

The transport layer: Dealing with performance problems

CS 456/656, Fall 2026

Mina Tahmasbi Arashloo

What we have so far

- **One packet at a time:** the sender moves on to the next packet only once it is sure the current one has made it correctly to the receiver
- **Feedback for every packet:** the receiver sends feedback for every packet, positive if the packet was fine, negative if it detects the packet was corrupted using checksums
- **Timers:** the sender uses a timer to detect loss and retransmit. A timer may trigger and cause a retransmission even if the packet is not lost, e.g., if the packet is delayed longer than expected
- **Duplicates:** an unnecessary retransmission can deliver a segment more than once; the receiver uses the segment ID to deal with that

What we have so far

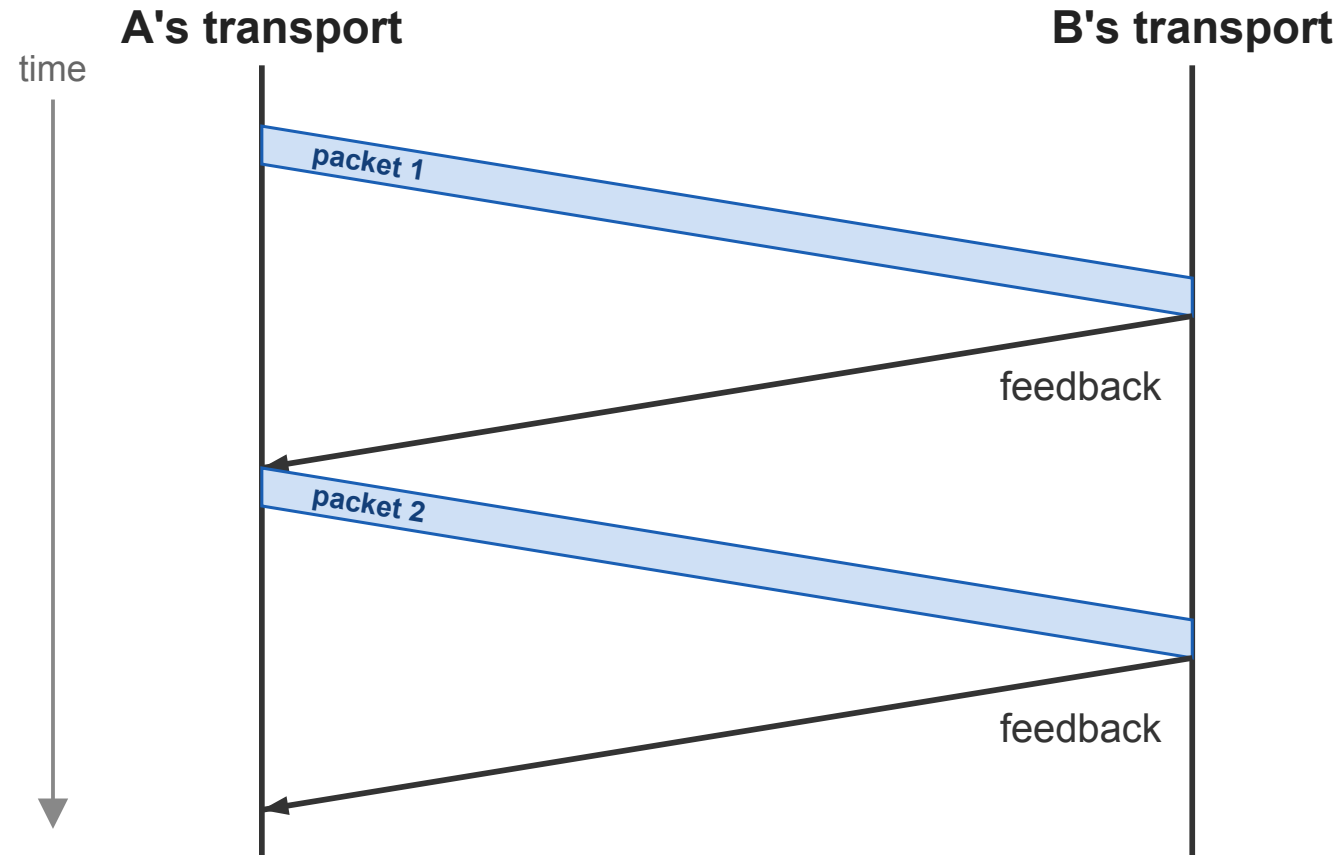
- **One packet at a time:** the sender moves on to the next packet only once it is sure the current one has made it correctly to the receiver
- **Feedback for every packet:** the receiver sends feedback for every packet, positive if the packet was fine, negative if it detects the packet was corrupted using checksums
- **Timers:** the sender uses a timer to detect loss and retransmit. A timer may trigger and cause a retransmission even if the packet is not lost, e.g., if the packet is delayed longer than expected
- **Duplicates:** an unnecessary retransmission can deliver a segment more than once; the receiver uses the segment ID to deal with that

Quick check: what are three scenarios in which an unnecessary retransmission happens?

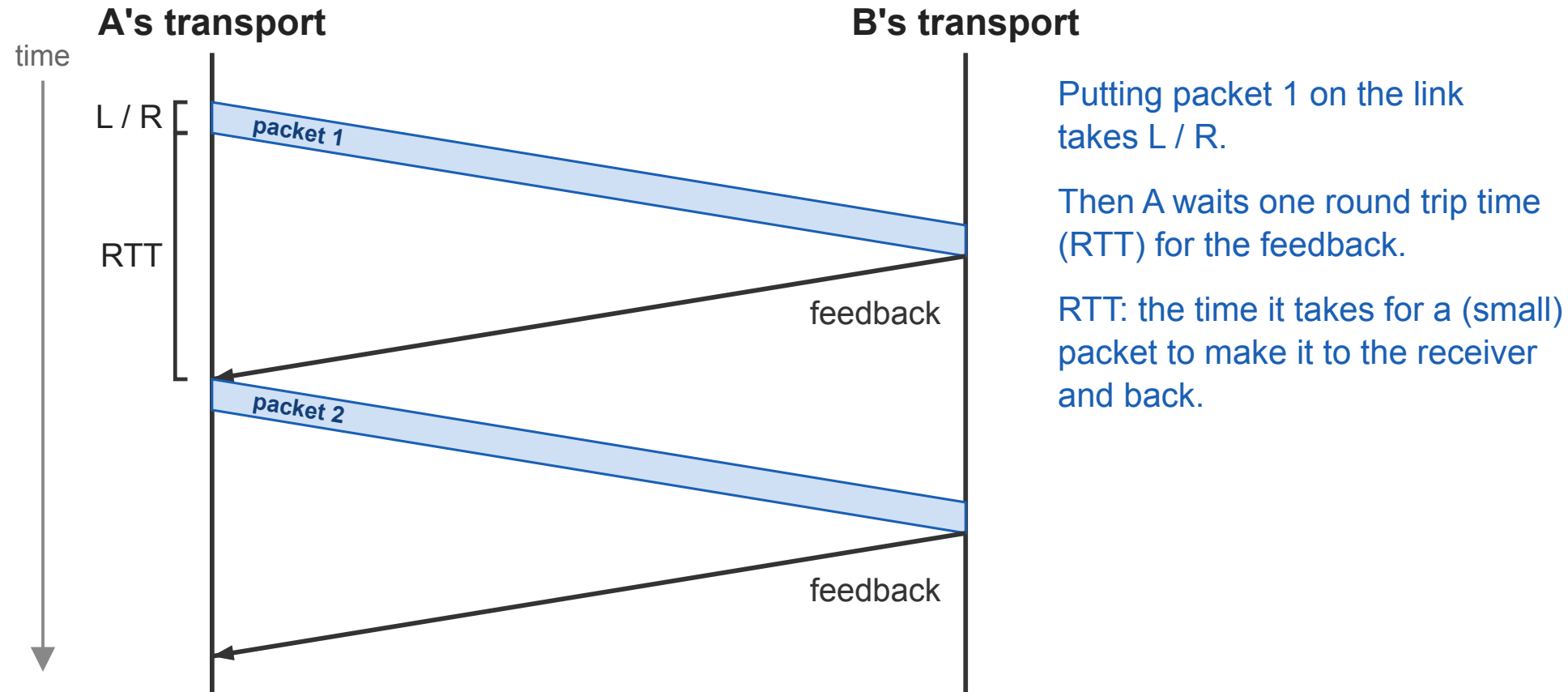
Is one packet at a time efficient enough?

The protocol we have gets every segment to B. But how efficiently does it do that?

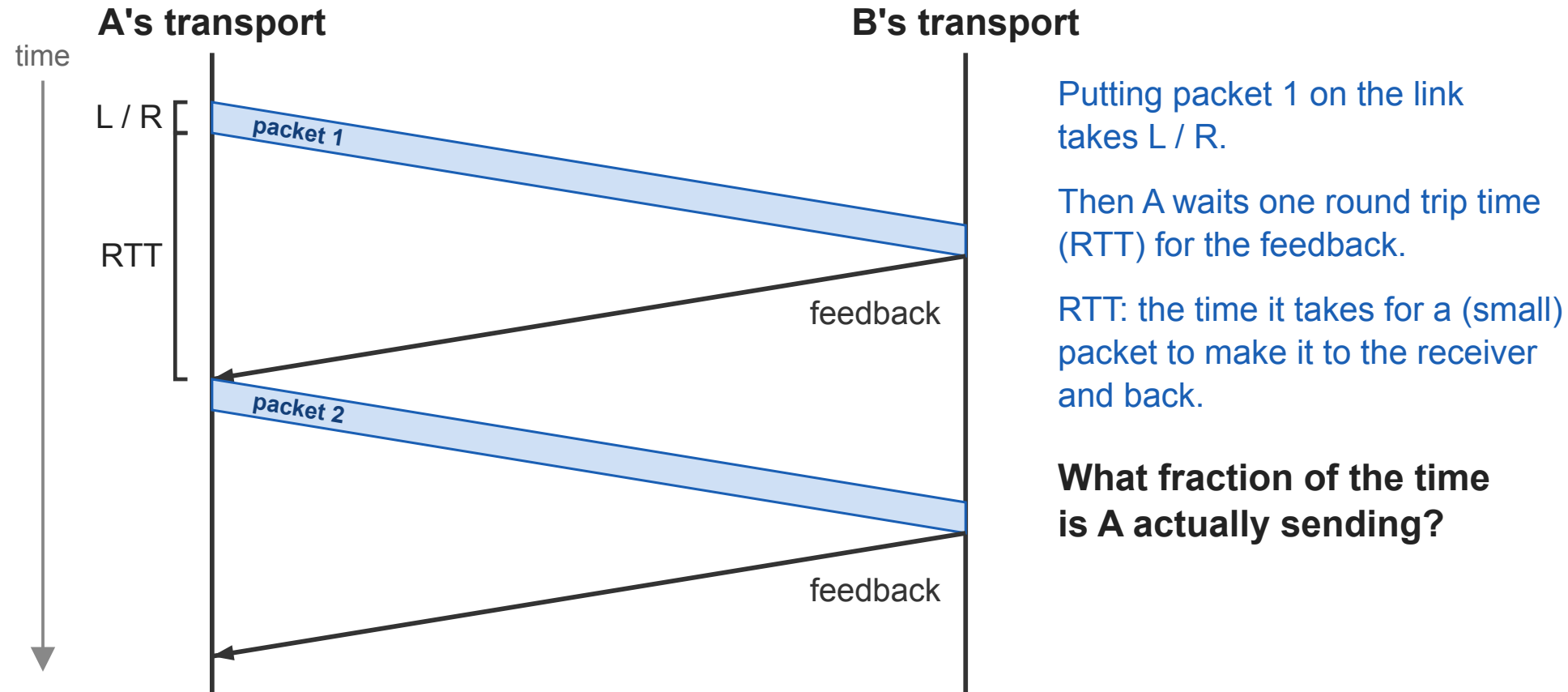
One packet at a time, on a timeline



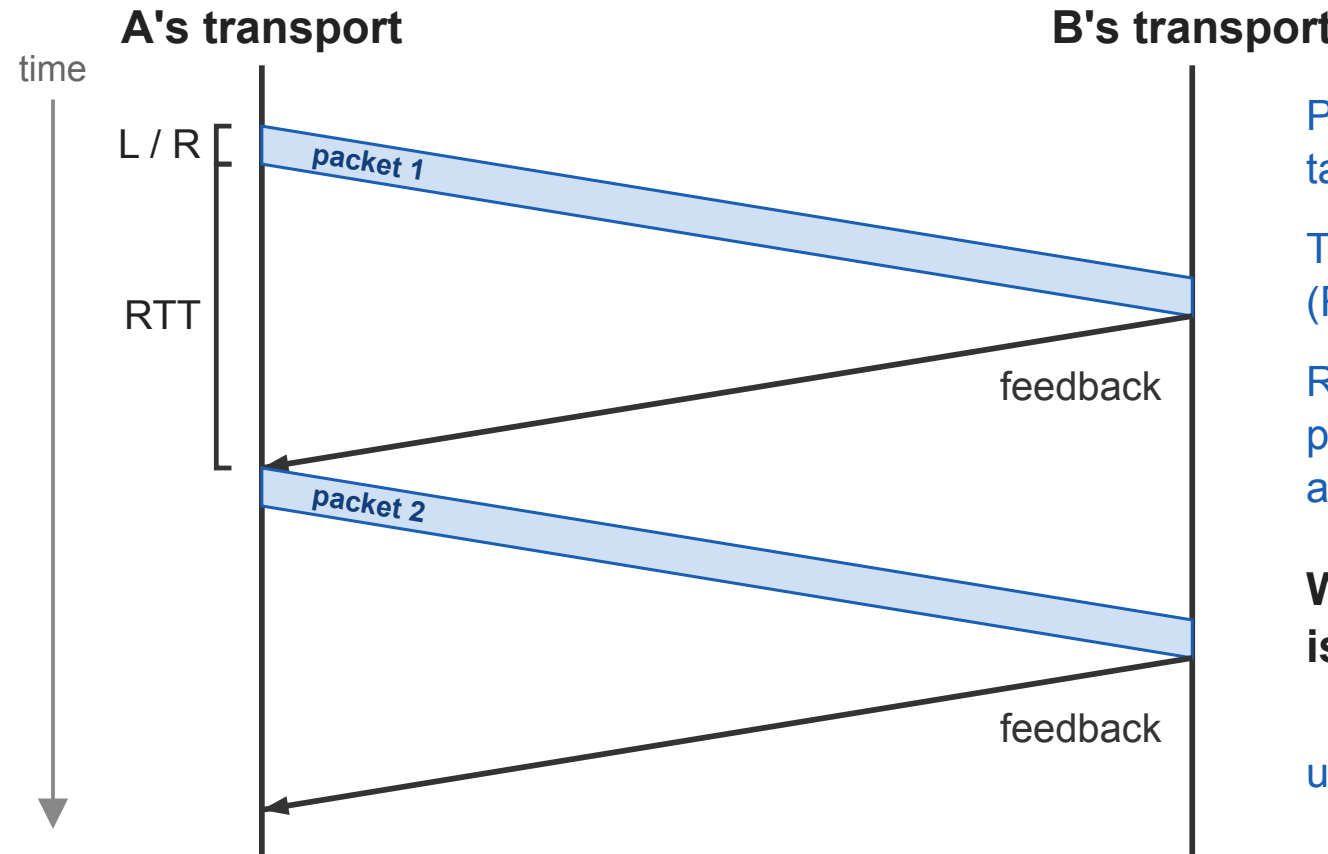
One packet at a time, on a timeline



One packet at a time, on a timeline



One packet at a time, on a timeline



Putting packet 1 on the link takes L/R .

Then A waits one round trip time (RTT) for the feedback.

RTT: the time it takes for a (small) packet to make it to the receiver and back.

What fraction of the time is A actually sending?

$$\text{utilization} = \frac{L/R}{RTT + L/R}$$

How much of the link is A using?

A link of 1 Gbps (10^9 bits per second), 15 ms propagation delay, packets of 8000 bits

How much of the link is A using?

A link of 1 Gbps (10^9 bits per second), 15 ms propagation delay, packets of 8000 bits

putting one packet on the link

$$L / R = 8000 / 10^9 \text{ s} = 0.008 \text{ ms}$$

How much of the link is A using?

A link of 1 Gbps (10^9 bits per second), 15 ms propagation delay, packets of 8000 bits

putting one packet on the link

$$L / R = 8000 / 10^9 \text{ s} = 0.008 \text{ ms}$$

one round trip (assume no waiting delay)

$$\text{RTT} = 2 \times 15 \text{ ms} = 30 \text{ ms}$$

How much of the link is A using?

A link of 1 Gbps (10^9 bits per second), 15 ms propagation delay, packets of 8000 bits

putting one packet on the link

$$L / R = 8000 / 10^9 \text{ s} = 0.008 \text{ ms}$$

one round trip (assume no waiting delay)

$$\text{RTT} = 2 \times 15 \text{ ms} = 30 \text{ ms}$$

fraction of the time A is sending

$$0.008 / (30 + 0.008) = 0.00027$$

How much of the link is A using?

A link of 1 Gbps (10^9 bits per second), 15 ms propagation delay, packets of 8000 bits

putting one packet on the link

$$L / R = 8000 / 10^9 \text{ s} = 0.008 \text{ ms}$$

one round trip (assume no waiting delay)

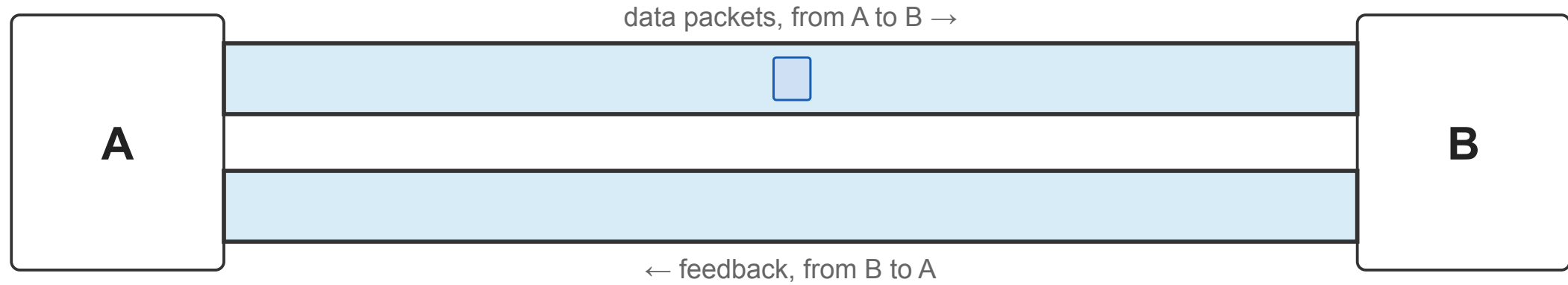
$$\text{RTT} = 2 \times 15 \text{ ms} = 30 \text{ ms}$$

fraction of the time A is sending

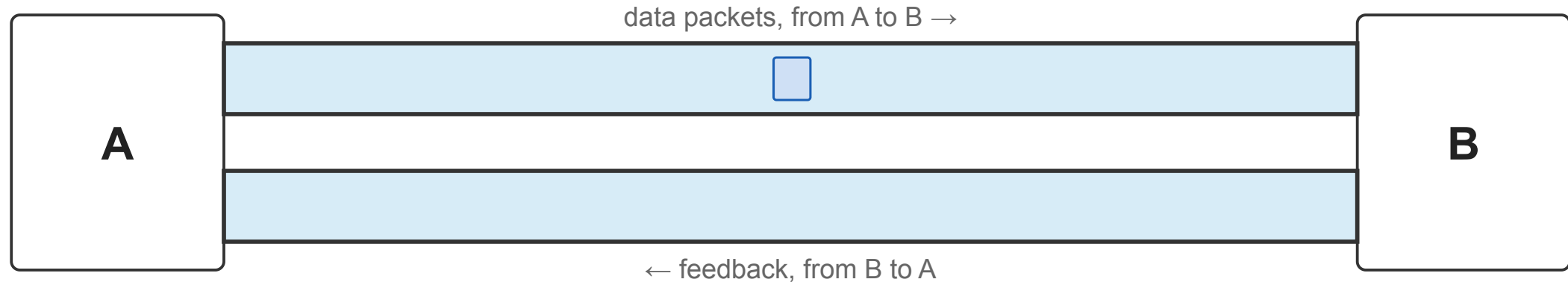
$$0.008 / (30 + 0.008) = 0.00027$$

**The link can carry 10^9 bits per second.
A is only sending about 267K bits per second.**

The link is empty almost all the time

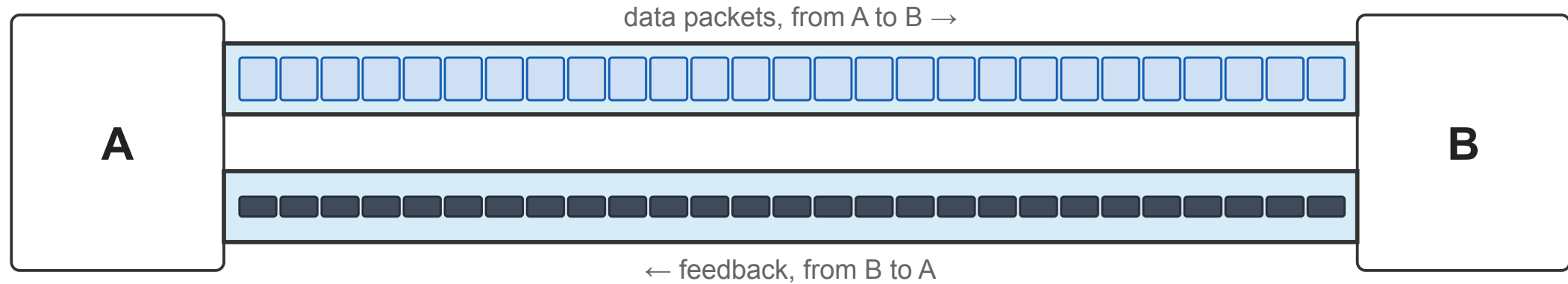


The link is empty almost all the time



How many packets could A send during one round trip?

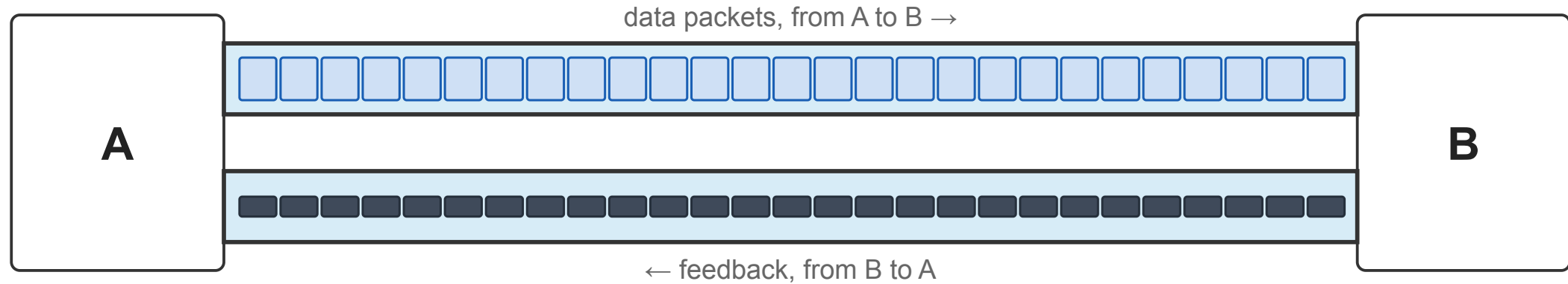
The link is empty almost all the time



How many packets could A send during one round trip?

$$10^9 \text{ bits per second} \times 0.030 \text{ s} = 30,000,000 \text{ bits} = 3750 \text{ packets}$$

The link is empty almost all the time

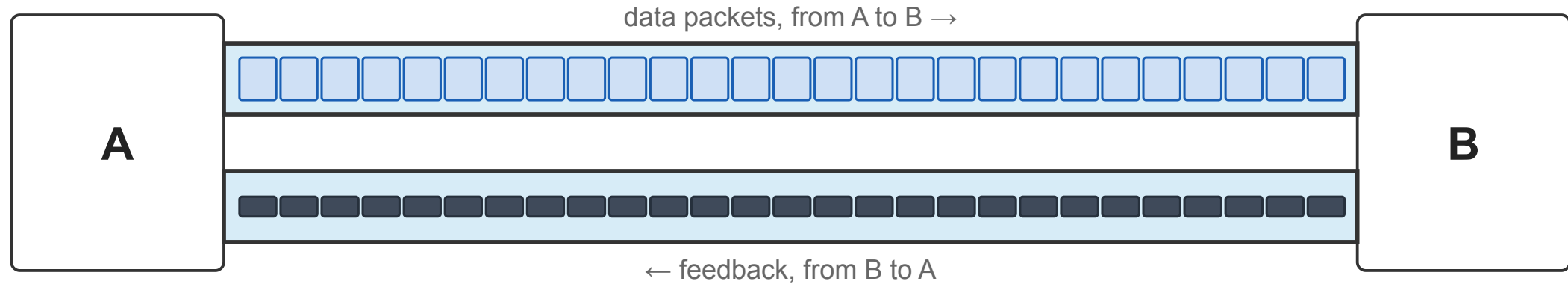


How many packets could A send during one round trip?

10^9 bits per second $\times$ 0.030 s = 30,000,000 bits = 3750 packets

1875 are on their way to B, and the feedback for another 1875 is on its way back

The link is empty almost all the time



How many packets could A send during one round trip?

10^9 bits per second $\times$ 0.030 s = 30,000,000 bits = 3750 packets

1875 are on their way to B, and the feedback for another 1875 is on its way back

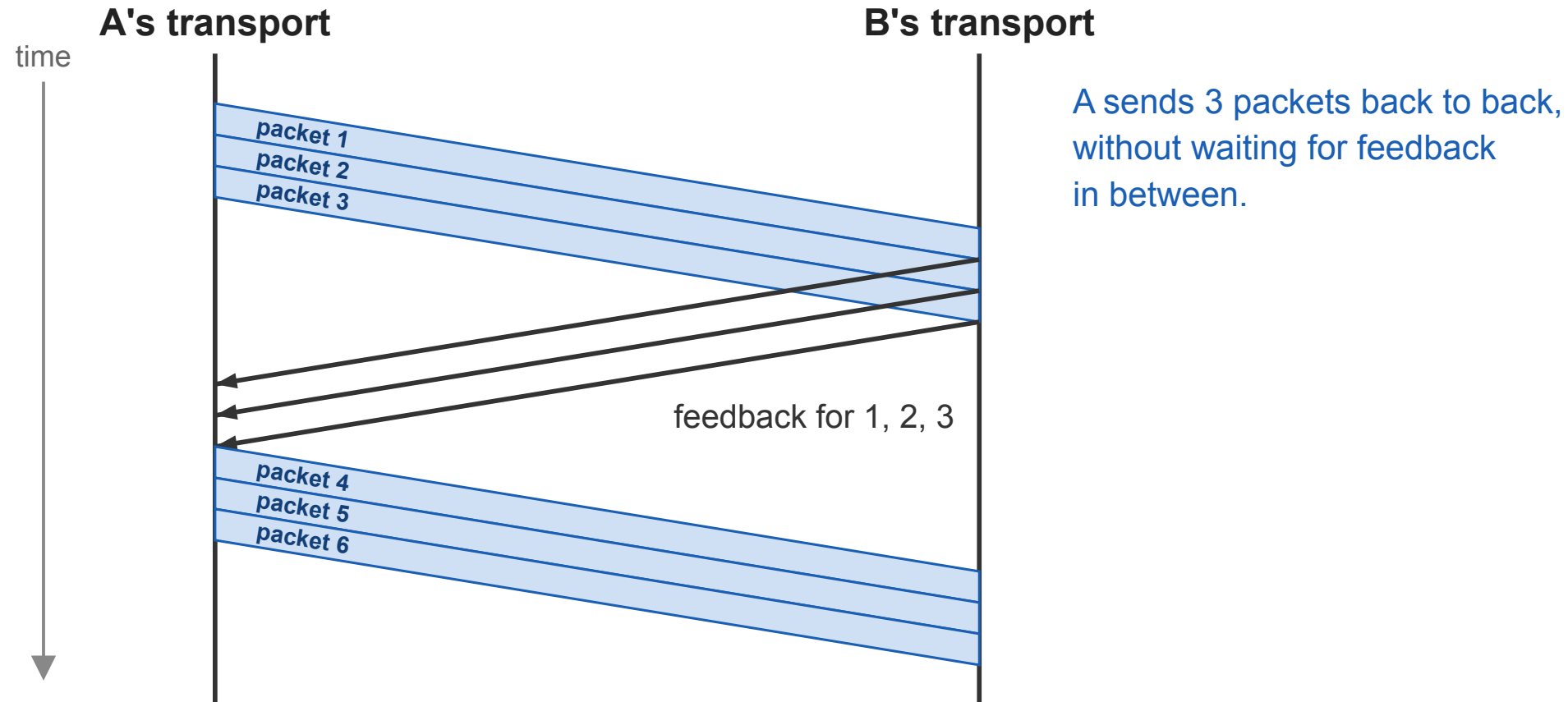
One packet at a time sends 1 of those 3750

What would you change?

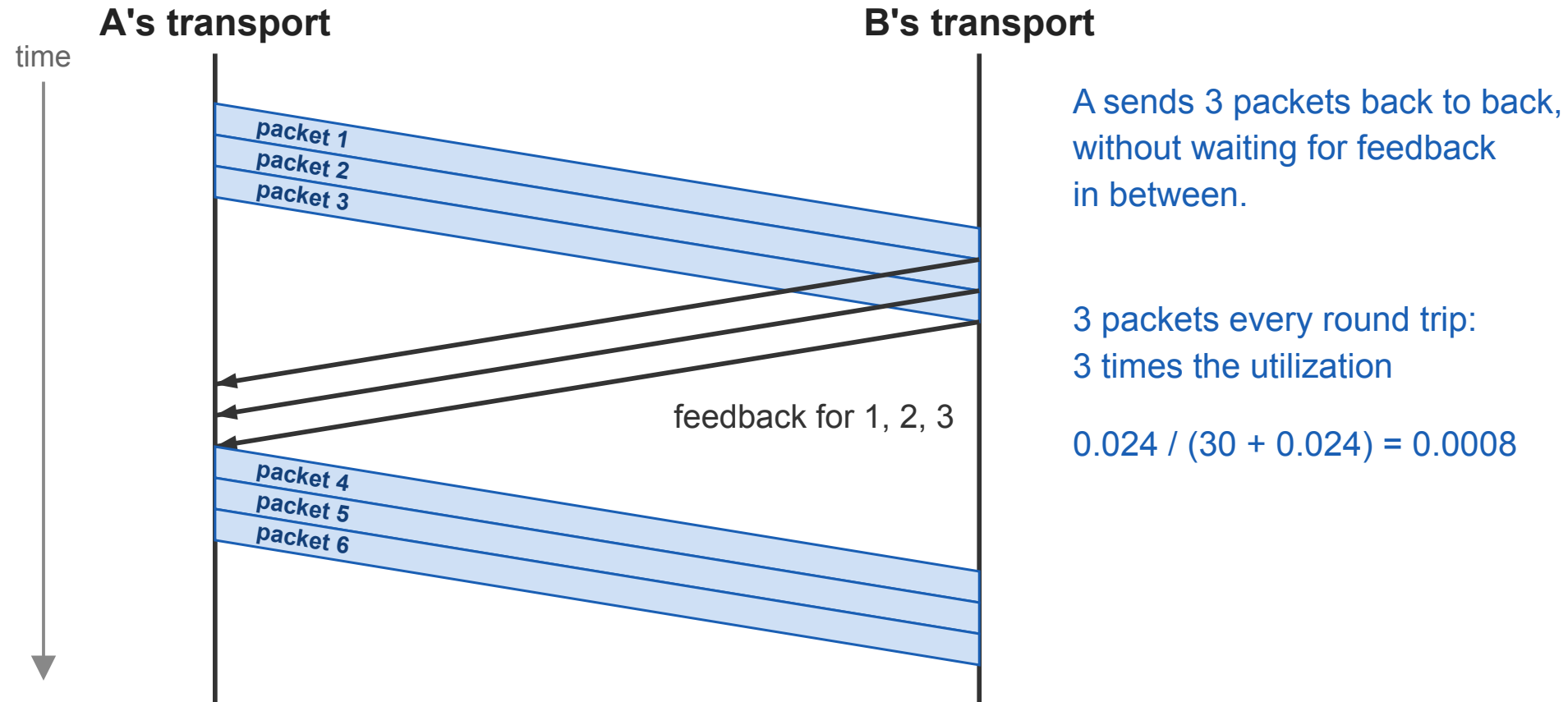
A large, empty rectangular box with a thin black border, intended for a user to write their response to the question "What would you change?".

Reset

Example: A batch of three packets at a time



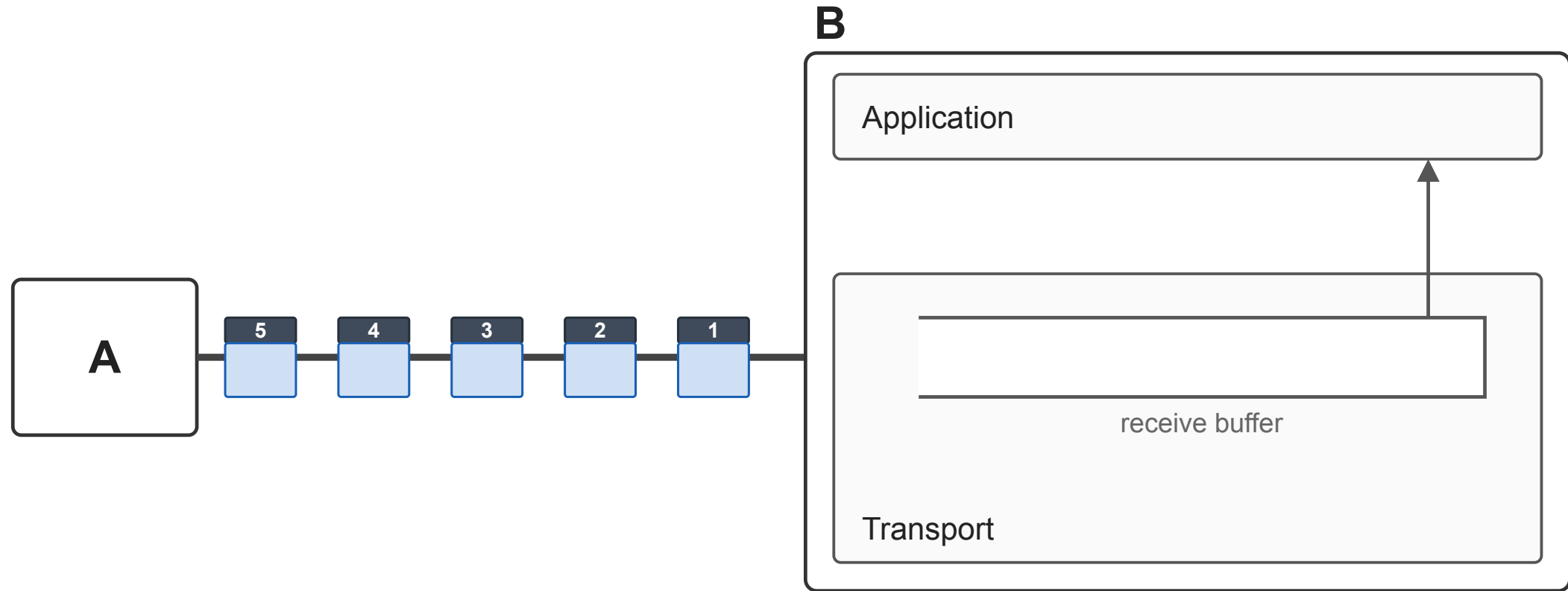
Example: A batch of three packets at a time



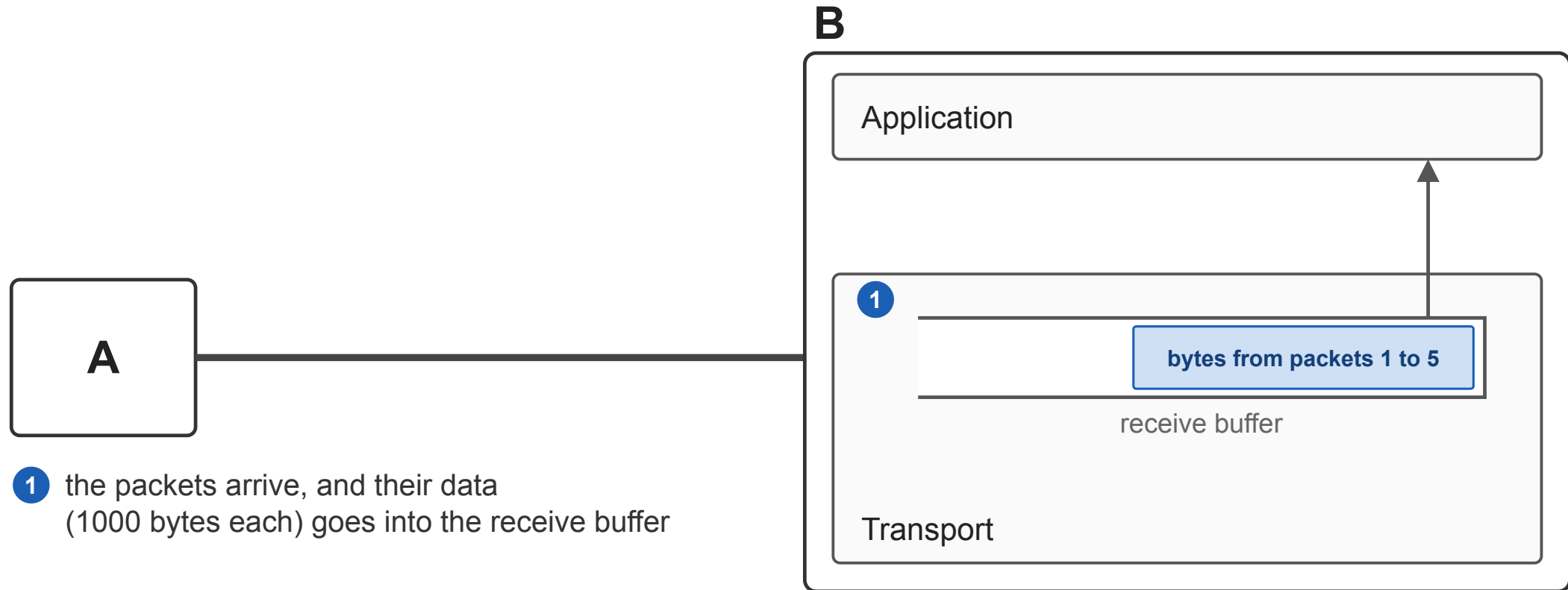
What is a good batch size?

- **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 (congestion control, Part 2)
 - The sender may “overwhelm” the receiver

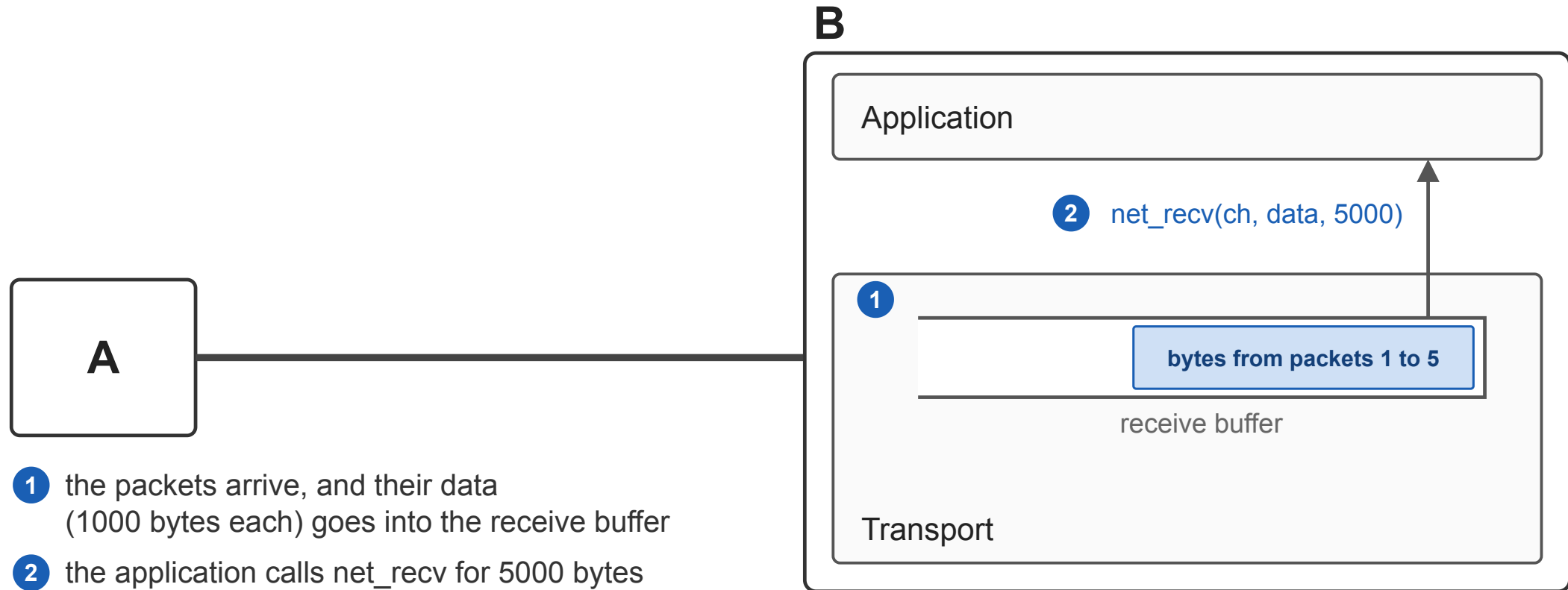
“Overwhelming” the receiver



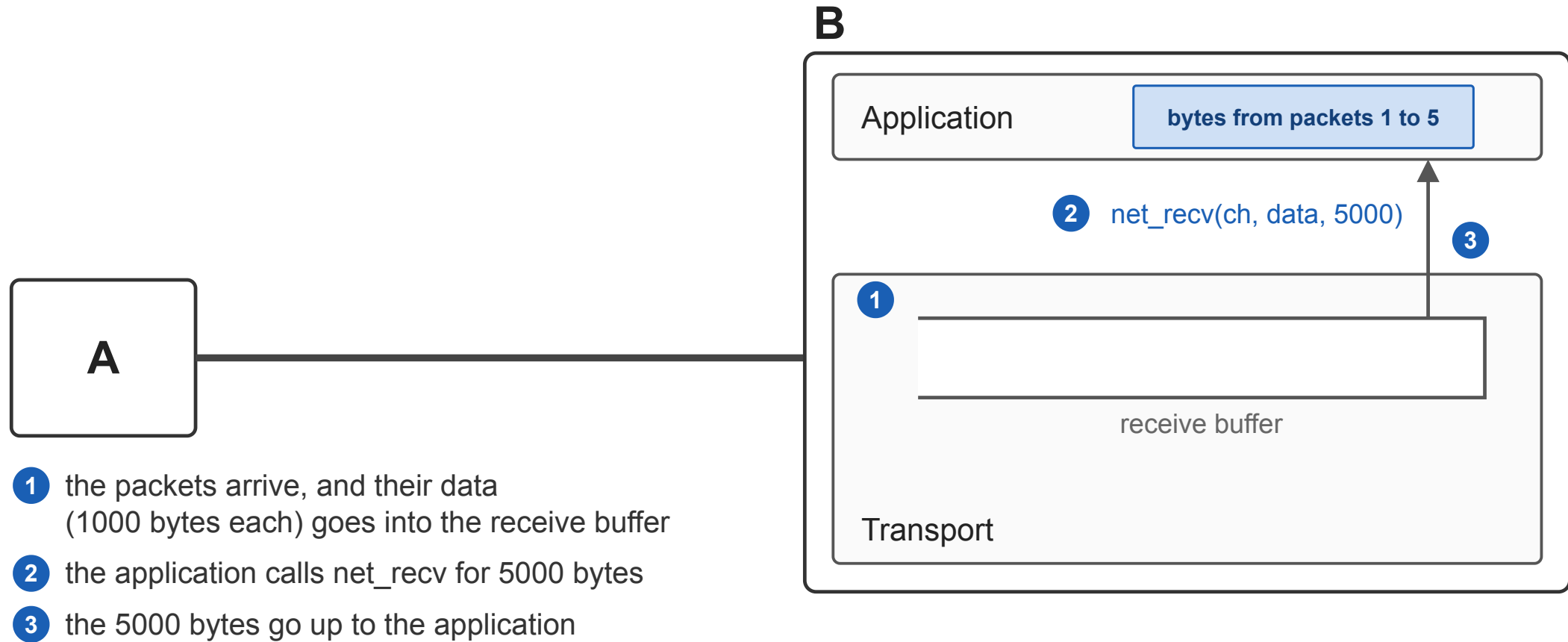
“Overwhelming” the receiver



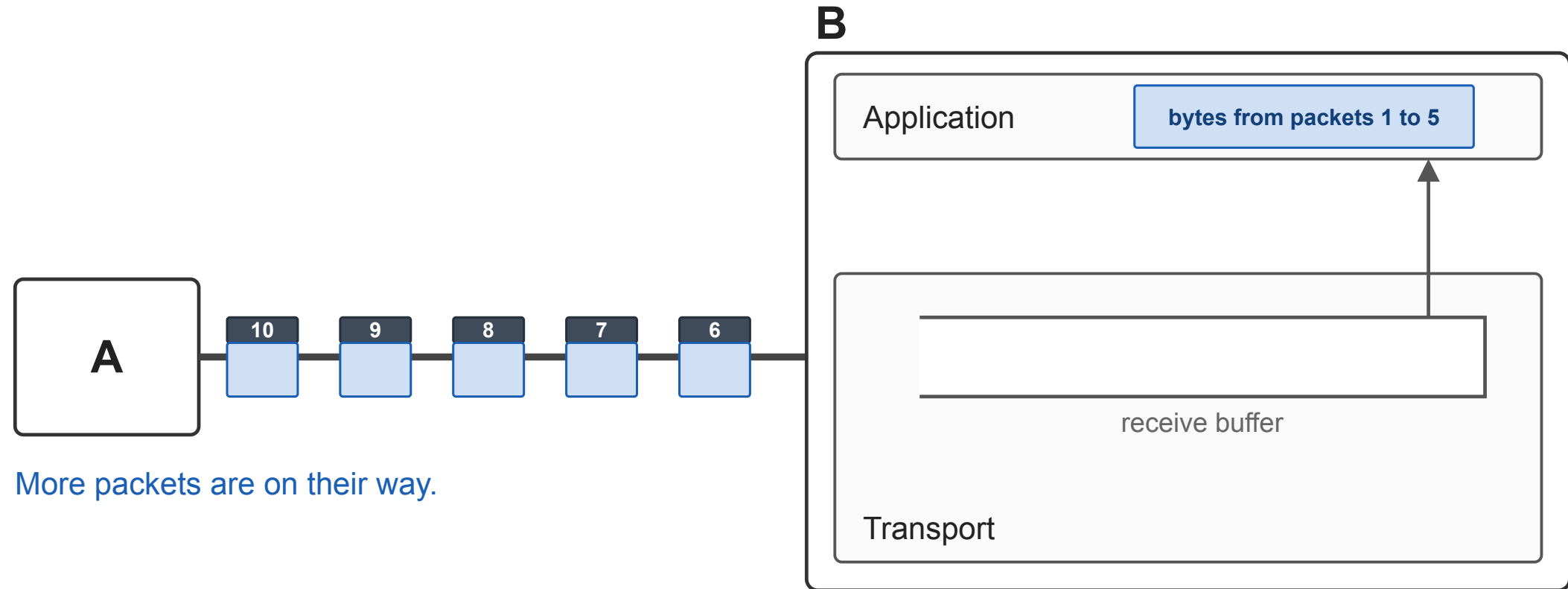
“Overwhelming” the receiver



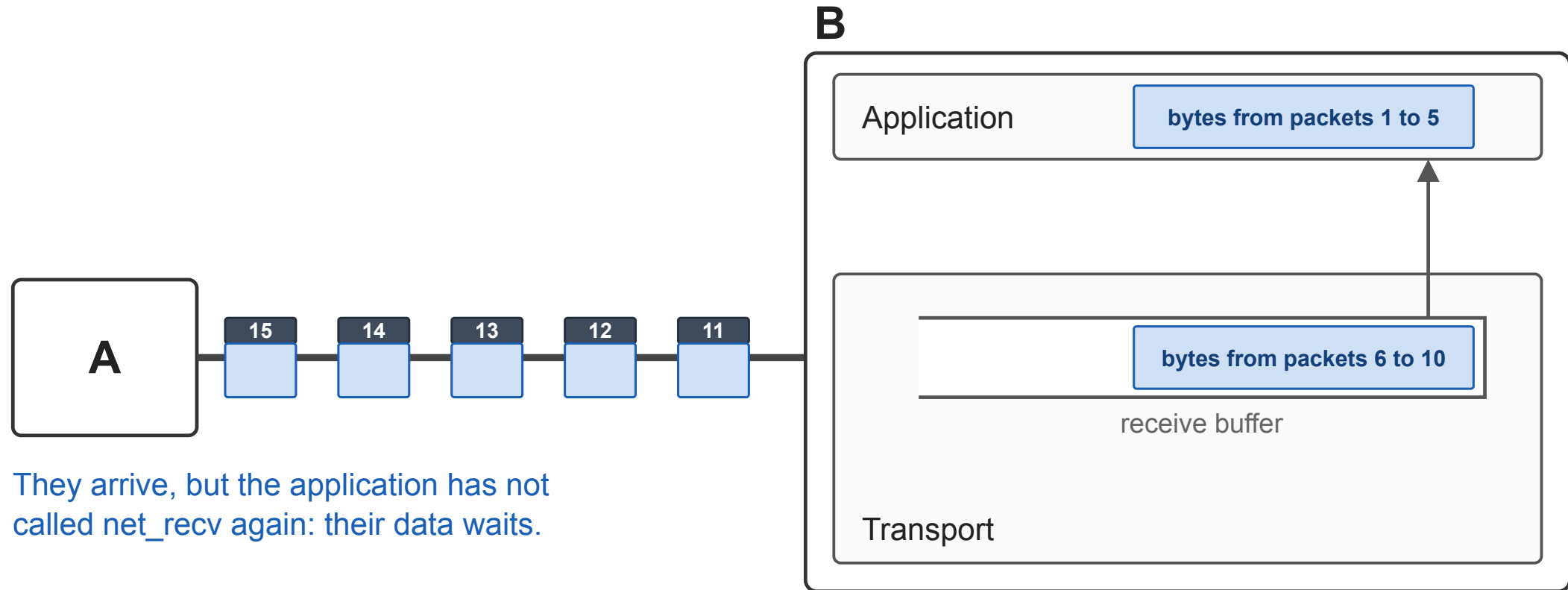
“Overwhelming” the receiver



“Overwhelming” the receiver

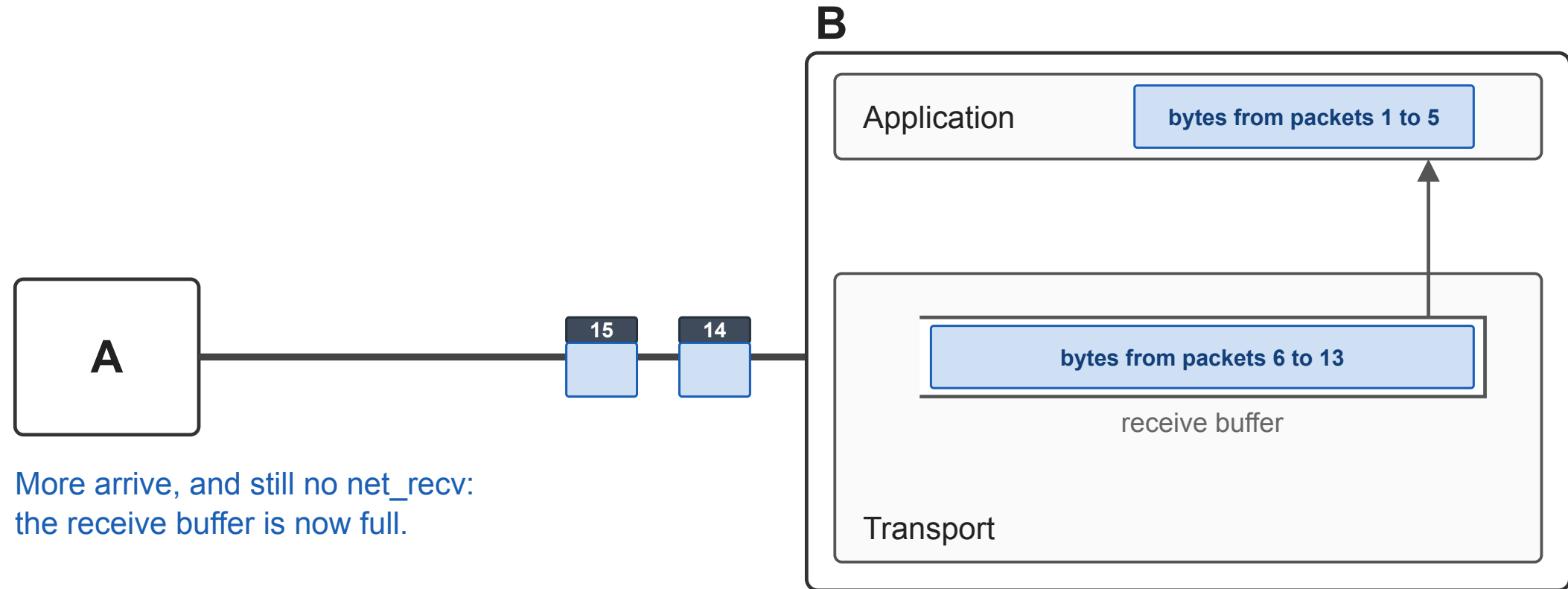


“Overwhelming” the receiver



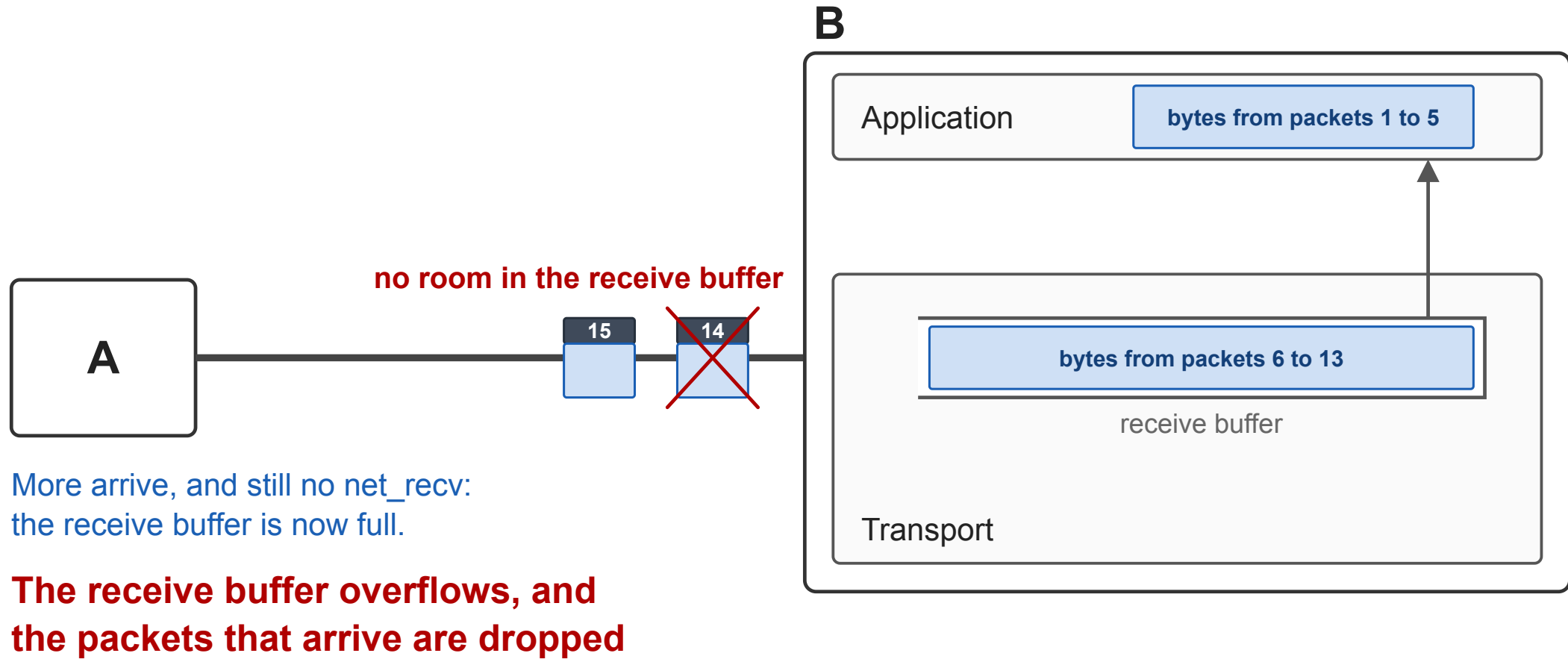
They arrive, but the application has not called `net_recv` again: their data waits.

“Overwhelming” the receiver



More arrive, and still no net_recv:
the receive buffer is now full.

“Overwhelming” the receiver

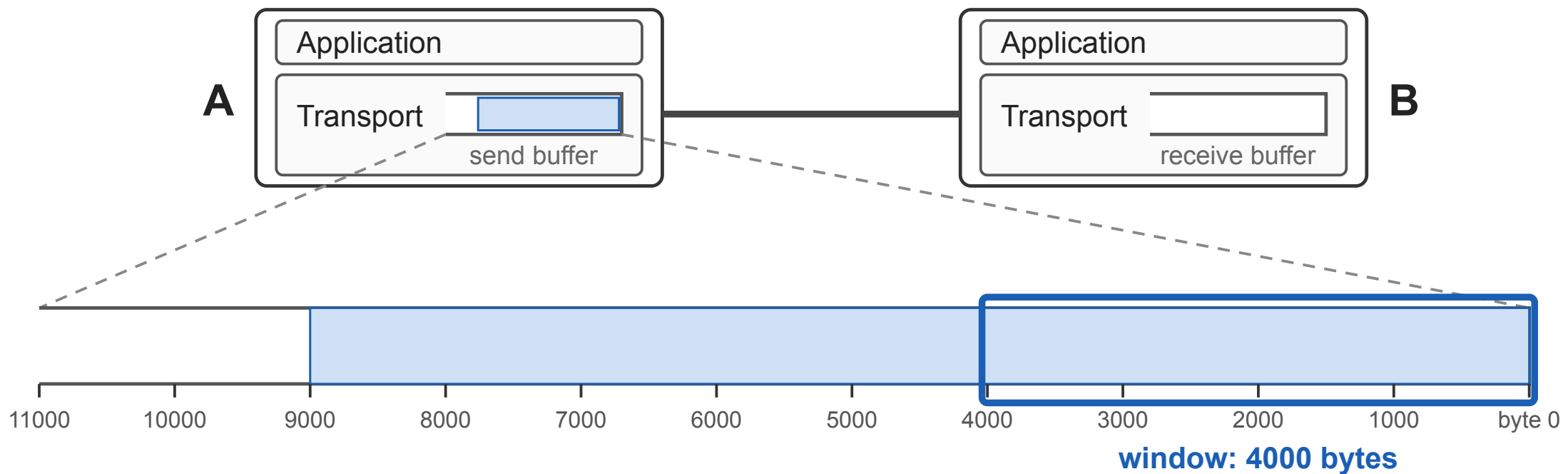


Flow control

Making sure the sender never sends more data than the receiver has room for.

One mechanism to implement flow control in the transport layer is using a sliding window.

A sliding window over the send buffer

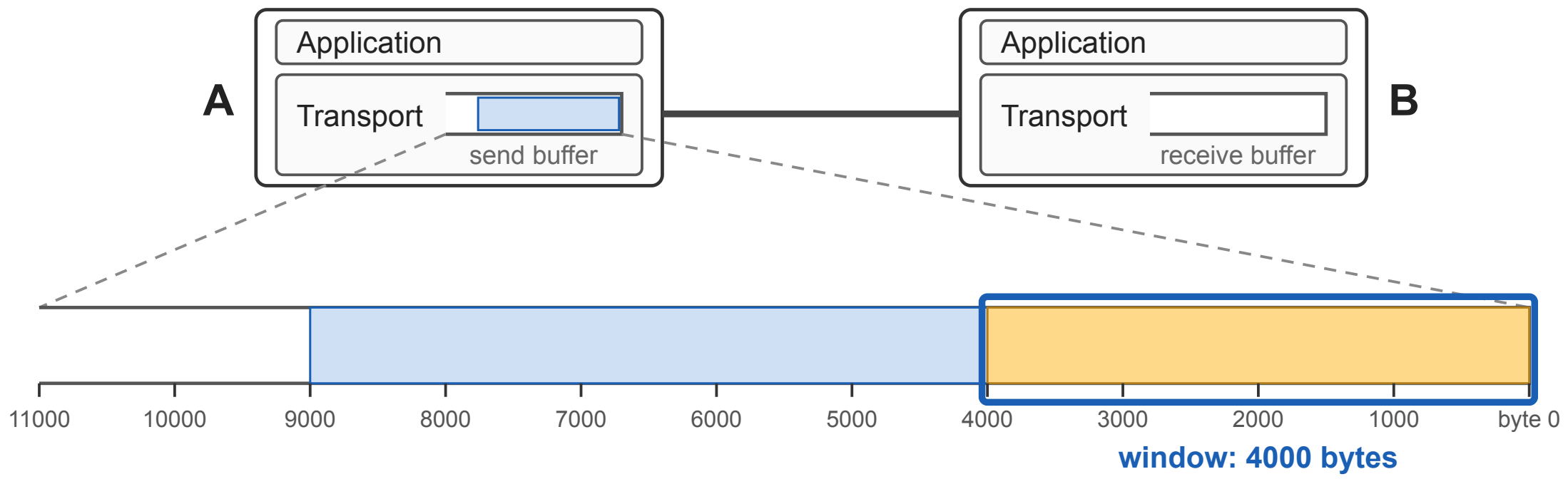


A only sends bytes that are inside the window. The window starts at the oldest byte without feedback.

The window covers the first 4000 bytes in the send buffer

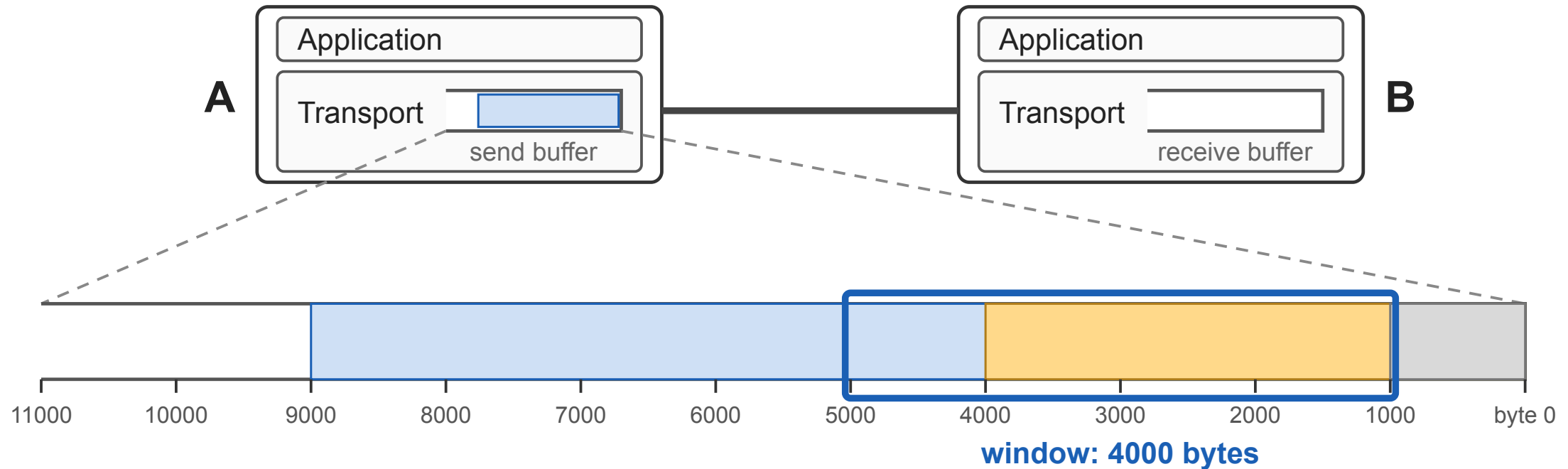
- successfully delivered
- sent, waiting for feedback
- not sent yet
- empty

A sliding window over the send buffer



A only sends bytes that are inside the window. The window starts at the oldest byte without feedback.
A sends the bytes in the window, as four packets of 1000 bytes, then has to wait

A sliding window over the send buffer

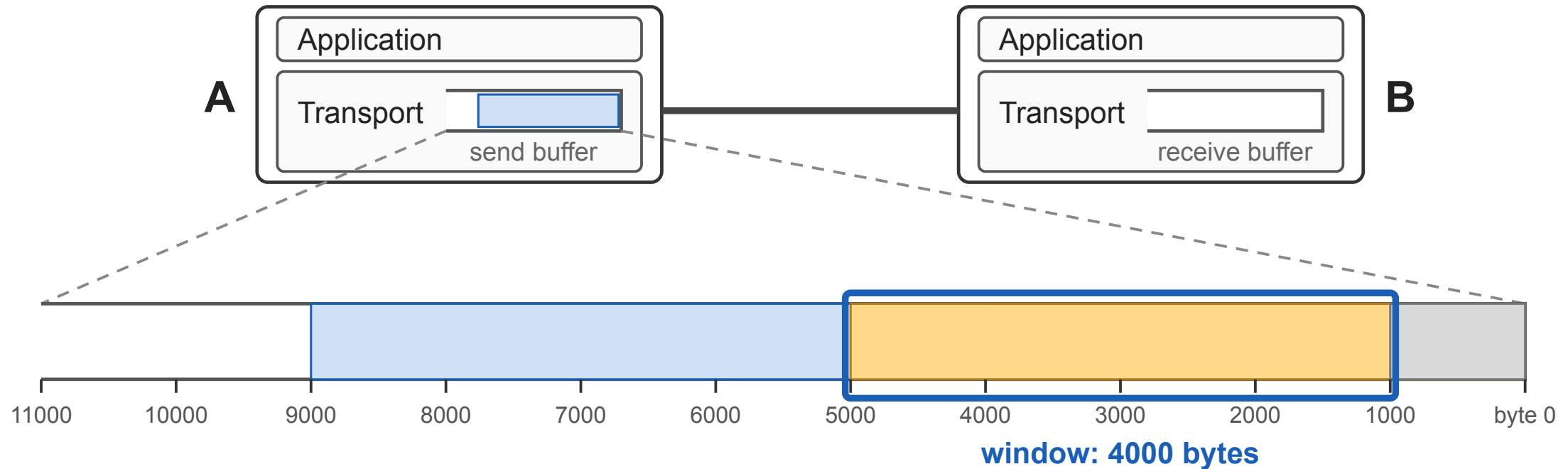


A only sends bytes that are inside the window. The window starts at the oldest byte without feedback.

Feedback for the first packet arrives: the window slides by 1000 bytes

 successfully delivered  sent, waiting for feedback  not sent yet  empty

A sliding window over the send buffer

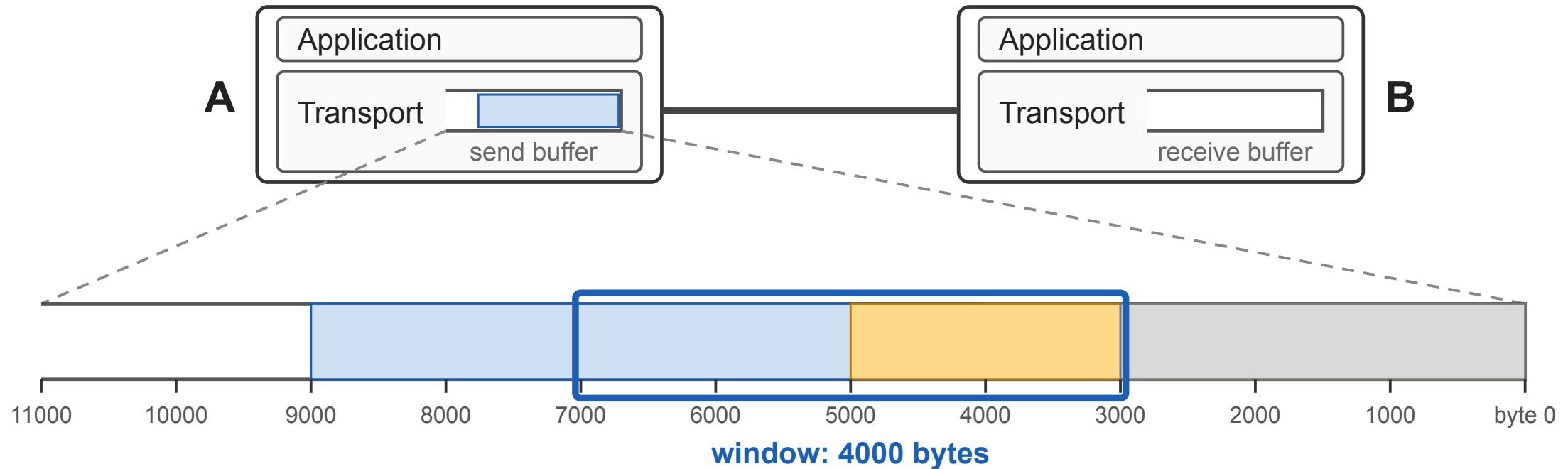


A only sends bytes that are inside the window. The window starts at the oldest byte without feedback.

Now A can send the next 1000 bytes

 successfully delivered  sent, waiting for feedback  not sent yet  empty

A sliding window over the send buffer

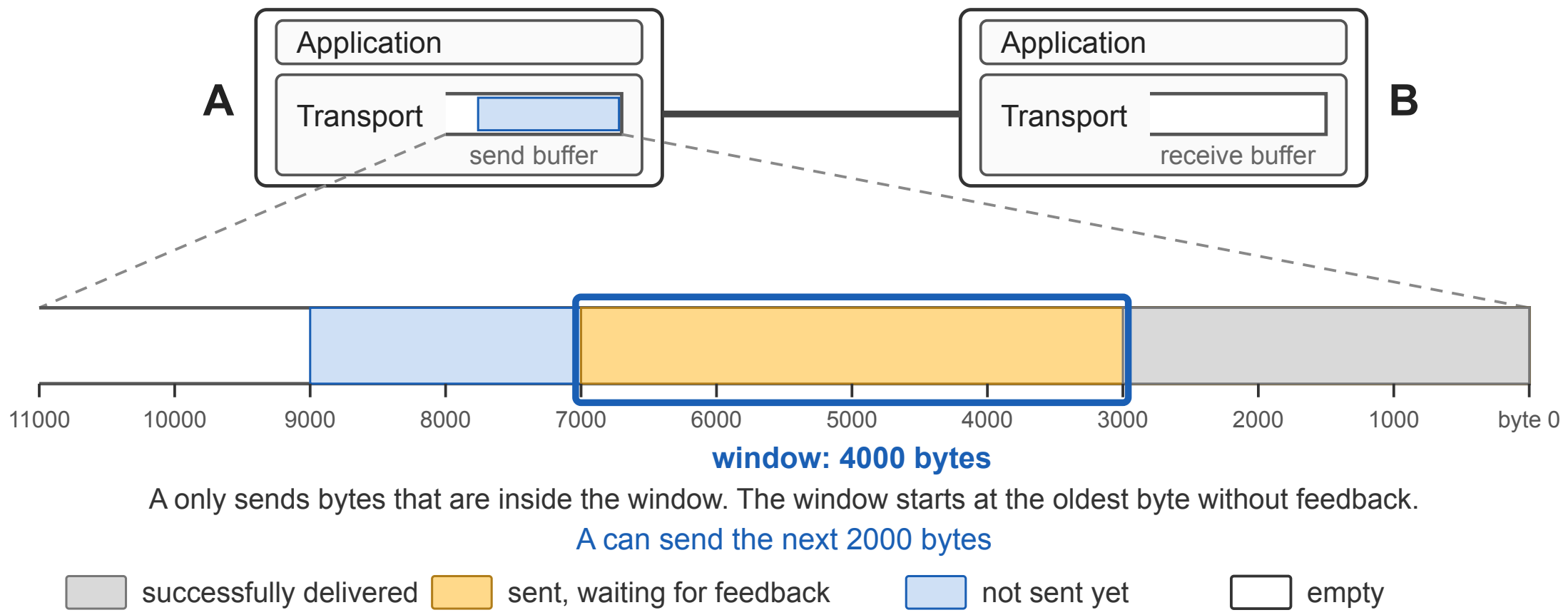


A only sends bytes that are inside the window. The window starts at the oldest byte without feedback.

Feedback for the next two packets arrives: the window slides by 2000 bytes

 successfully delivered  sent, waiting for feedback  not sent yet  empty

A sliding window over the send buffer



Flow control, with a window

A's window should never be larger than the room left in B's receive buffer.

Flow control, with a window

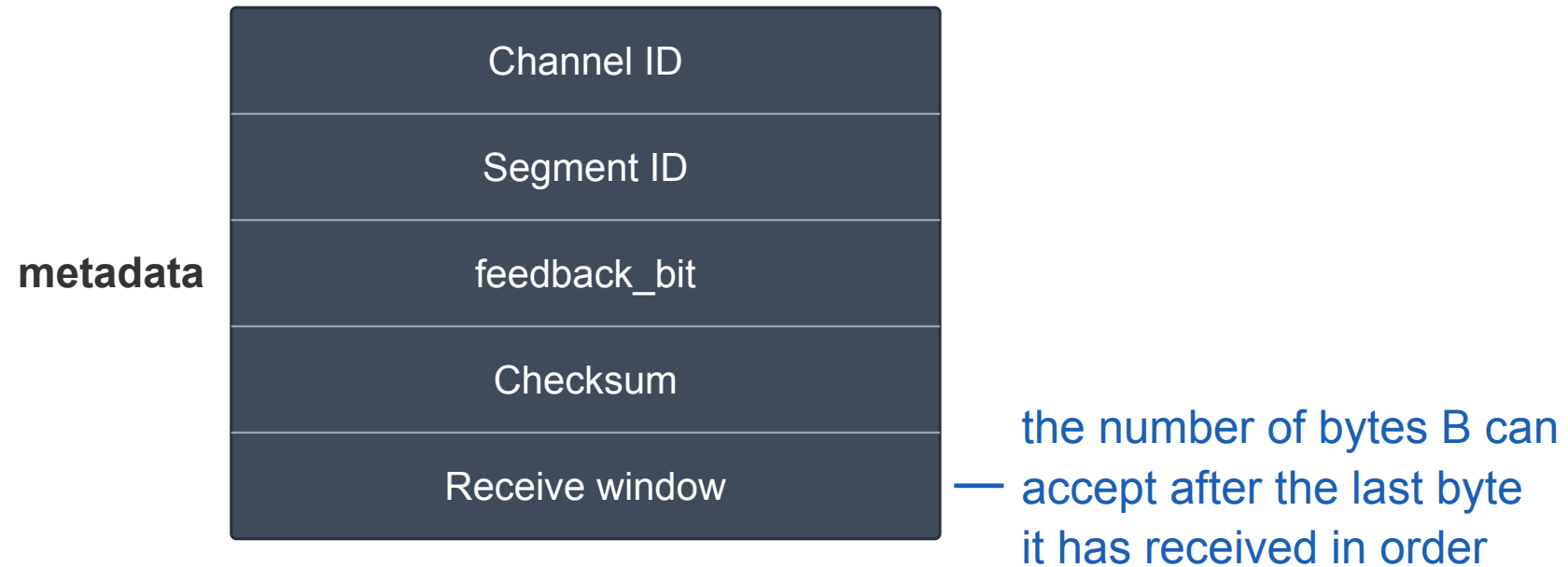
A's window should never be larger than the room left in B's receive buffer.

How can A find out how much room B has?

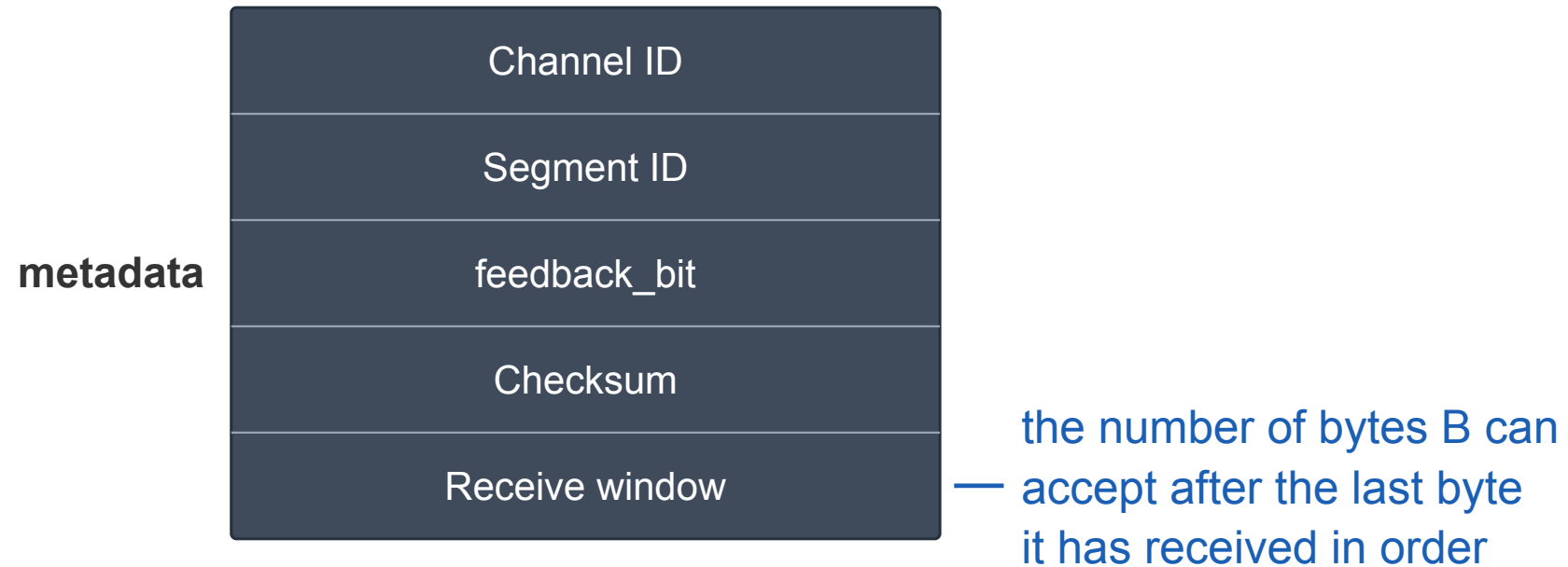
The feedback, with a receive window

metadata	Channel ID
	Segment ID
	feedback_bit
	Checksum

The feedback, with a receive window

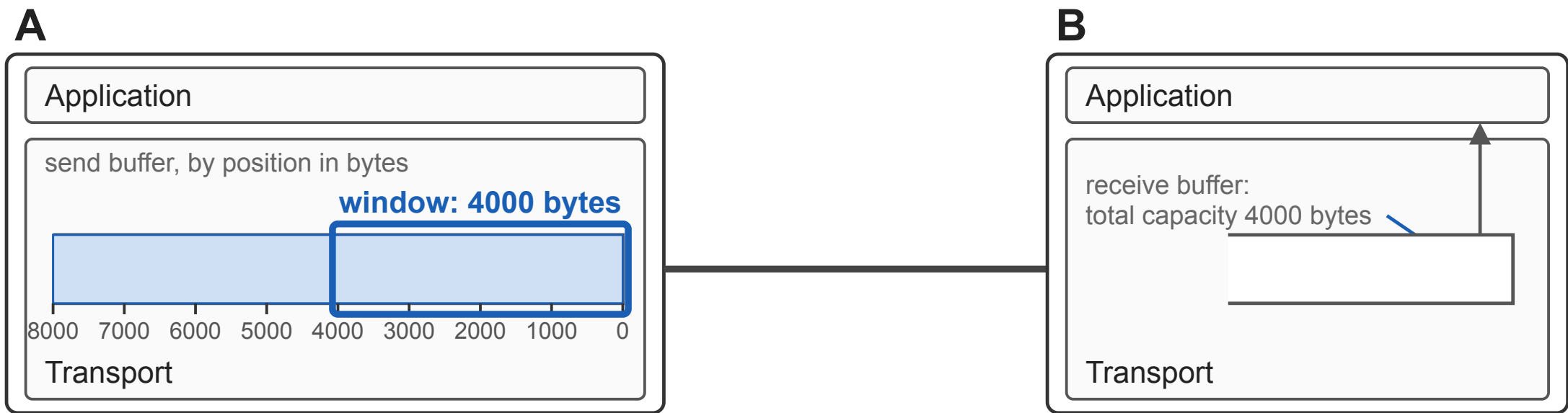


The feedback, with a receive window



A sets the size of its sliding window to the receive window in the latest feedback

Flow control, step by step



B's receive buffer is empty, and B has told A: receive window = 4000.
A's window is 4000 bytes.



successfully delivered

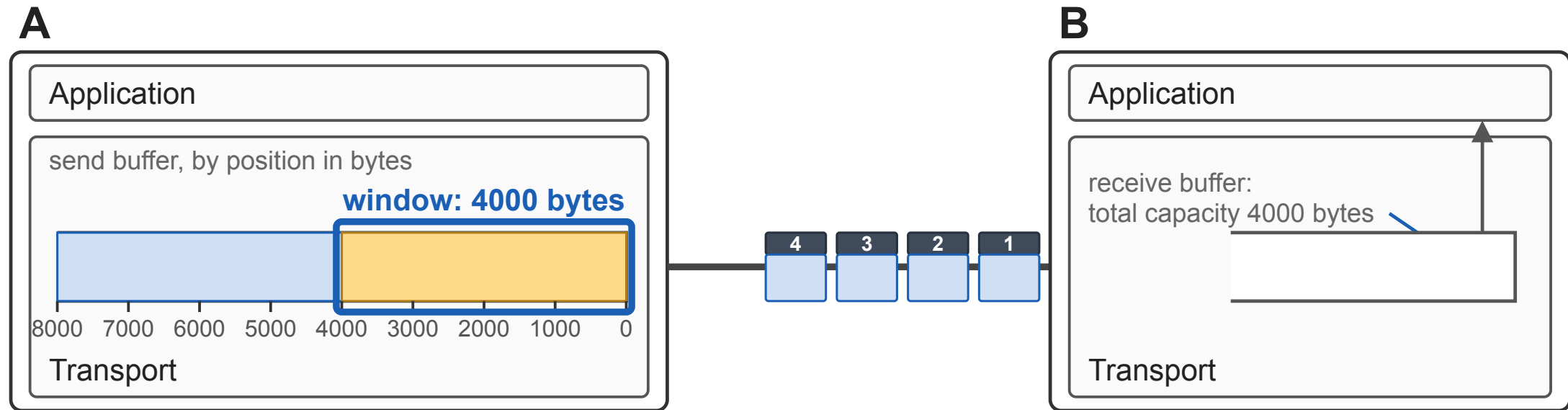


sent, waiting for feedback



not sent yet

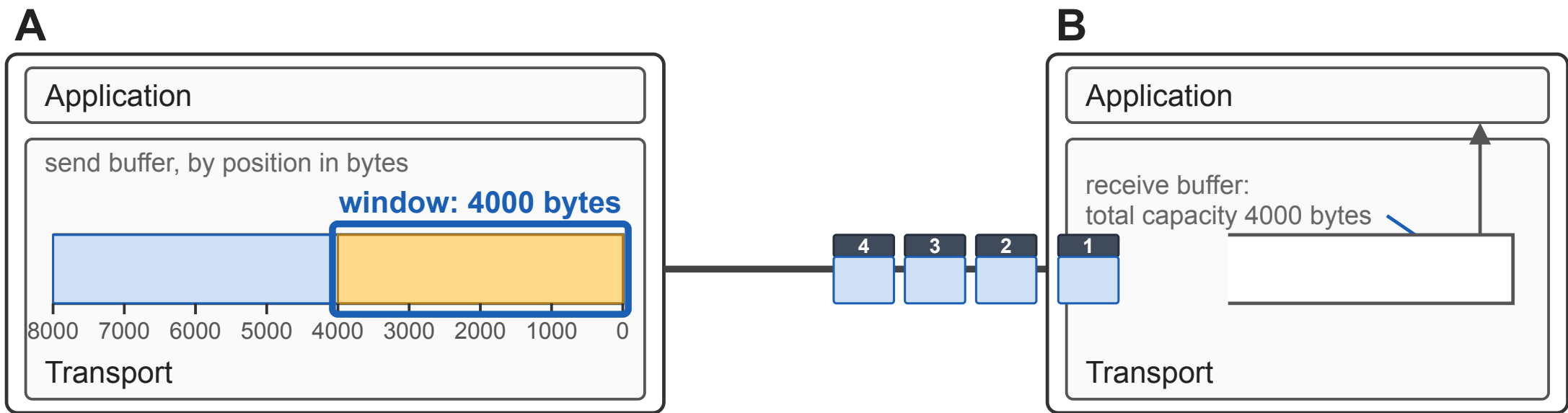
Flow control, step by step



A sends the 4000 bytes in its window, as packets 1 to 4.

 successfully delivered  sent, waiting for feedback  not sent yet

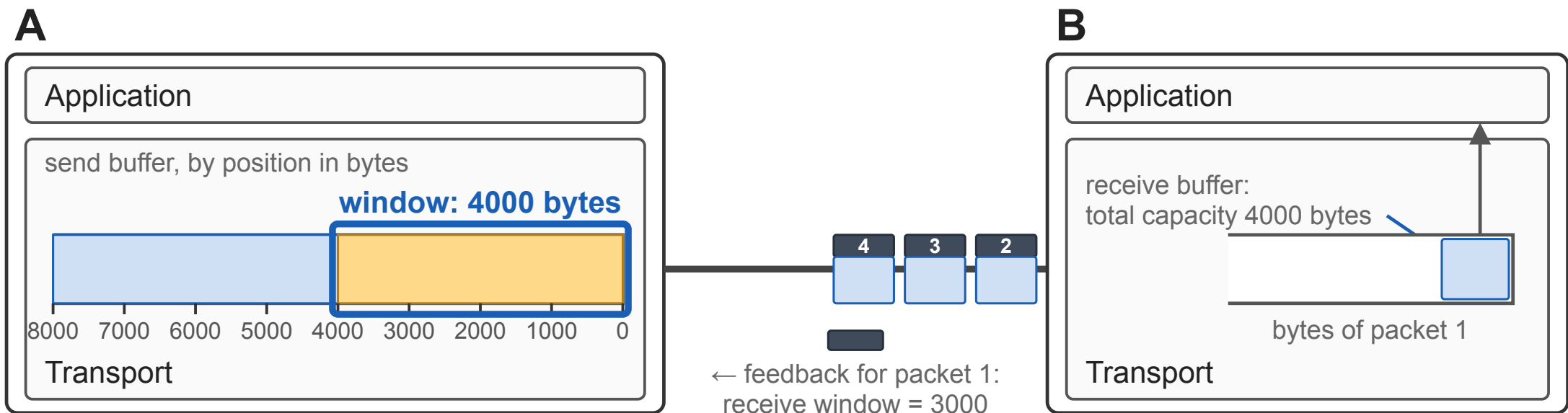
Flow control, step by step



Packet 1 arrives at B.

 successfully delivered  sent, waiting for feedback  not sent yet

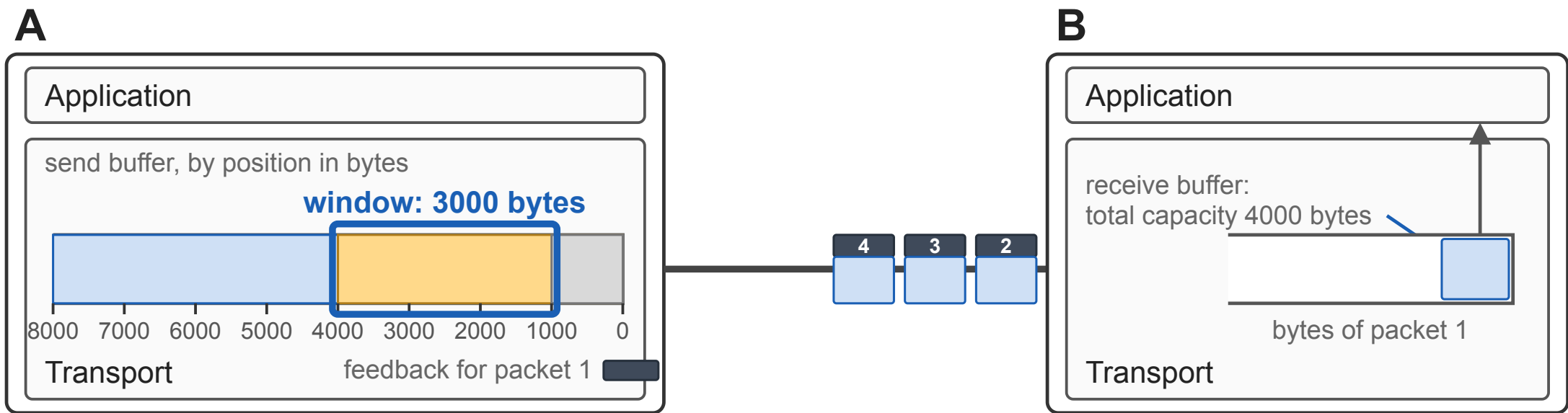
Flow control, step by step



B stores its bytes. B has space left for 3000 more bytes, and specifies that in its feedback for packet 1: receive window = 3000.

successfully delivered sent, waiting for feedback not sent yet

Flow control, step by step



The feedback for packet 1 reaches A. A's window now starts at byte 1000 and is 3000 bytes long.
All of it is sent already, so A sends nothing new.



successfully delivered

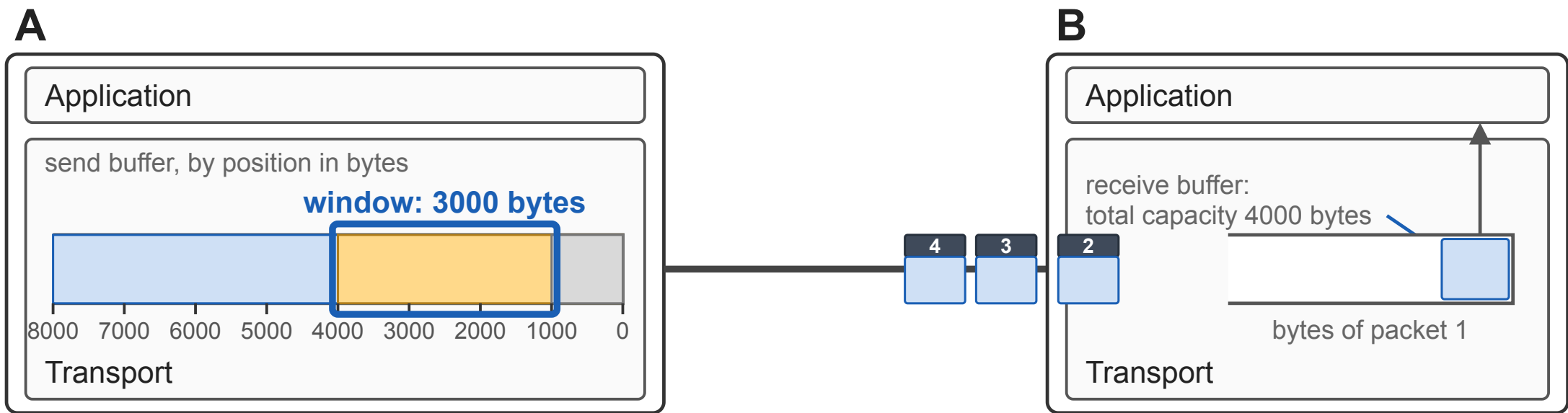


sent, waiting for feedback



not sent yet

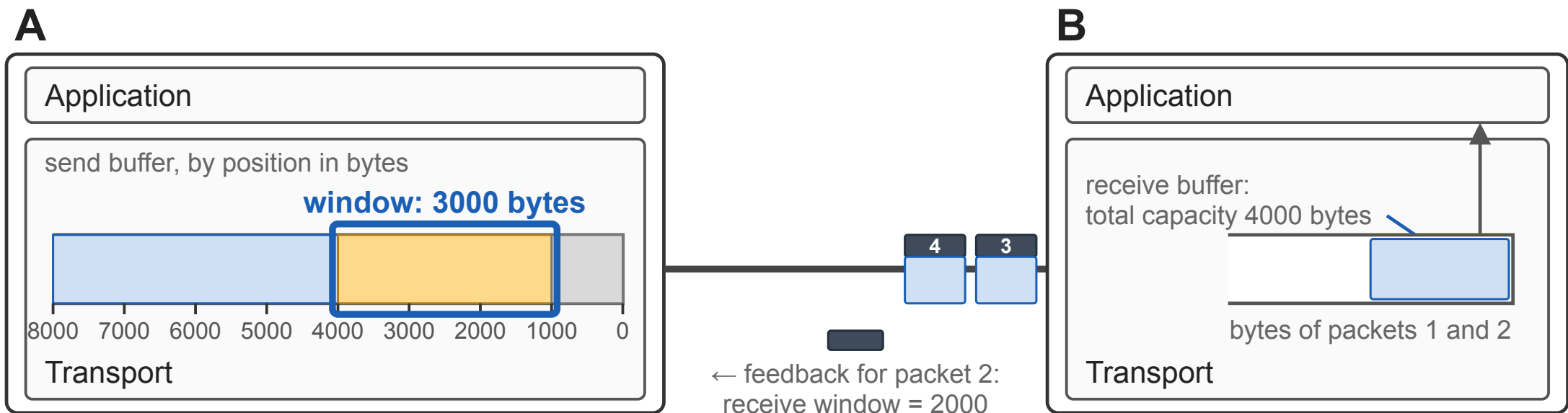
Flow control, step by step



Packet 2 arrives at B.

successfully delivered sent, waiting for feedback not sent yet

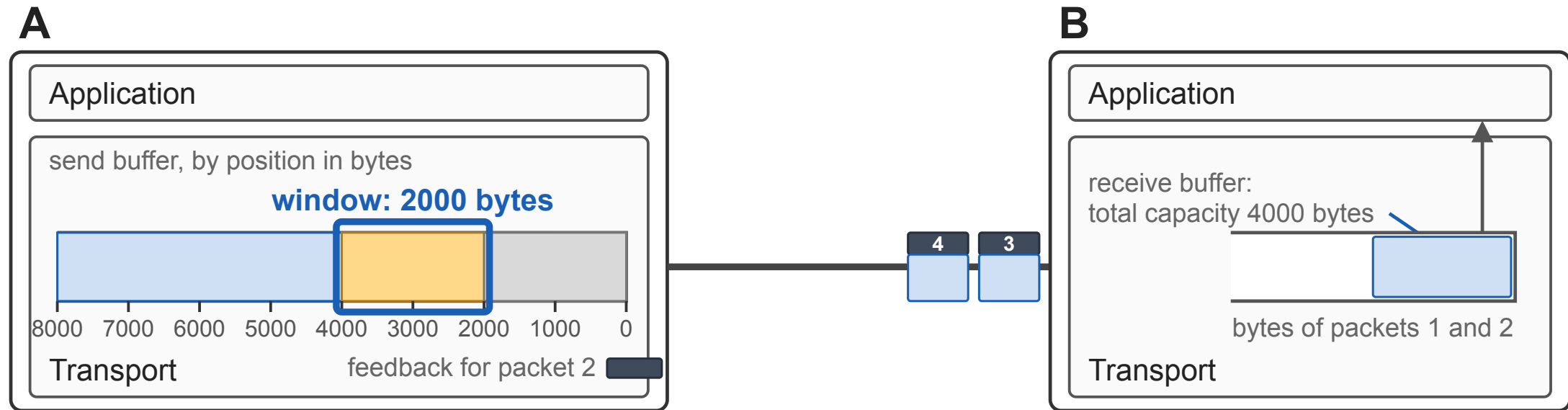
Flow control, step by step



B stores its bytes: space left for 2000 bytes, receive window = 2000.

 successfully delivered  sent, waiting for feedback  not sent yet

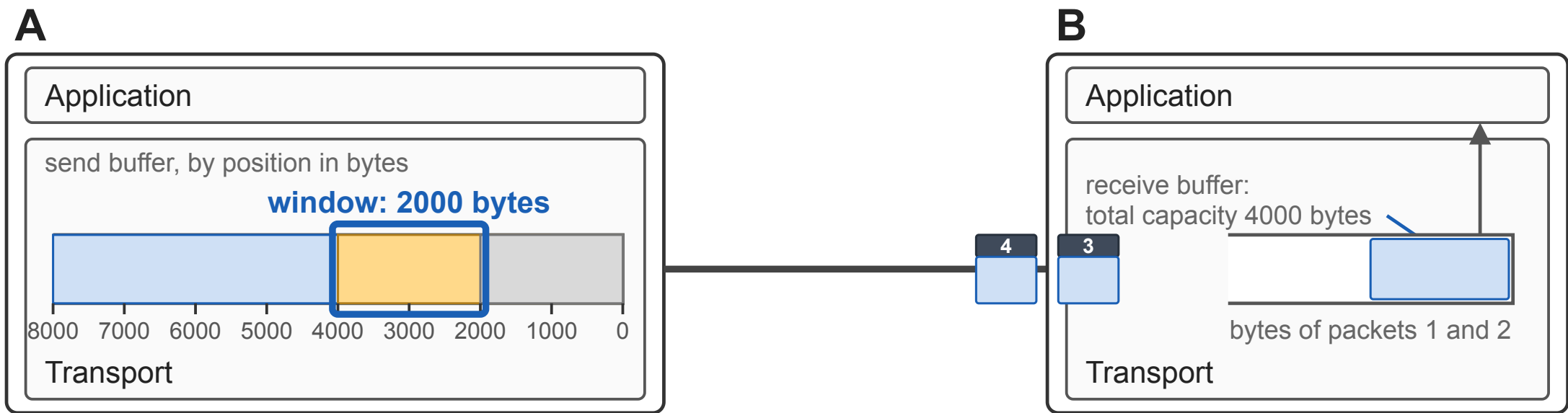
Flow control, step by step



The feedback for packet 2 reaches A: A's window now starts at byte 2000 and is 2000 bytes long.

 successfully delivered  sent, waiting for feedback  not sent yet

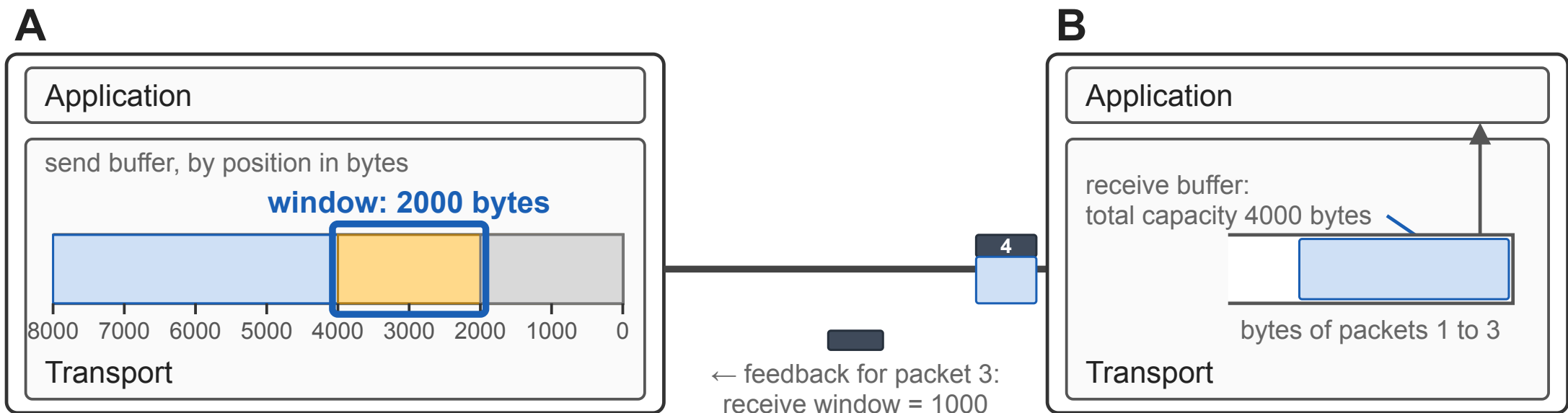
Flow control, step by step



Packet 3 arrives at B.

 successfully delivered  sent, waiting for feedback  not sent yet

Flow control, step by step



B stores its bytes: space left for 1000 bytes, receive window = 1000.



successfully delivered

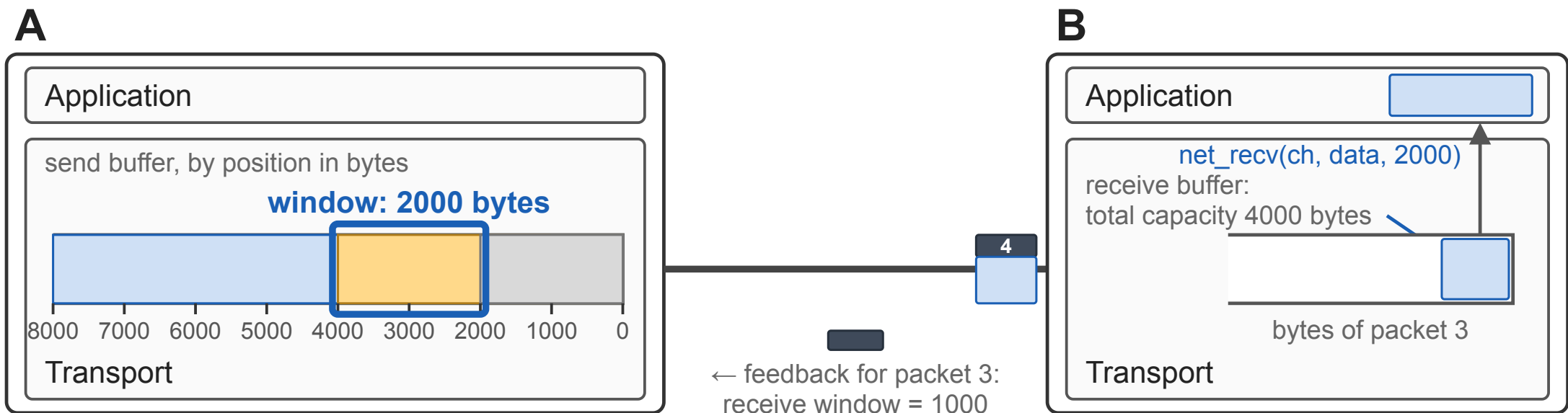


sent, waiting for feedback



not sent yet

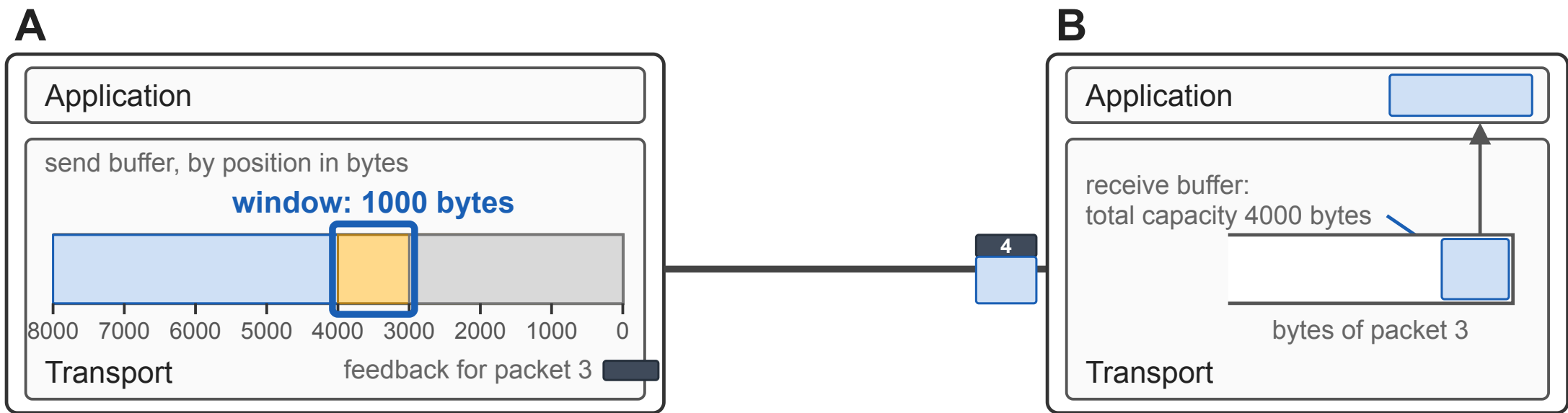
Flow control, step by step



The application at B calls `net_rcv` for 2000 bytes:
the bytes of packets 1 and 2 leave the receive buffer.

 successfully delivered  sent, waiting for feedback  not sent yet

Flow control, step by step



The feedback for packet 3 reaches A: A's window now starts at byte 3000 and is 1000 bytes long.



successfully delivered

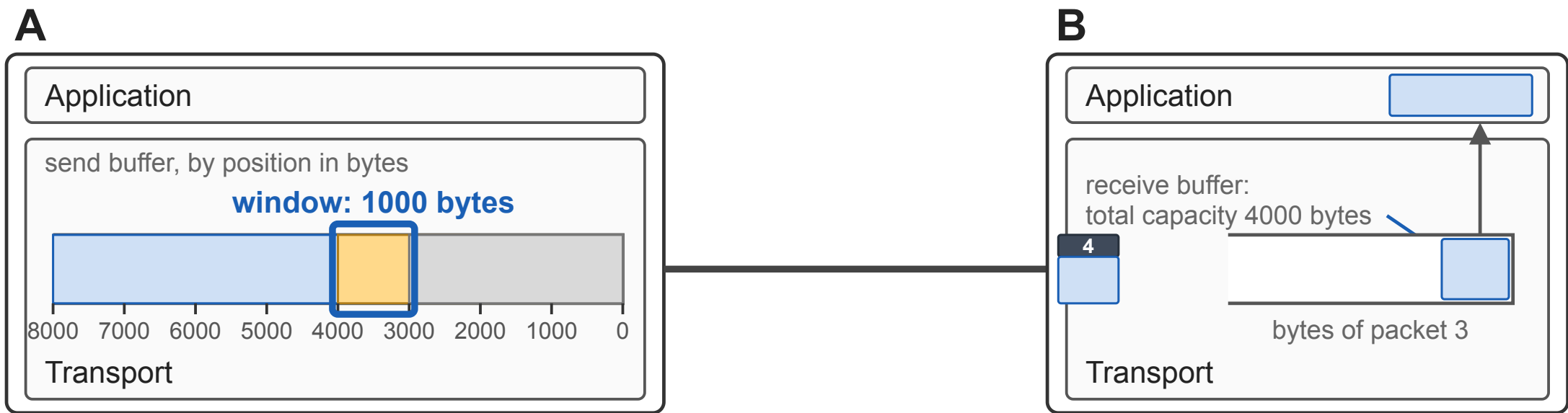


sent, waiting for feedback



not sent yet

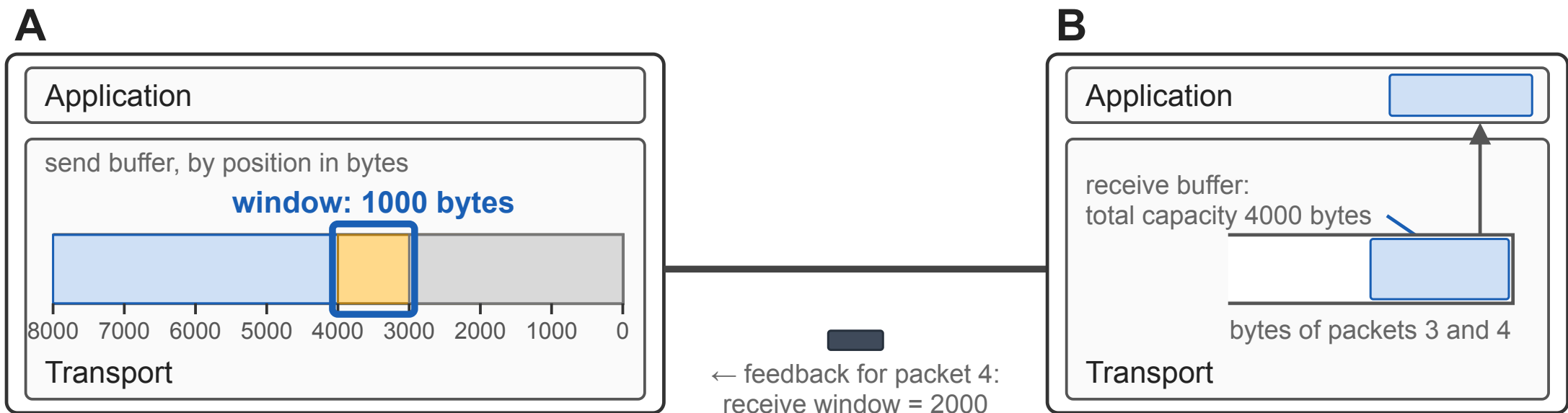
Flow control, step by step



Packet 4 arrives at B.

 successfully delivered  sent, waiting for feedback  not sent yet

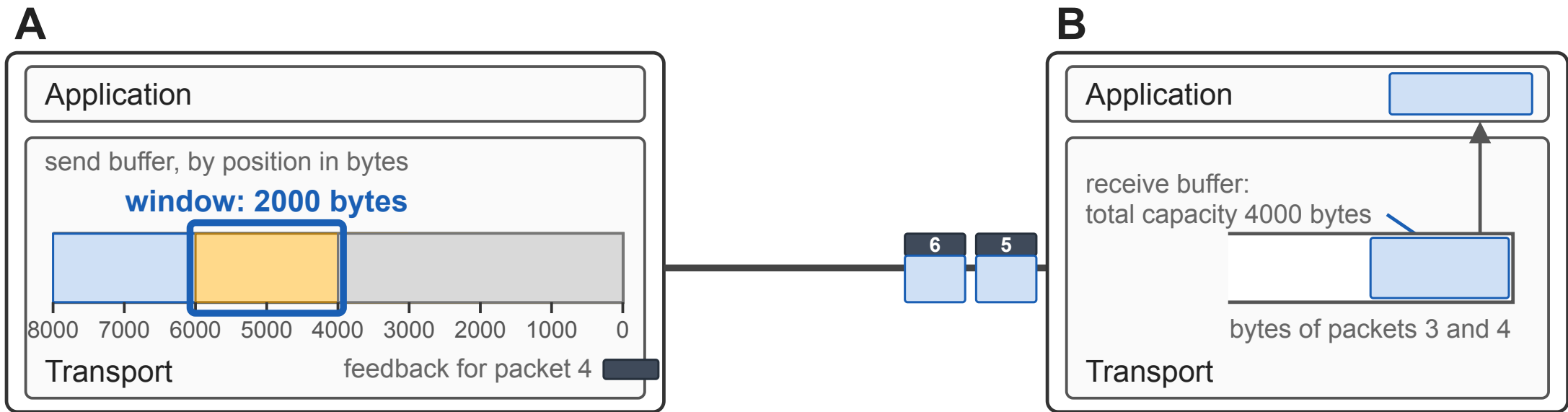
Flow control, step by step



B stores its bytes. 2000 bytes are waiting in the buffer: space left for 2000 bytes, receive window = 2000.

 successfully delivered  sent, waiting for feedback  not sent yet

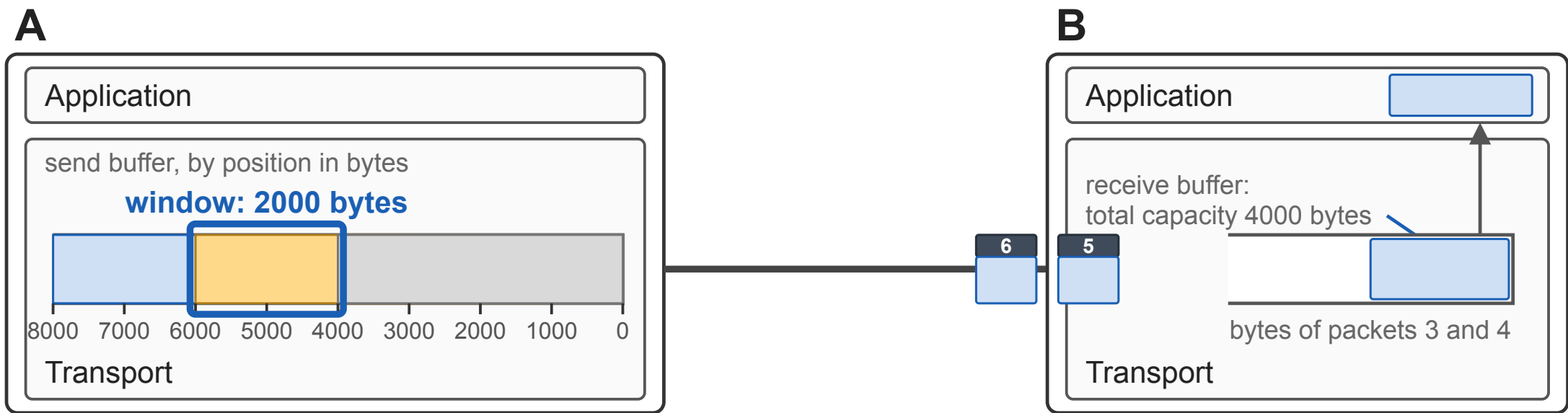
Flow control, step by step



The feedback for packet 4 reaches A. A's window now starts from byte 4000 and is 2000 bytes long:
A sends packets 5 and 6.

 successfully delivered  sent, waiting for feedback  not sent yet

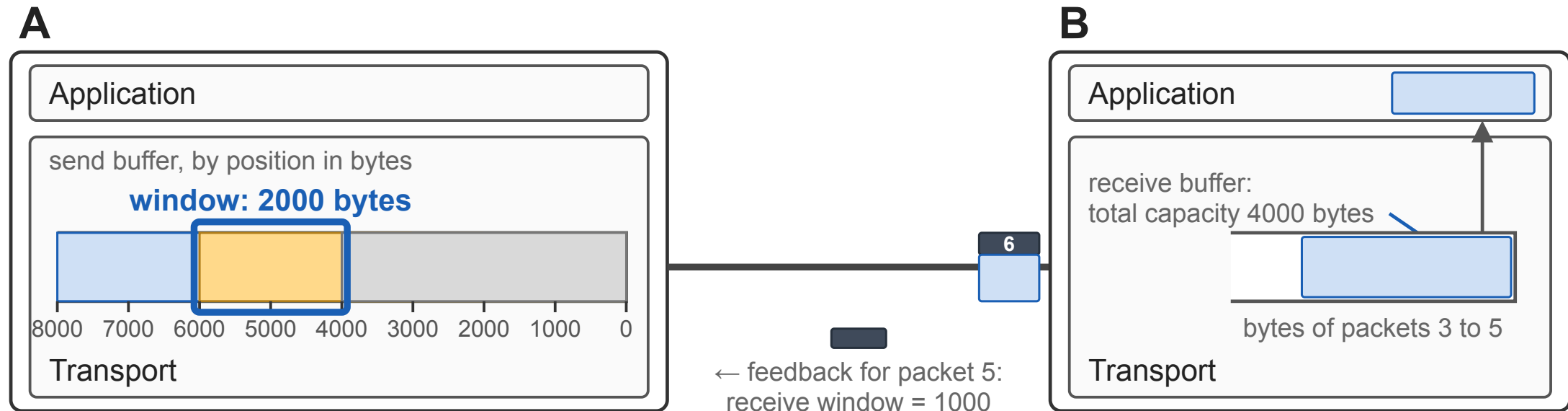
Flow control, step by step



Packet 5 arrives at B.

successfully delivered sent, waiting for feedback not sent yet

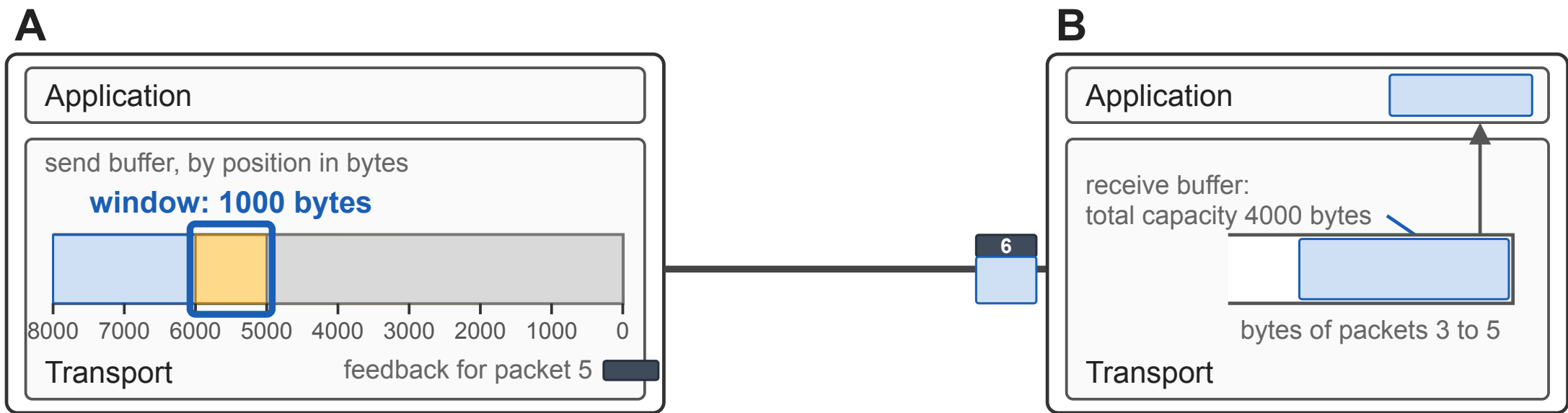
Flow control, step by step



B stores its bytes, and the application has not called `net_rcv` again:
space left for 1000 bytes, receive window = 1000.

 successfully delivered  sent, waiting for feedback  not sent yet

Flow control, step by step



The feedback for packet 5 reaches A: A's window now starts at byte 5000 and is 1000 bytes long.



successfully delivered

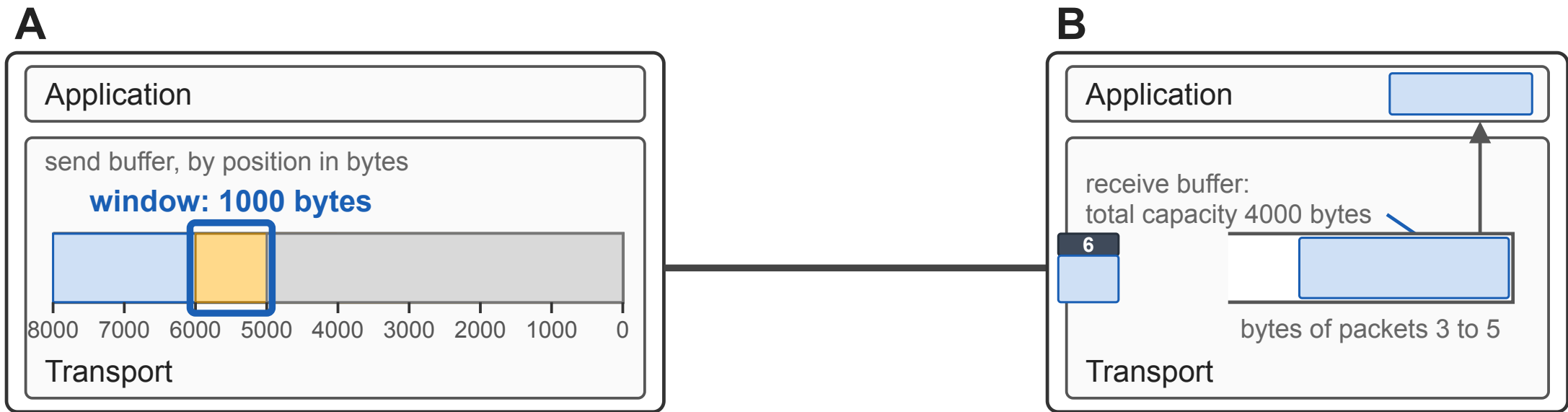


sent, waiting for feedback



not sent yet

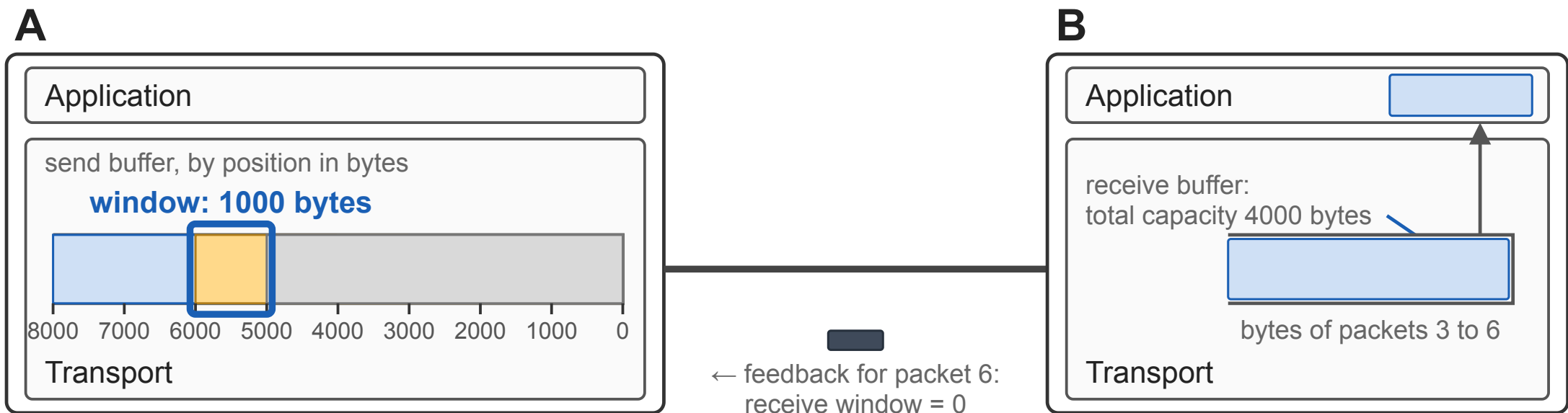
Flow control, step by step



Packet 6 arrives at B.

successfully delivered sent, waiting for feedback not sent yet

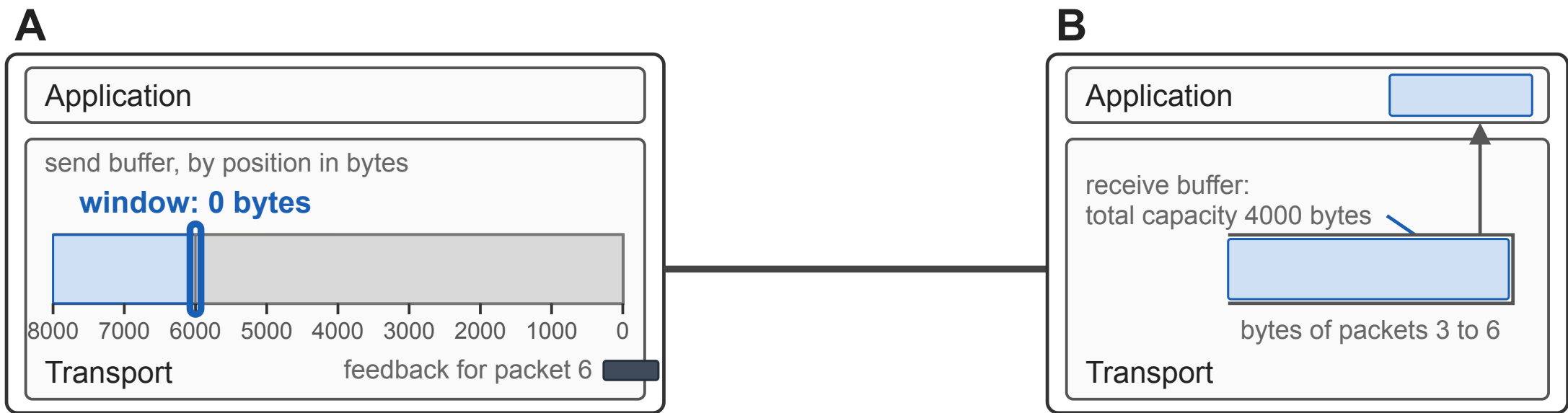
Flow control, step by step



B stores its bytes, and the buffer is full: no space left, receive window = 0.

 successfully delivered  sent, waiting for feedback  not sent yet

Flow control, step by step



The feedback for packet 6 reaches A. A's window is now 0 bytes: A sends nothing until B has room again.
The receive buffer never overflows, and nothing is dropped.

 successfully delivered  sent, waiting for feedback  not sent yet

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

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

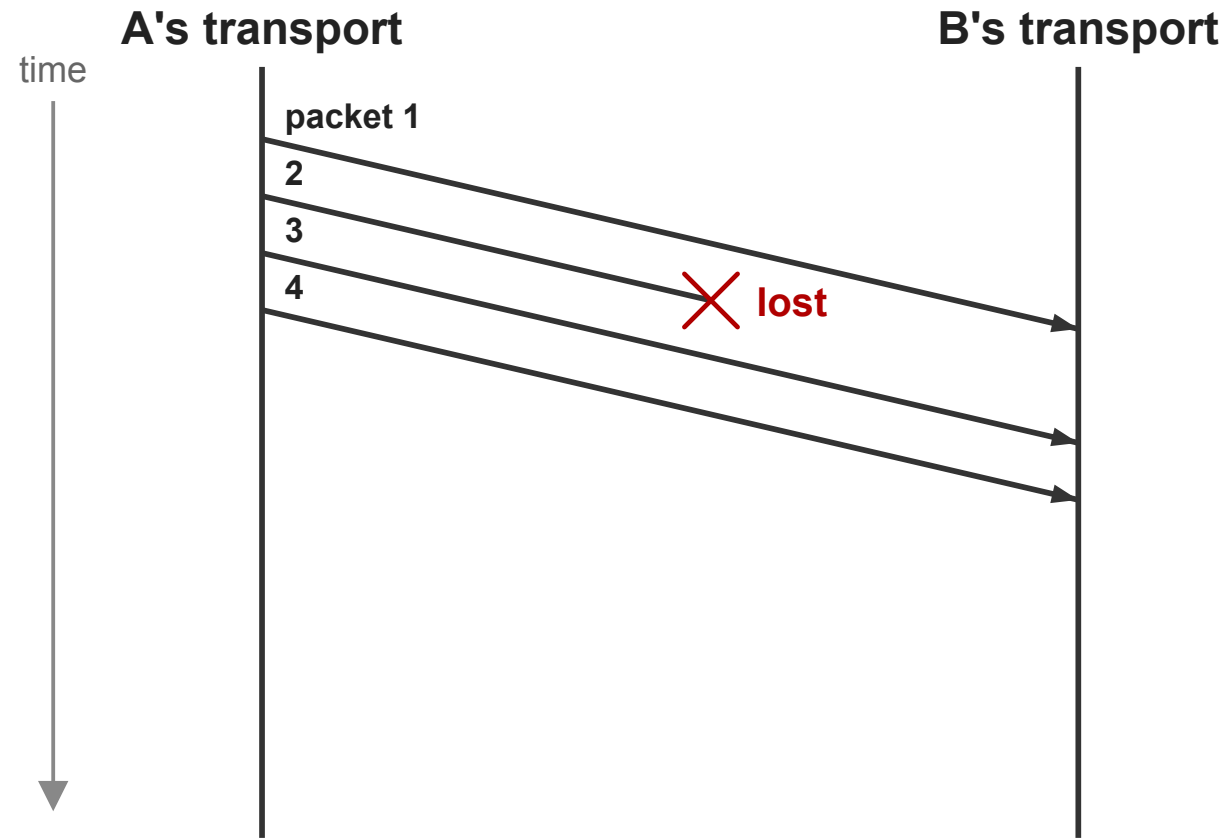
What is a good batch size?

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

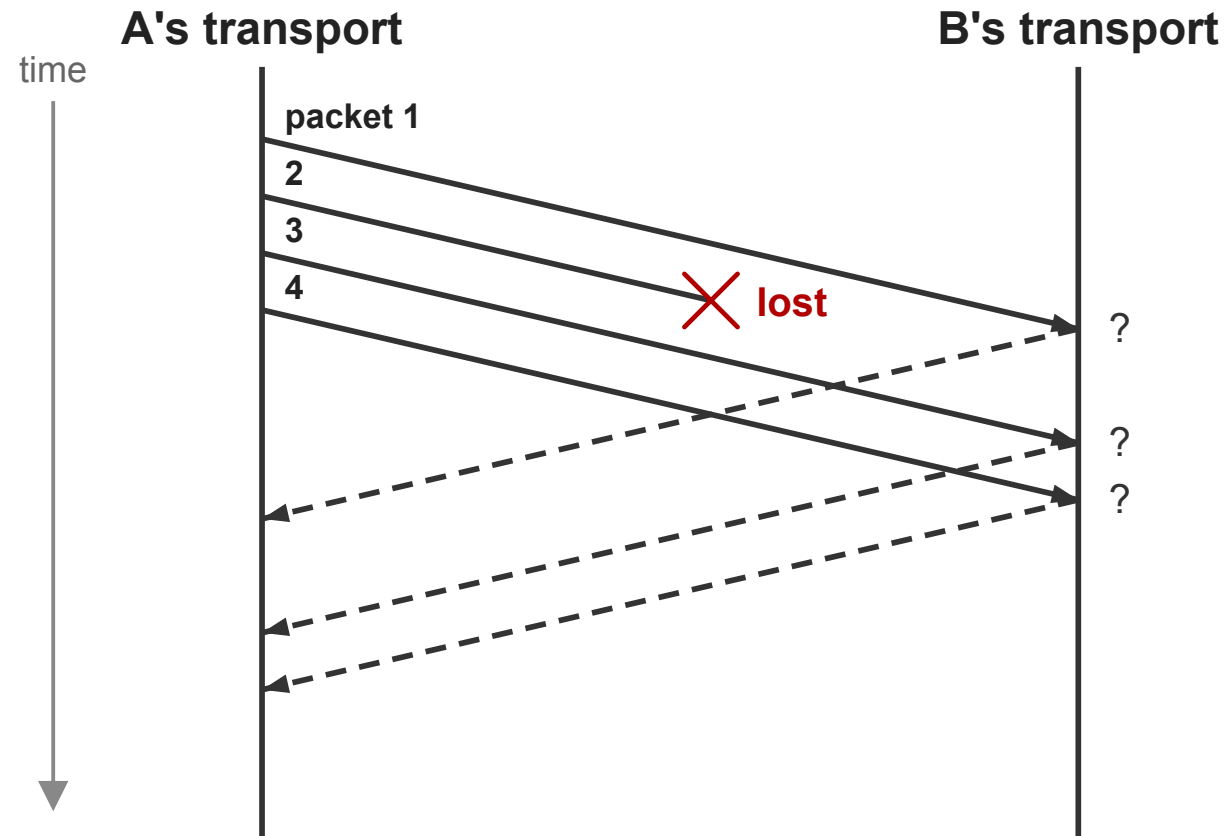
✓ flow control with a sliding window

Dealing with loss



Suppose the window size is 4.
A sends segments 1 to 4.
Segment 2 is lost.

Dealing with loss

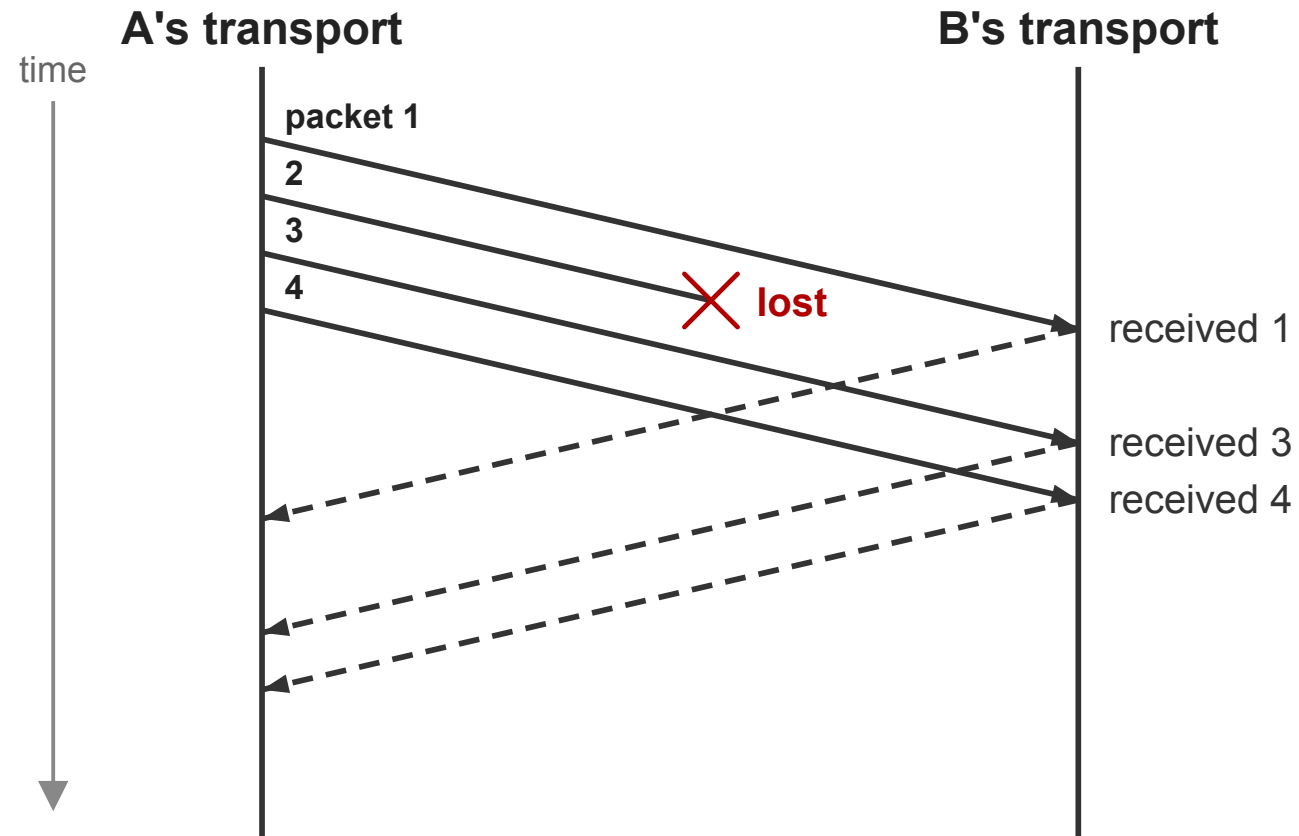


Suppose the window size is 4.
A sends segments 1 to 4.
Segment 2 is lost.

B sends positive feedback
for segments 1, 3 and 4.

**What metadata should each
feedback carry, now that
several segments are in
flight at once?**

Option 1: selective feedback

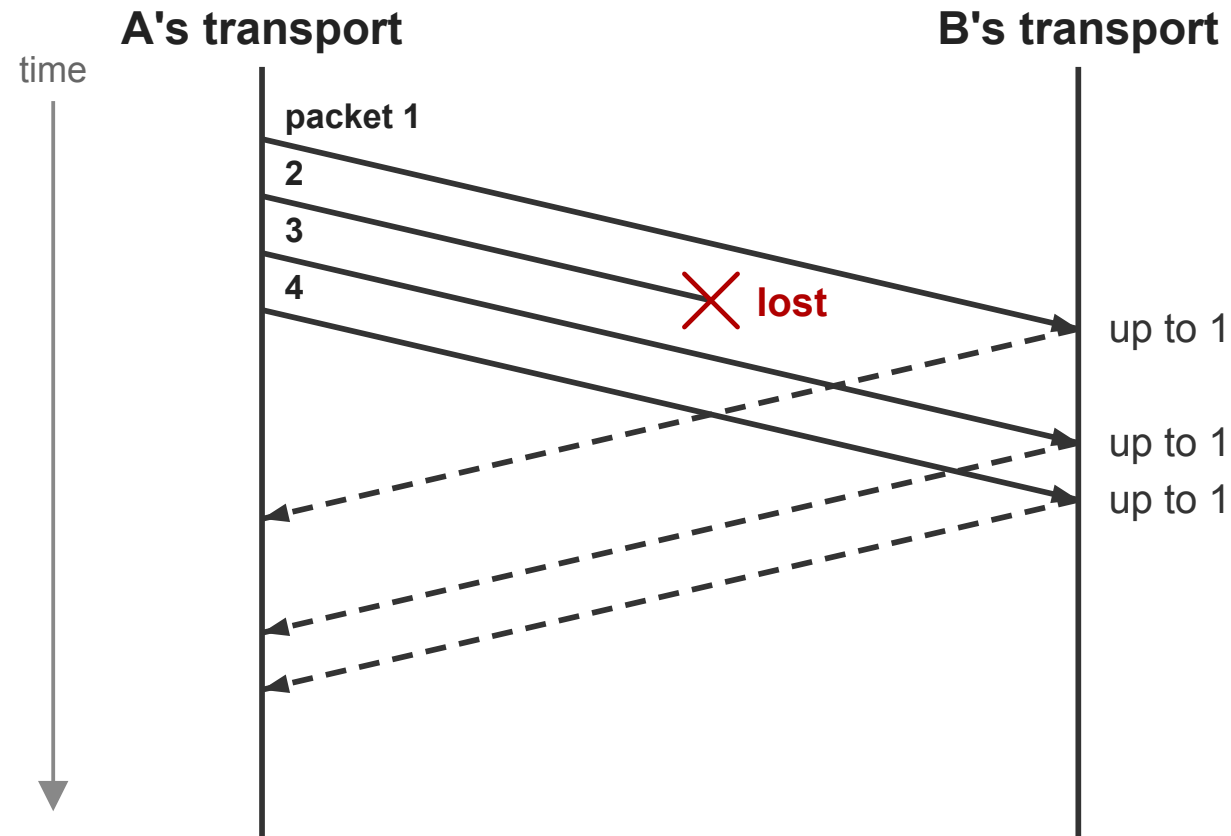


Selective feedback

“I acknowledge I have received segment X.”

One feedback for each packet. The feedback includes the segment ID for the data carried in that packet. This is what we have had so far.

Option 2: cumulative feedback



Cumulative feedback

"I have every segment up to and including $X - 1$.
I don't have X ."

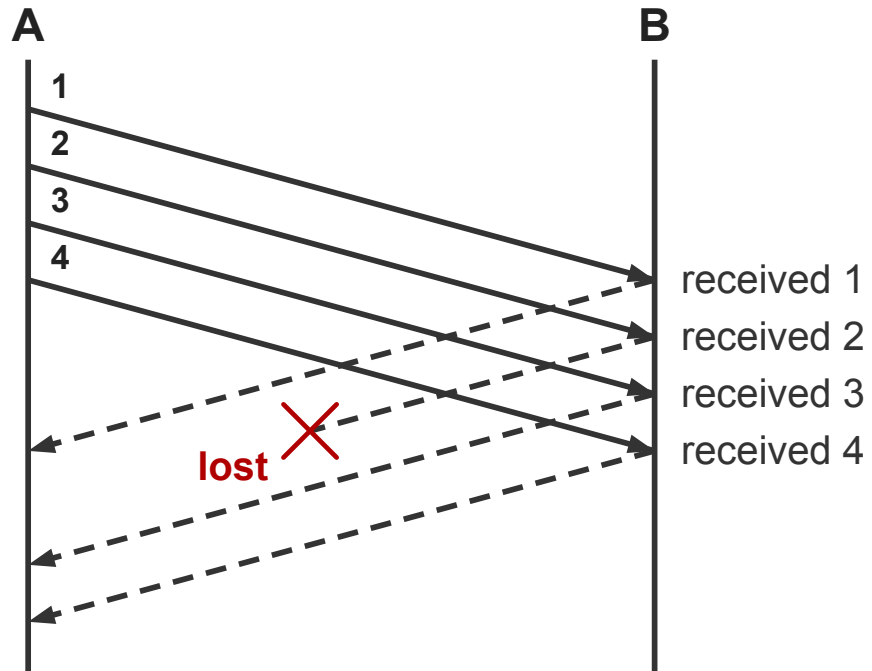
Still one feedback for each packet.

When a feedback is lost (in-class)

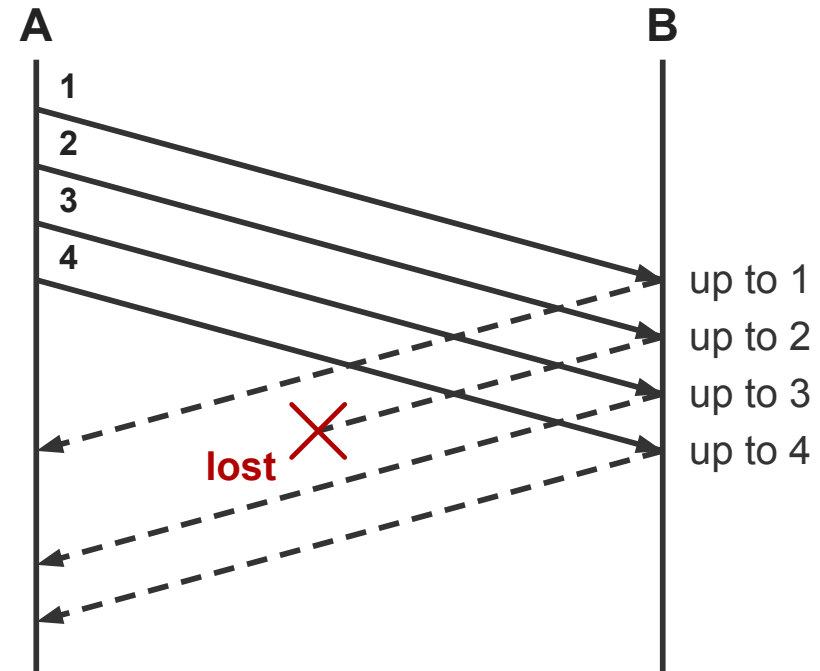
For each kind of acknowledgement, what happens when a feedback is lost? Would it result in a difference in the sender's retransmission behavior?

When a feedback is lost

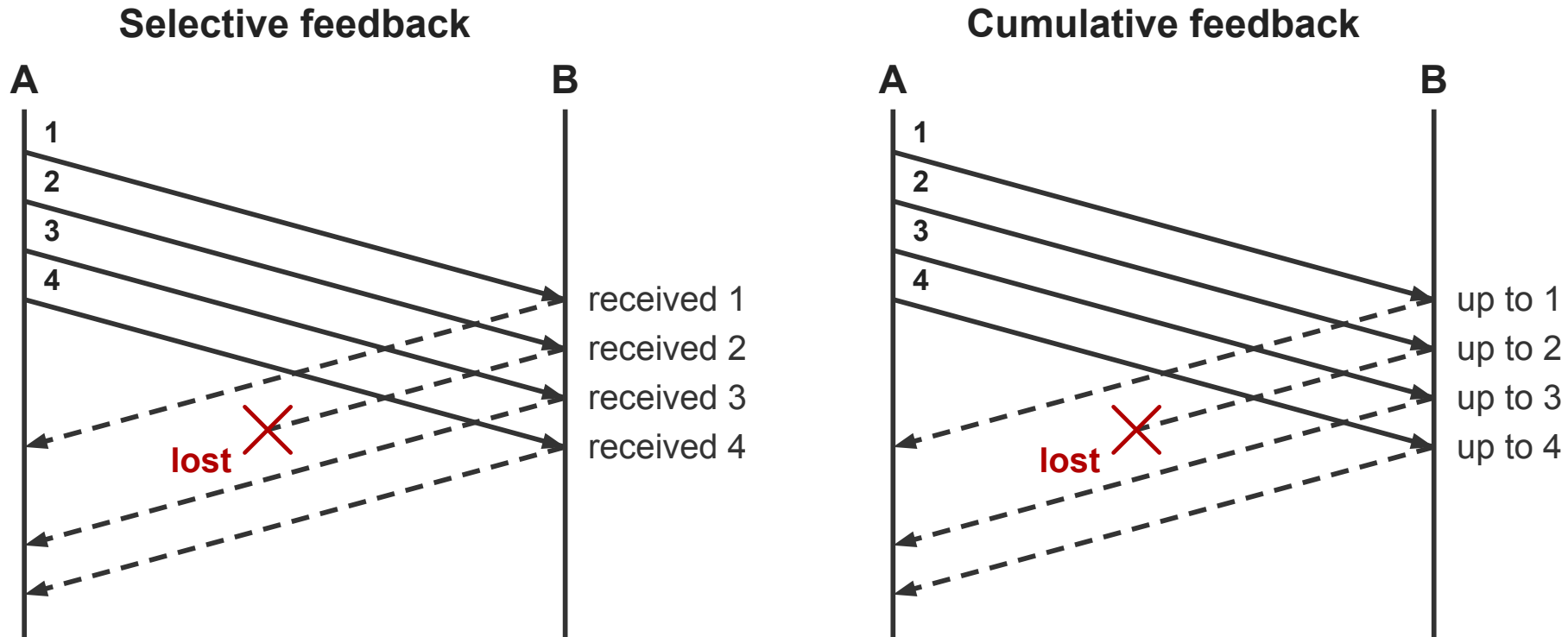
Selective feedback



Cumulative feedback



When a feedback is lost



In each case: does A have to resend packet 2?

Managing timers

- **One timer for each packet:** A starts a unique timer for each packet. When it goes off, it retransmits that packet.
- **One timer:** A keeps one single timer, for the “oldest” packet whose data is still unacknowledged. Once the corresponding data segment is acknowledged, A finds the next “oldest” packet waiting for feedback, and resets the timer.
-

Managing timers

- **One timer for each packet:** A starts a unique timer for each packet. When it goes off, it retransmits that packet.
- **One timer:** A keeps one single timer, for the “oldest” packet whose data is still unacknowledged. Once the corresponding data segment is acknowledged, A finds the next “oldest” packet waiting for feedback, and resets the timer.
- **Quick check:** In the one-timer case, which packet gets retransmitted on timer expiry?

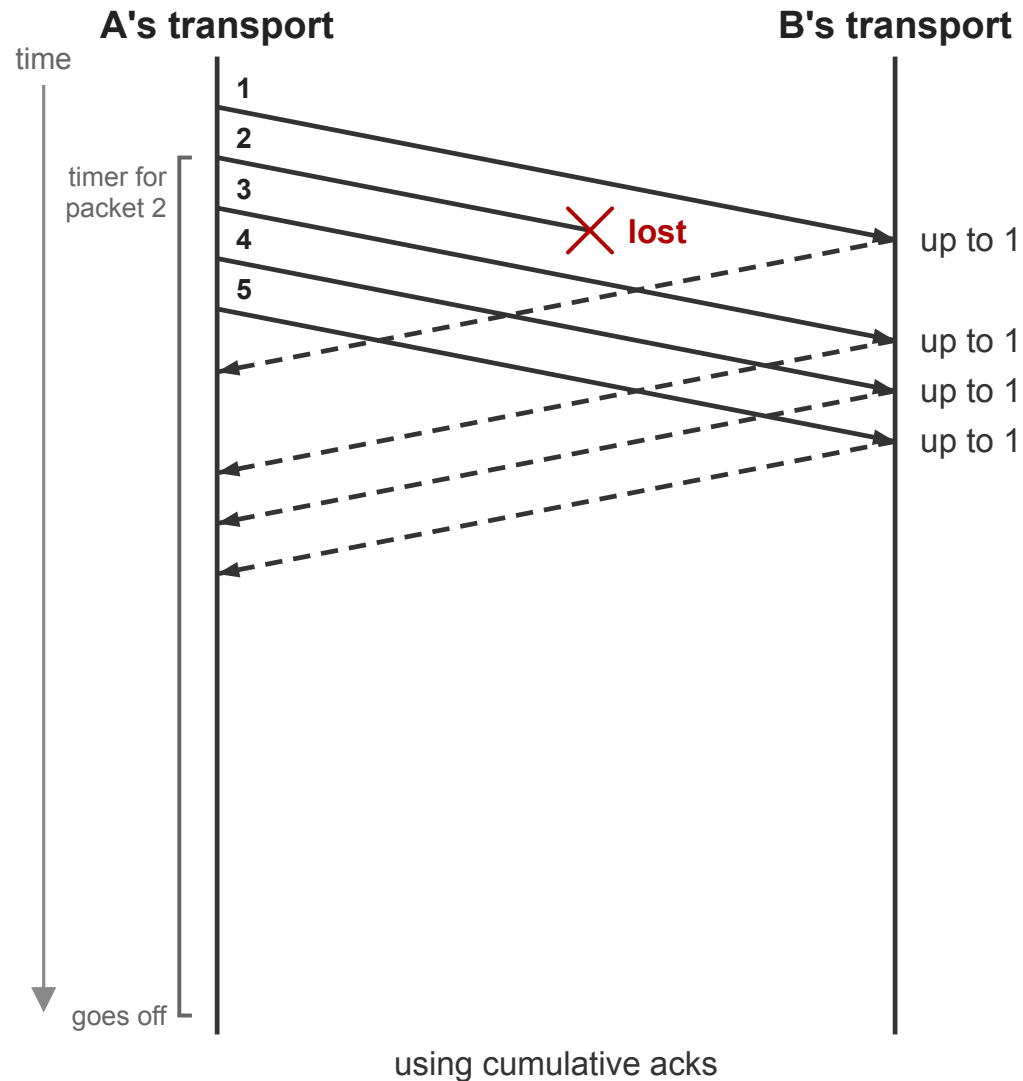
Pros and cons (in-class)

What are the pros and cons of each acknowledgement option? What are the pros and cons of each timer option?

(in class discussion): extensive discussion with examples about cumulative vs selective acks, combining these two types of acks, trade-offs, and retransmission strategies when loss is detected with cumulative acks.

Reset

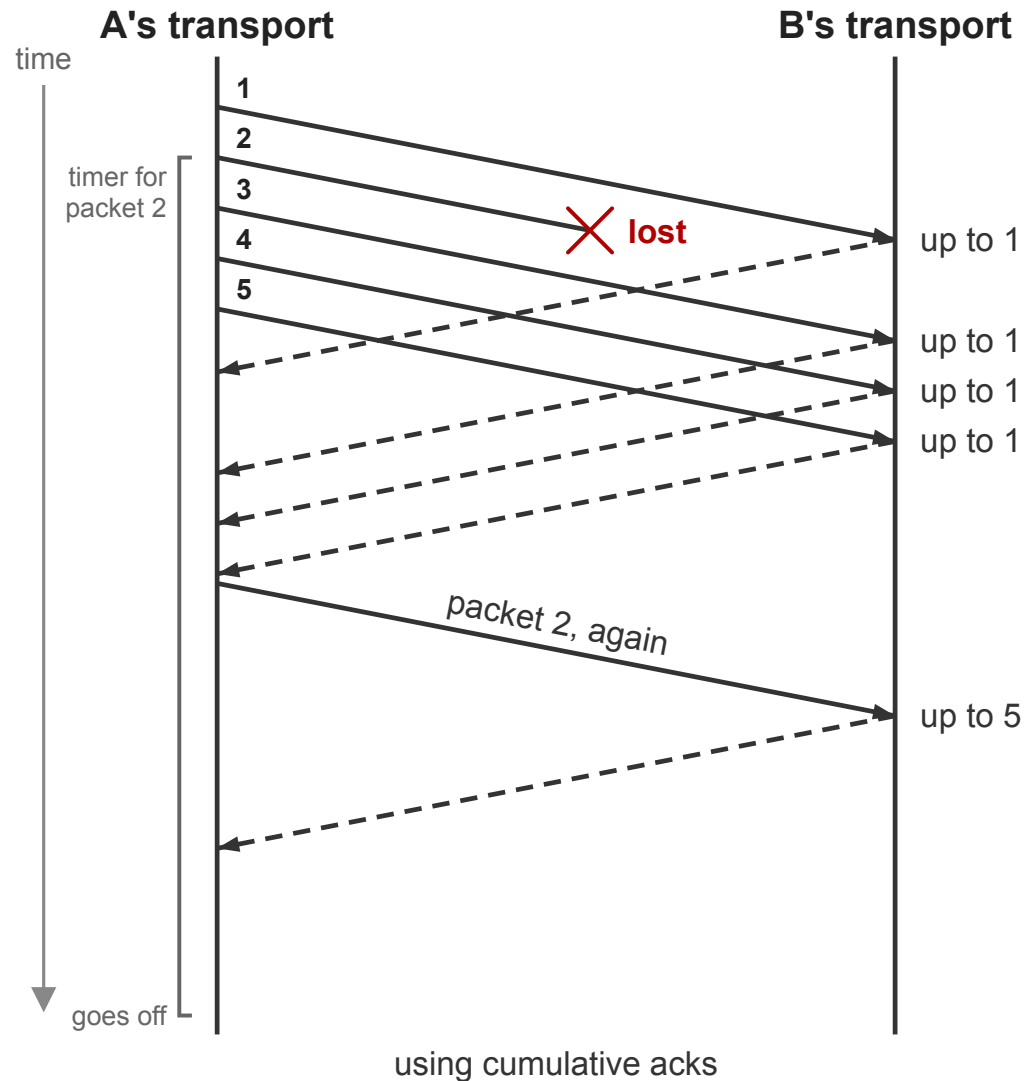
Other “faster” loss signals



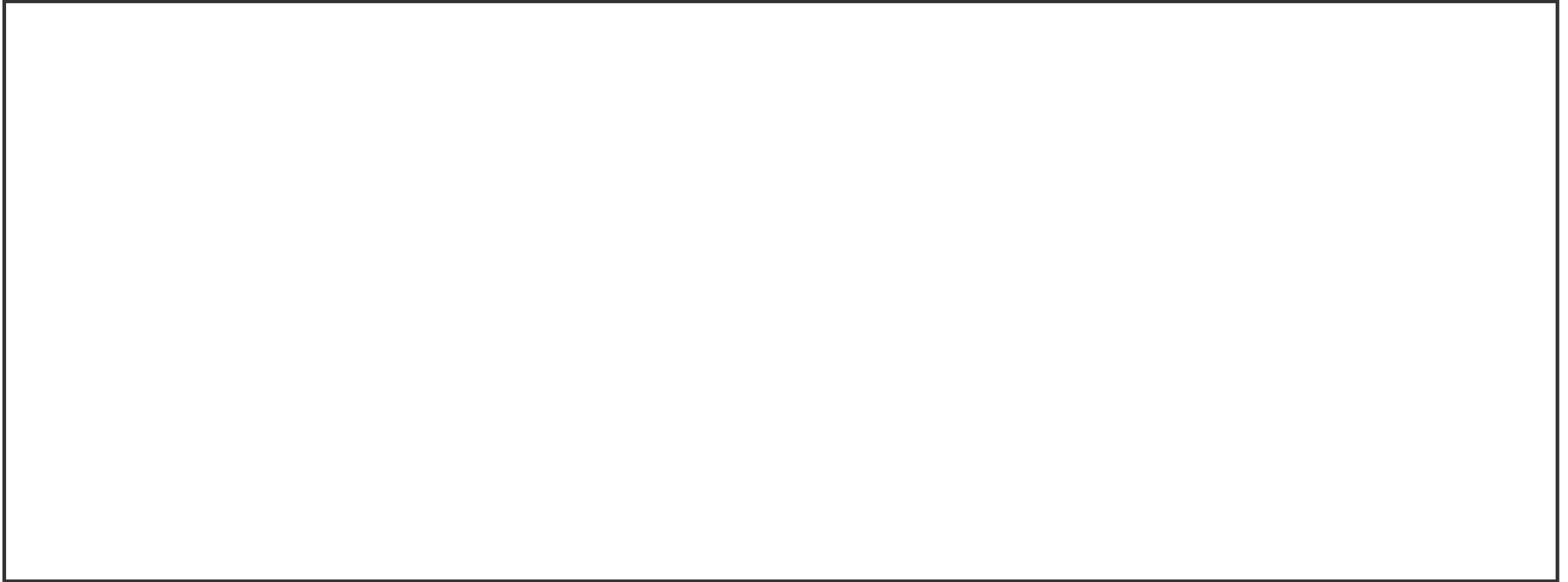
So far, timer expiry was the signal for detecting loss.

Is there a way we can detect loss faster?

Fast retransmit using “duplicate” acks

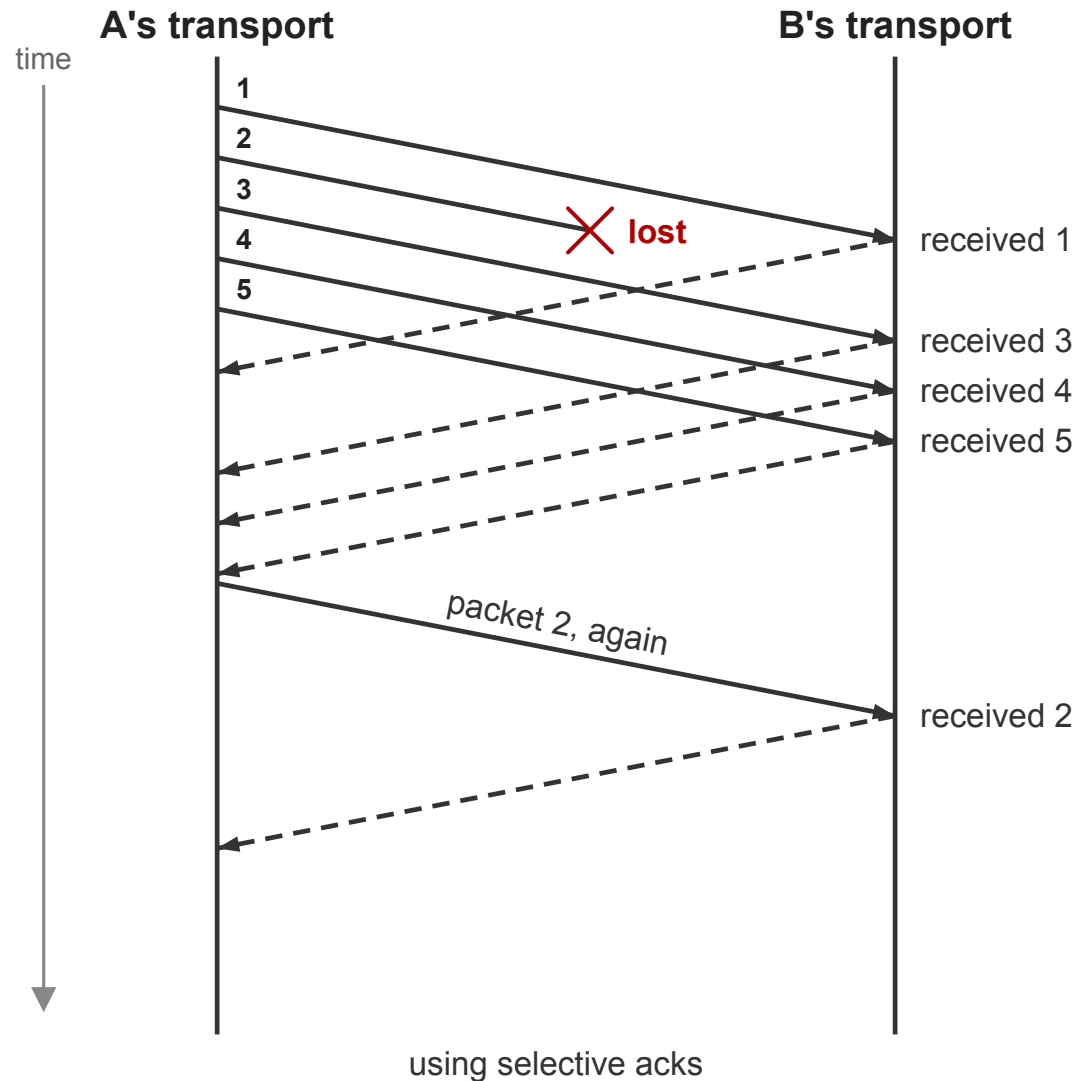


Can you do the same thing with selective acks? (in-class)



Reset

Fast retransmit using selective acks



What we added today

- Allowing multiple in-flight packets
- Controlling the number of in-flight packets using a sliding window
- Flow control: Receiver tells sender how much room it has for outstanding data and the sender adjusts the in-flight data accordingly.
- Selective and cumulative acks
- Fast retransmission using acks as signal for loss rather than timeouts.