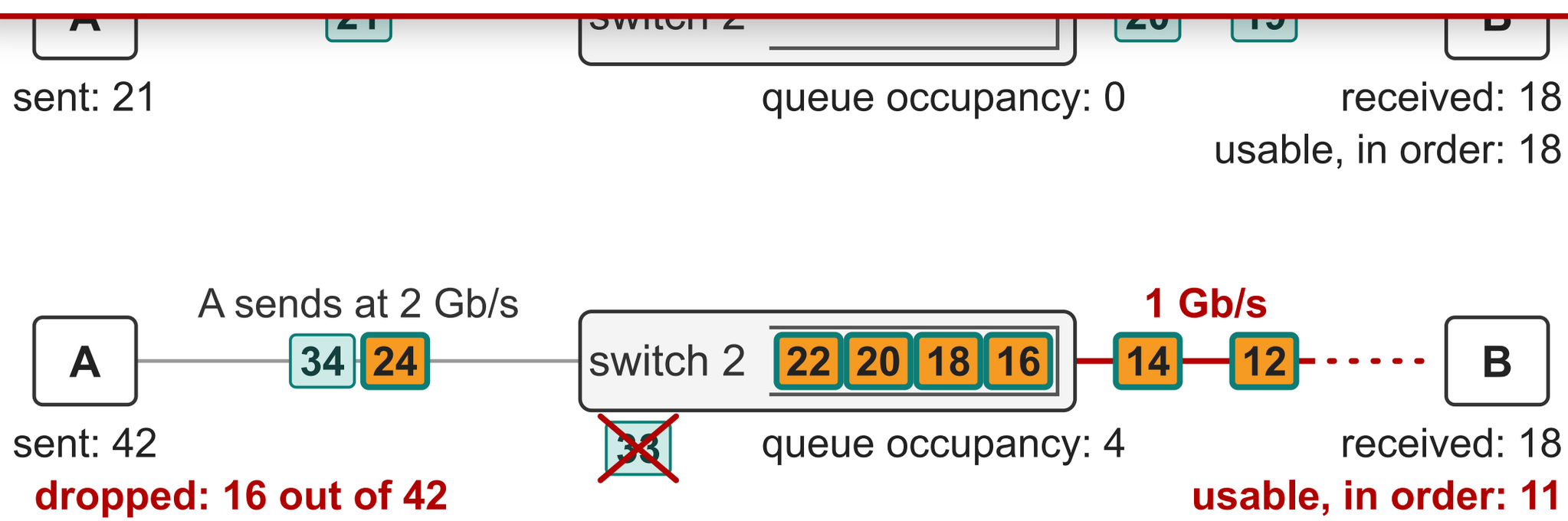


What if A sends faster than the slowest link?

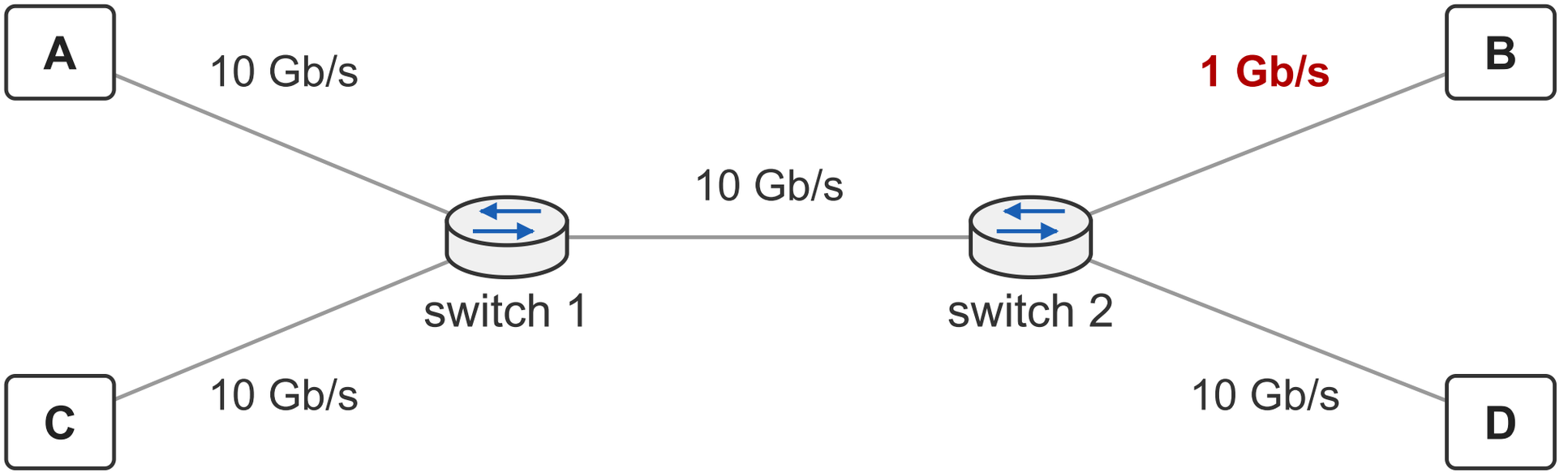
When
(orar
gain

Sending faster does not get more data through. If anything, it adds losses and wasted sends.

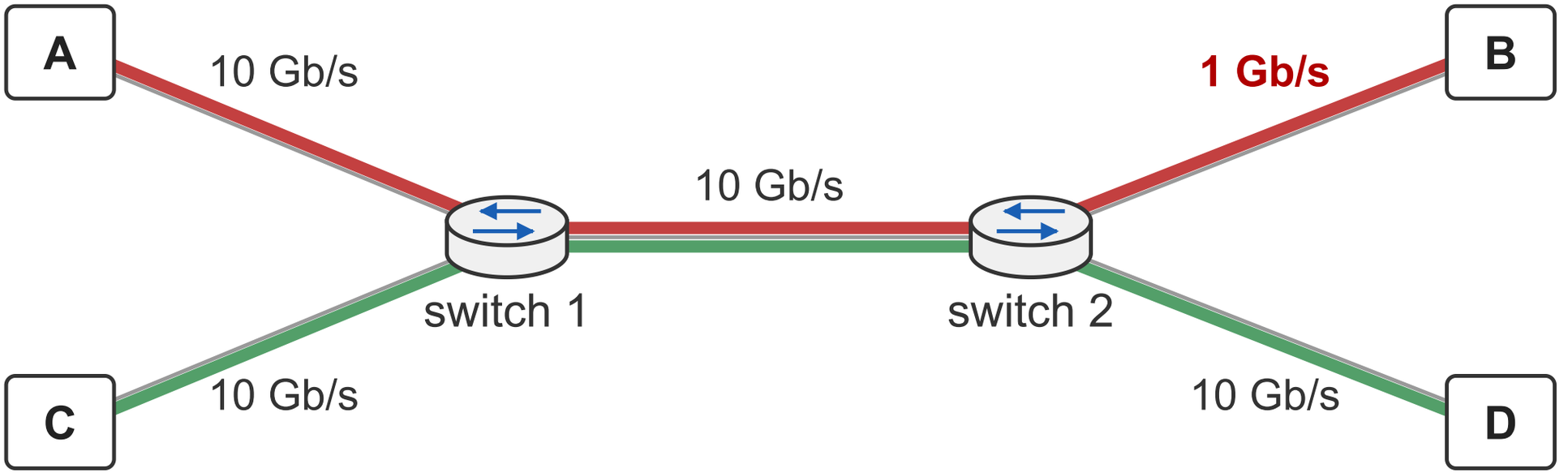
It can have severe consequences on a shared network.



Losses downstream waste upstream work

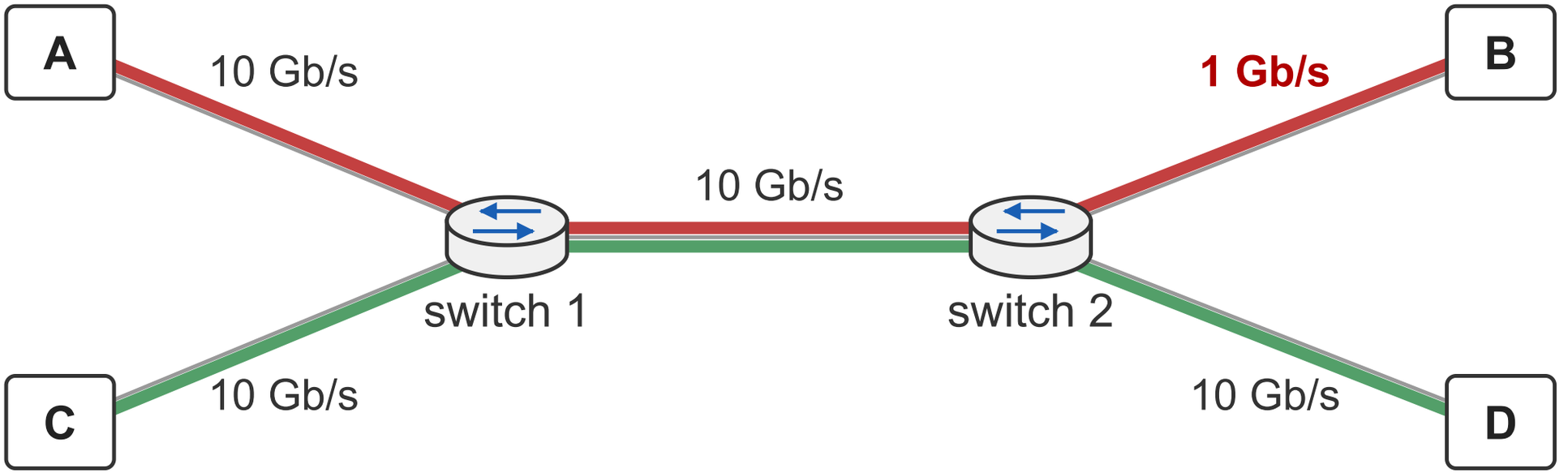


Losses downstream waste upstream work



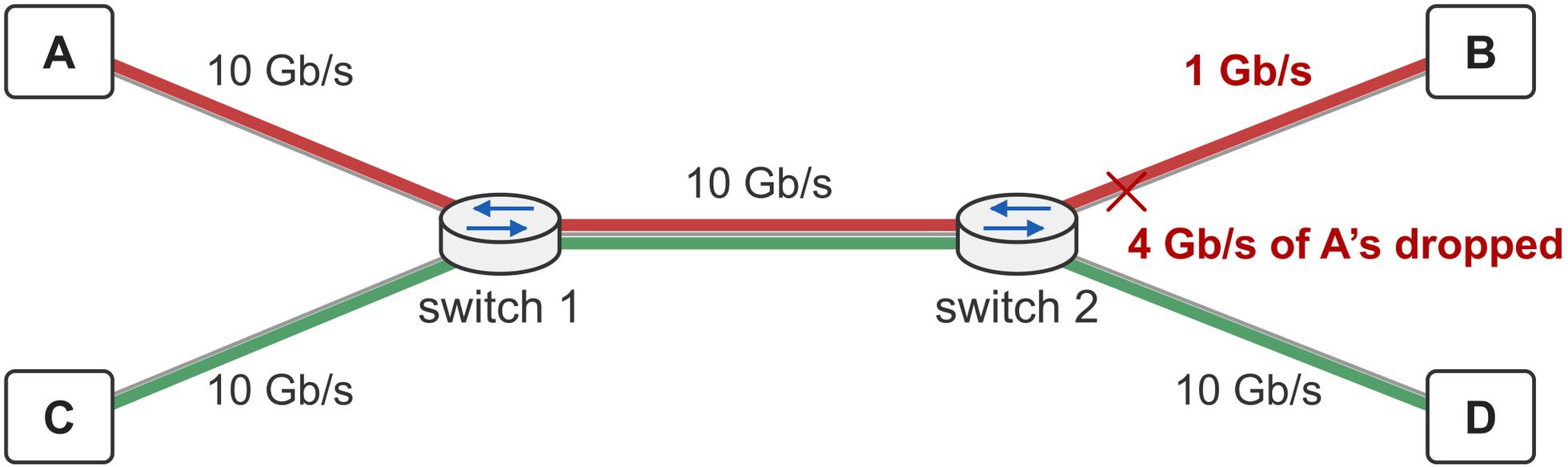
A and C each send as fast as their own link allows: 10 Gb/s

Losses downstream waste upstream work



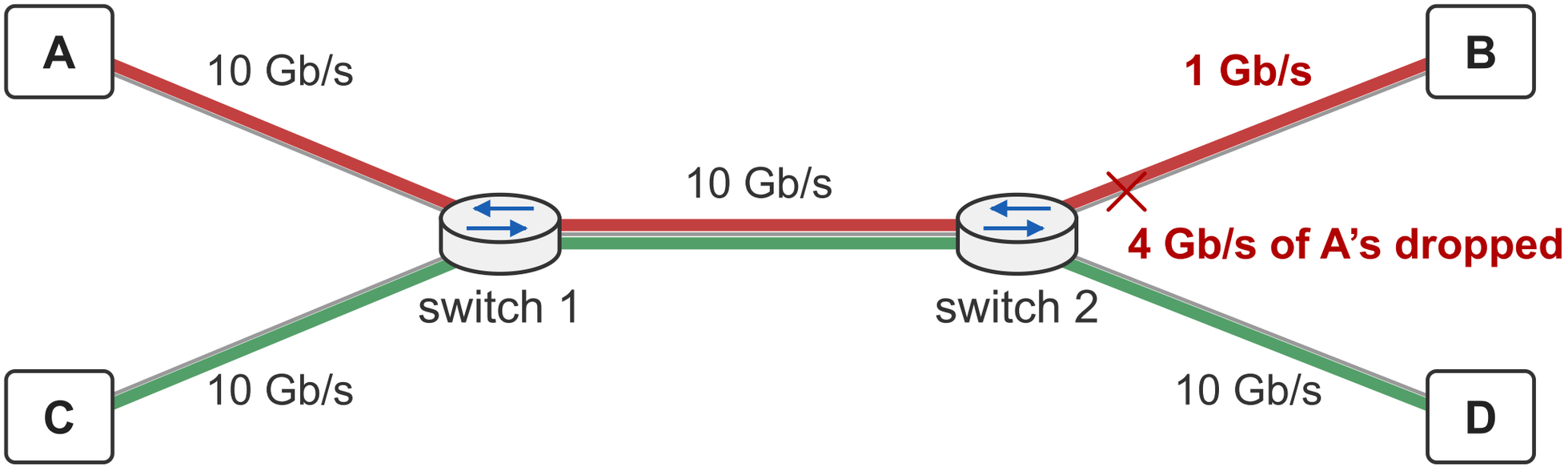
A and C each send as fast as their own link allows: 10 Gb/s
Switch 1 to switch 2 is 10 Gb/s. About 5 Gb/s of each flow gets through

Losses downstream waste upstream work



A and C each send as fast as their own link allows: 10 Gb/s
Switch 1 to switch 2 is 10 Gb/s. About 5 Gb/s of each flow gets through
Switch 2 to B is 1 Gb/s. Out of A's 5 Gb/s, 4 Gb/s is dropped

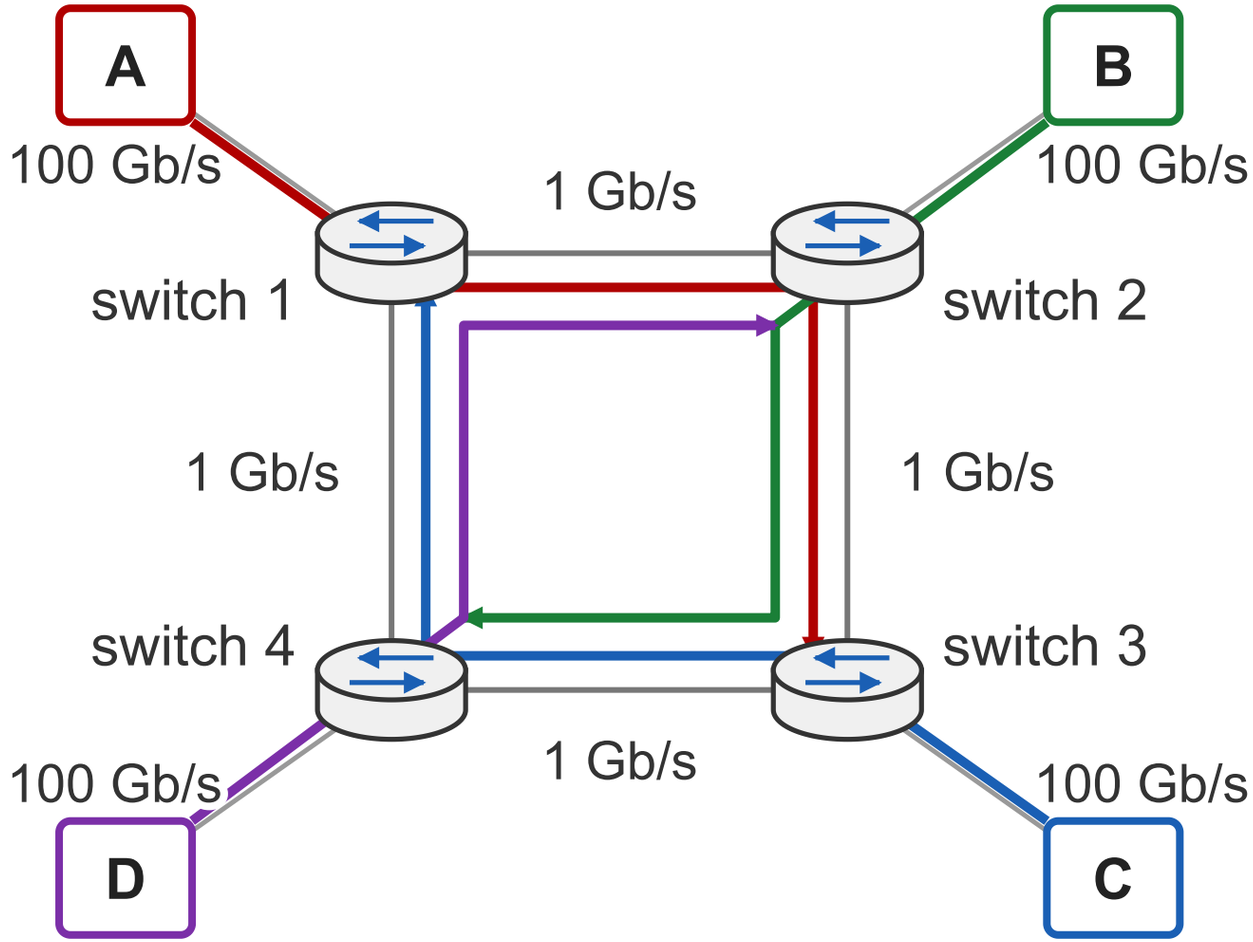
Losses downstream waste upstream work



A and C each send as fast as their own link allows: 10 Gb/s
Switch 1 to switch 2 is 10 Gb/s. About 5 Gb/s of each flow gets through
Switch 2 to B is 1 Gb/s. Out of A's 5 Gb/s, 4 Gb/s is dropped
B gets 1 Gb/s and D gets 5 Gb/s: 6 Gb/s total
Had A sent 1 Gb/s from the beginning, the total could have been 1 + 9 = 10 Gb/s

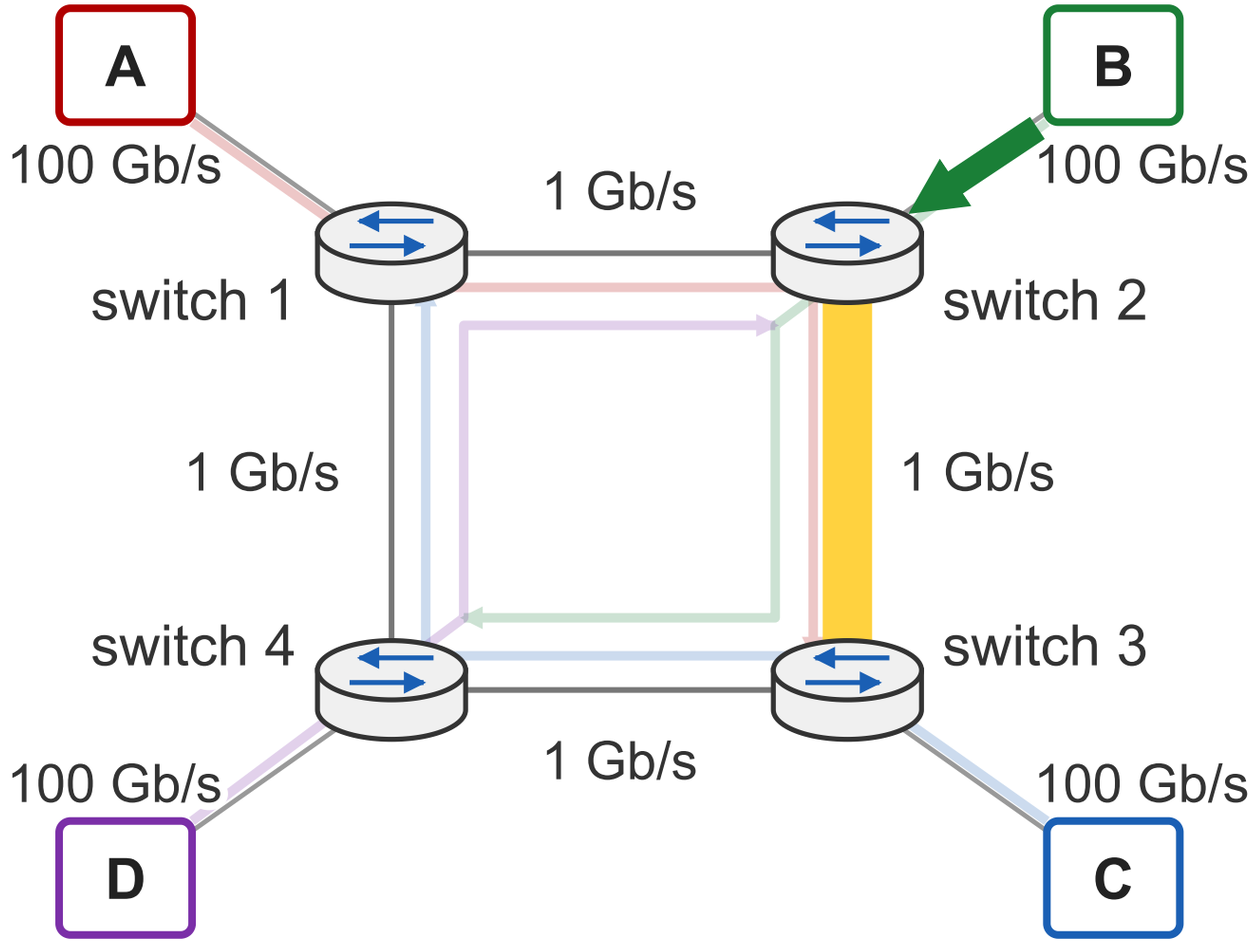
Congestion collapse

Each host sends to the opposite corner (A → C, B → D, C → A, D → B), as fast as it can.



Congestion collapse

Each host sends to the opposite corner (A → C, B → D, C → A, D → B), as fast as it can.

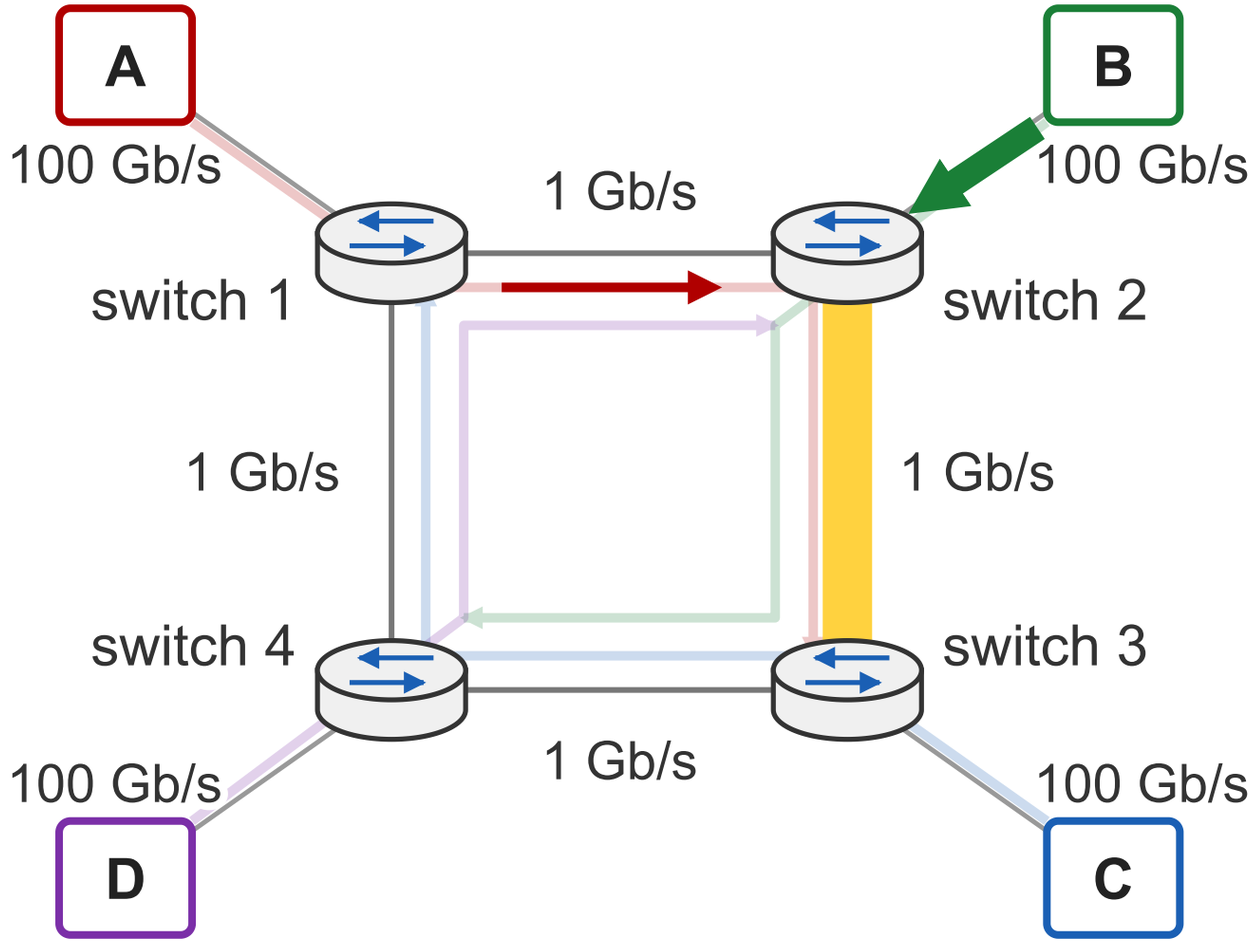


Consider switch 2's queue for the link to switch 3:

- B is sending 100 Gb/s into the queue

Congestion collapse

Each host sends to the opposite corner (A → C, B → D, C → A, D → B), as fast as it can.

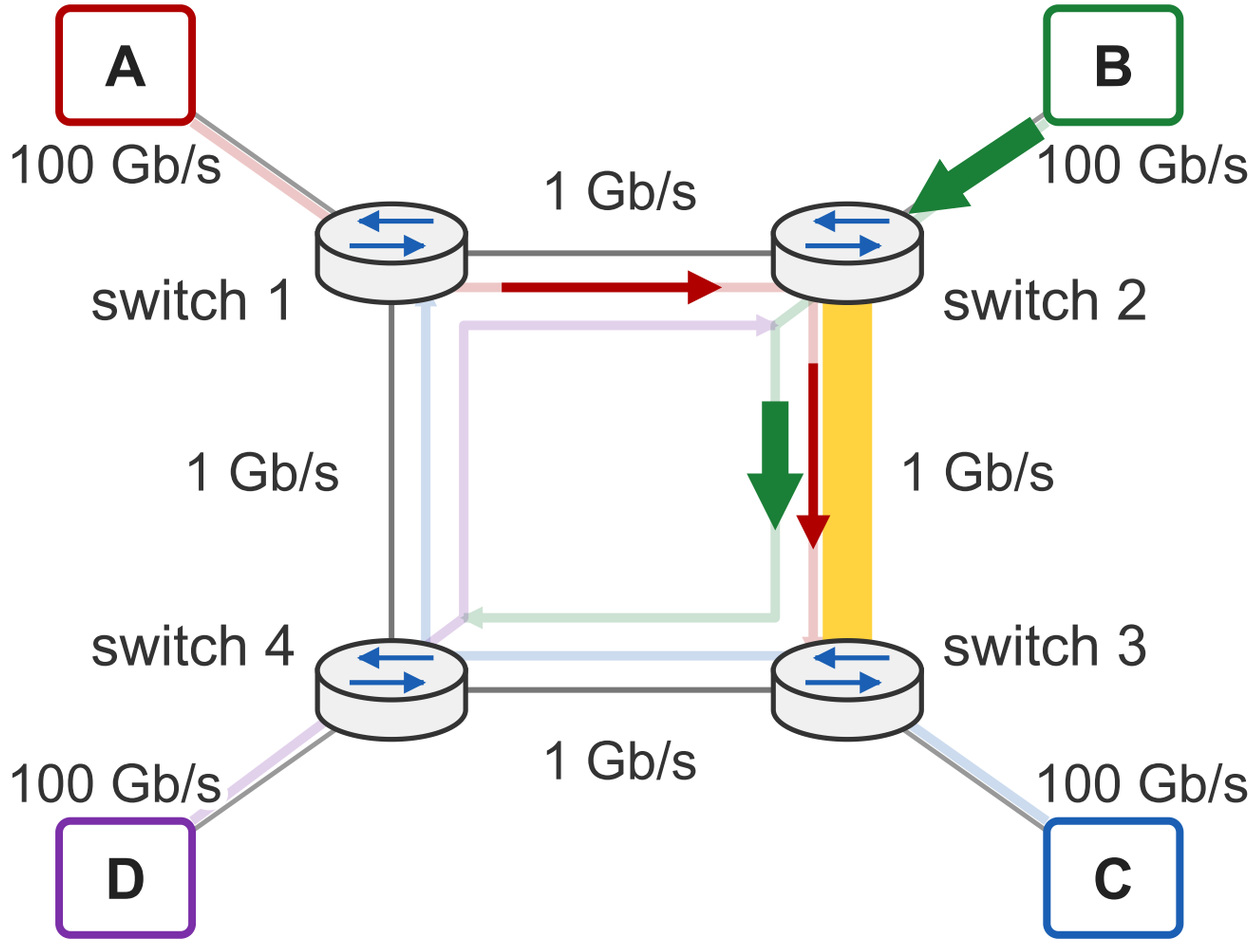


Consider switch 2's queue for the link to switch 3:

- B is sending 100 Gb/s into the queue
- A is sending $X \leq 1$ Gb/s into the queue

Congestion collapse

Each host sends to the opposite corner (A → C, B → D, C → A, D → B), as fast as it can.

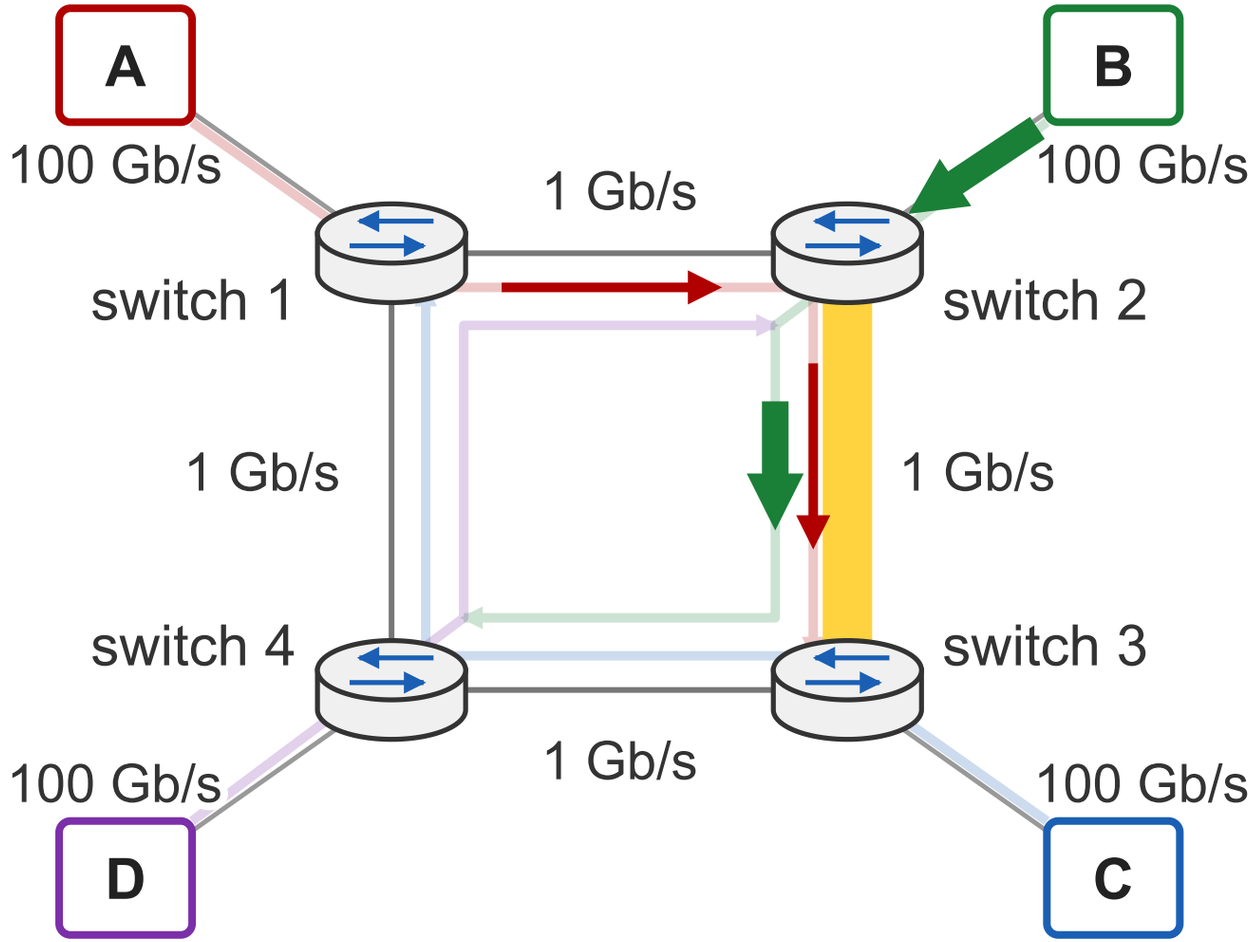


Consider switch 2's queue for the link to switch 3:

- B is sending 100 Gb/s into the queue
- A is sending $X \leq 1$ Gb/s into the queue
- A's share = $X / (X + 100)$
- B's share = $100 / (X + 100)$

Congestion collapse

Each host sends to the opposite corner (A → C, B → D, C → A, D → B), as fast as it can.

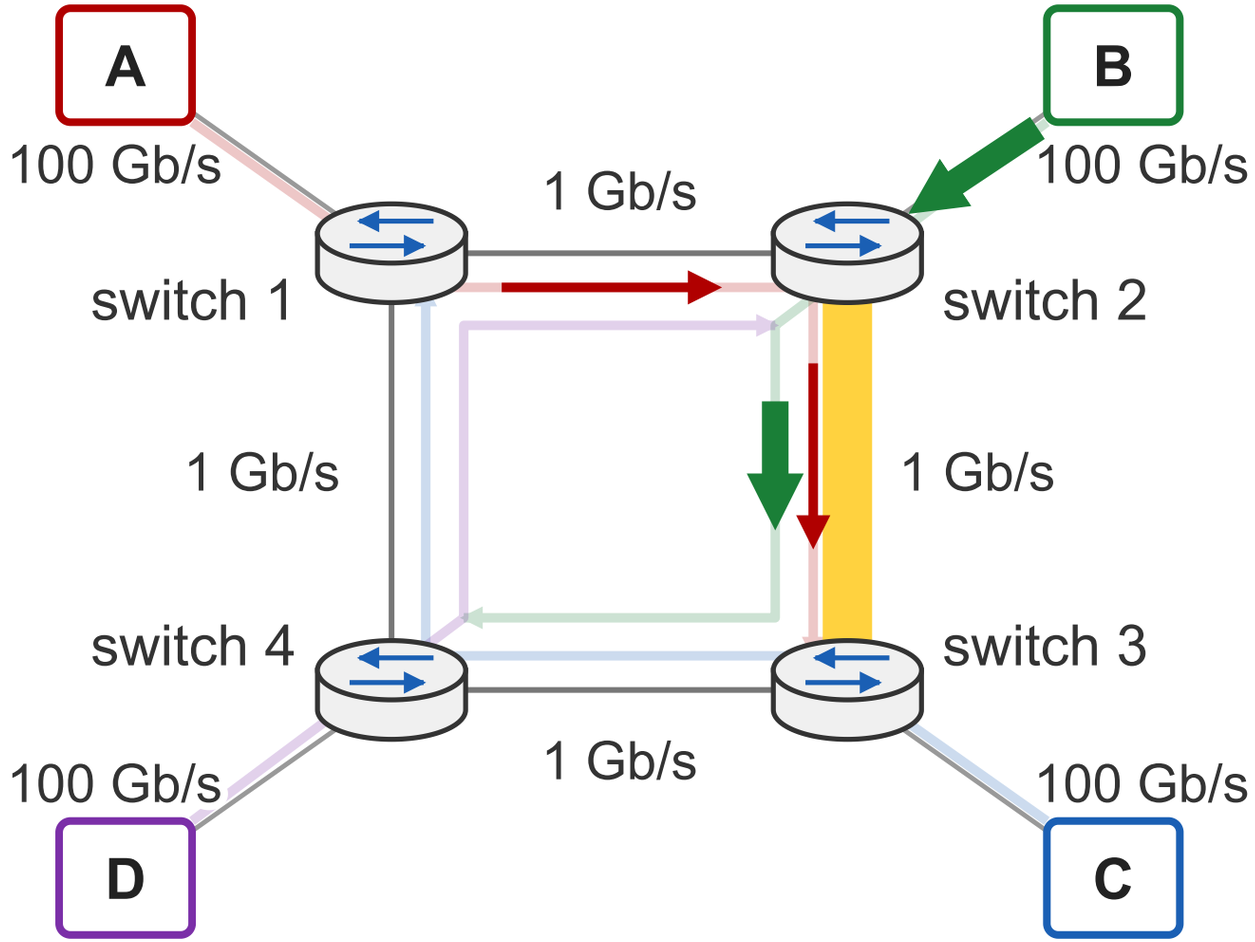


Consider switch 2's queue for the link to switch 3:

- B is sending 100 Gb/s into the queue
- A is sending $X \leq 1$ Gb/s into the queue
- A's share = $X / (X + 100)$
- B's share = $100 / (X + 100)$
- B is getting most of the 1 Gb/s

Congestion collapse

Each host sends to the opposite corner (A → C, B → D, C → A, D → B), as fast as it can.



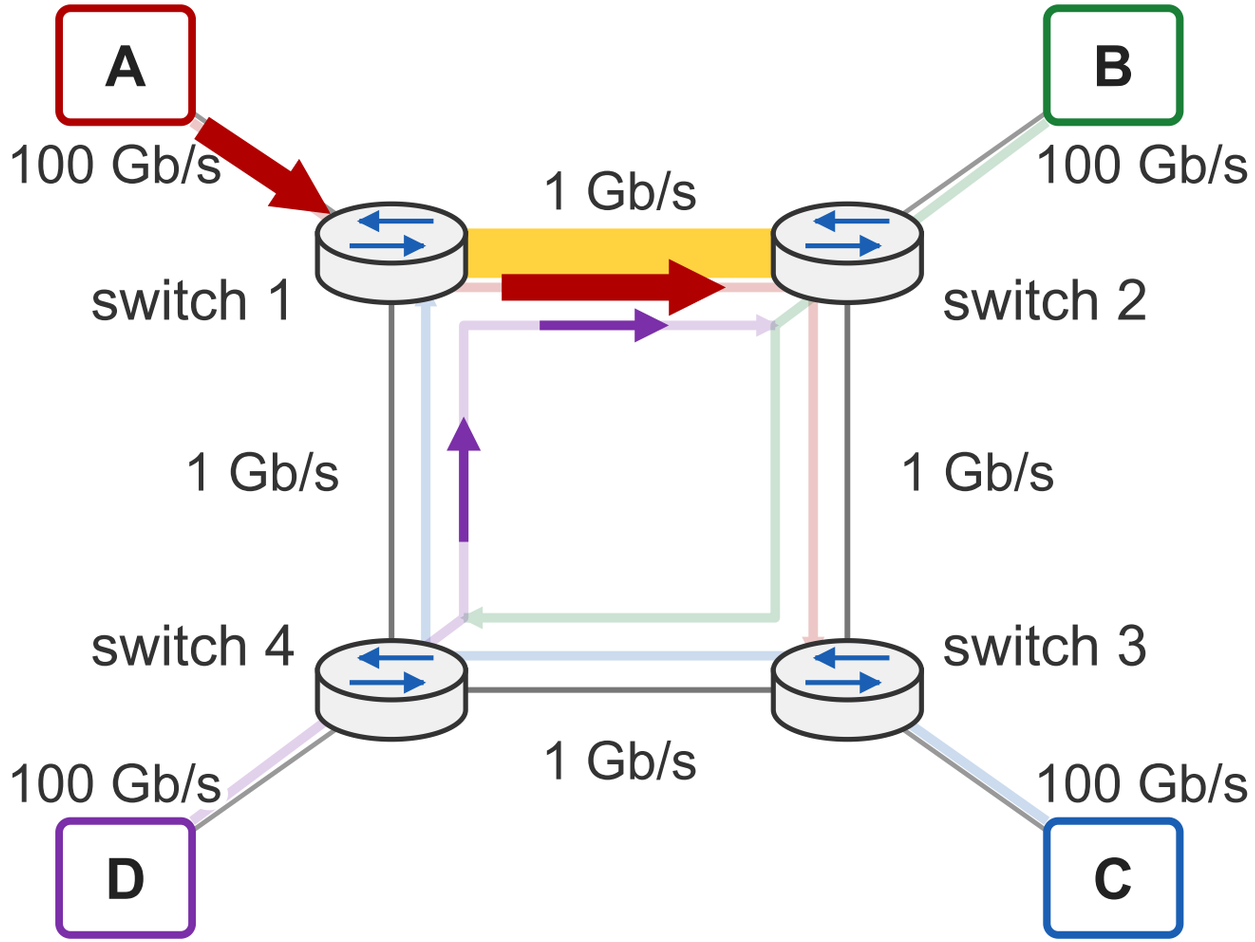
Consider switch 2's queue for the link to switch 3:

- B is sending 100 Gb/s into the queue
- A is sending X ≤ 1 Gb/s into the queue
- A's share = X / (X + 100)
- B's share = 100 / (X + 100)
- B is getting most of the 1 Gb/s

What is X?

Congestion collapse

Each host sends to the opposite corner (A → C, B → D, C → A, D → B), as fast as it can.

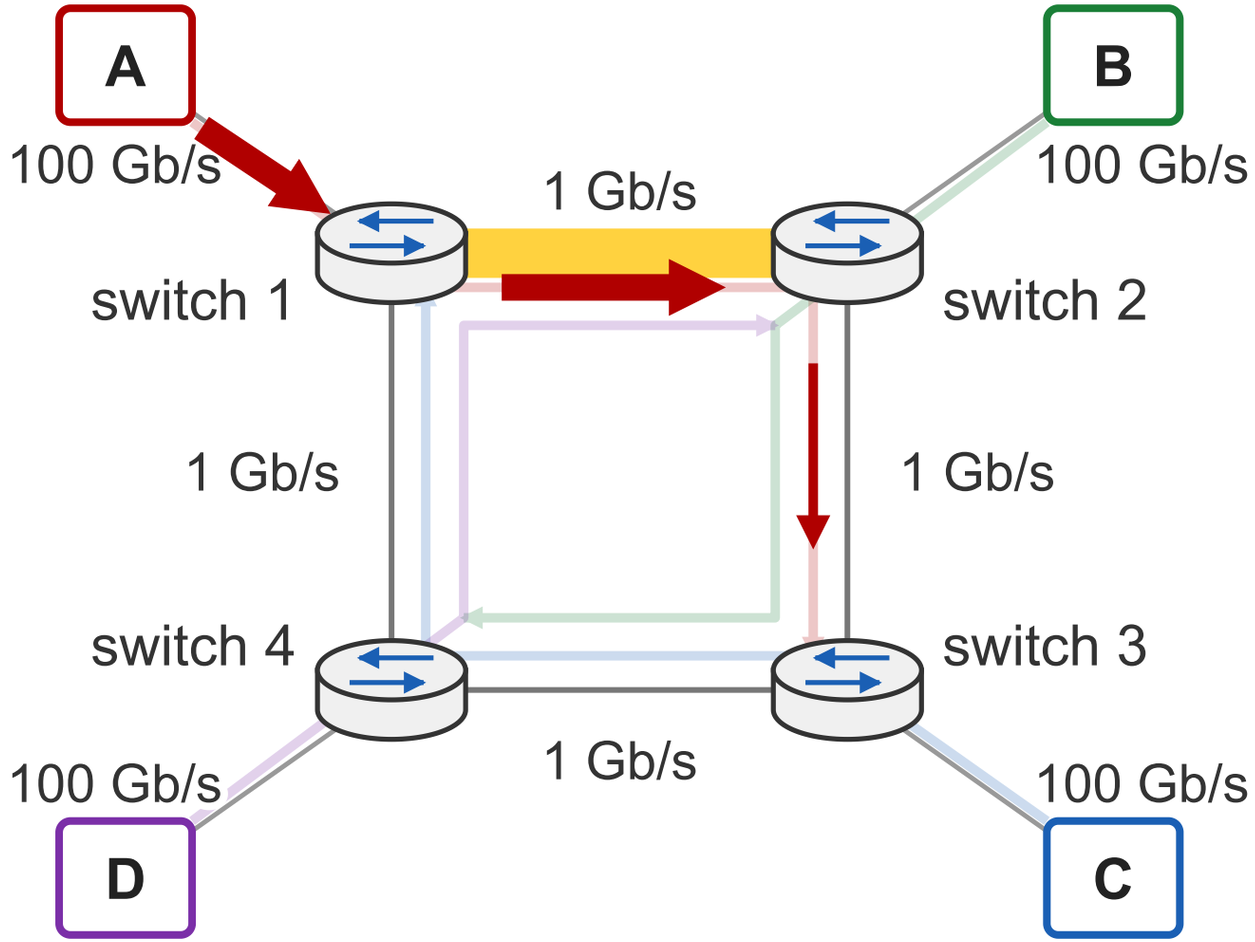


At switch 1, A is in B's place:

- A sends 100 Gb/s
- D's packets at most 1 Gb/s
- A's traffic is getting most of the 1 Gb/s
- So, X is almost 1 Gb/s

Congestion collapse

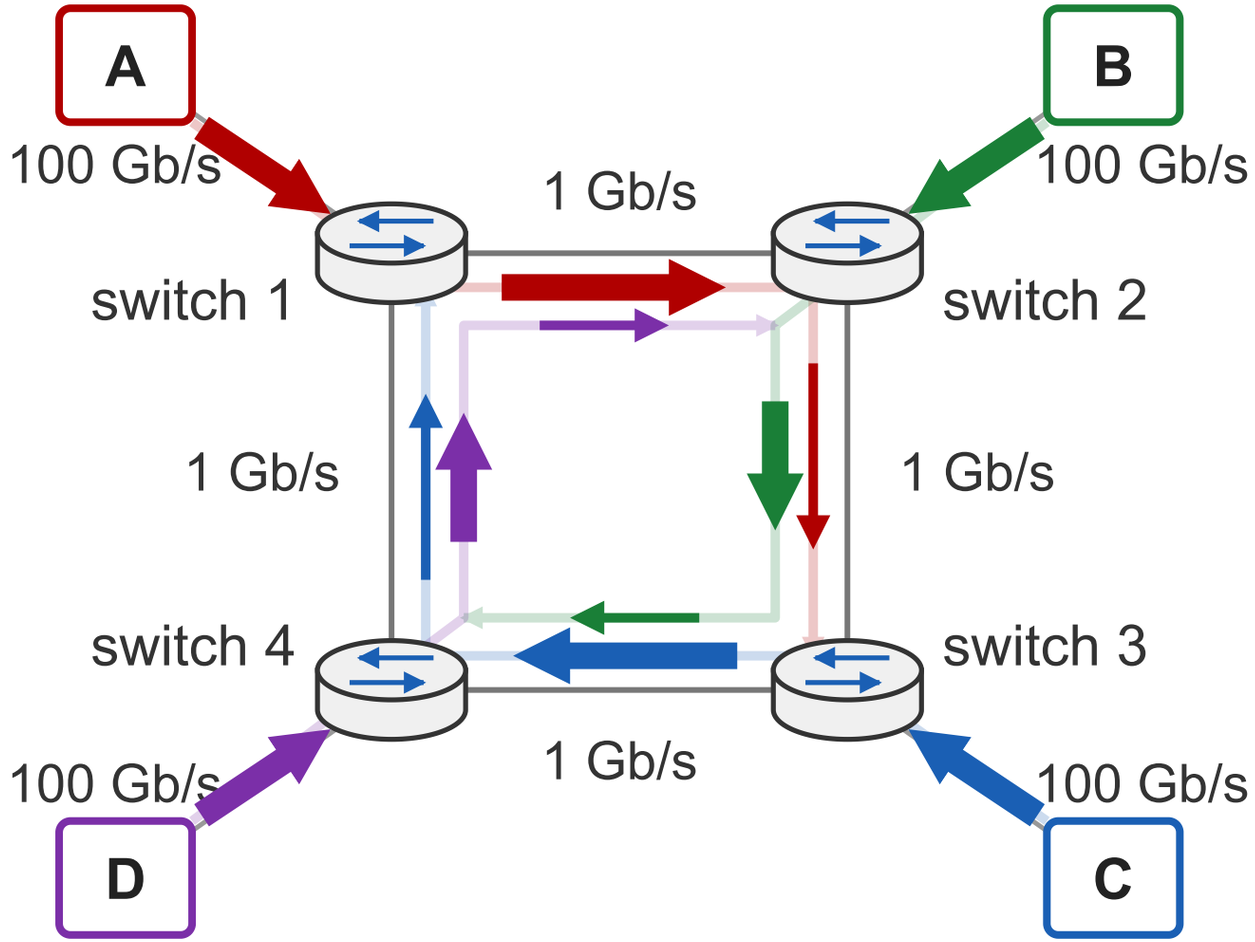
Each host sends to the opposite corner (A → C, B → D, C → A, D → B), as fast as it can.



- So, A is taking most of the link from switch 1 to switch 2
- It lets almost nothing of D's packets through
- But then, almost all of its own packets are dropped at switch 2
  - Because B doesn't let A's packets through, similar to how A didn't let D's packets through

Congestion collapse

Each host sends to the opposite corner (A → C, B → D, C → A, D → B), as fast as it can.

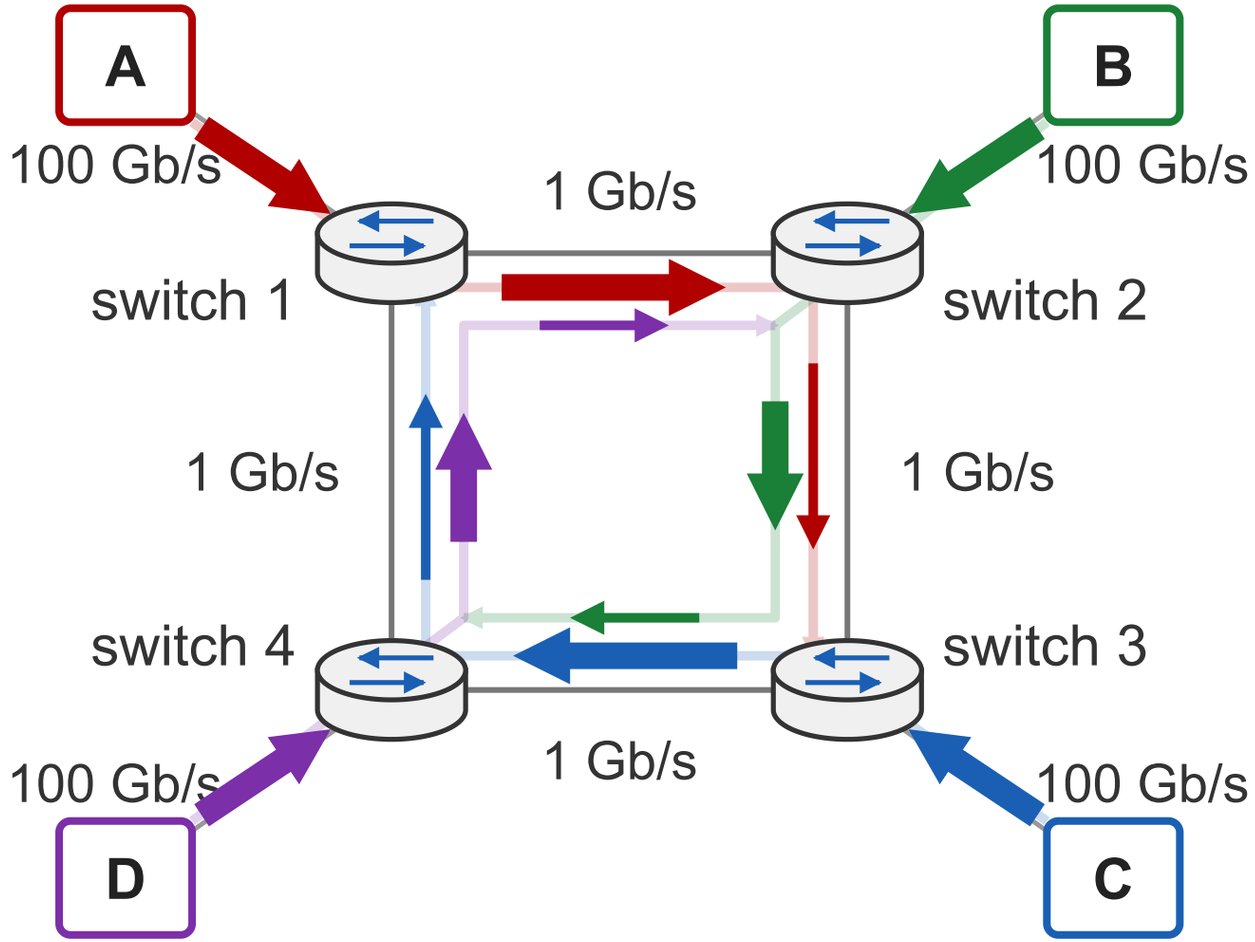


This is the same for every flow:

- Takes most of the bottleneck at its first hop
- Crowds out the other flow on that link
- Then, it is mostly dropped at the next hop

Congestion collapse

Each host sends to the opposite corner (A → C, B → D, C → A, D → B), as fast as it can.



This is the same for every flow:

- Takes most of the bottleneck at its first hop
- Crowds out the other flow on that link
- Then, it is mostly dropped at the next hop

The network links are always busy, but they are not actually delivering data to the end points

How could this have been better? (in-class)

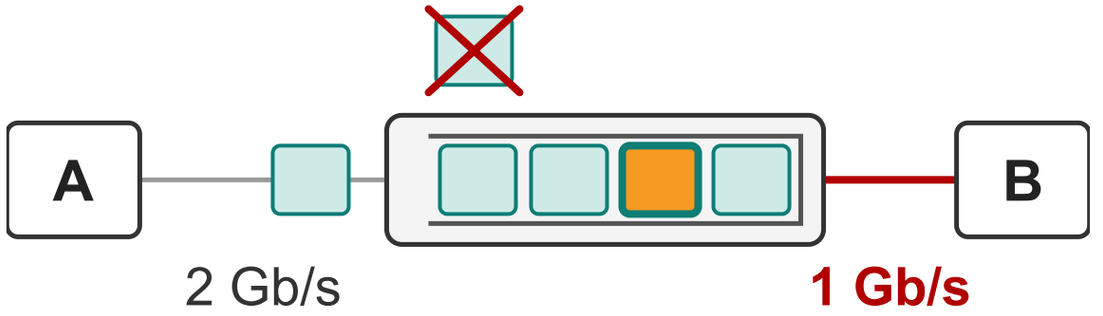
(in-class discussion)

- How much would you have each host send instead?

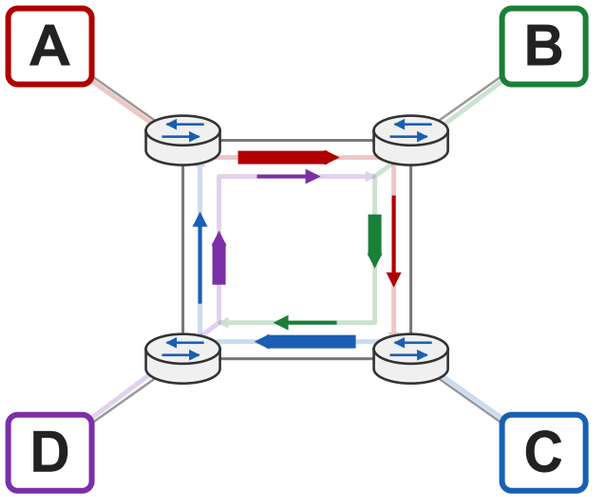
Reset

Recap so far: send only as much as your share of the bottleneck link allows

Only flow on the path: sending faster than the bottleneck link does not get more data through. If anything, it adds losses and wasted sends.

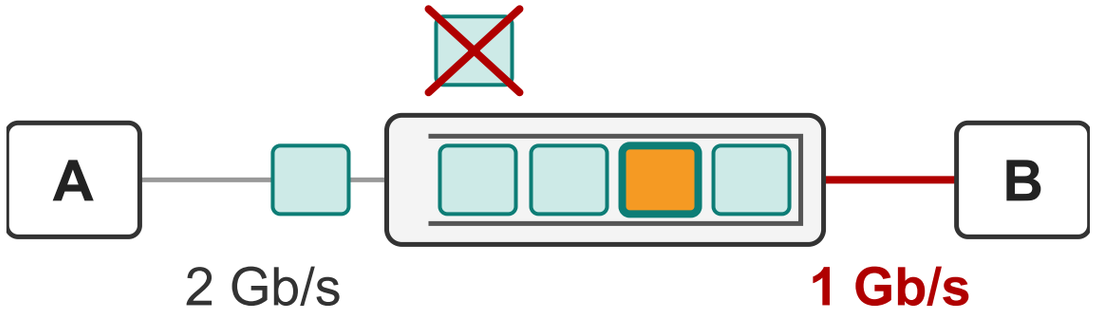


Multiple flows sharing bottleneck links: the network can be busy and still deliver almost nothing (congestion collapse) if flows do not adjust to their share.



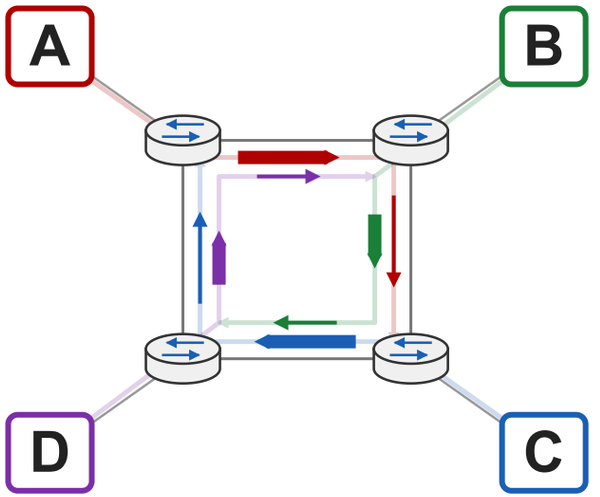
Recap so far: send only as much as your share of the bottleneck link allows

Only flow on the path: sending faster than the bottleneck link does not get more data through. If anything, it adds losses and wasted sends.

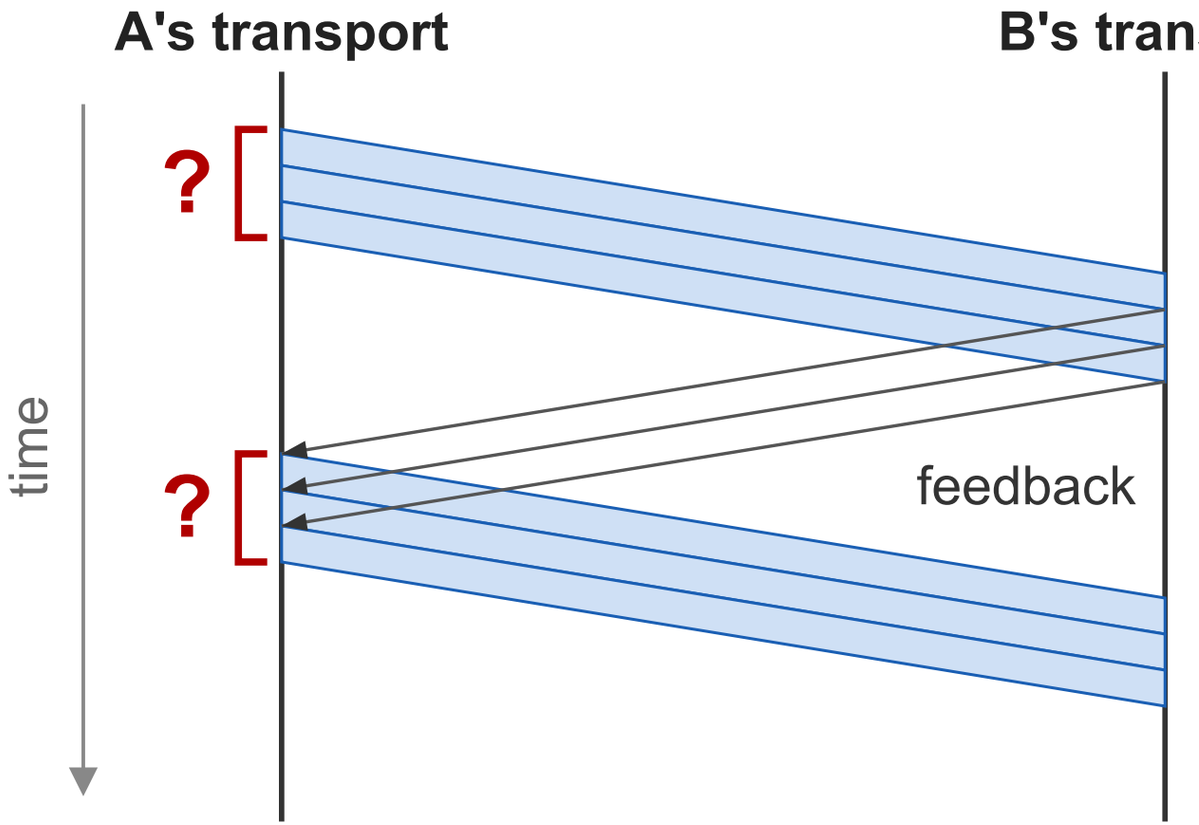


Multiple flows sharing bottleneck links: the network can be busy and still deliver almost nothing (congestion collapse) if flows do not adjust to their share.

One common way: bound the number of in-flight bytes

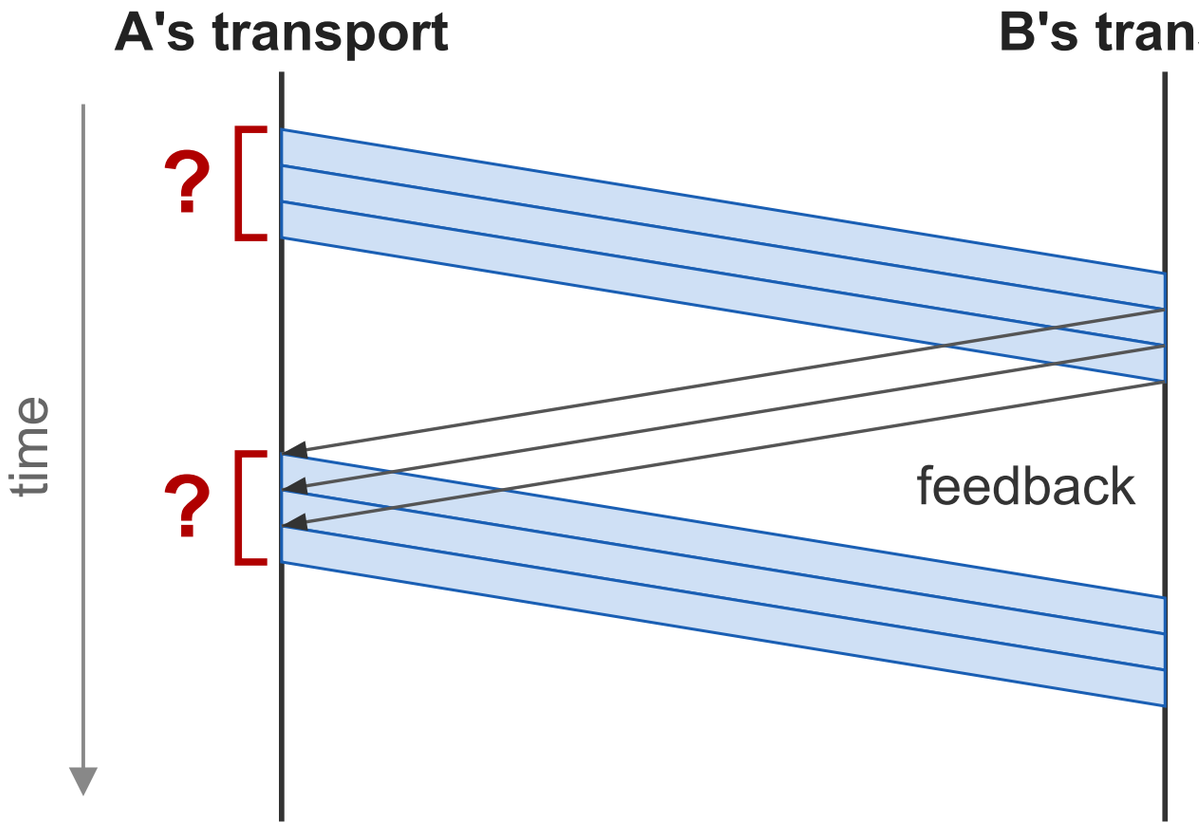


Back to the batch size



A can have a “batch size” of segments in flight.

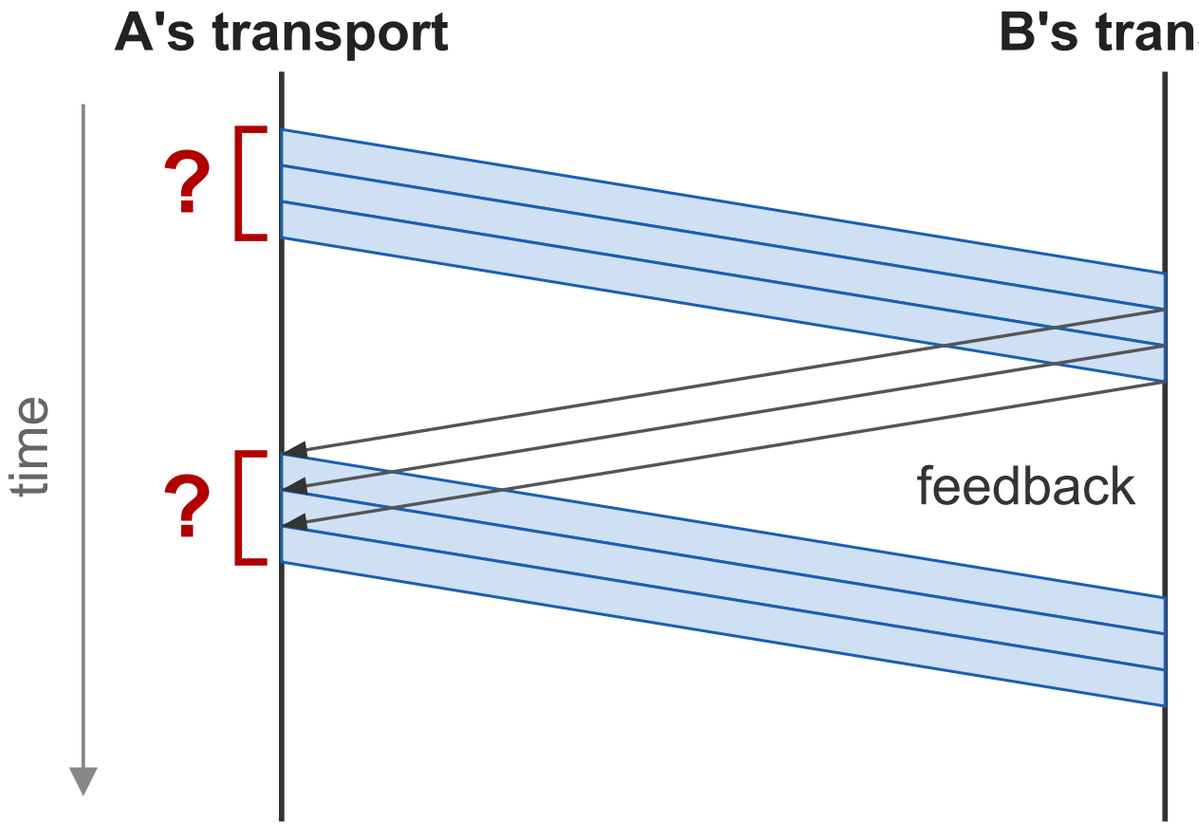
Back to the batch size



A can have a “batch size” of segments in flight.

Sending rate ≈ batch size / RTT

Back to the batch size

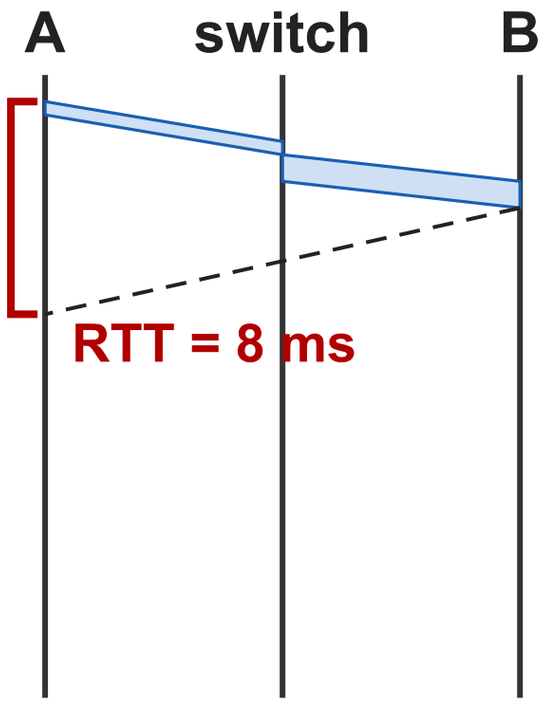


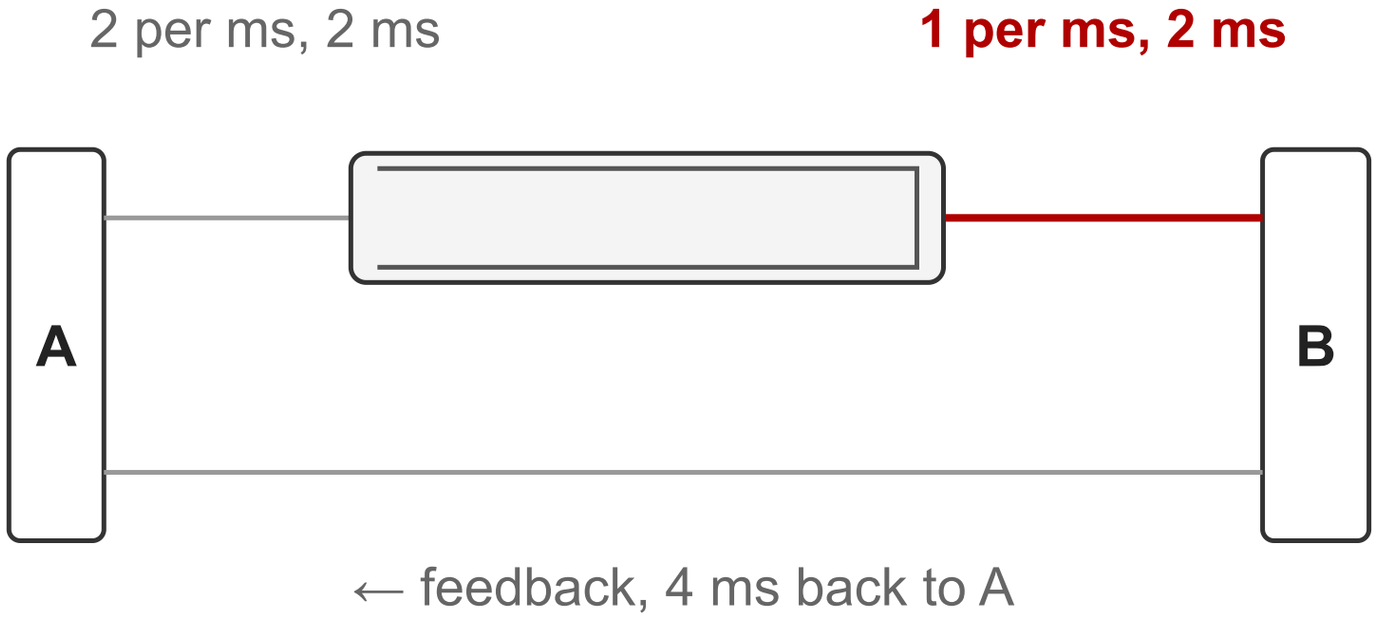
A can have a “batch size”
of segments in flight.

Sending rate ≈
batch size / RTT

So what is the right batch size?

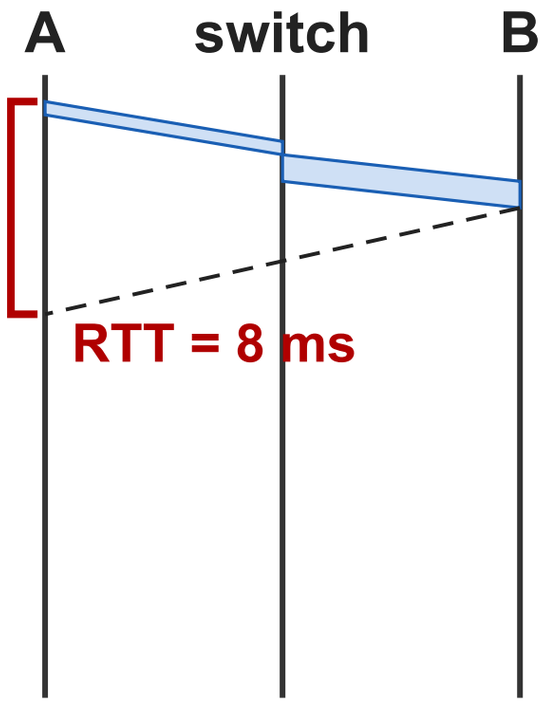
Bounding the number of in-flight bytes

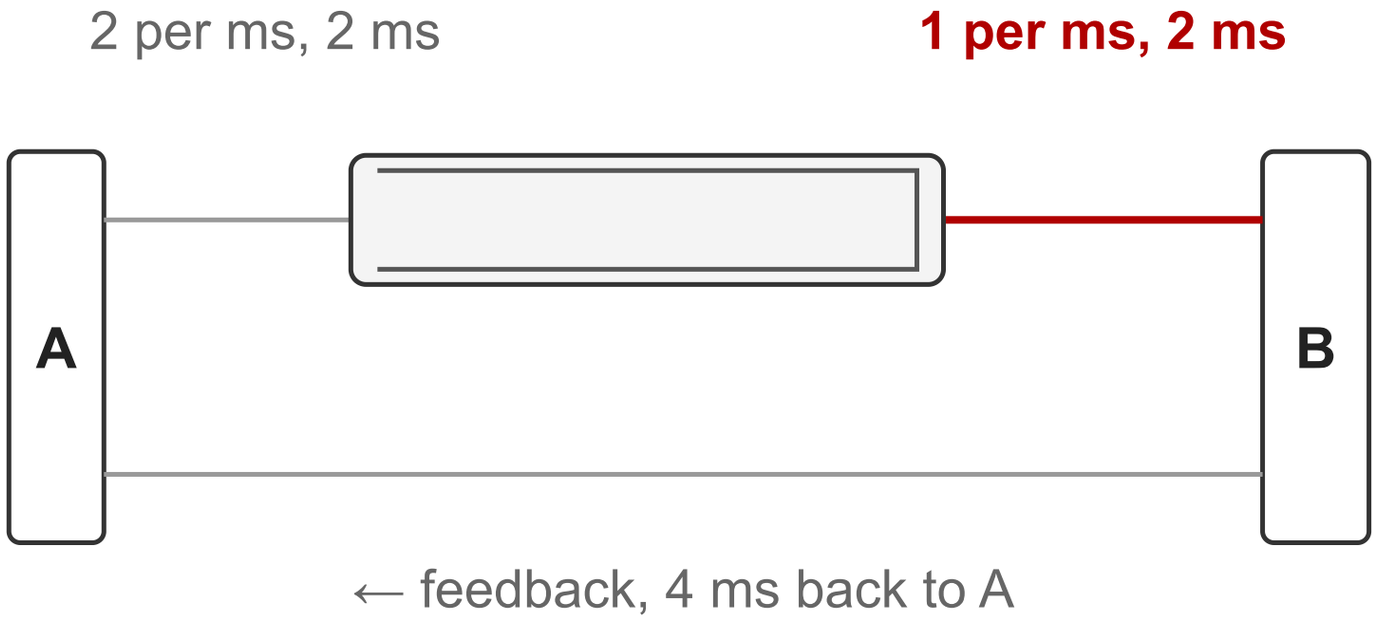




Each packet is 1500B. RTT (with no queueing) is 8ms.

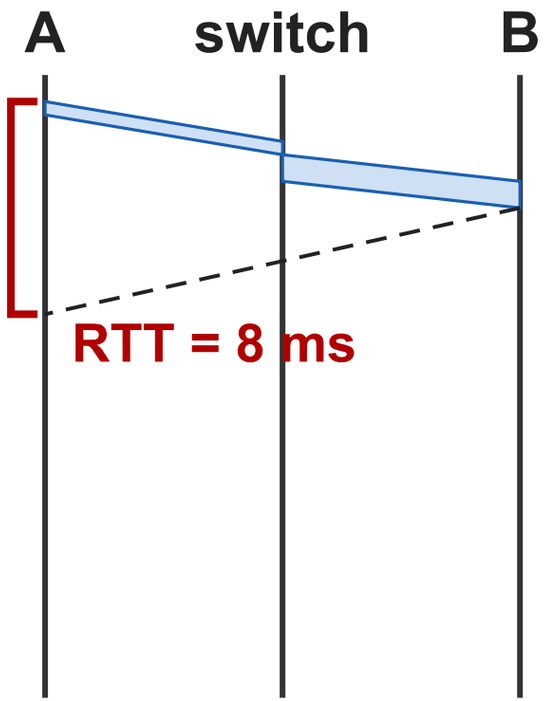
Bounding the number of in-flight bytes

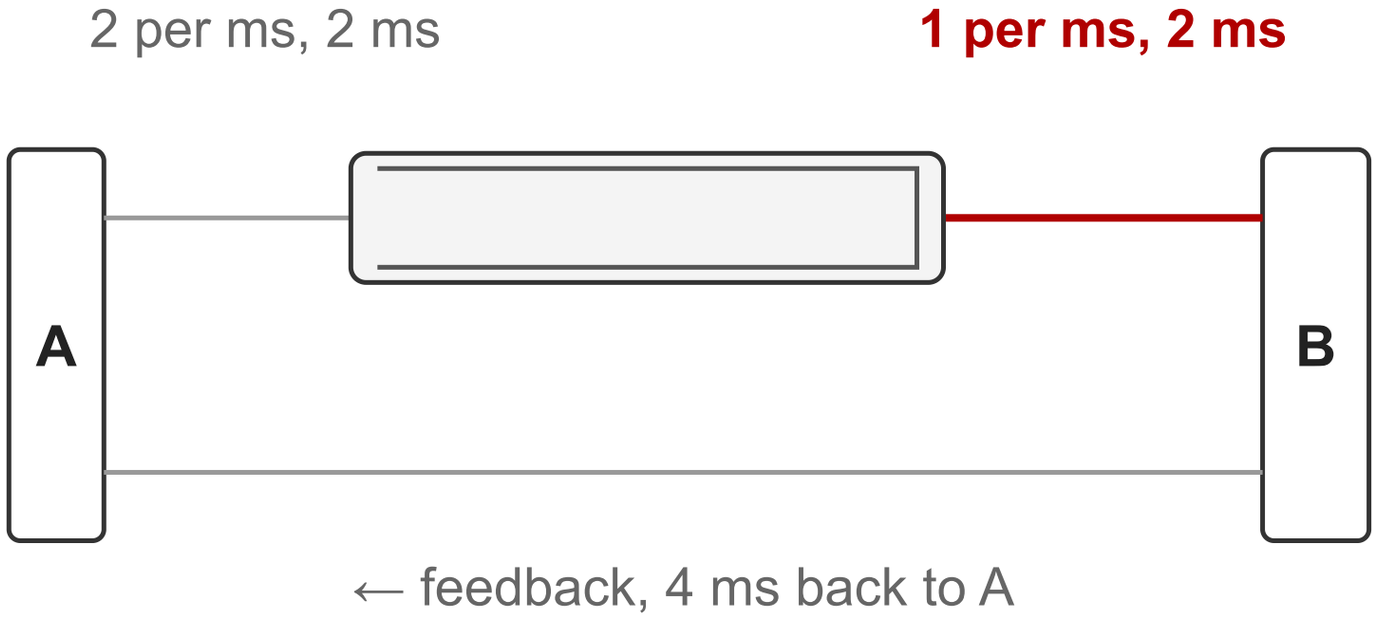




How many bytes could the bottleneck send in one RTT?

Bounding the number of in-flight bytes

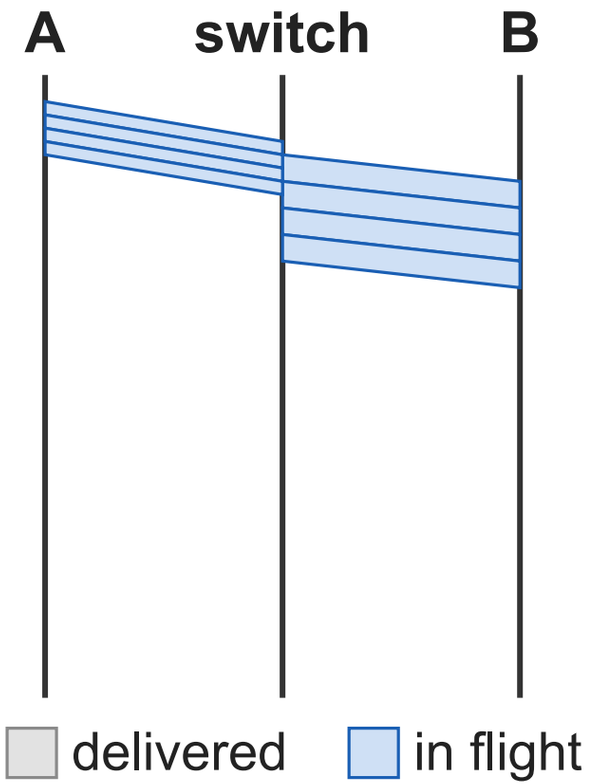


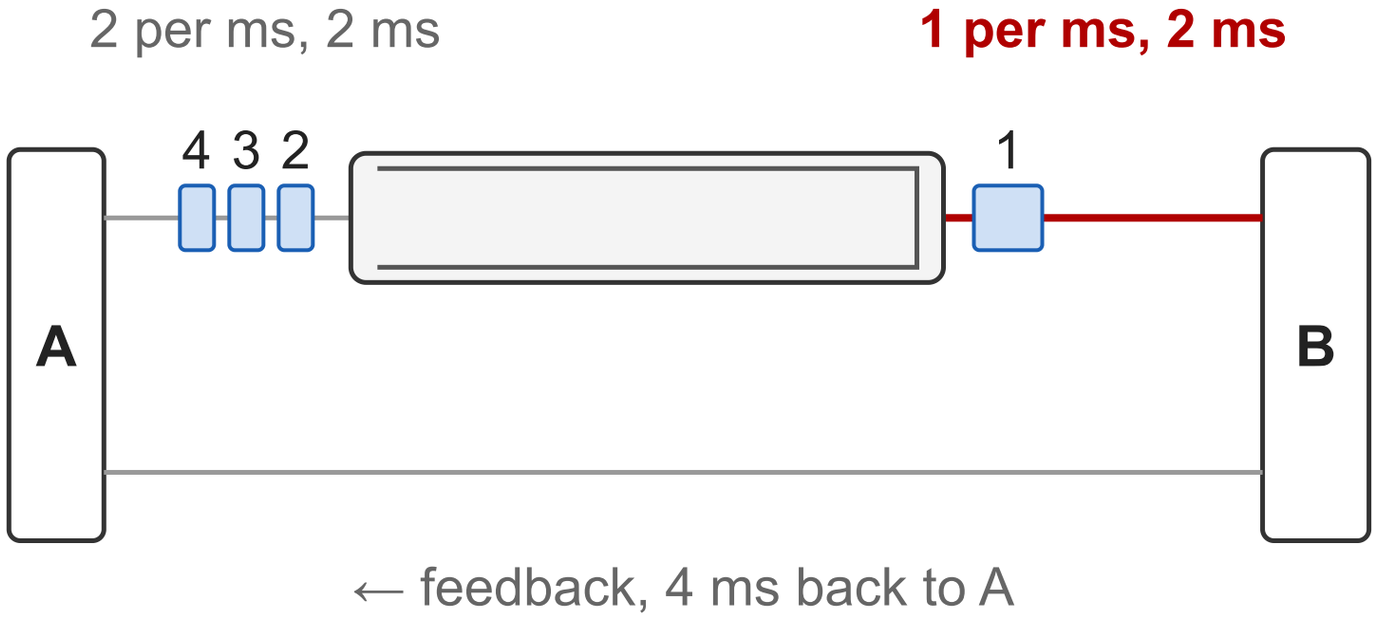


How many bytes could the bottleneck send in one RTT?

1 packet per ms × 8 ms = 8 packets = 12,000 bytes: the bandwidth-delay product (BDP)

Bounding the number of in-flight bytes



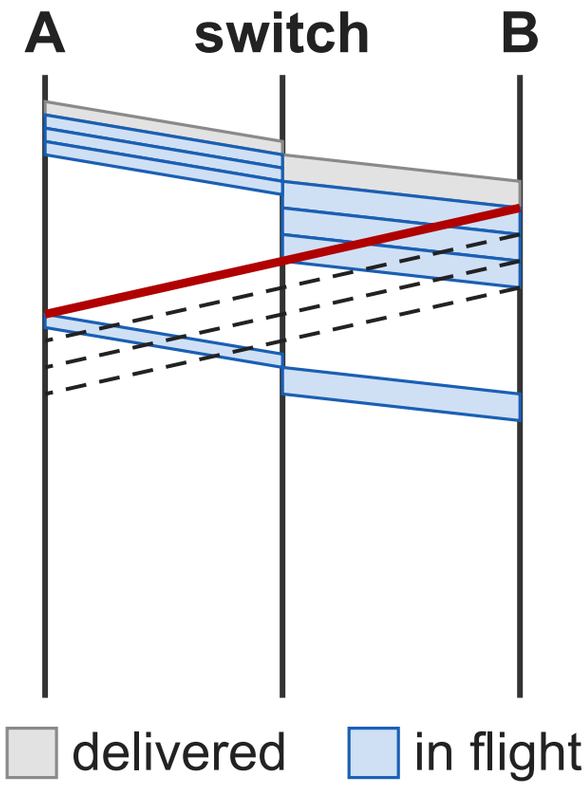


Bound 6,000 bytes (4 packets): below the BDP

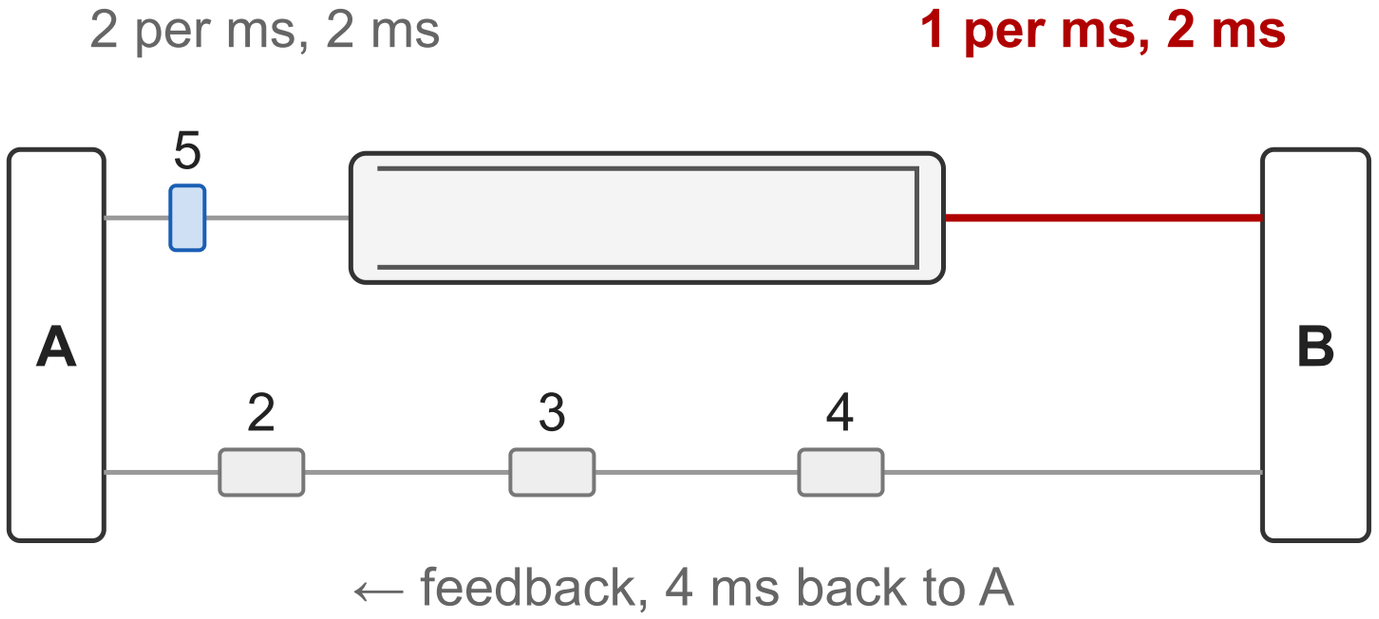
A sends packets 1 to 4 back to back.
No feedback yet: all 4 are in flight.

In flight now:
4 packets, 6,000 bytes

Bounding the number of in-flight bytes



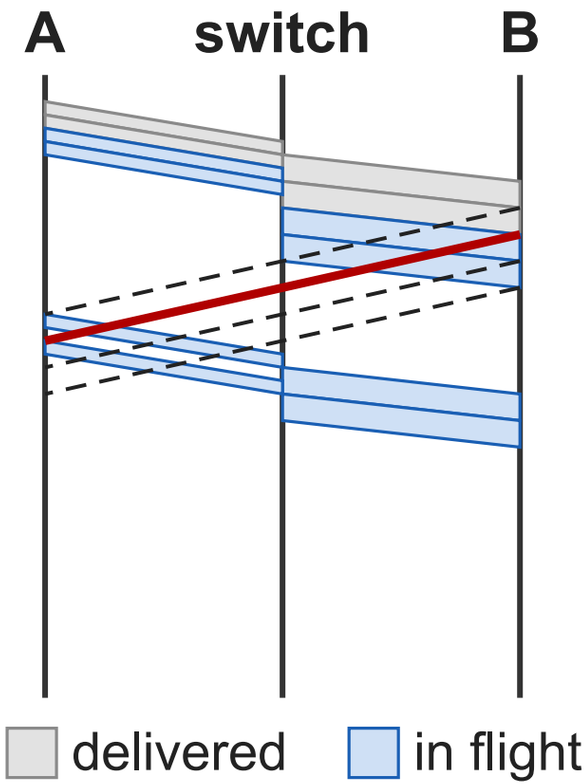
In flight now:
4 packets, 6,000 bytes



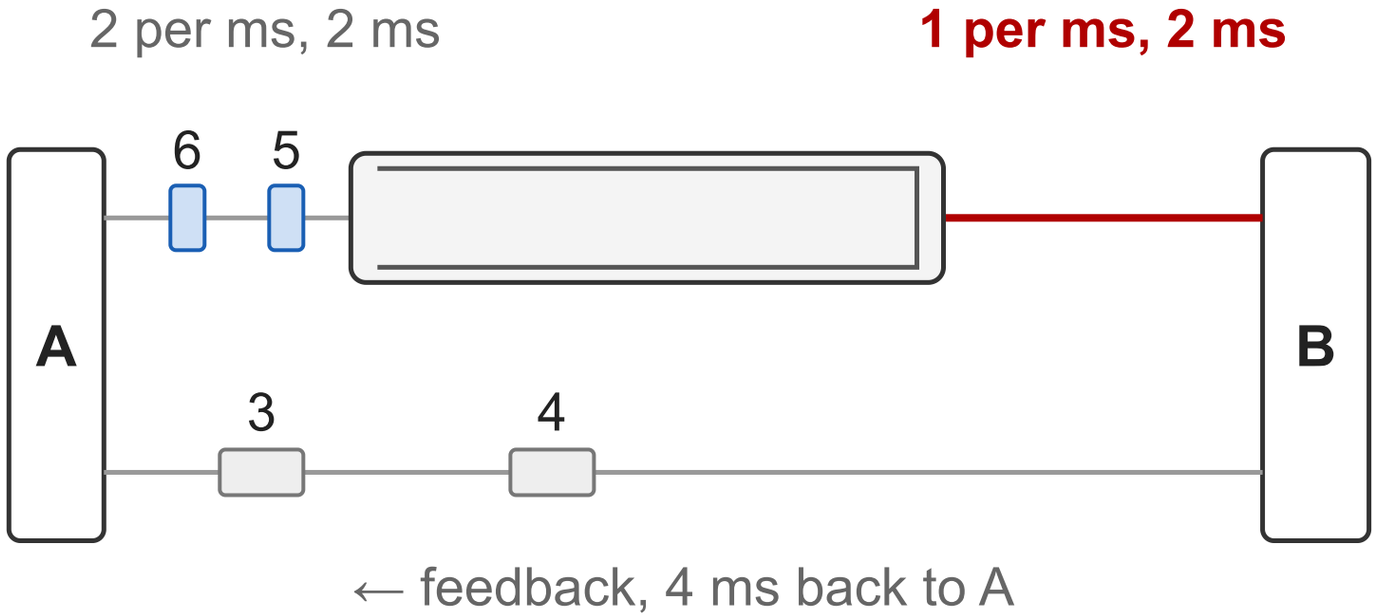
Bound 6,000 bytes (4 packets): below the BDP

Feedback for packet 1 comes back: 1 is delivered, and A sends packet 5.
Still 4 in flight: packets 2 to 5.

Bounding the number of in-flight bytes



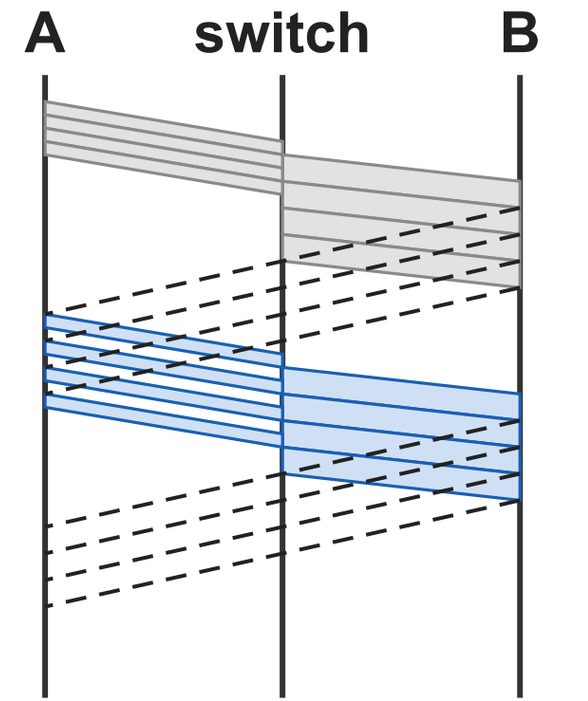
In flight now:
4 packets, 6,000 bytes



Bound 6,000 bytes (4 packets): below the BDP

Feedback for packet 2 comes back: 2 is delivered, and A sends packet 6.
Still 4 in flight: packets 3 to 6.

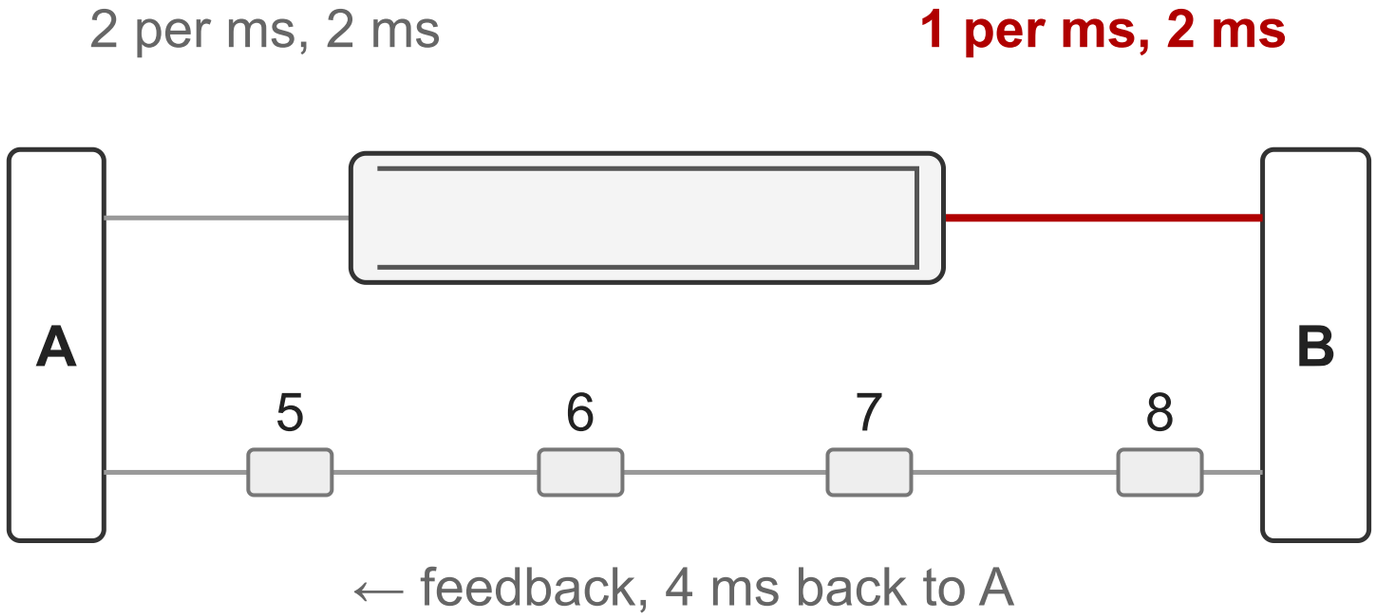
Bounding the number of in-flight bytes



delivered in flight

No packets wait in the queue

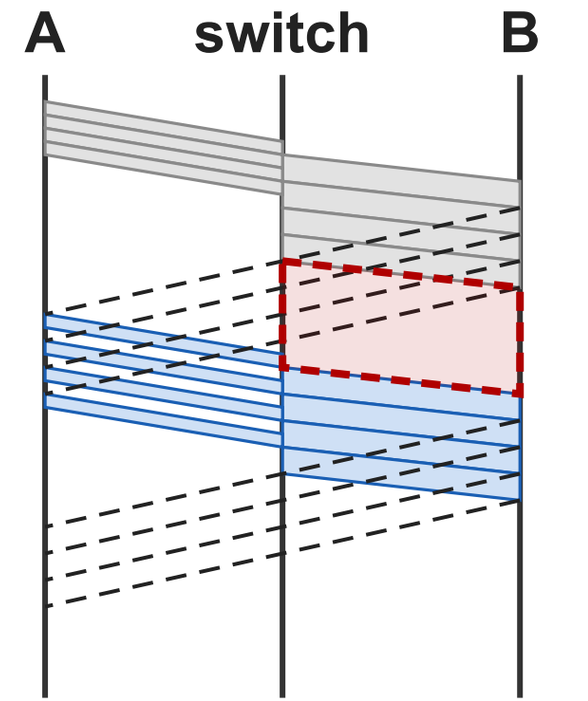
In flight now: 4 packets, 6,000 bytes



Bound 6,000 bytes (4 packets): below the BDP

A sends 4 back to back, then waits for the first feedback. B receives 4 packets, then sits idle for 4 ms.

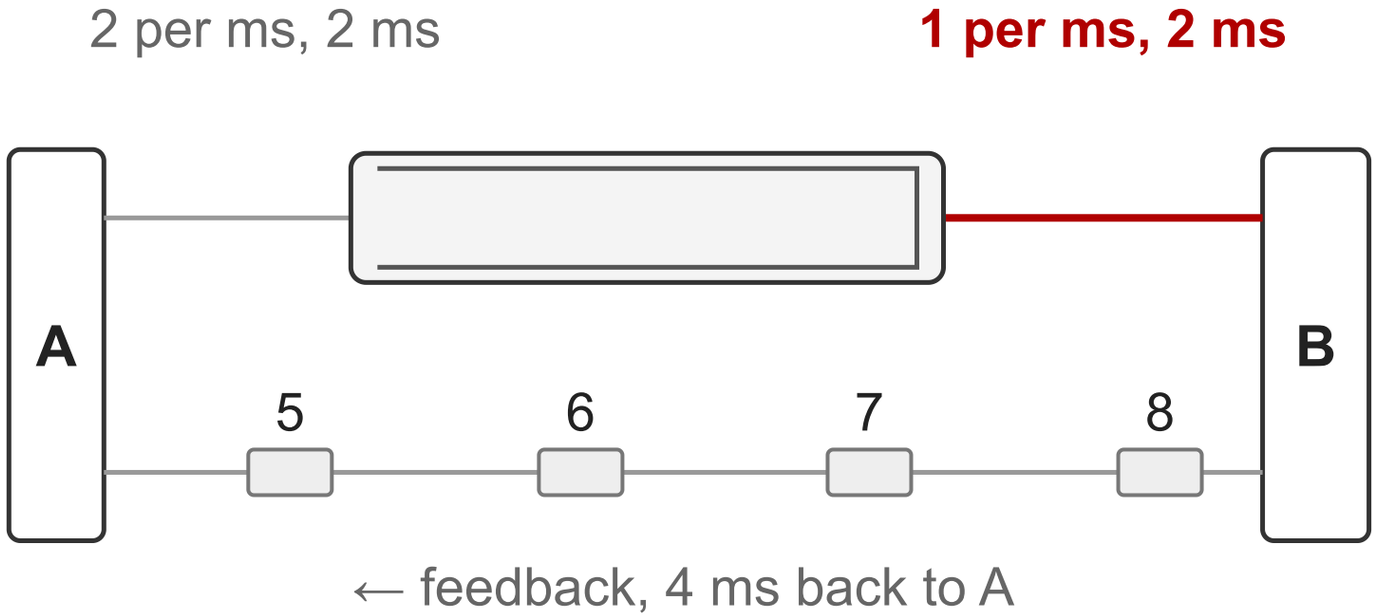
Bounding the number of in-flight bytes



delivered in flight

No packets wait in the queue

In flight now: 4 packets, 6,000 bytes

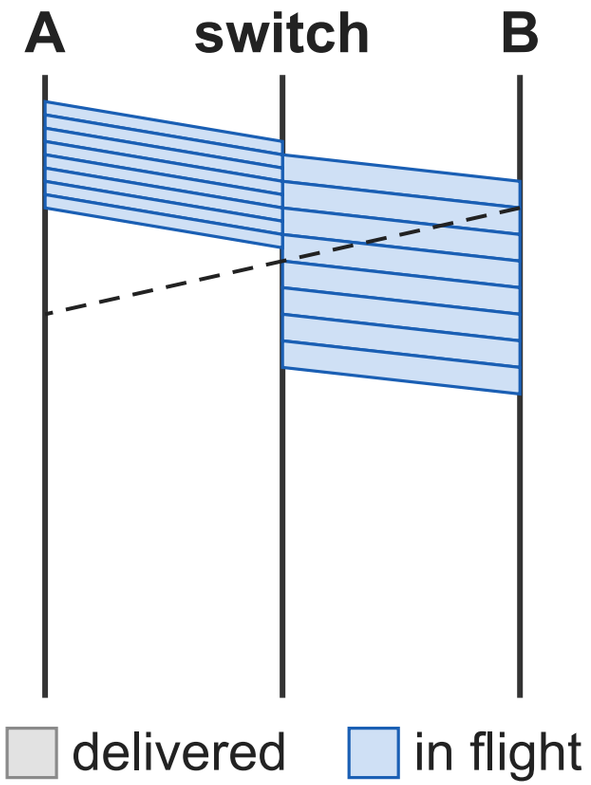


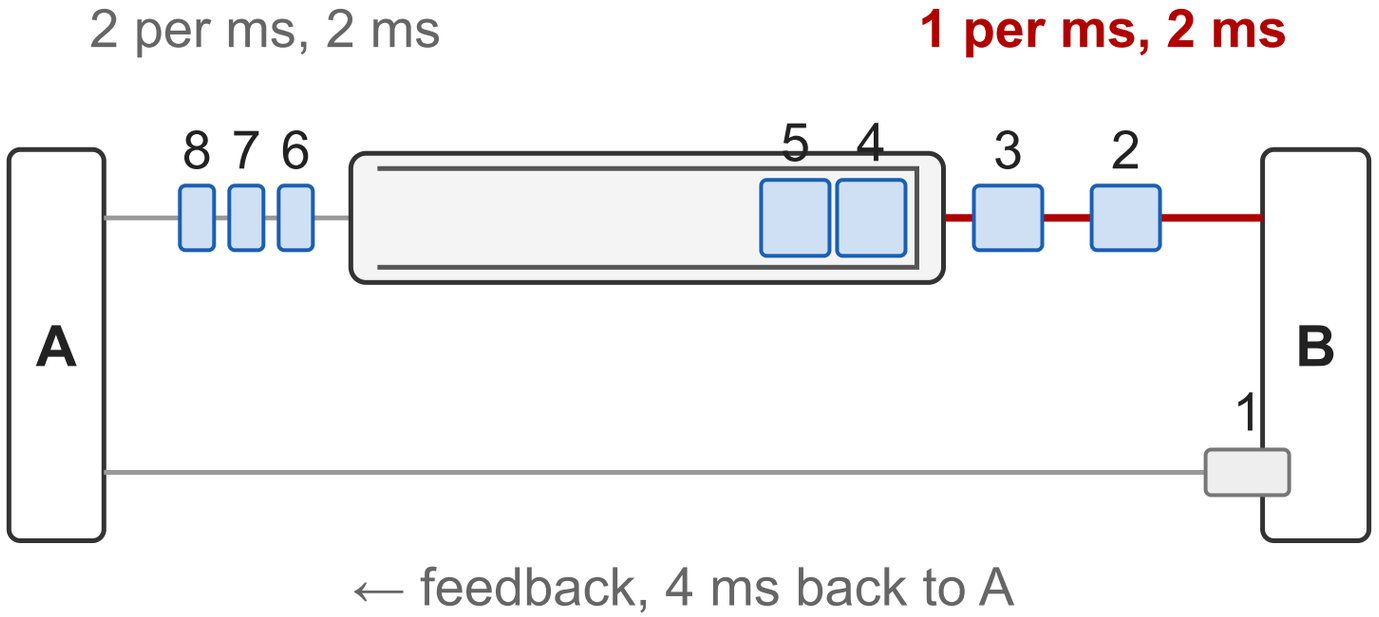
Bound 6,000 bytes (4 packets): below the BDP

A sends 4 back to back, then waits for the first feedback. B receives 4 packets, then sits idle for 4 ms.

But the bottleneck is not fully utilized

Bounding the number of in-flight bytes



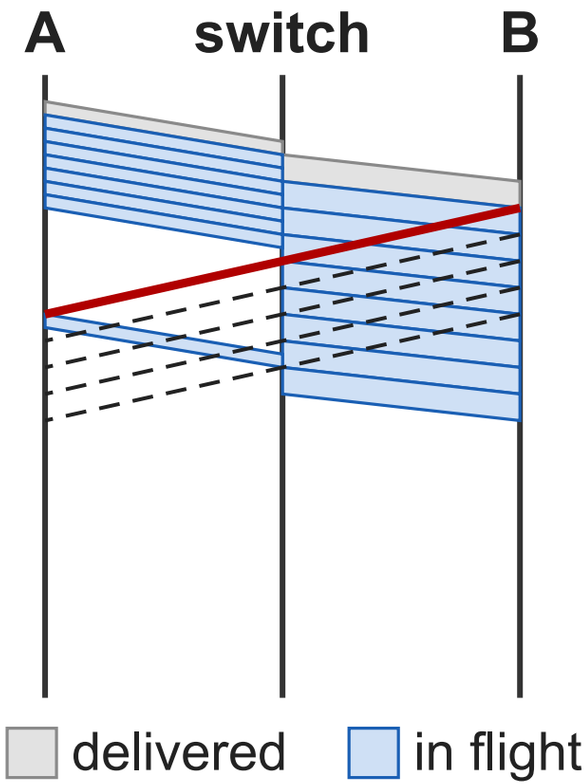


Bound 12,000 bytes (8 packets): the BDP

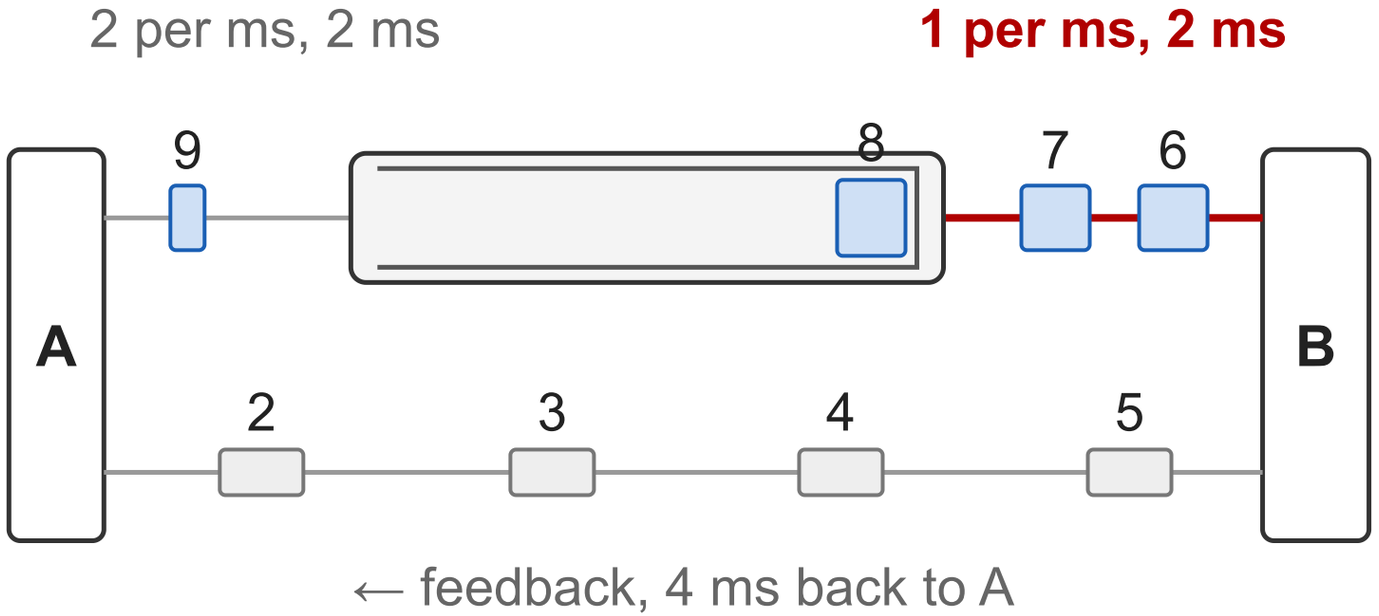
A sends packets 1 to 8 back to back.
No feedback yet: all 8 are in flight.

In flight now:
8 packets, 12,000 bytes

Bounding the number of in-flight bytes



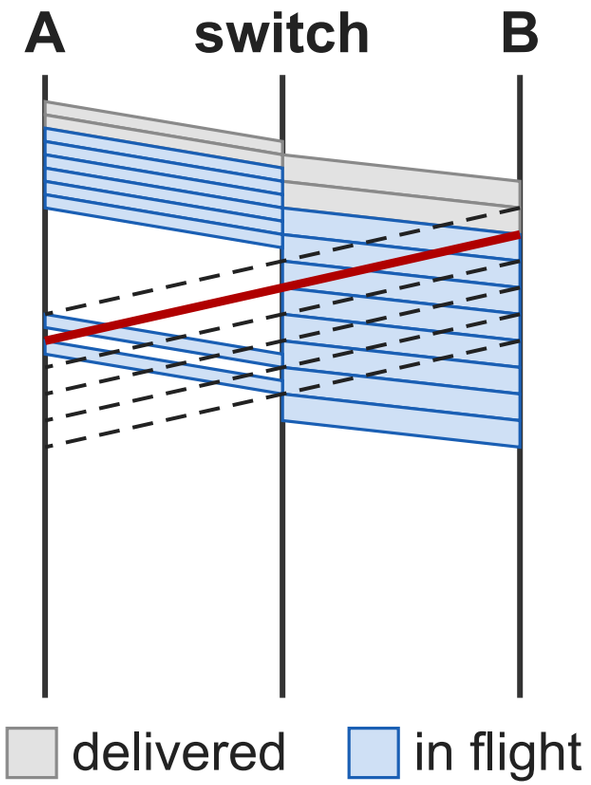
In flight now:
8 packets, 12,000 bytes



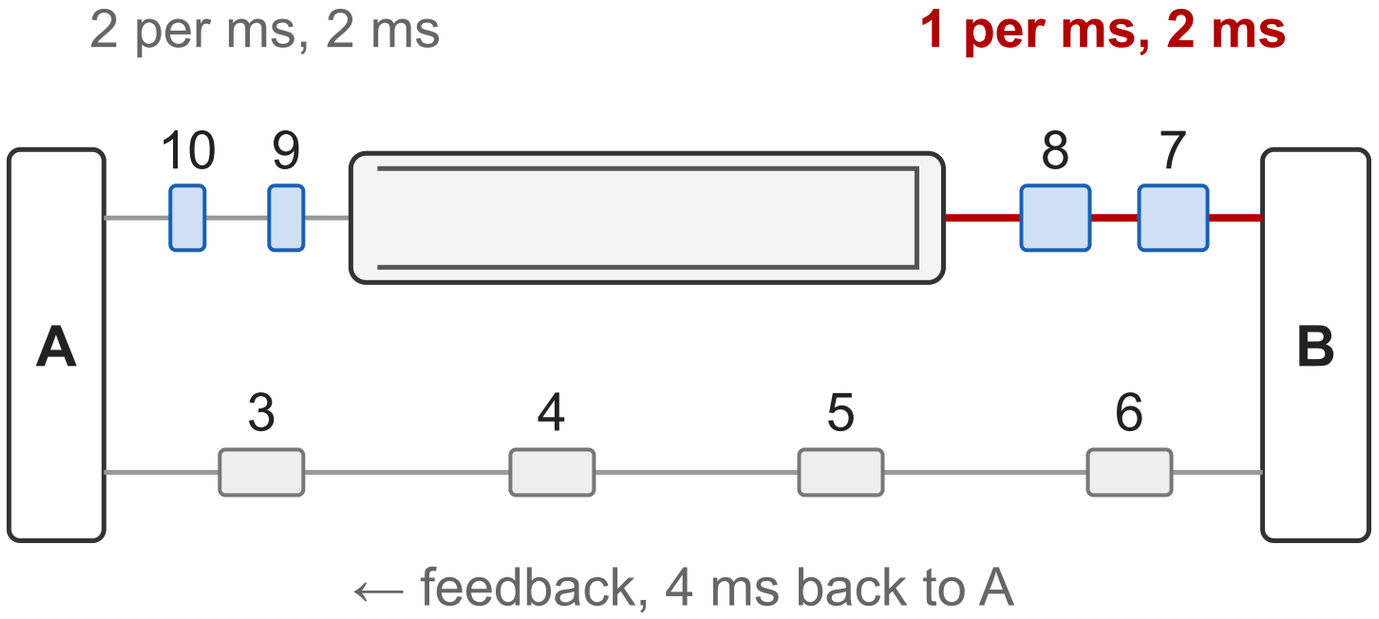
Bound 12,000 bytes (8 packets): the BDP

Feedback for packet 1 comes back: 1 is delivered, and A sends packet 9.
Still 8 in flight: packets 2 to 9.

Bounding the number of in-flight bytes



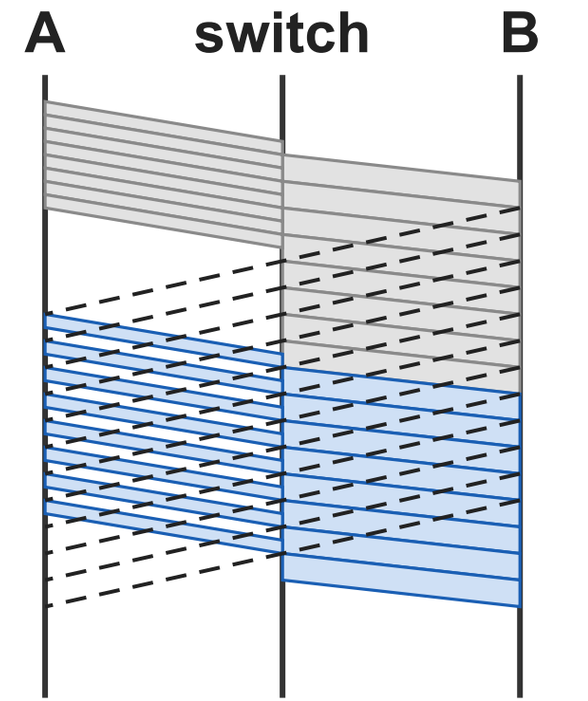
In flight now:
8 packets, 12,000 bytes



Bound 12,000 bytes (8 packets): the BDP

Feedback for packet 2 comes back: 2 is delivered, and A sends packet 10.
Still 8 in flight: packets 3 to 10.

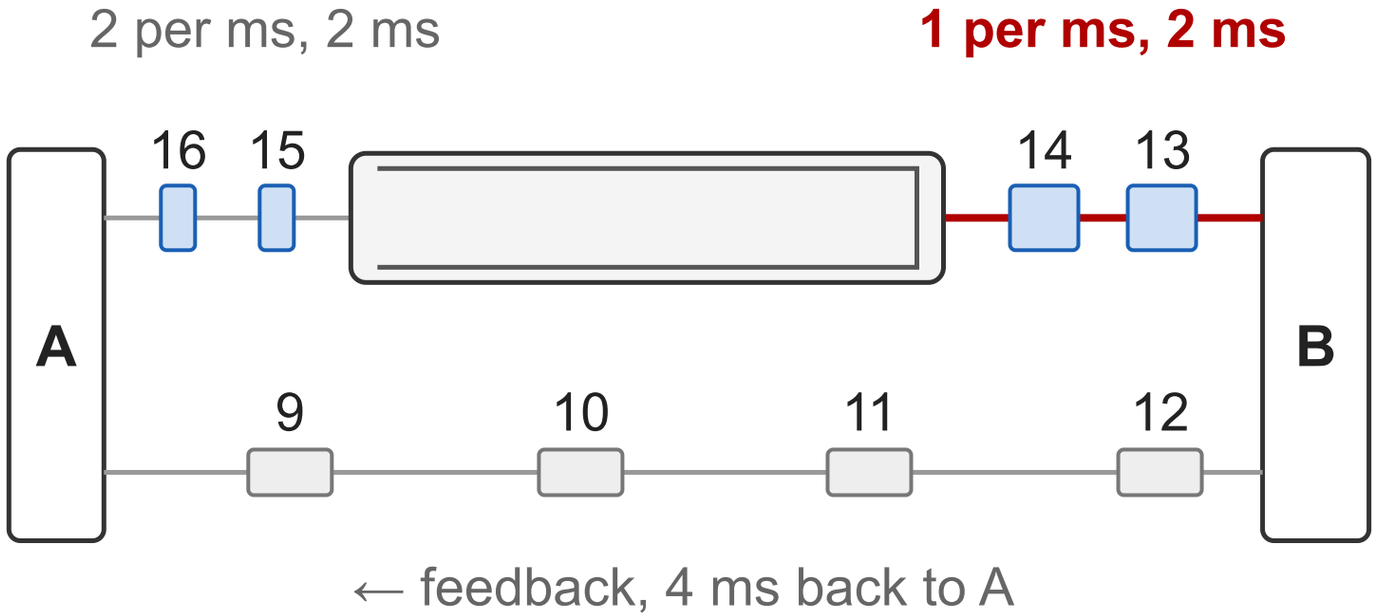
Bounding the number of in-flight bytes



delivered in flight

No packets wait in the queue

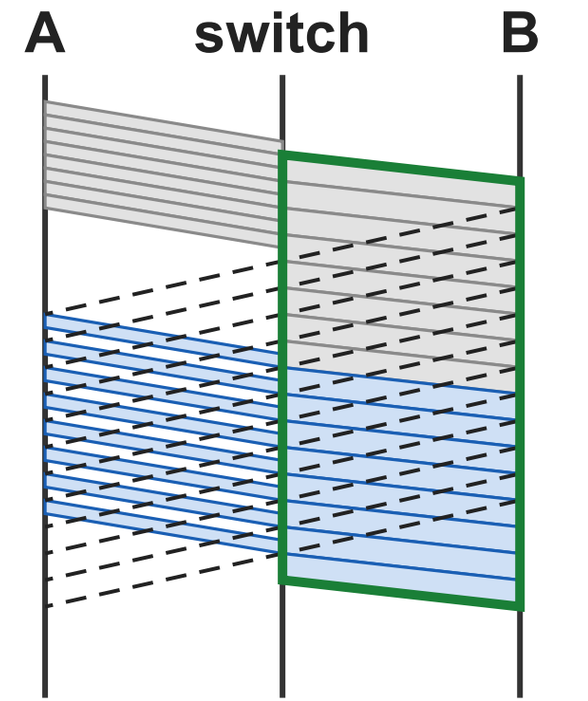
In flight now:
8 packets, 12,000 bytes



Bound 12,000 bytes (8 packets): the BDP

After the first round, the round trip is full and the queue empty. B continuously receives packets, with no idle time.

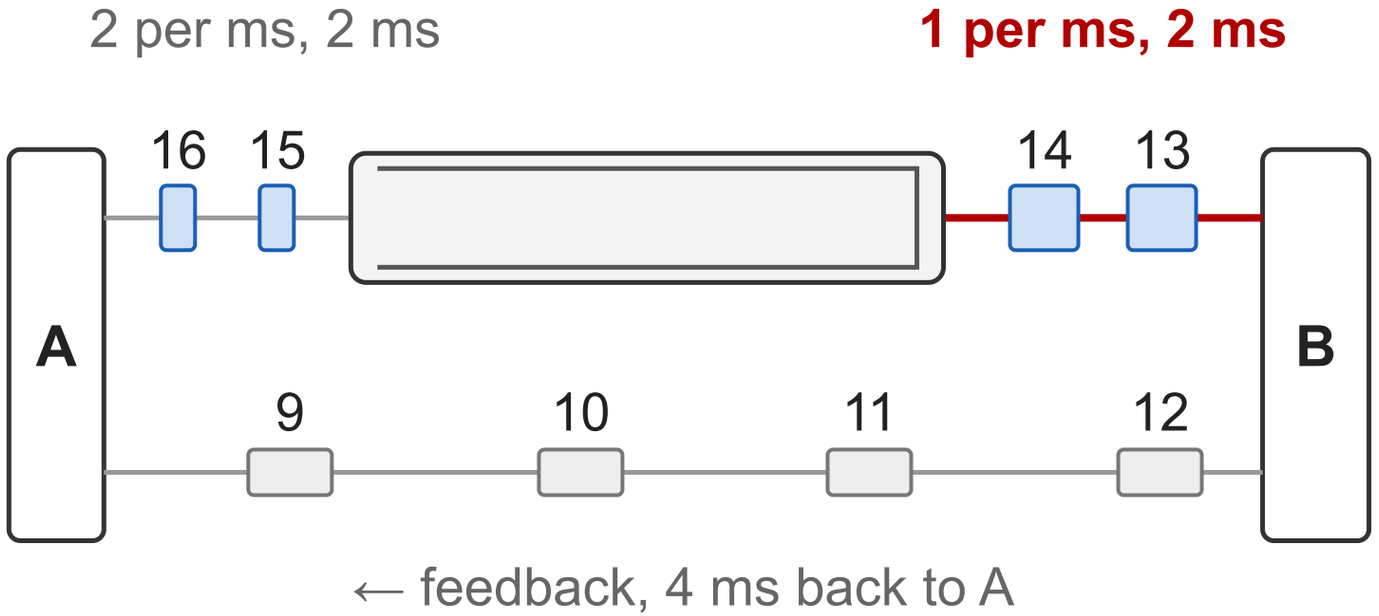
Bounding the number of in-flight bytes



delivered in flight

No packets wait in the queue

In flight now: 8 packets, 12,000 bytes

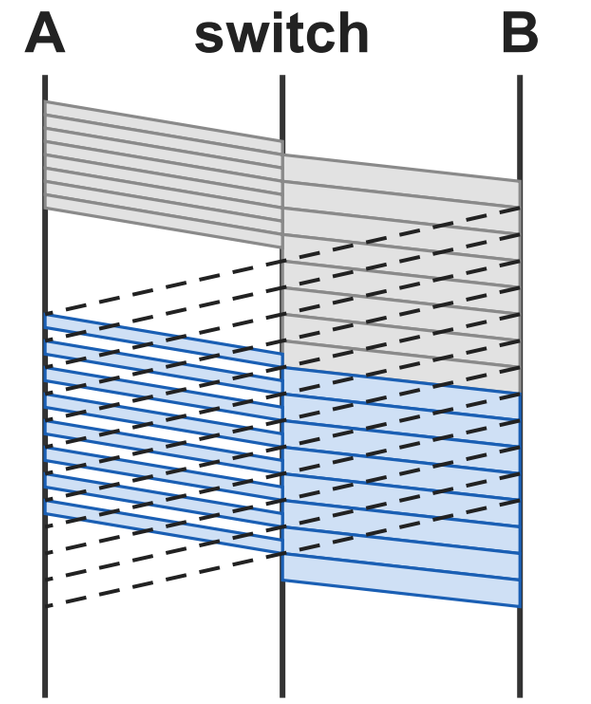


Bound 12,000 bytes (8 packets): the BDP

After the first round, the round trip is full and the queue empty. B continuously receives packets, with no idle time.

The bottleneck is now fully utilized

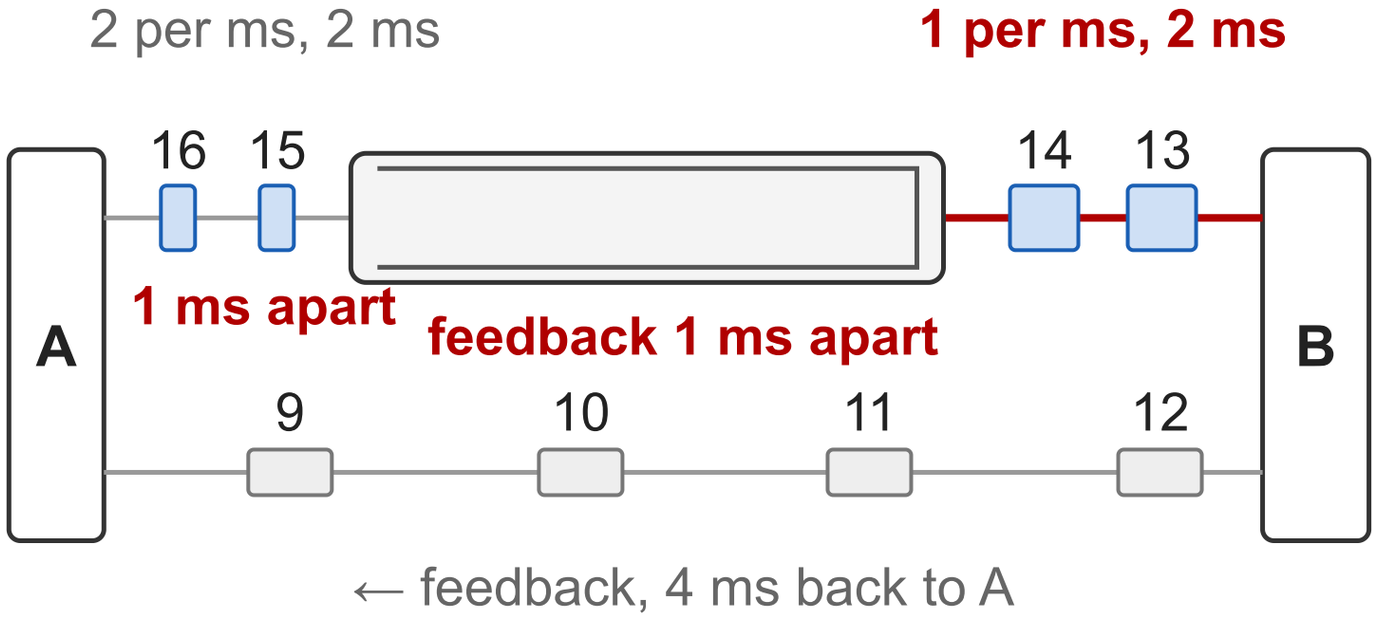
Bounding the number of in-flight bytes



delivered in flight

No packets wait in the queue

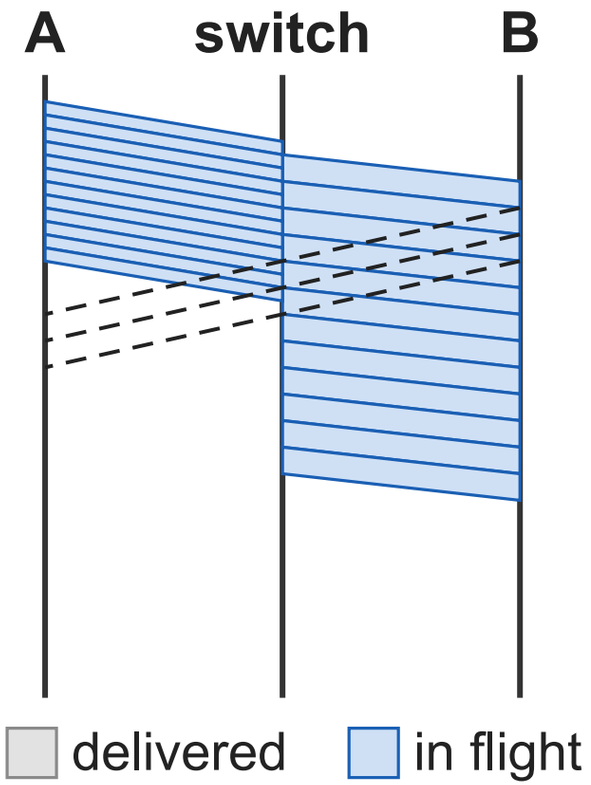
In flight now:
8 packets, 12,000 bytes

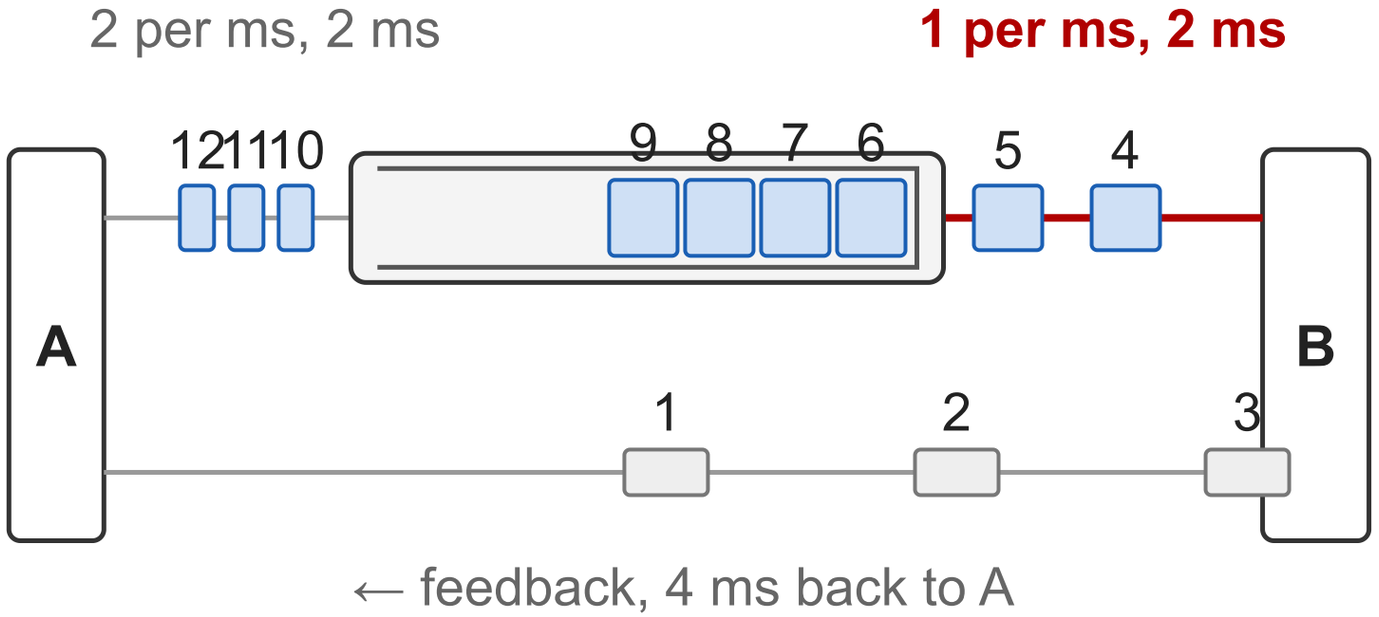


Self-clocking

After the first batch is sent out, the sliding window only moves forward when feedback comes back. So, new data enters the network at the rate old data is delivered and its feedback is received.

Bounding the number of in-flight bytes



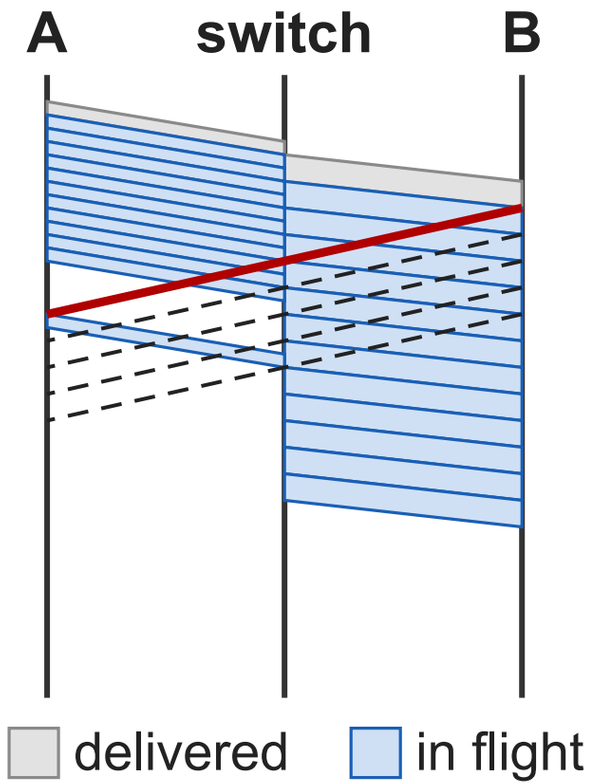


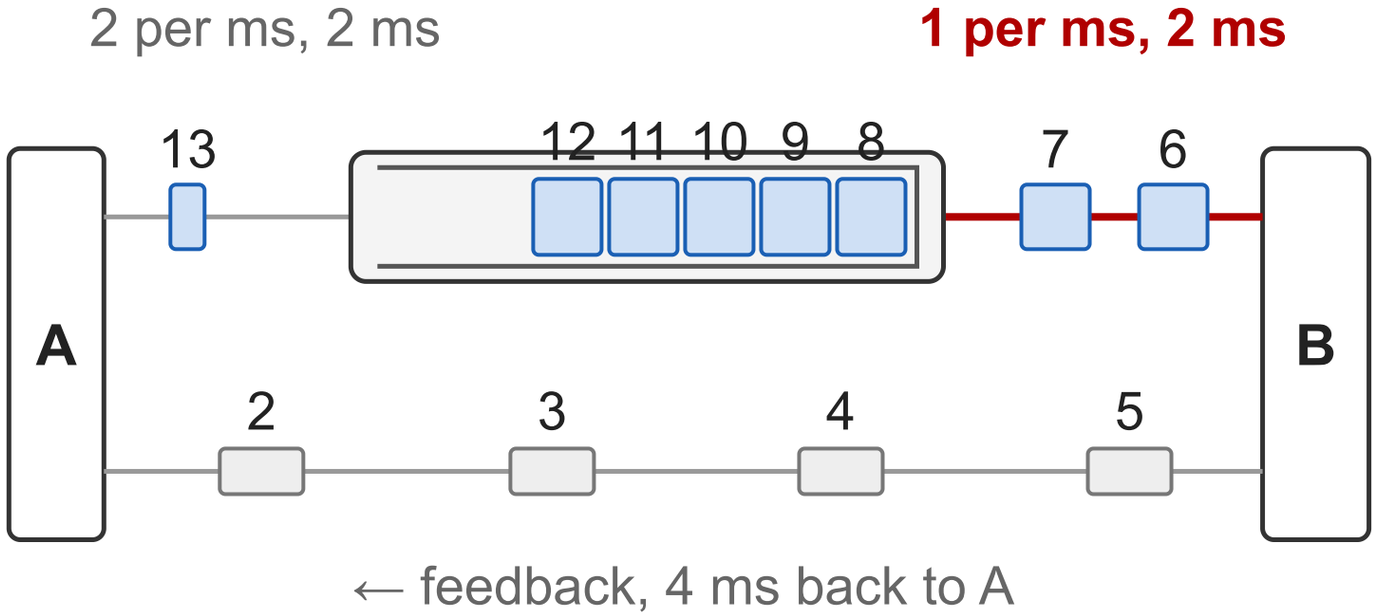
Bound 18,000 bytes (12 packets): above the BDP

A sends packets 1 to 12 back to back.
No feedback yet: all 12 are in flight.

In flight now:
12 packets, 18,000 bytes

Bounding the number of in-flight bytes



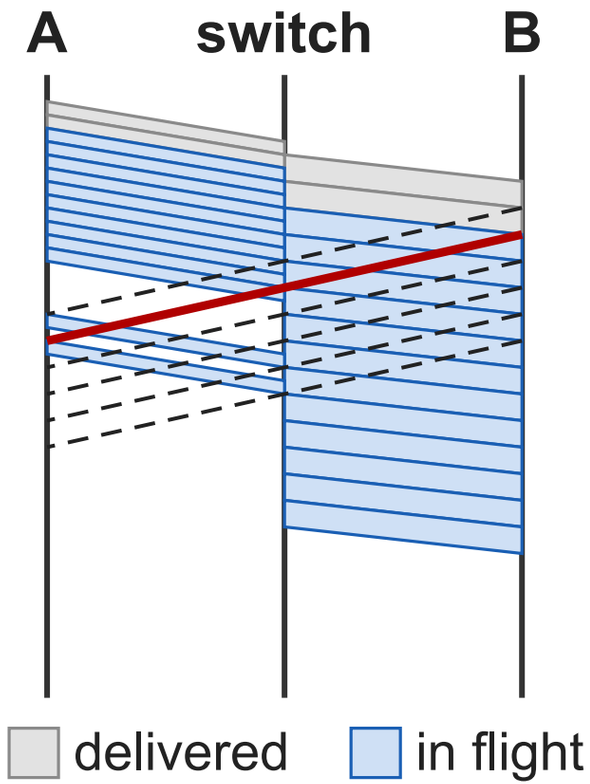


Bound 18,000 bytes (12 packets): above the BDP

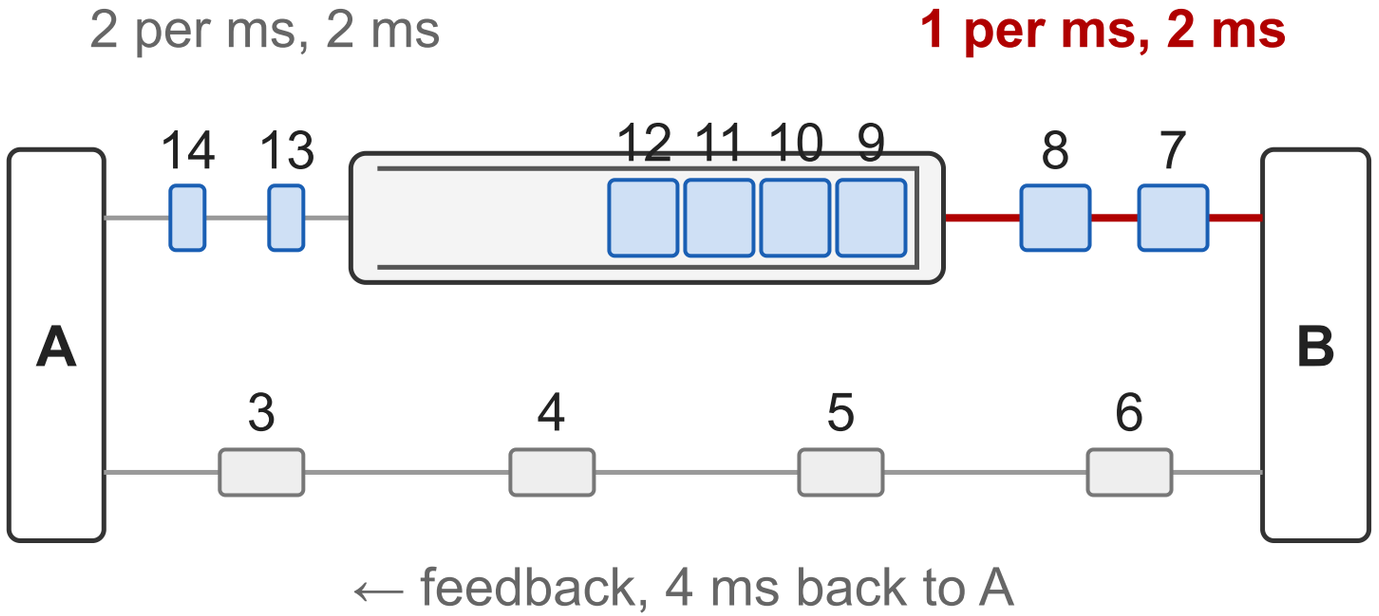
Feedback for packet 1 comes back: 1 is delivered, and A sends packet 13.
Still 12 in flight: packets 2 to 13.

In flight now:
12 packets, 18,000 bytes

Bounding the number of in-flight bytes



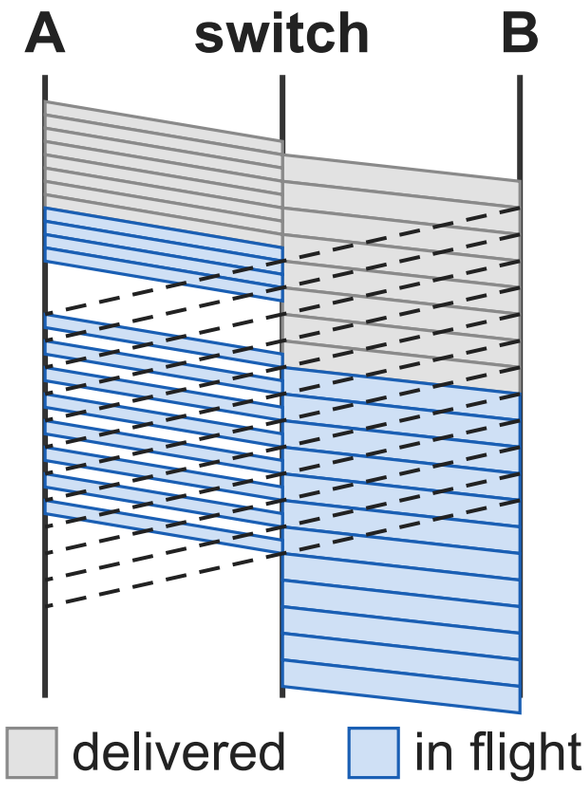
In flight now:
12 packets, 18,000 bytes



Bound 18,000 bytes (12 packets): above the BDP

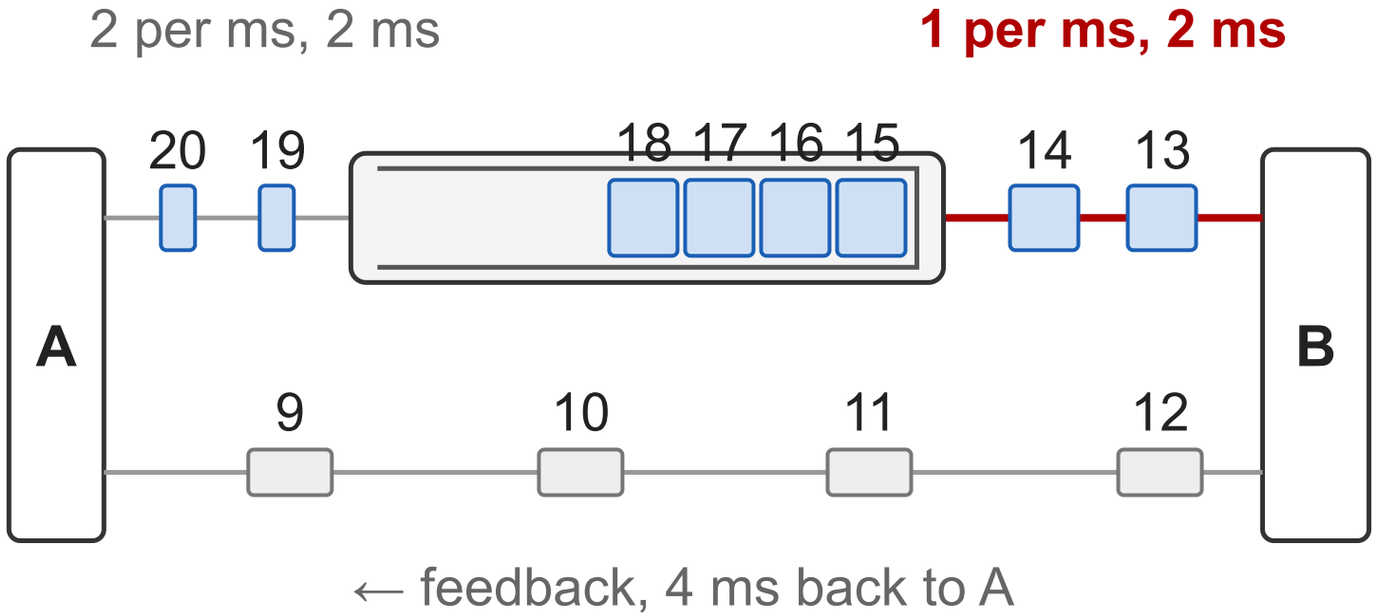
Feedback for packet 2 comes back: 2 is delivered, and A sends packet 14.
Still 12 in flight: packets 3 to 14.

Bounding the number of in-flight bytes



4 packets always wait in the queue

In flight now:
12 packets, 18,000 bytes

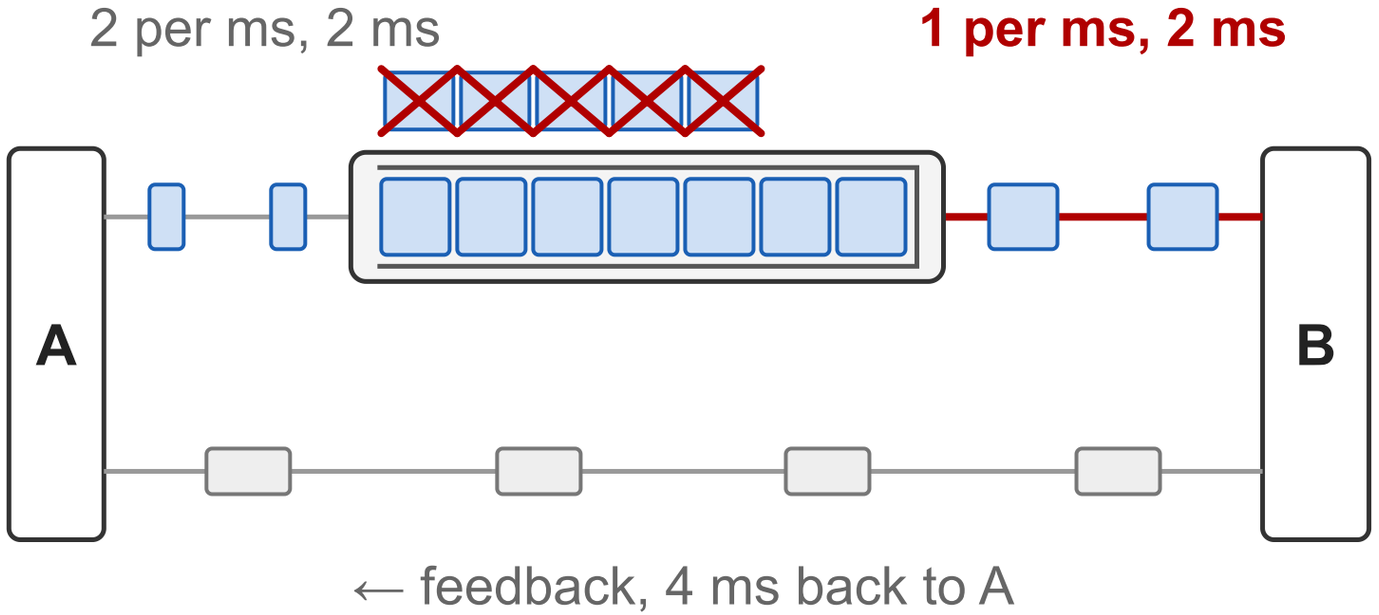


Bound 18,000 bytes (12 packets): above the BDP

The round trip is full **and** four packets always wait in the queue. B still gets one every 1 ms, with no idle time.

But the roundtrip is now 12 ms

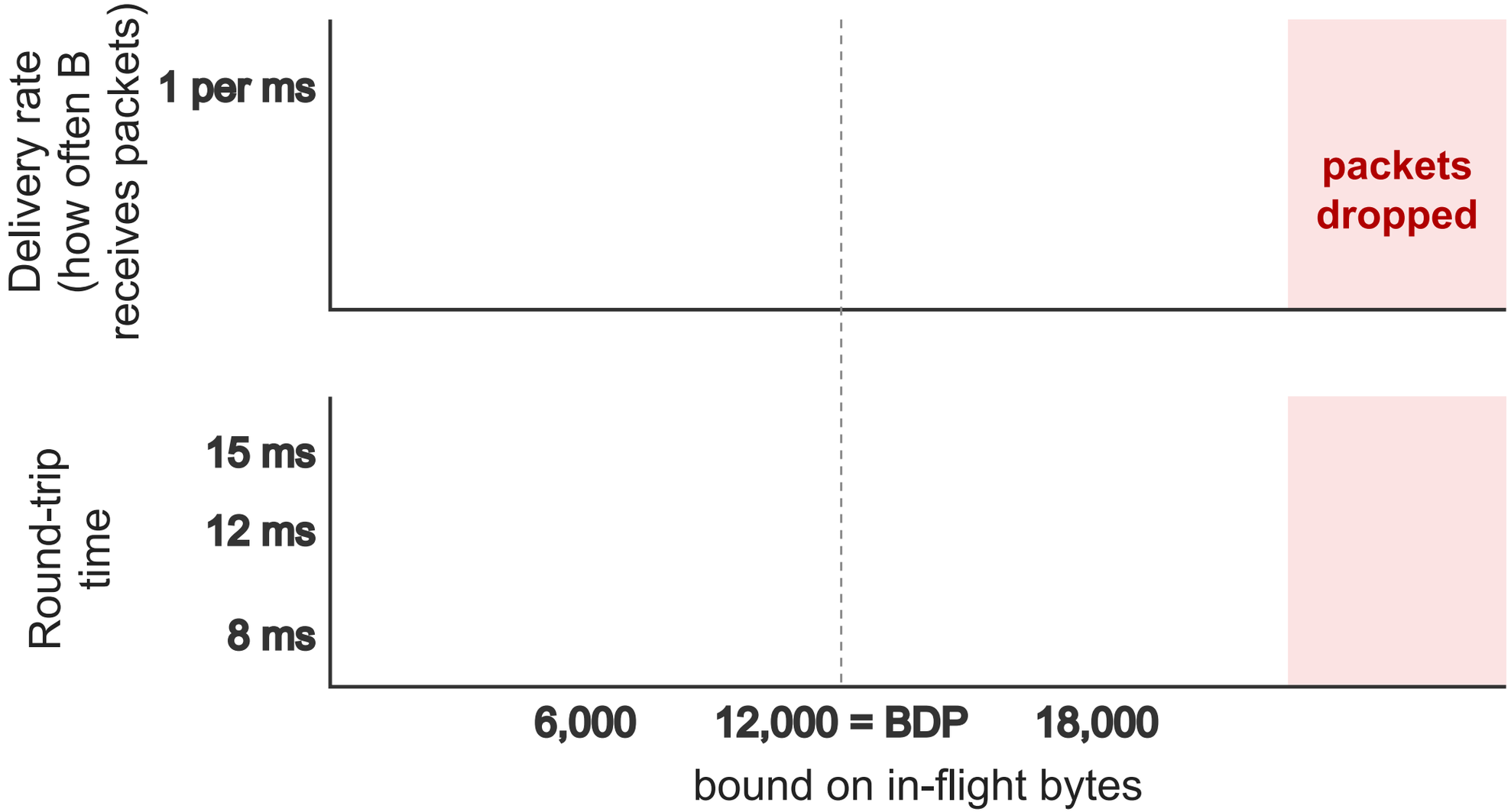
Bounding the number of in-flight bytes



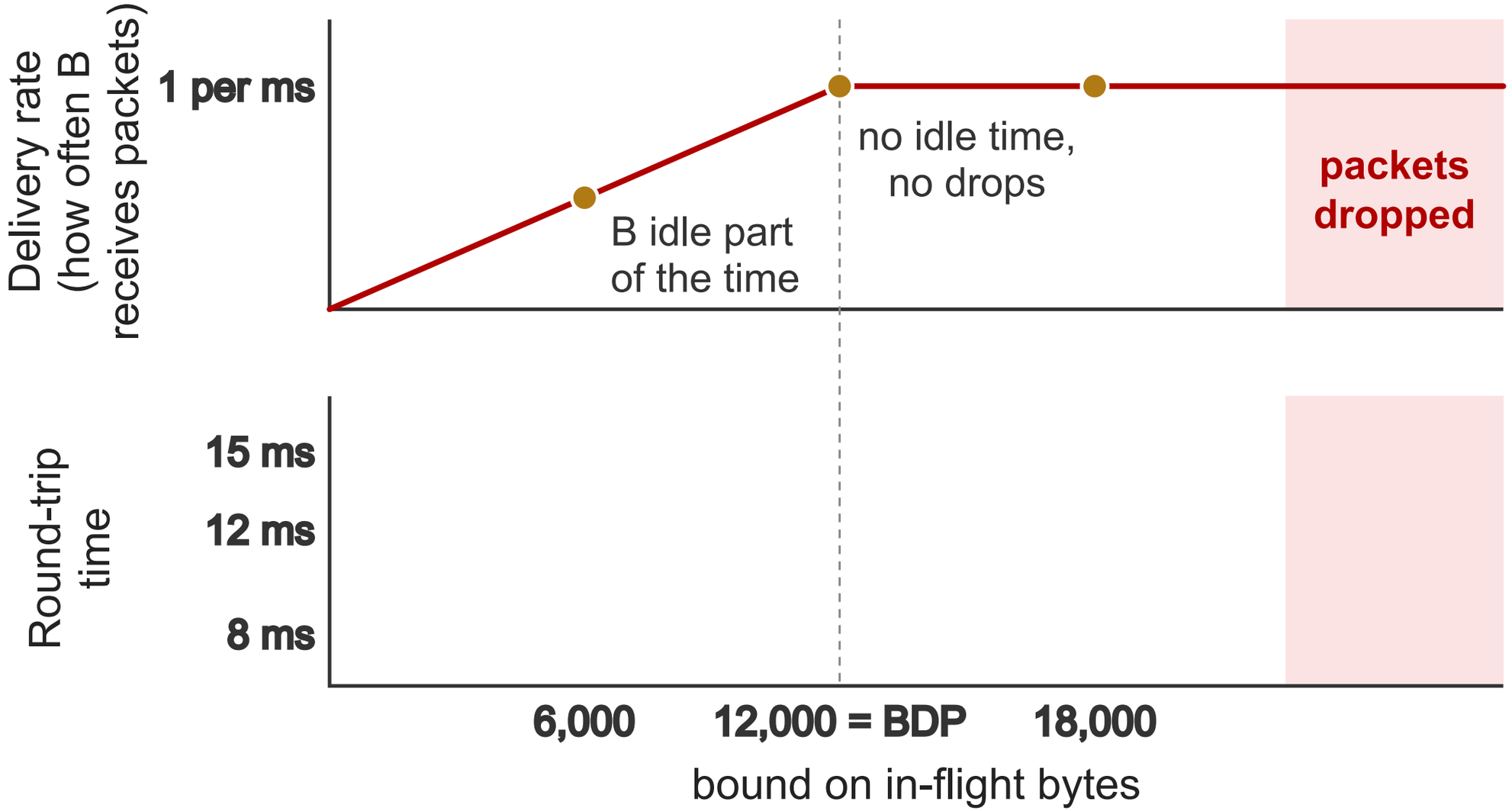
Bound 30,000 bytes (20 packets): far above the BDP

Round trip full, queue full (7), 5 **dropped**
every round trip. B still gets one every 1 ms,
but some are **retransmissions**.
The RTT is now 15 ms

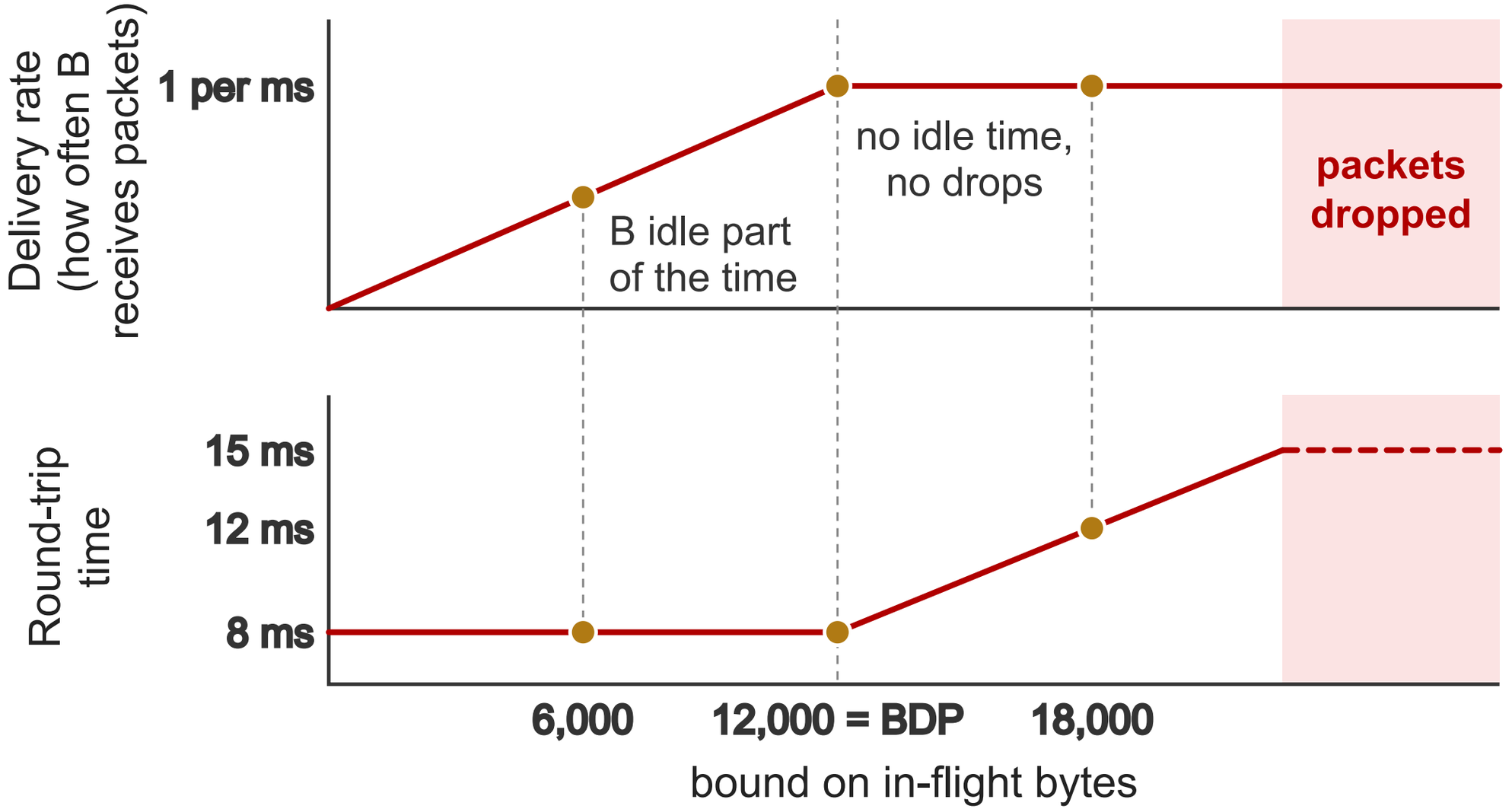
Bounding the number of in-flight bytes



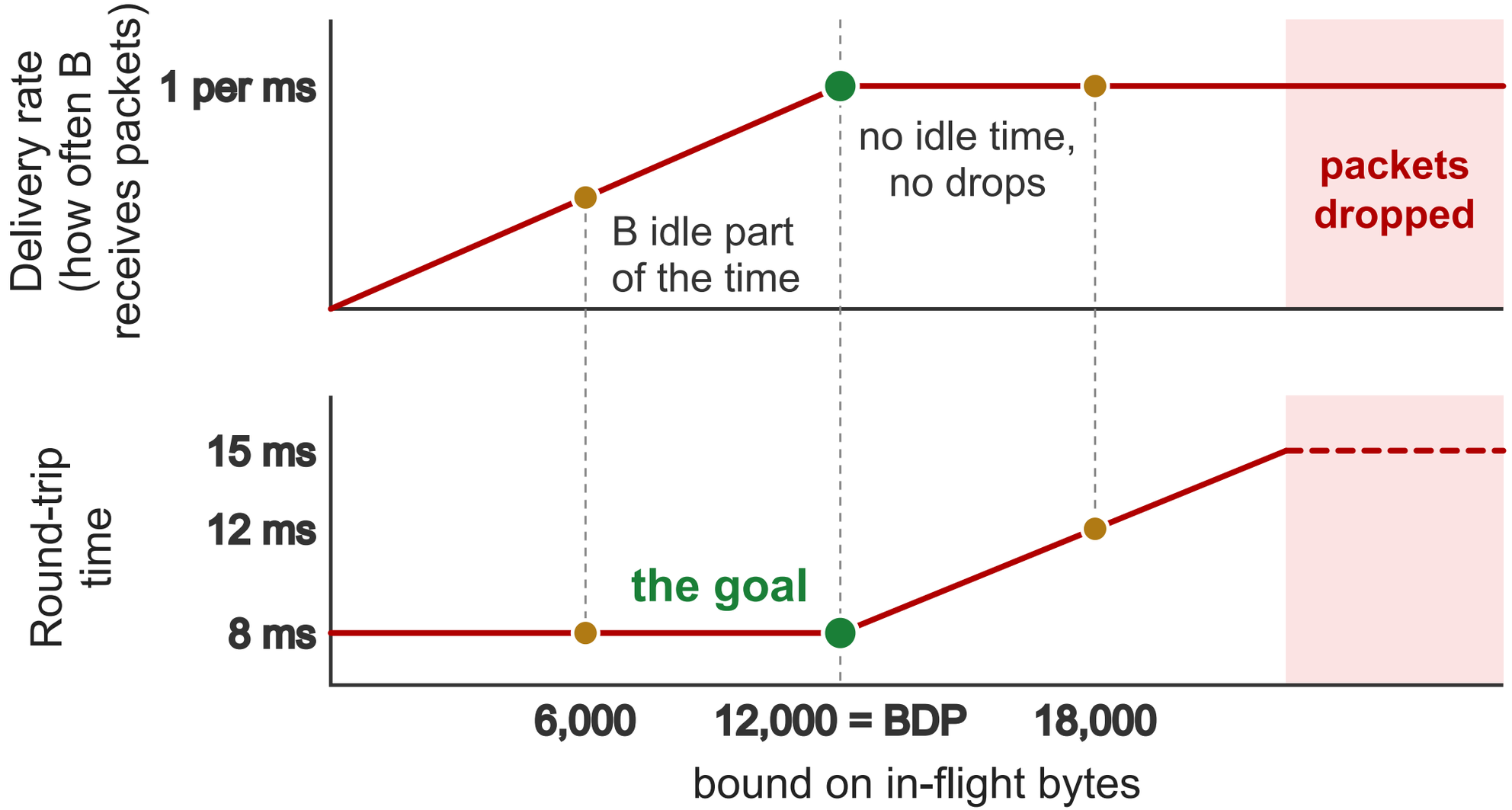
Bounding the number of in-flight bytes



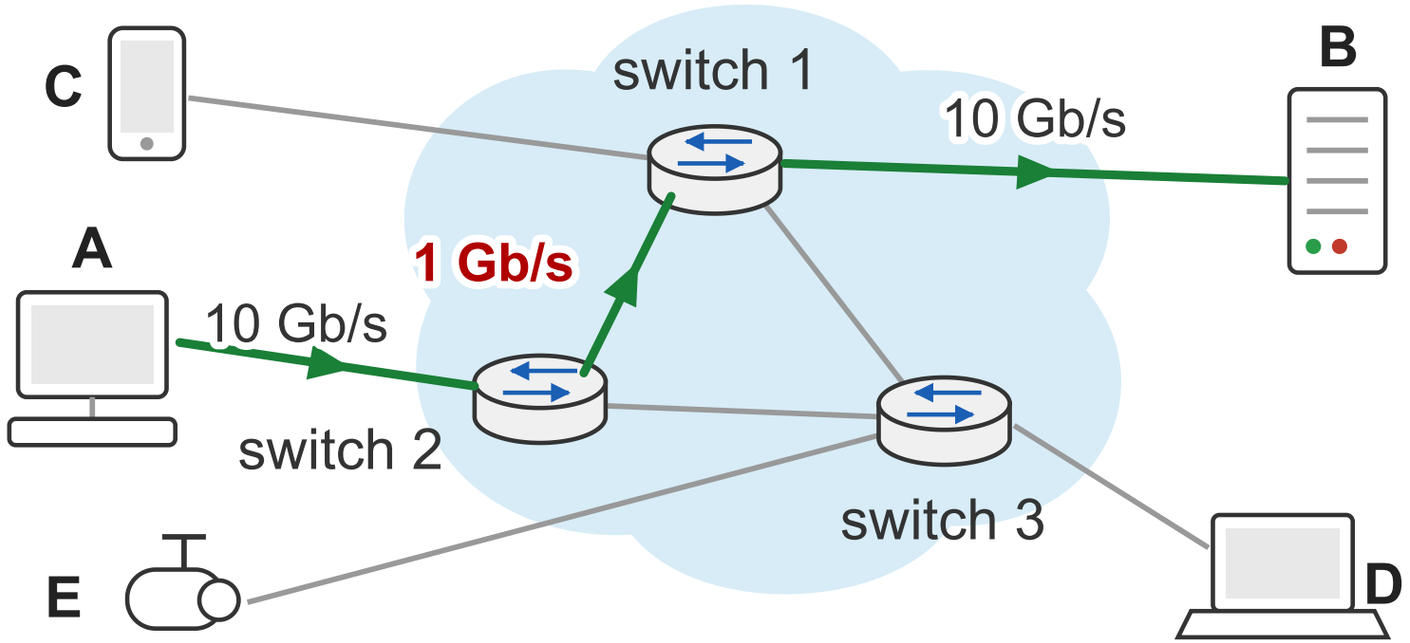
Bounding the number of in-flight bytes



Bounding the number of in-flight bytes



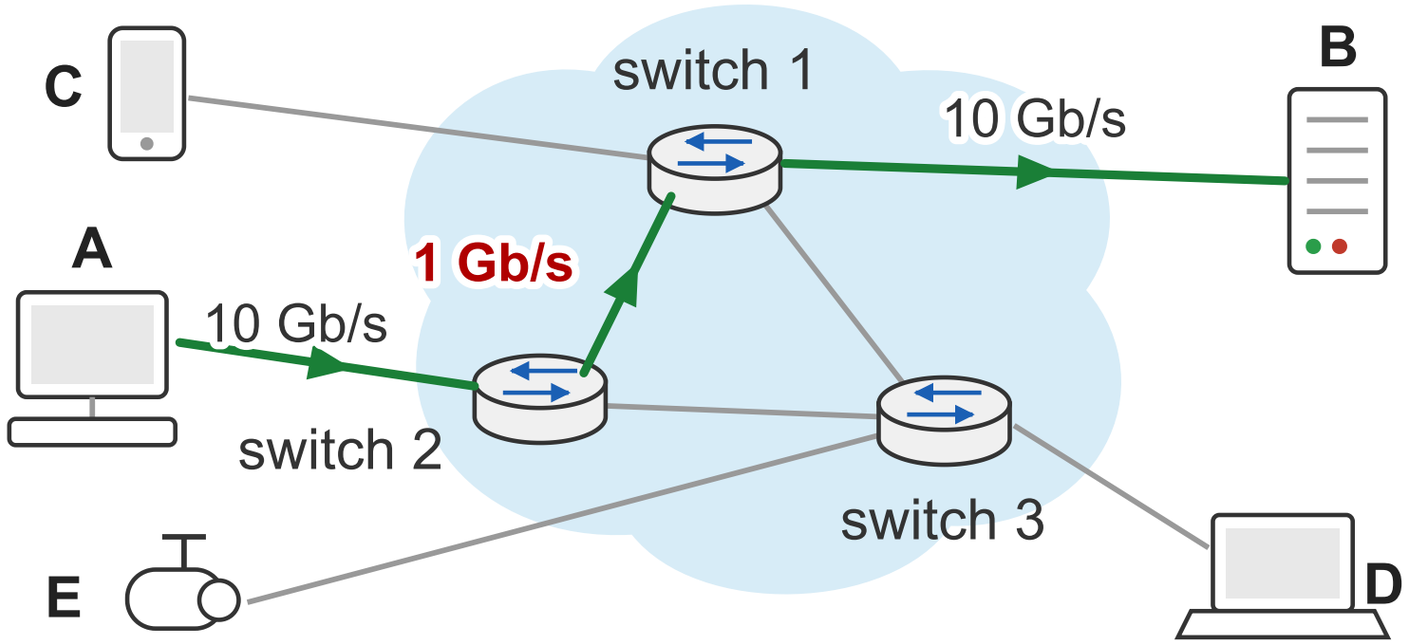
How fast should the sender send?



Consider a sender and a receiver:

- **The sender knows exactly what path its traffic will take to the destination**, and the properties (rate, propagation delay, etc.) of each link
- The sender always has more data to send

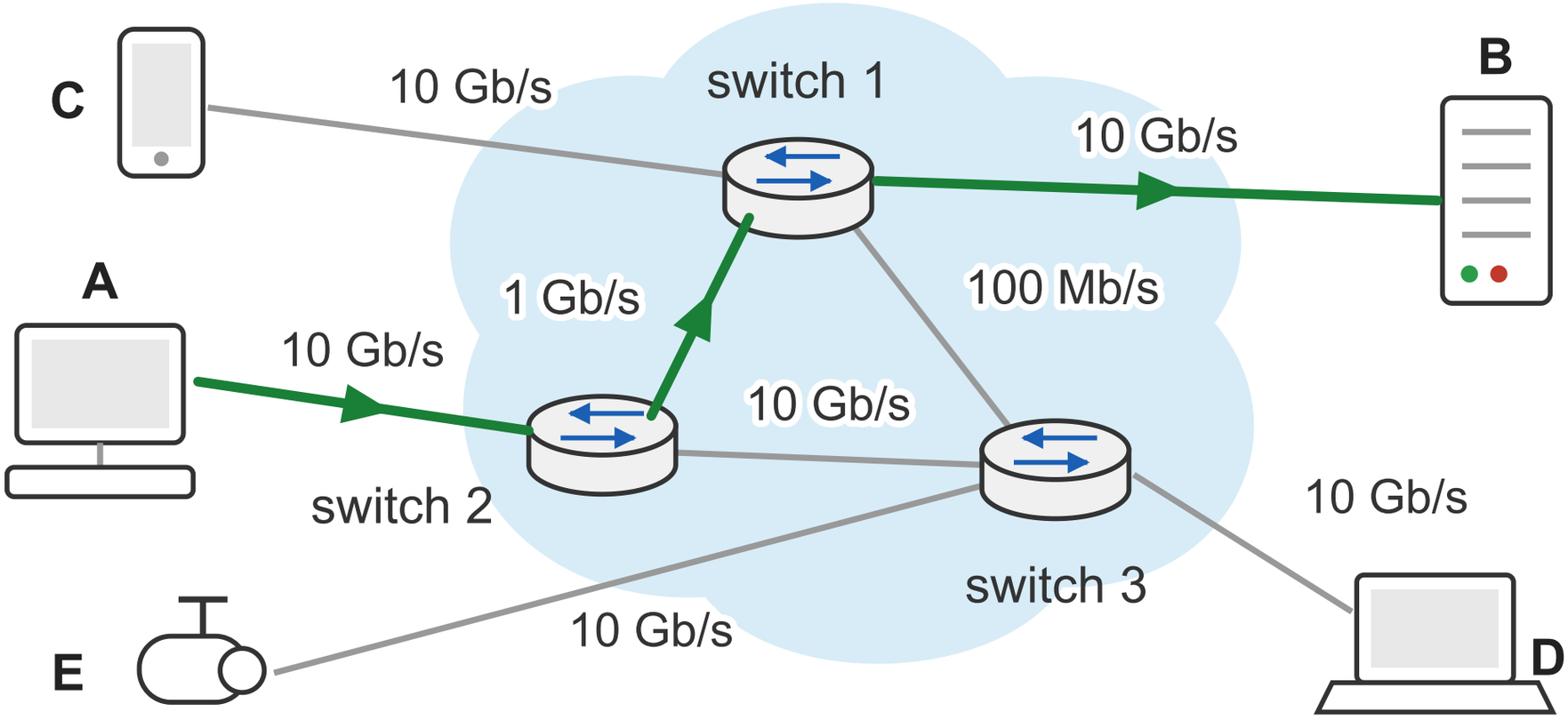
How fast should the sender send?



Consider a sender and a receiver:

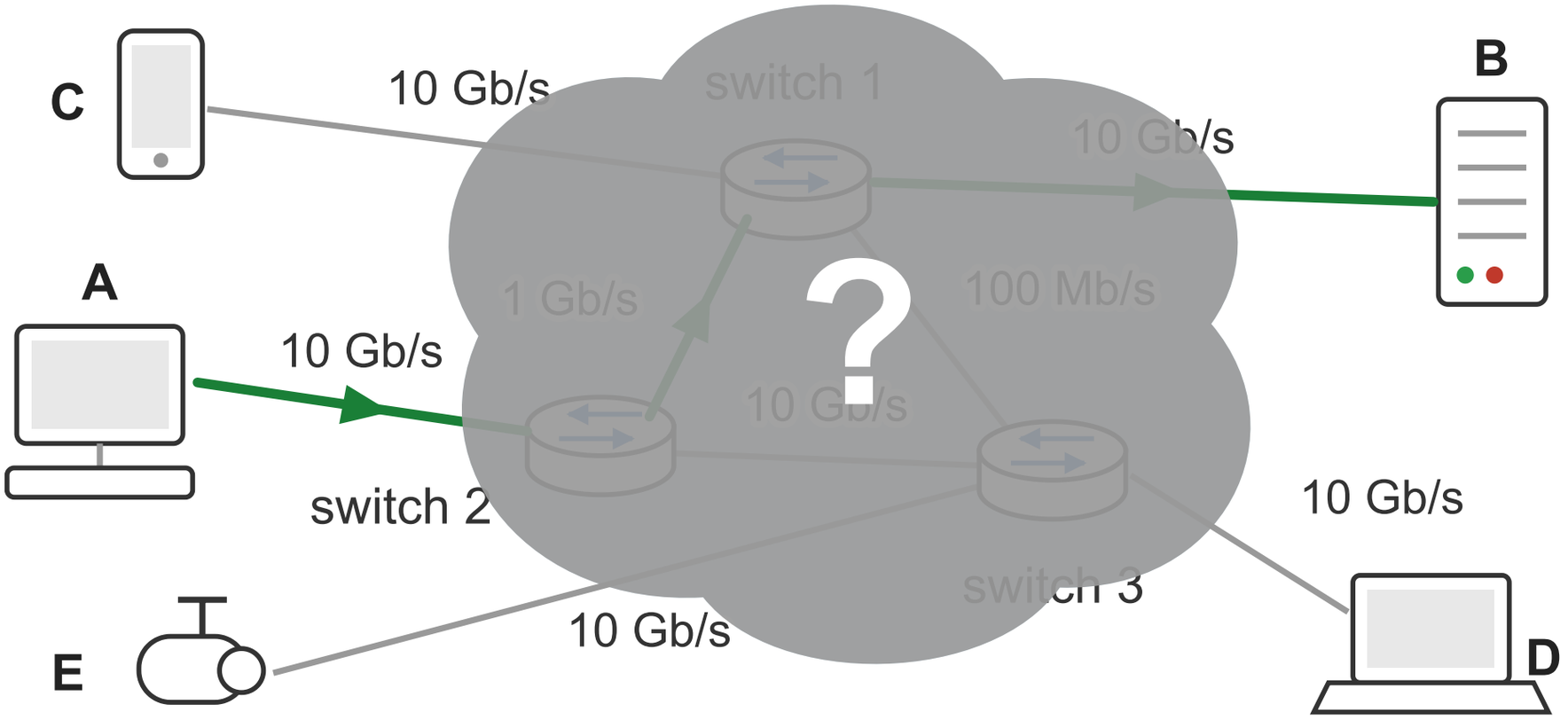
- **The sender knows exactly what path its traffic will take to the destination**, and the properties (rate, propagation delay, etc.) of each link
- The sender always has more data to send

In reality



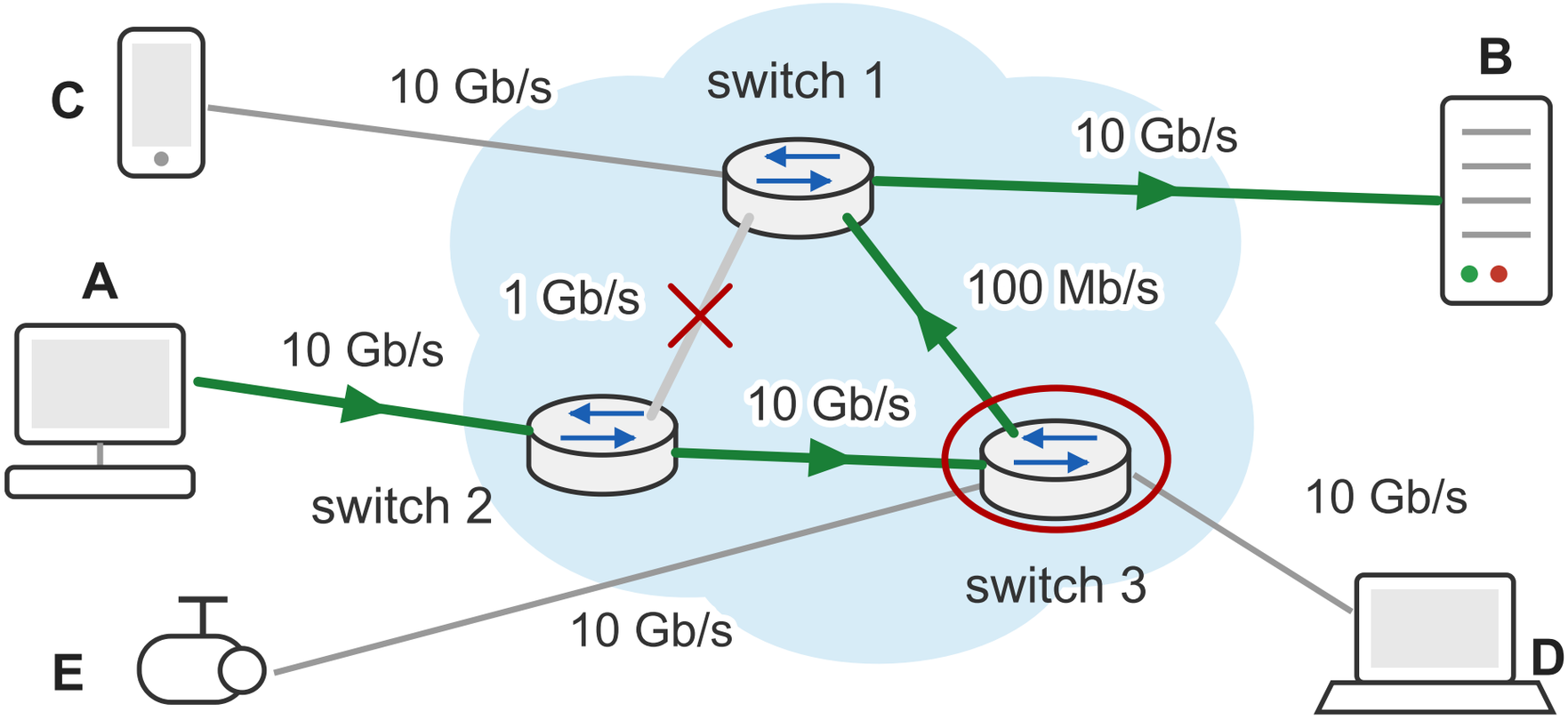
So far: A knew its path (green) and every link's rate

In reality



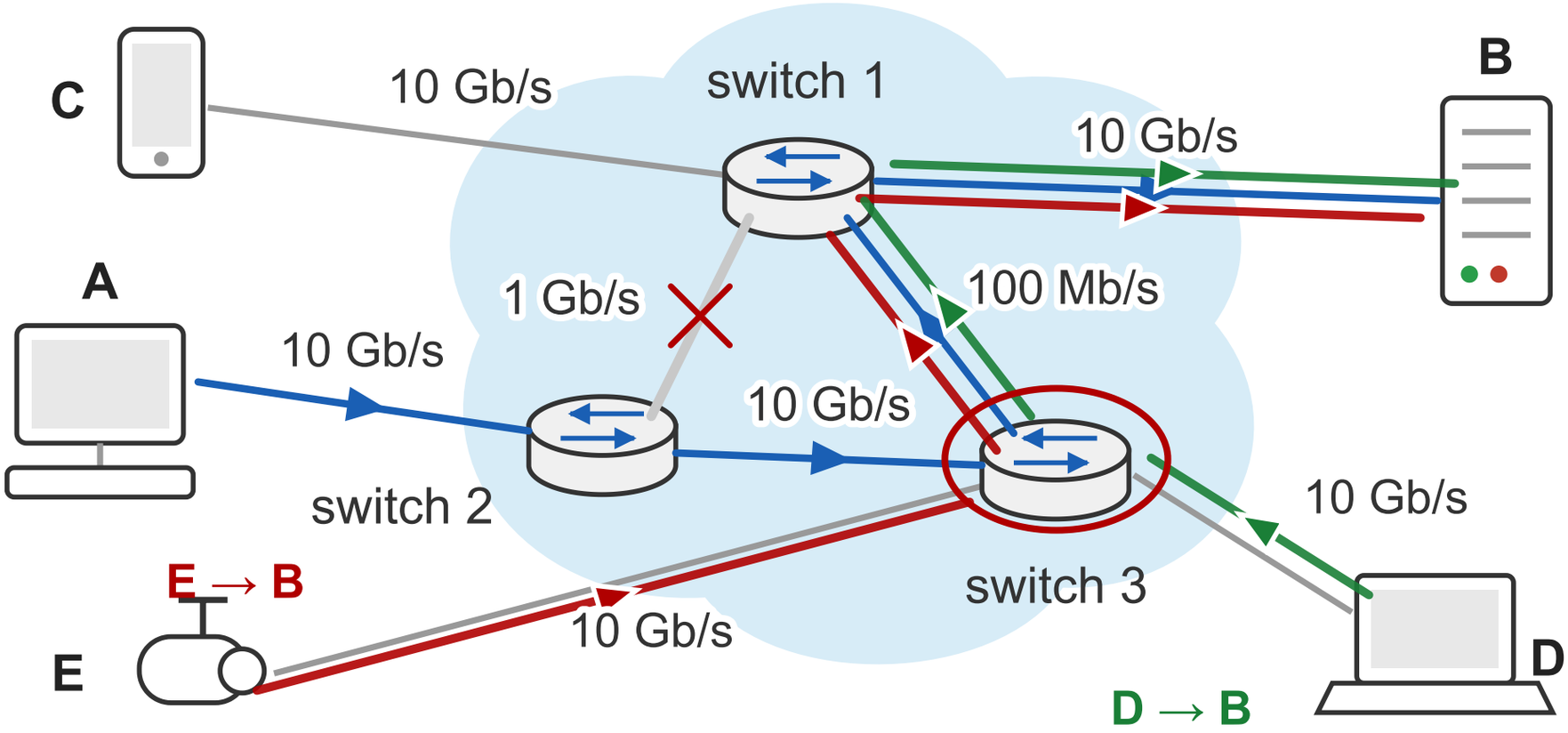
1. A does not know its path, or the rates on it

In reality



2. Even if A somehow found out once, that path might change (link failures, traffic changes, …)

In reality



3. Other flows may start sharing the bottleneck, and reduce A's share

How would you find the right share? (in-class)

(in-class discussion)

- You are a sender communicating with a receiver. You don’t get any extra information from the network or other senders and receivers that share the network with you.
- How can you figure out your “right” share, and notice when it changes?

Reset

Congestion control

Congestion control

- Congestion control algorithms use control packets and measurements to decide when to send more data into the network:

Congestion control

- Congestion control algorithms use control packets and measurements to decide when to send more data into the network:
  - Typically part of the transport layer (Why?)

Congestion control

- Congestion control algorithms use control packets and measurements to decide when to send more data into the network:
  - Typically part of the transport layer (Why?)
- This is a hard problem:

Congestion control

- Congestion control algorithms use control packets and measurements to decide when to send more data into the network:
  - Typically part of the transport layer (Why?)
- This is a hard problem:
  - Keep the bottleneck busy

Congestion control

- Congestion control algorithms use control packets and measurements to decide when to send more data into the network:
  - Typically part of the transport layer (Why?)
- This is a hard problem:
  - Keep the bottleneck busy
  - Keep the queues short

Congestion control

- Congestion control algorithms use control packets and measurements to decide when to send more data into the network:
  - Typically part of the transport layer (Why?)
- This is a hard problem:
  - Keep the bottleneck busy
  - Keep the queues short
  - Share fairly with other flows

Congestion control

- Congestion control algorithms use control packets and measurements to decide when to send more data into the network:
  - Typically part of the transport layer (Why?)
- This is a hard problem:
  - Keep the bottleneck busy
  - Keep the queues short
  - Share fairly with other flows
  - React quickly when things change

Congestion control

- Congestion control algorithms use control packets and measurements to decide when to send more data into the network:
  - Typically part of the transport layer (Why?)
- This is a hard problem:
  - Keep the bottleneck busy
  - Keep the queues short
  - Share fairly with other flows
  - React quickly when things change
  - Without much direct input from the network

Congestion control

- Congestion control algorithms use control packets and measurements to decide when to send more data into the network:
  - Typically part of the transport layer (Why?)
- This is a hard problem:
  - Keep the bottleneck busy
  - Keep the queues short
  - Share fairly with other flows
  - React quickly when things change
  - Without much direct input from the network
  - Without coordination with other flows

Congestion control

- Congestion control algorithms use control packets and measurements to decide when to send more data into the network:
  - Typically part of the transport layer (Why?)
- This is a hard problem:
  - Keep the bottleneck busy
  - Keep the queues short
  - Share fairly with other flows
  - React quickly when things change
  - Without much direct input from the network
  - Without coordination with other flows
  - …

Congestion control

- Congestion control algorithms use control packets and measurements to decide when to send more data into the network:
  - Typically part of the transport layer (Why?)
- This is a hard problem:
  - Keep the bottleneck busy
  - Keep the queues short
  - Share fairly with other flows
  - React quickly when things change
  - Without much direct input from the network
  - Without coordination with other flows
  - …
- Four decades of research

Congestion control

- Congestion control algorithms use control packets and measurements to decide when to send more data into the network:
  - Typically part of the transport layer (Why?)
- This is a hard problem:
  - Keep the bottleneck busy
  - Keep the queues short
  - Share fairly with other flows
  - React quickly when things change
  - Without much direct input from the network
  - Without coordination with other flows
  - …
- Four decades of research
- We'll see some example congestion control algorithms in the next lecture