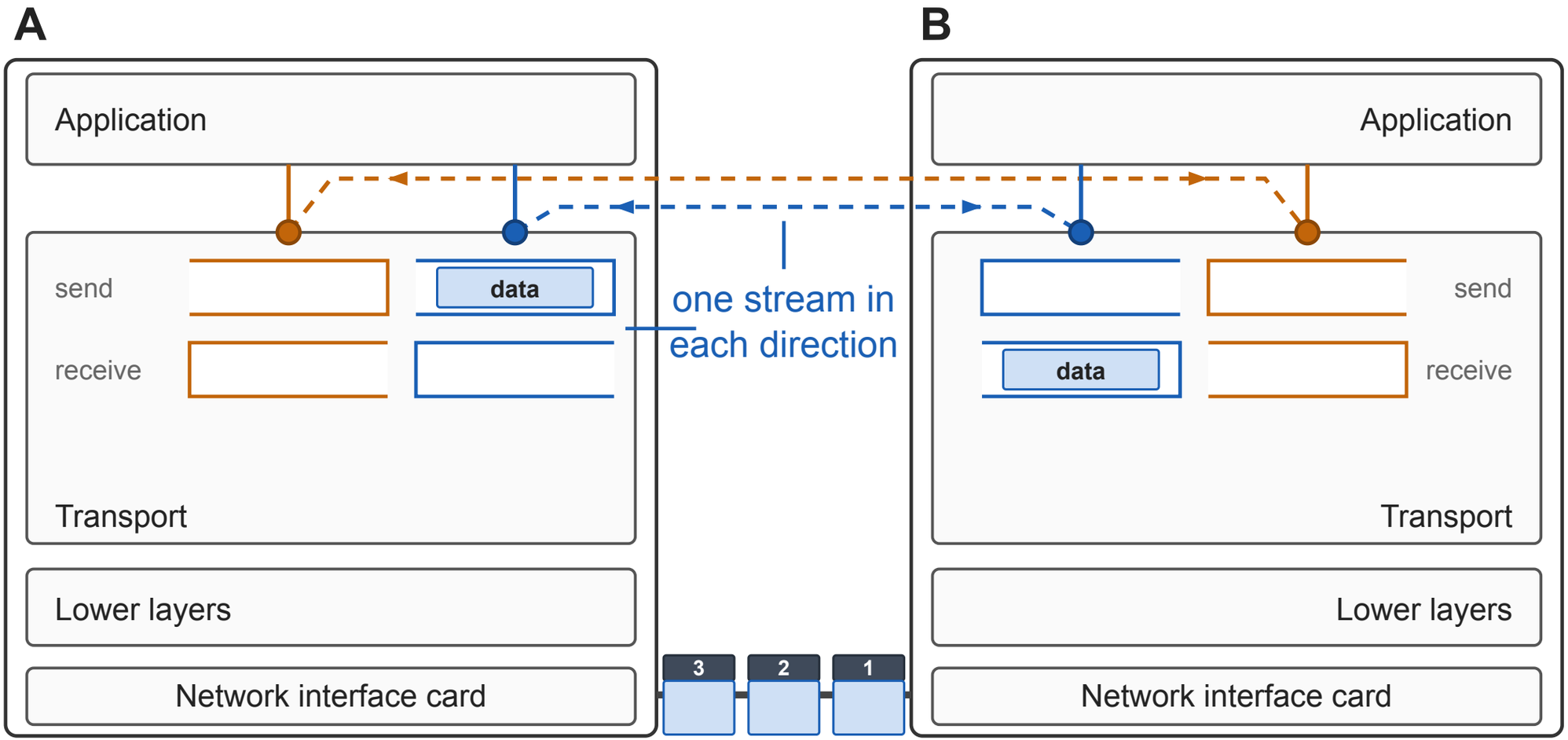
UNIVERSITY OF WATERLOO

The transport layer: Dealing with bit errors and loss

CS 456/656, Fall 2026

Mina Tahmasbi Arashloo

Where we are



each packet carries its channel ID and its segment ID

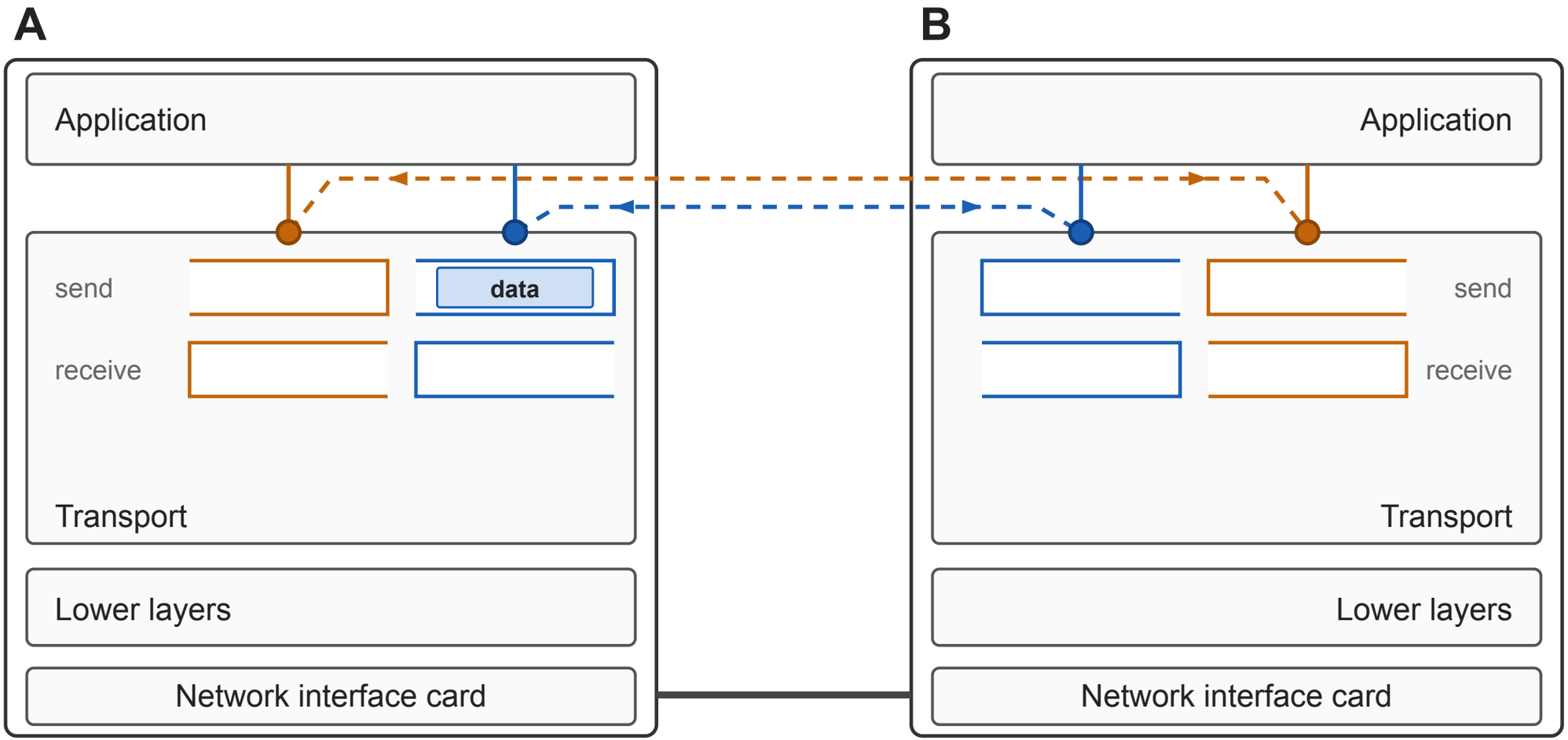
What does the transport layer have to take care of?

1. Get the data to the right application on the destination computer
2. Segmentation and reassembly
3. What to do when the lower layers lose, corrupt, or slow down packets

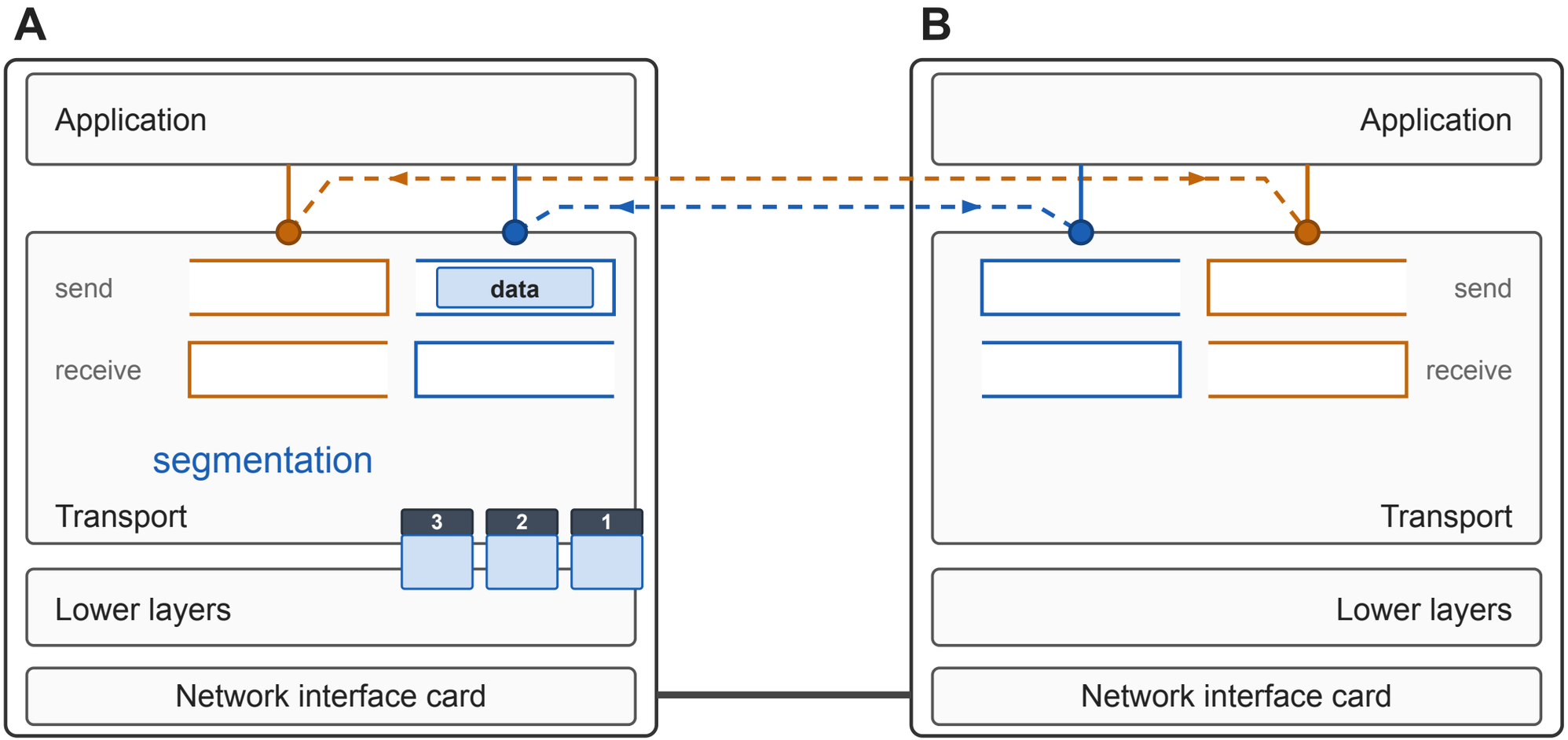
What does the transport layer have to take care of?

1. Get the data to the right application on the destination computer
2. Segmentation and reassembly
3. What to do when the lower layers lose, corrupt, or slow down packets

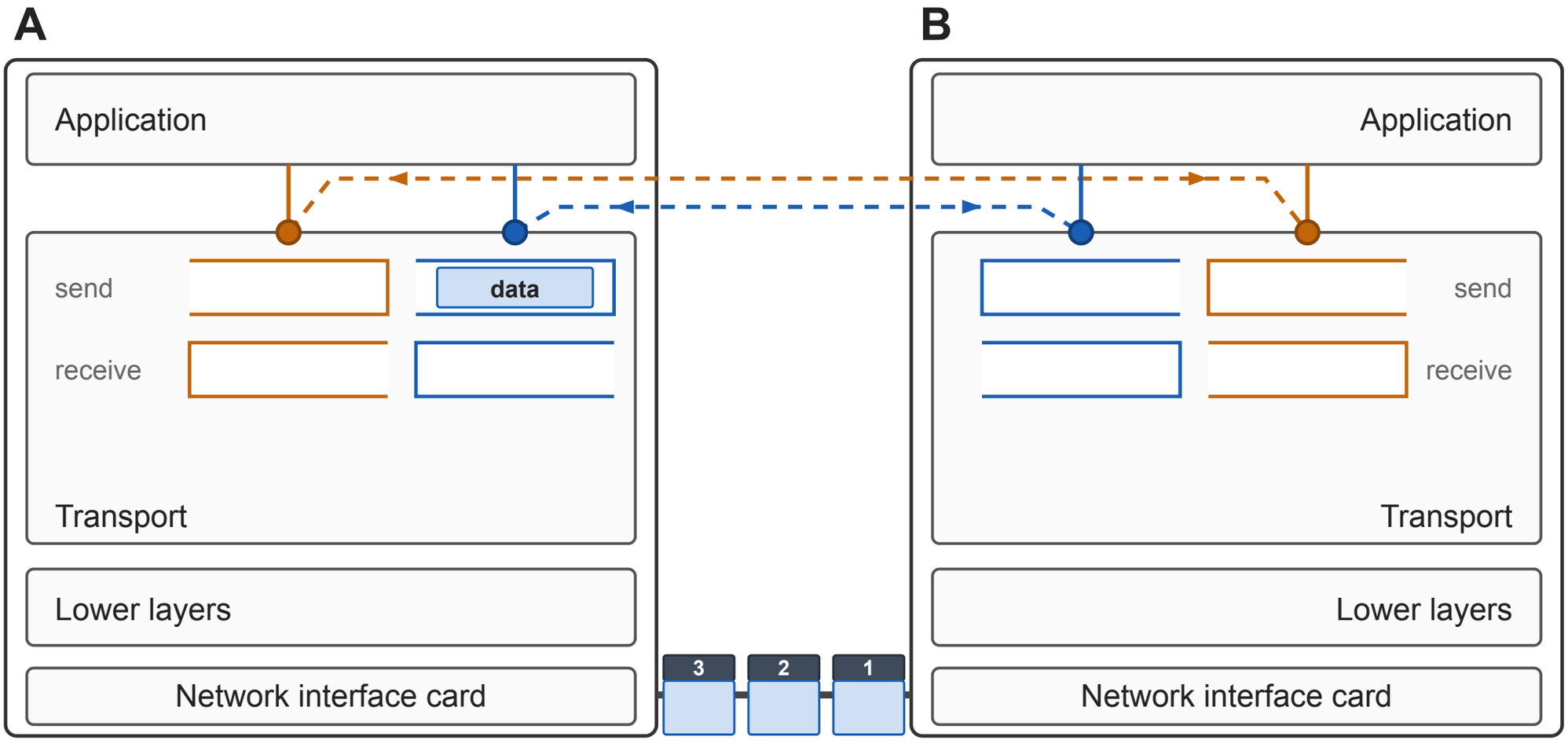
What happens to the packets?



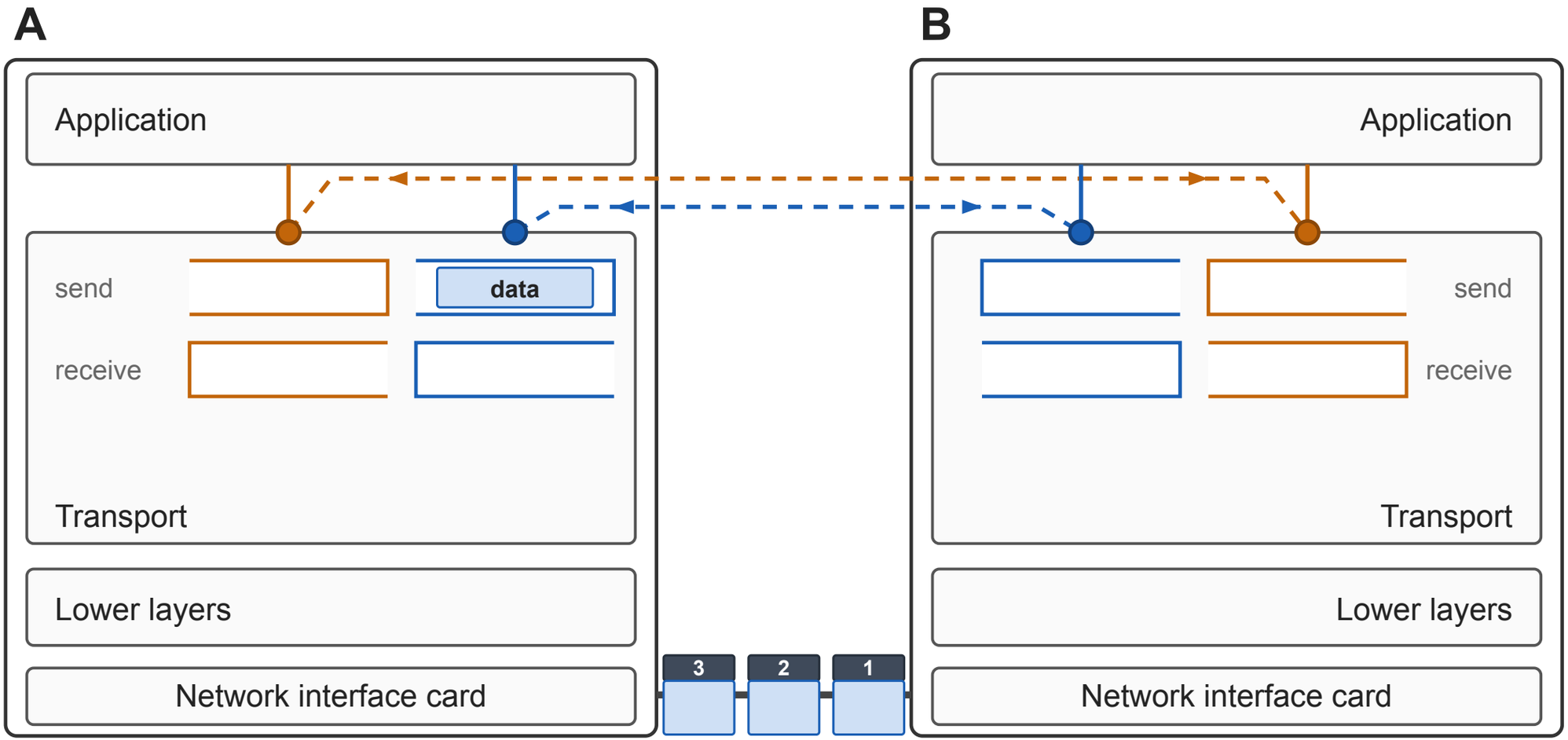
What happens to the packets?



What happens to the packets?



What happens to the packets?

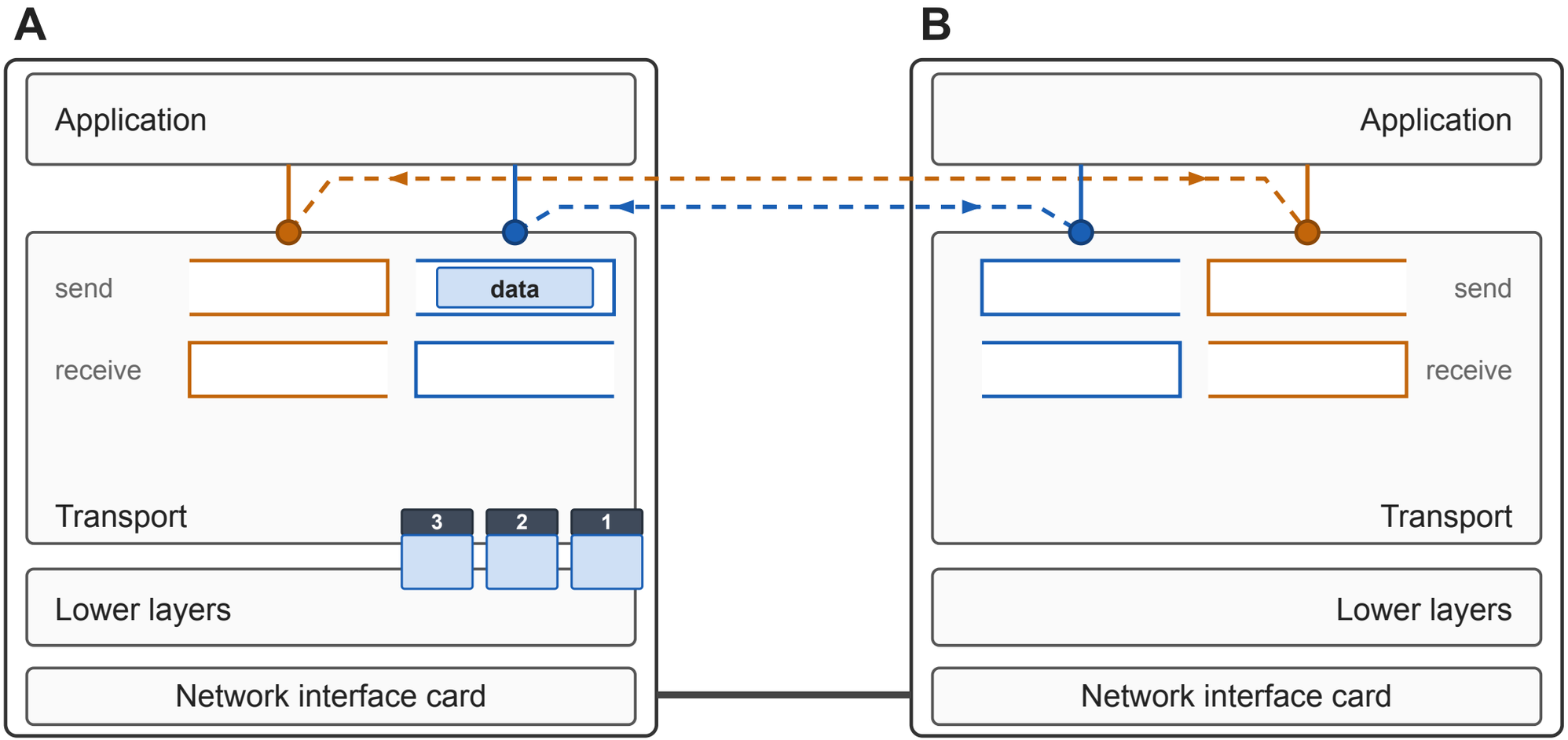


The lower layers may corrupt, reorder, or drop these packets

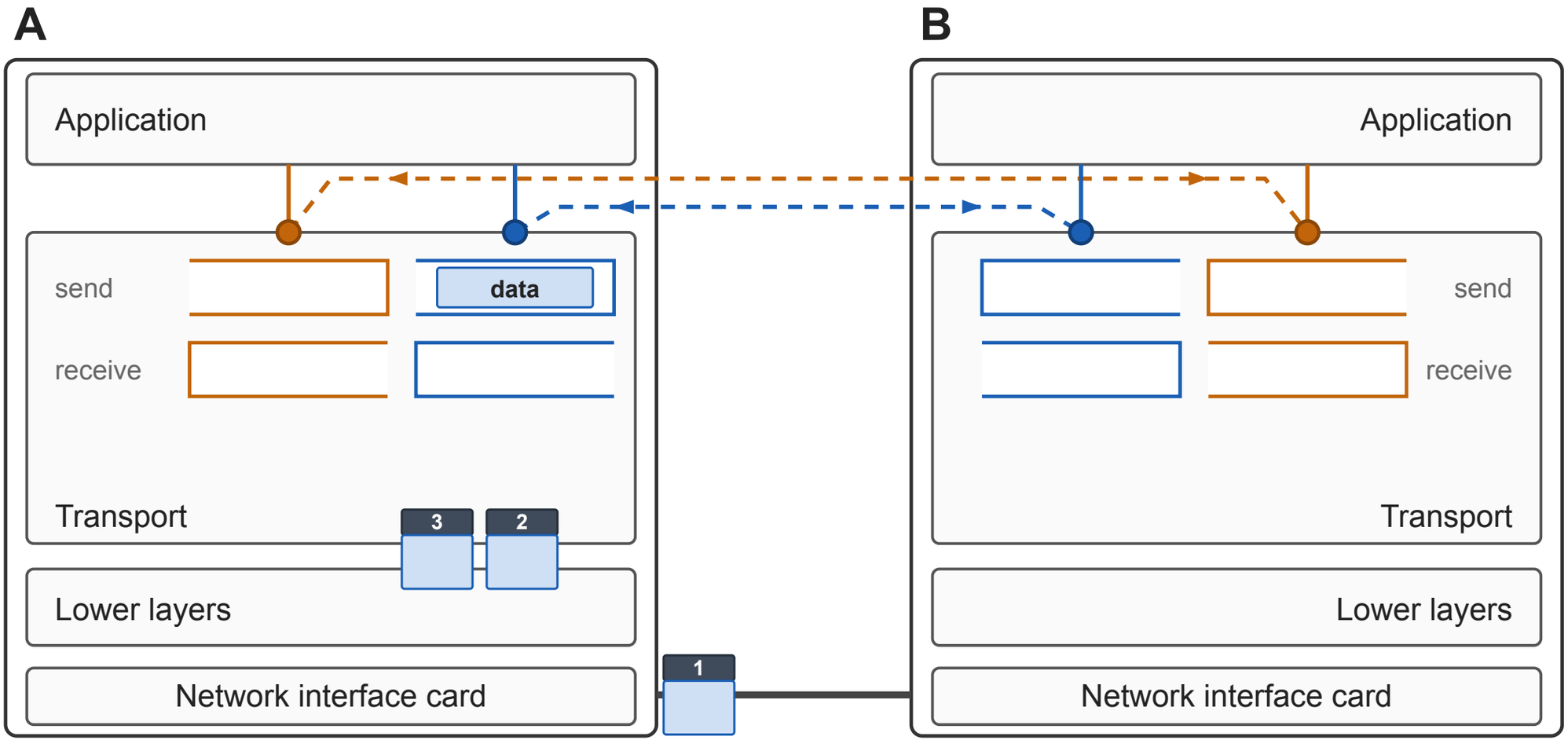
First, one packet at a time

A’s transport layer makes sure one packet has successfully been received at B, and only then moves on to the next one.

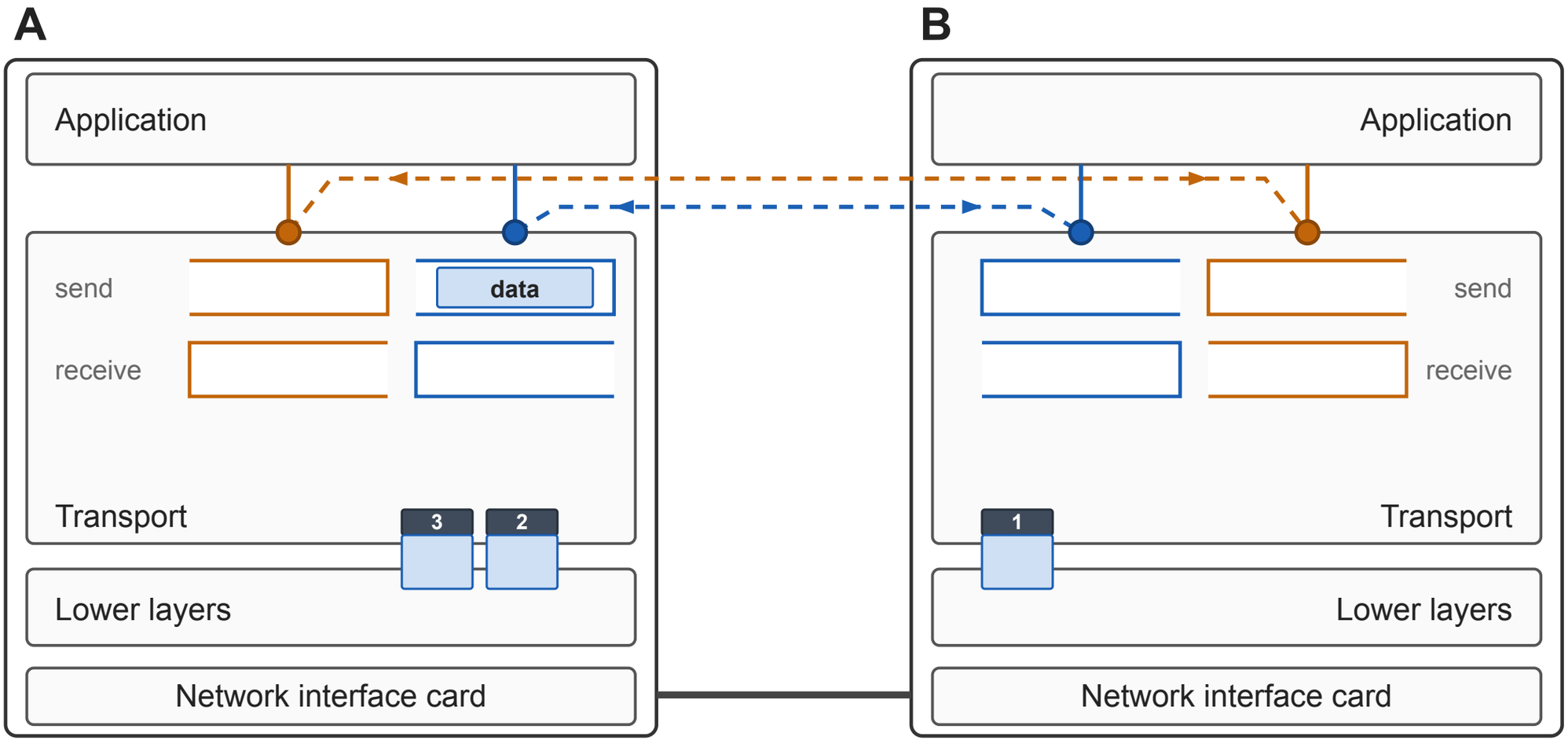
One packet at a time



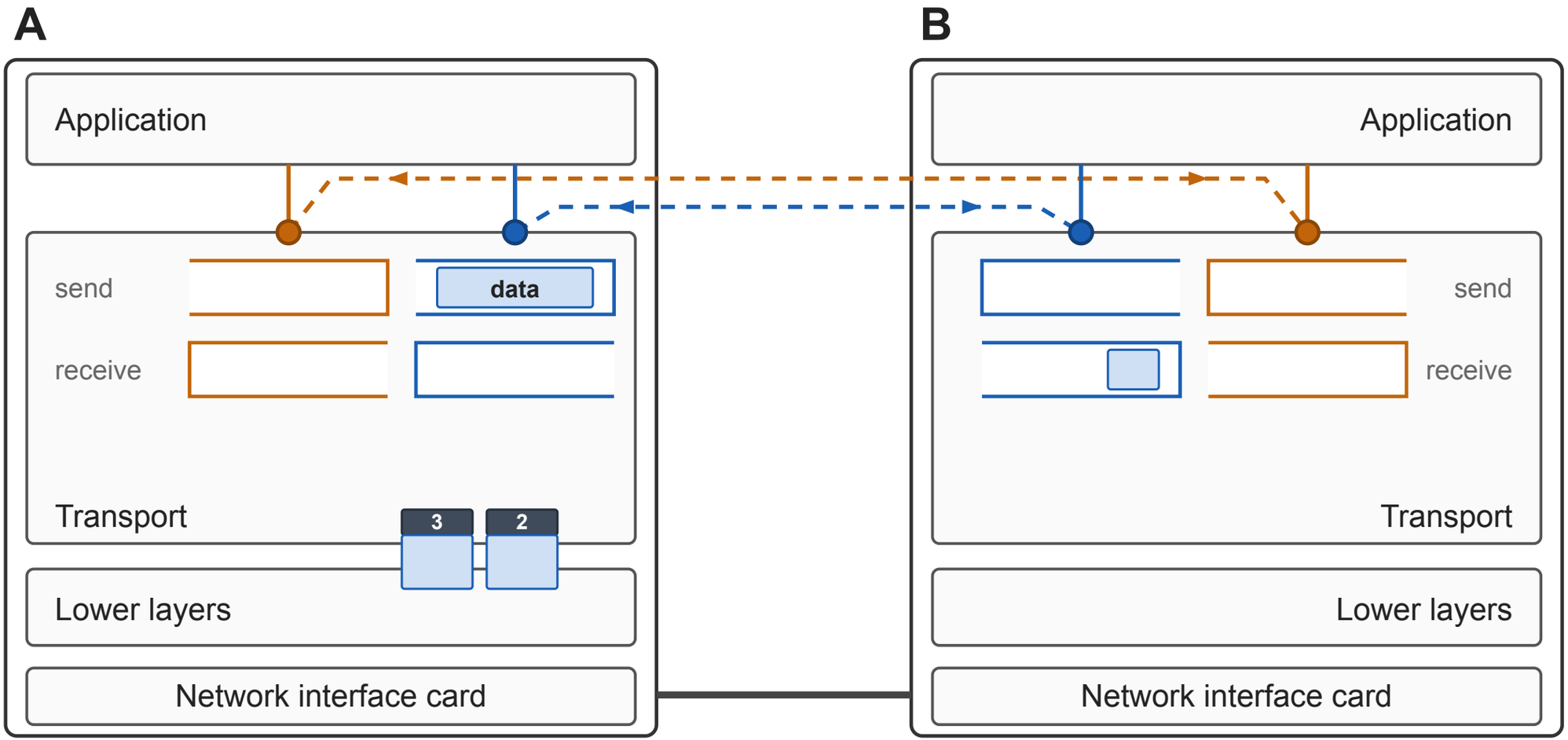
One packet at a time



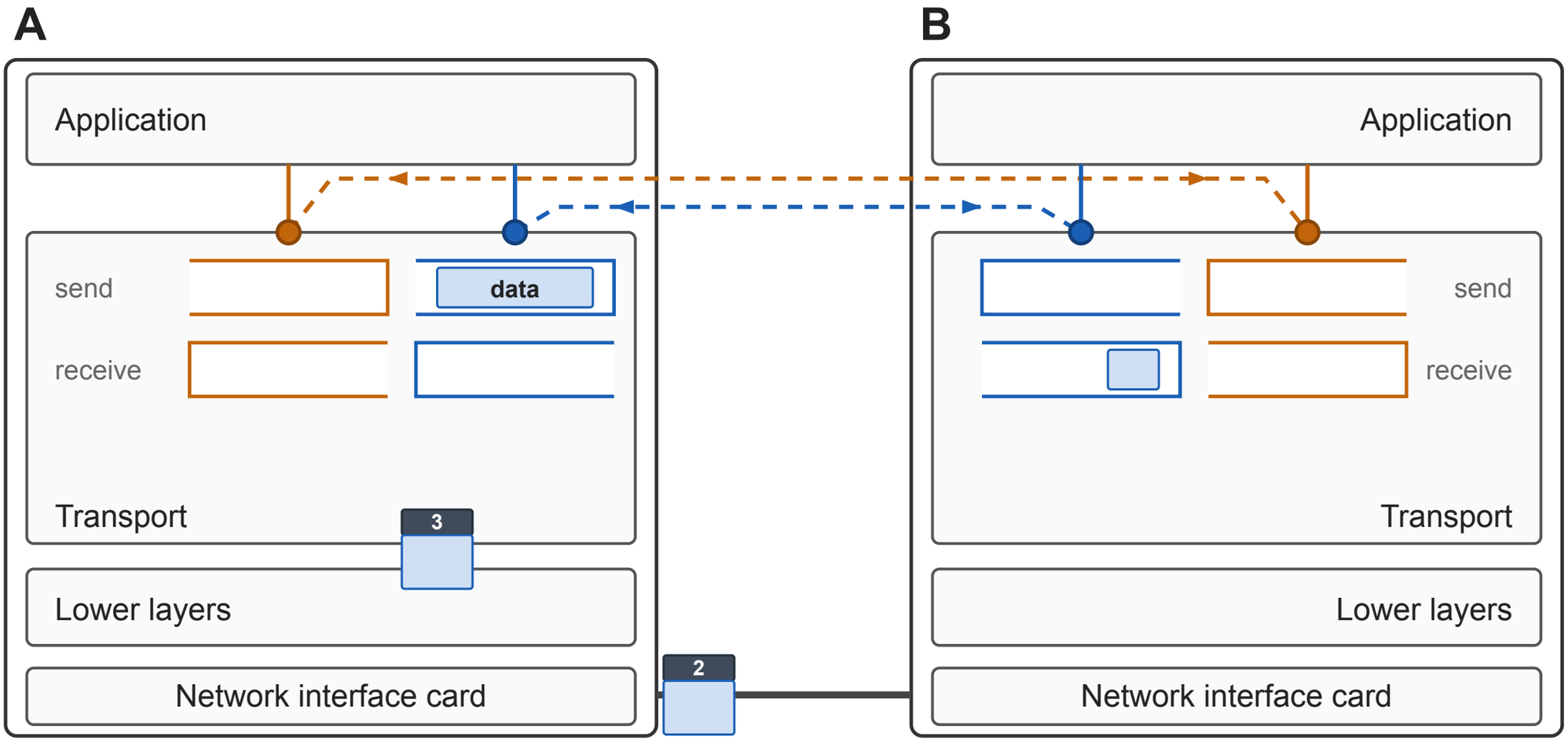
One packet at a time



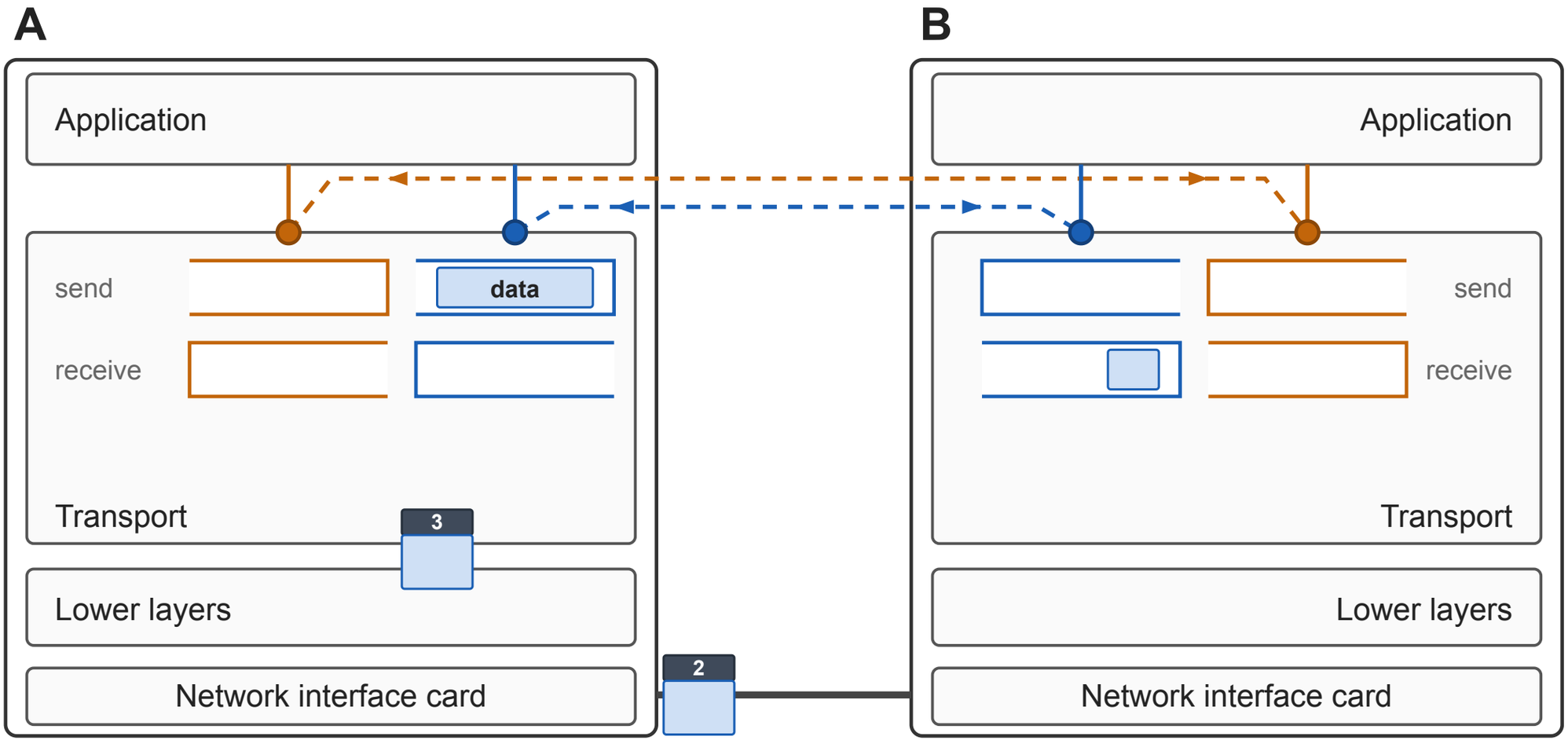
One packet at a time



One packet at a time

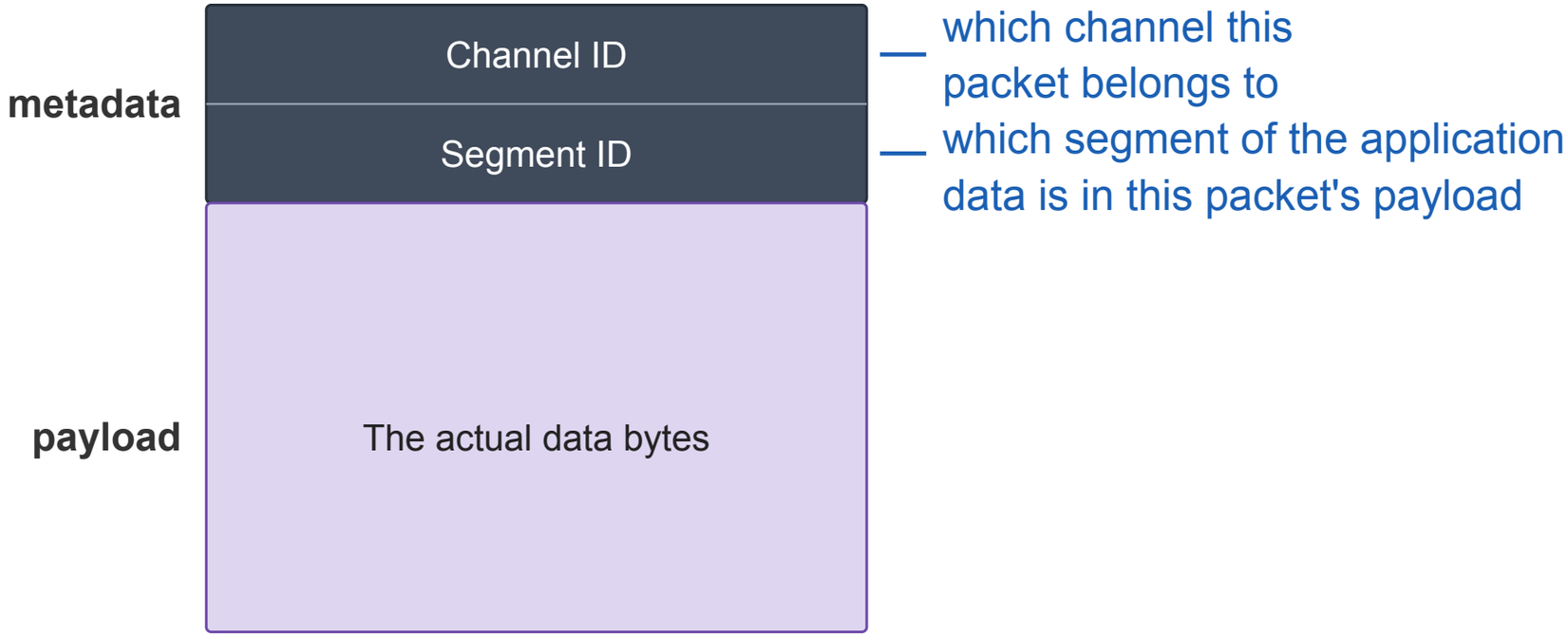


One packet at a time



What should the protocol be, so that this one packet gets to B?

Reminder: transport metadata so far



Inside the transport layer, so far

This is pseudocode. Real implementations optimize quite a bit and look different.

`to_lower(pkt)` hands a packet to the lower layers. `from_lower()` blocks until they provide a received packet.

(*) If the segments are not all `max_payload_size` long, this does not quite work. We overlook that for now.

Transport at A (sender)

```
buffer send_buf[];   // the data, one per channel
int next_id[];       // next segment ID, per channel

void net_send(ch, data, n) {
    send_buf[ch].append(data, n);
}

void send_loop(ch) {
    while (true) {
        seg = send_buf[ch].take(max_payload_size);
        pkt = add_header(seg, ch, next_id[ch]);
        next_id[ch] = next_id[ch] + 1;
        to_lower(pkt);
    }
}
```

Transport at B (receiver)

```
buffer recv_buf[];   // the data, one per channel

void net_recv(ch, data, n) {
    recv_buf[ch].take(data, n);
}

void receive_loop() {
    while (true) {
        pkt = from_lower();
        seg = strip_header(pkt);
        offset = pkt.seg_id * max_payload_size;  // (*)
        recv_buf[pkt.ch].write(offset, seg);
    }
}
```

Inside the transport layer, so far

This is pseudocode. Real implementations optimize quite a bit and look different.

`to_lower(pkt)` hands a packet to the lower layers. `from_lower()` blocks until they provide a received packet.

(*) If the segments are not all `max_payload_size` long, this does not quite work. We overlook that for now.

Transport at A (sender)

```
buffer send_buf[];   // the data, one per channel
int next_id[];       // next segment ID, per channel

void net_send(ch, data, n) {
    send_buf[ch].append(data, n);
}

void send_loop(ch) {
    while (true) {
        seg = send_buf[ch].take(max_payload_size);
        pkt = add_header(seg, ch, next_id[ch]);
        next_id[ch] = next_id[ch] + 1;
        to_lower(pkt);
    }
}
```

Transport at B (receiver)

```
buffer recv_buf[];   // the data, one per channel

void net_recv(ch, data, n) {
    recv_buf[ch].take(data, n);
}

void receive_loop() {
    while (true) {
        pkt = from_lower();
        seg = strip_header(pkt);
        offset = pkt.seg_id * max_payload_size;  // (*)
        recv_buf[pkt.ch].write(offset, seg);
    }
}
```

If the link is perfect (no corruption or loss) and B has infinite memory, this will be fine.

Dealing with bit corruption

Every packet still arrives, in the order it was sent. But some of its bits may be flipped on the way.

Dealing with bit corruption

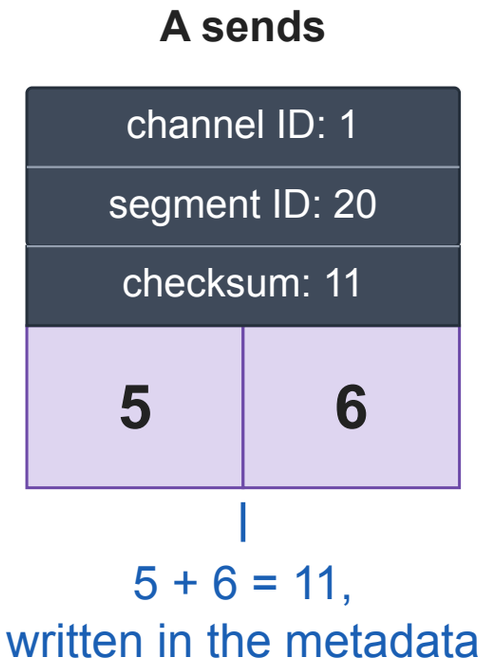
Every packet still arrives, in the order it was sent. But some of its bits may be flipped on the way.

> How can an endpoint tell that a packet it received is corrupted?

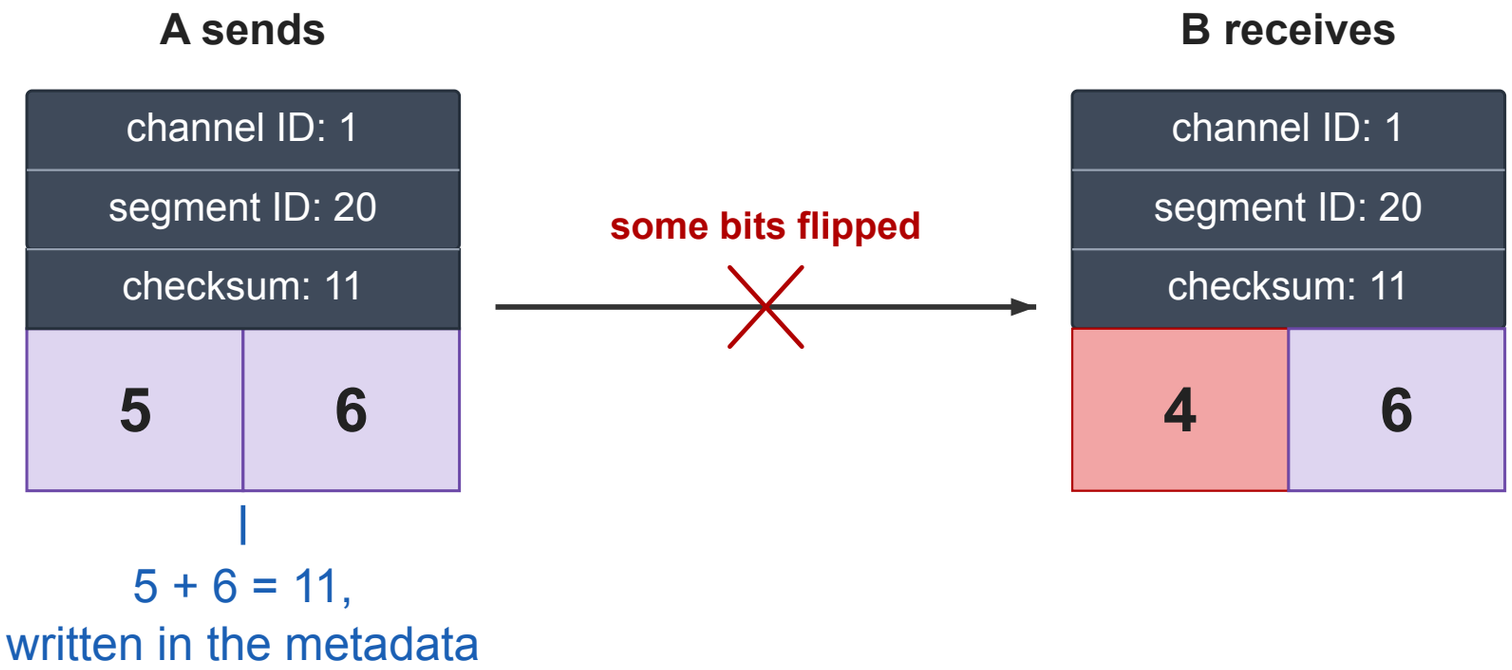
Checksum

A sends

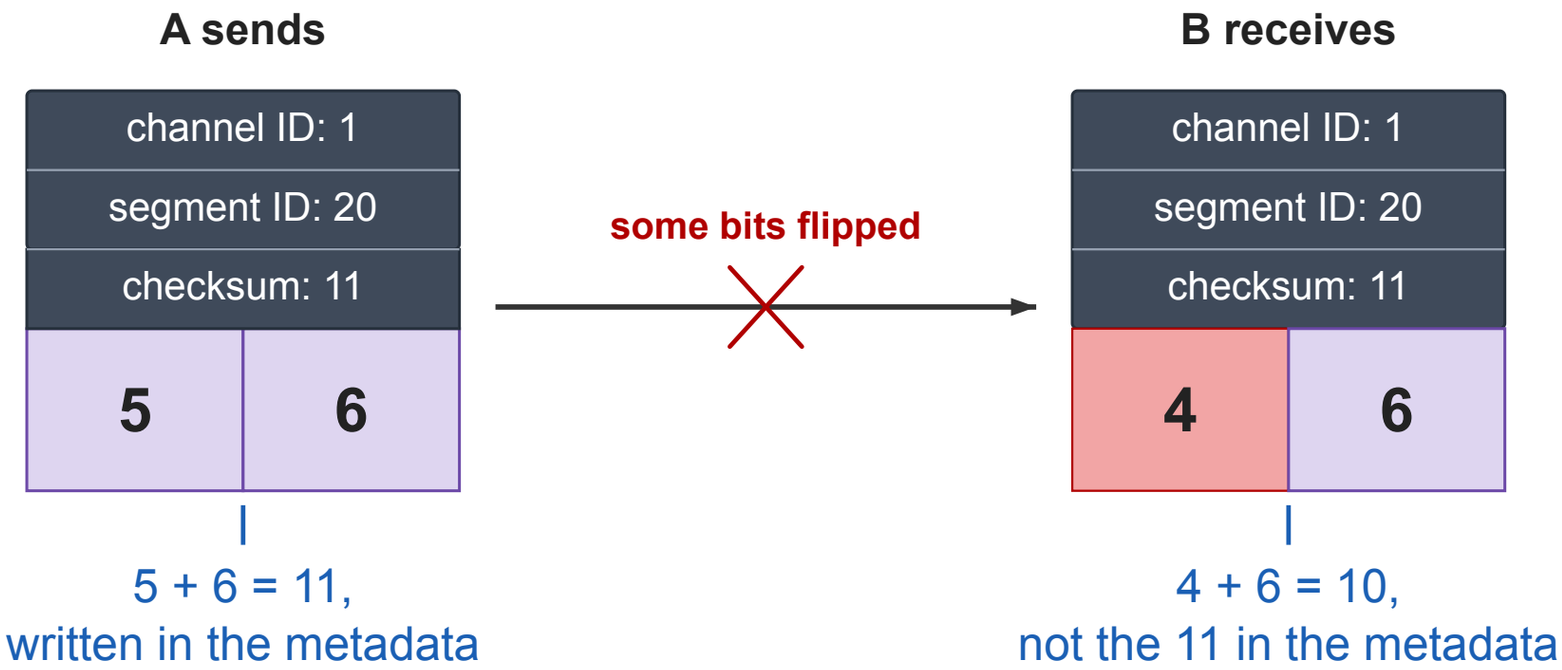
Checksum


A sends
channel ID: 1
segment ID: 20
checksum: 11
5
6
5 + 6 = 11,
written in the metadata

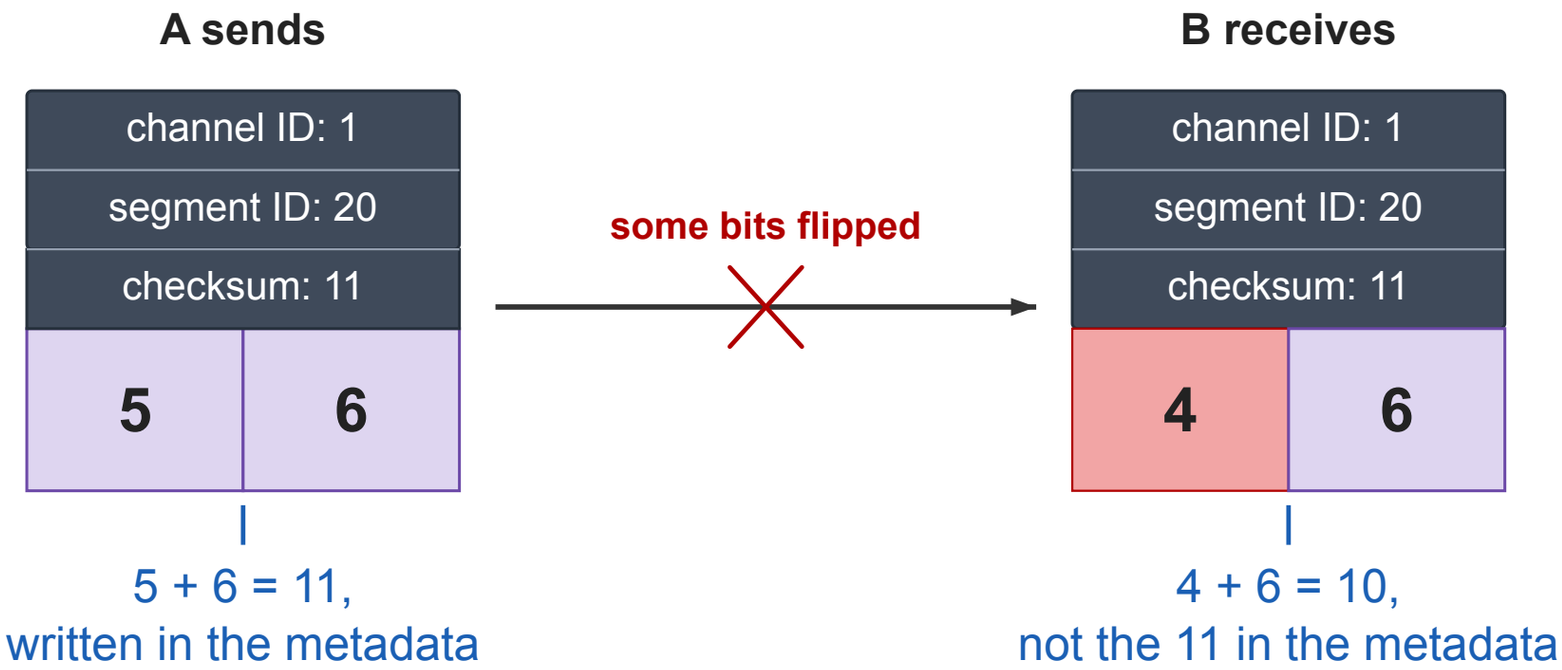
Checksum


A sends
channel ID: 1
segment ID: 20
checksum: 11
5
6
5 + 6 = 11,
written in the metadata
some bits flipped
B receives
channel ID: 1
segment ID: 20
checksum: 11
4
6

Checksum


A sends
channel ID: 1
segment ID: 20
checksum: 11
5
6
5 + 6 = 11,
written in the metadata
some bits flipped
B receives
channel ID: 1
segment ID: 20
checksum: 11
4
6
4 + 6 = 10,
not the 11 in the metadata

Checksum



B knows some bits flipped on the way (not which ones, though)

The Internet checksum

- **Sender:**
  - treat the metadata and payload as a sequence of 16-bit integers
  - add them all up, with one’s complement addition
  - put the complement of the sum in the metadata, as the checksum
- **Receiver:** compute the same thing over what arrived, and compare it with the checksum in the metadata
  - not equal: an error is detected
  - equal: no error is detected (it does not mean there was no error)

Internet checksum: an example

two 16-bit integers

```
  0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
+ 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
-----------------------------------
```

Internet checksum: an example

two 16-bit integers

```
  0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
+ 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
-----------------------------------
```

their sum

```
  1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
```

Internet checksum: an example

two 16-bit integers

```
  0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
+ 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
-----------------------------------
```

their sum

```
  1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
```

checksum: flip every bit

```
  0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
```

this goes in the packet's metadata

Internet checksum does not always detect bit errors

A sends

two 16-bit integers

```
  0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
+ 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
-----------------------------------
```

their sum

```
  1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
```

checksum: flip every bit

```
  0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
```

Internet checksum does not always detect bit errors

A sends

two 16-bit integers

```
  0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
+ 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
-----------------------------------
```

their sum

```
  1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
```

checksum: flip every bit

```
  0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
```

B receives

two bits flipped

```
  0 1 0 0 0 1 1 0 0 1 1 0 0 1 1 0
+ 0 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1
-----------------------------------
```

Internet checksum does not always detect bit errors

A sends

two 16-bit integers

```
  0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
+ 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
-----------------------------------
```

their sum

```
  1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
```

checksum: flip every bit

```
  0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
```

B receives

two bits flipped

```
  0 1 0 0 0 1 1 0 0 1 1 0 0 1 1 0
+ 0 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1
-----------------------------------
```

the same sum

```
  1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
```

so the same checksum

```
  0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
```

Internet checksum does not always detect bit errors

A sends

two 16-bit integers

```
  0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
+ 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
-----------------------------------
```

their sum

```
  1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
```

checksum: flip every bit

```
  0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
```

B receives

two bits flipped

```
  0 1 0 0 0 1 1 0 0 1 1 0 0 1 1 0
+ 0 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1
-----------------------------------
```

the same sum

```
  1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
```

so the same checksum

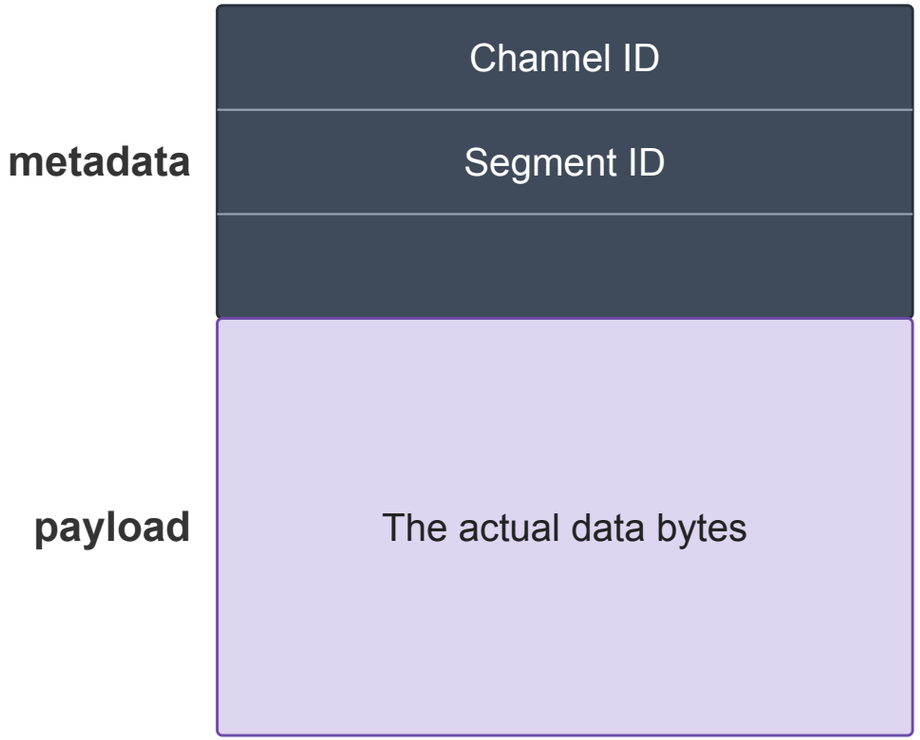
```
  0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
```

Although the bits changed, the checksum did not change, so this corruption goes unnoticed

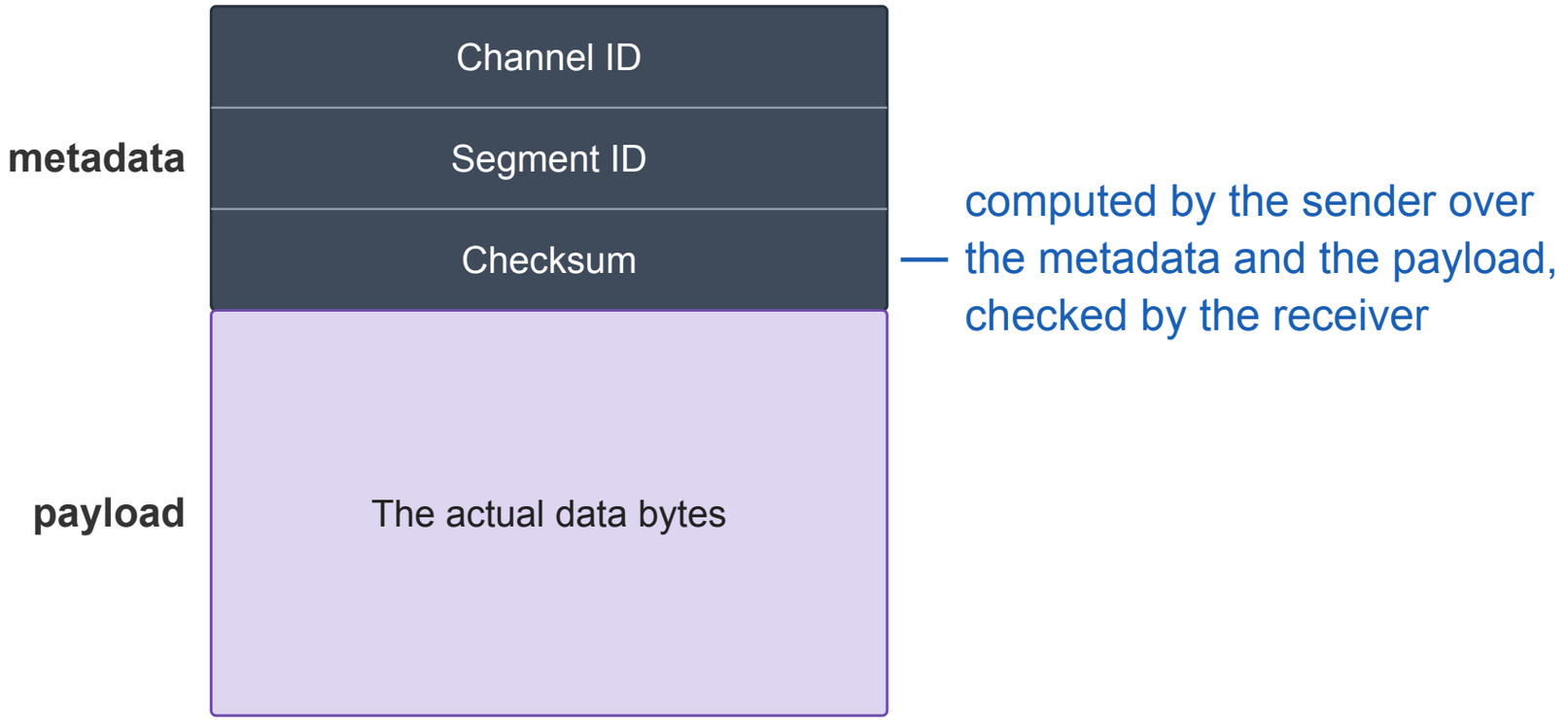
Who do you think should deal with unnoticed corruption?

Reset

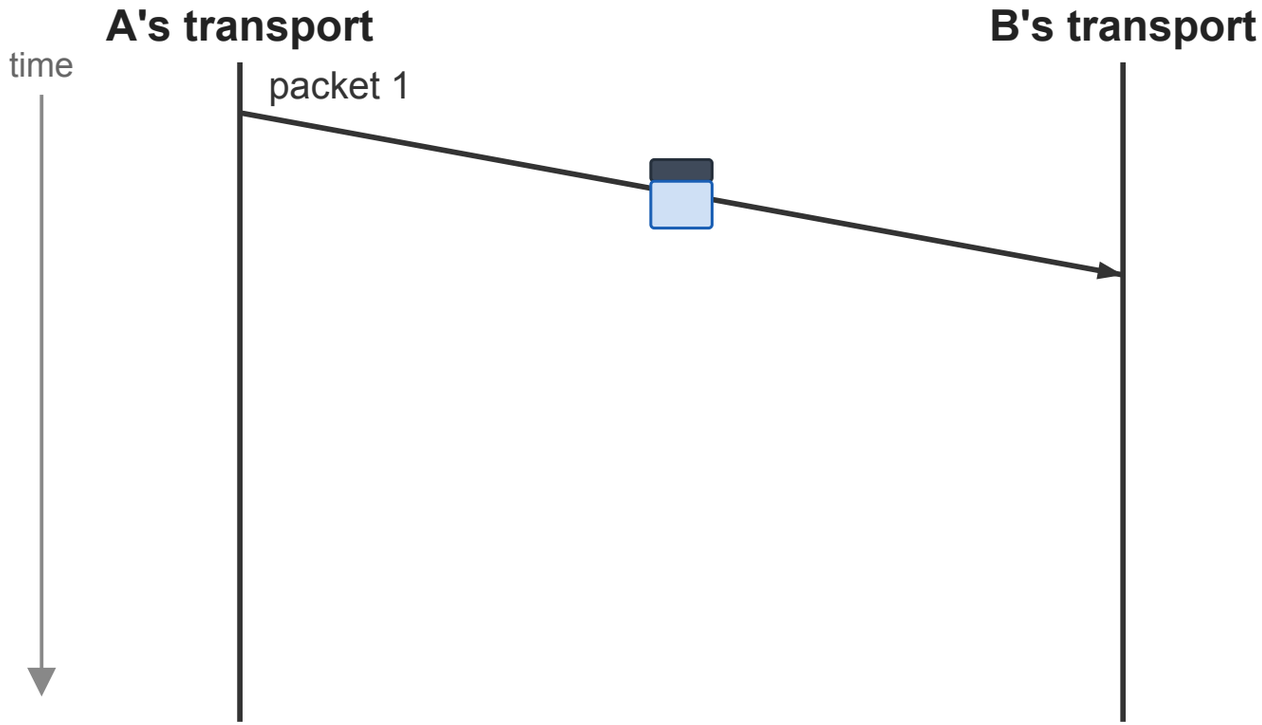
The packet, with a checksum



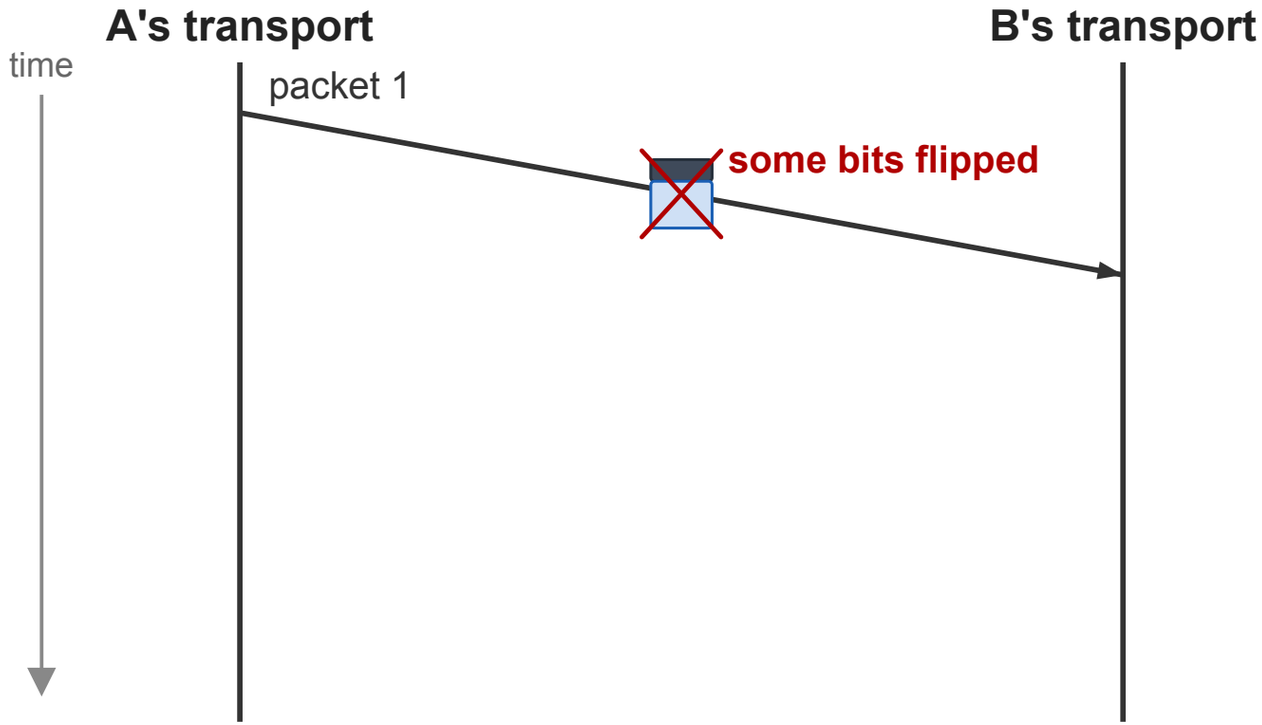
The packet, with a checksum



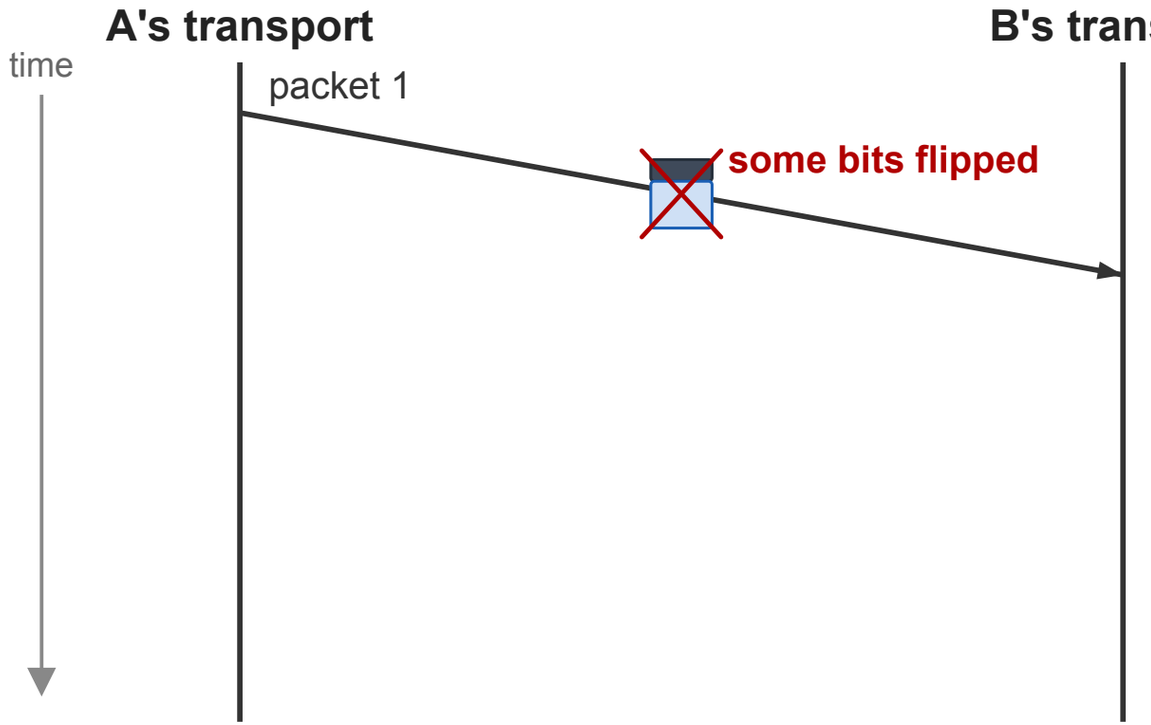
When the data packet is corrupted



When the data packet is corrupted

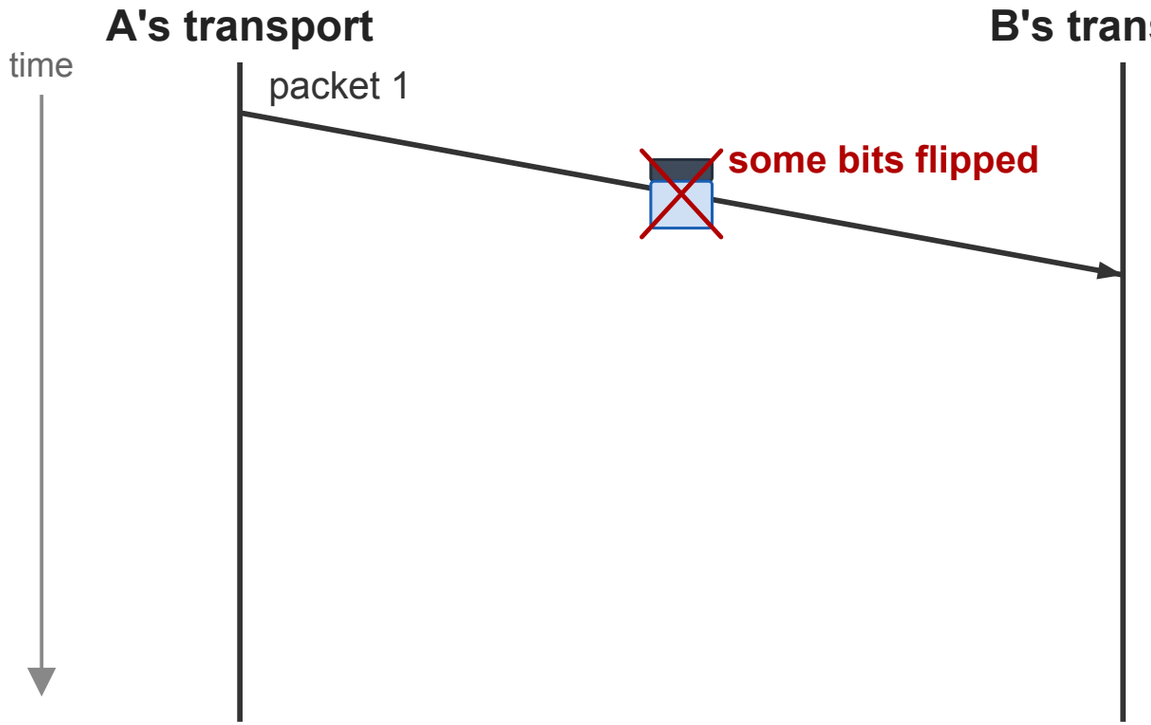


When the data packet is corrupted



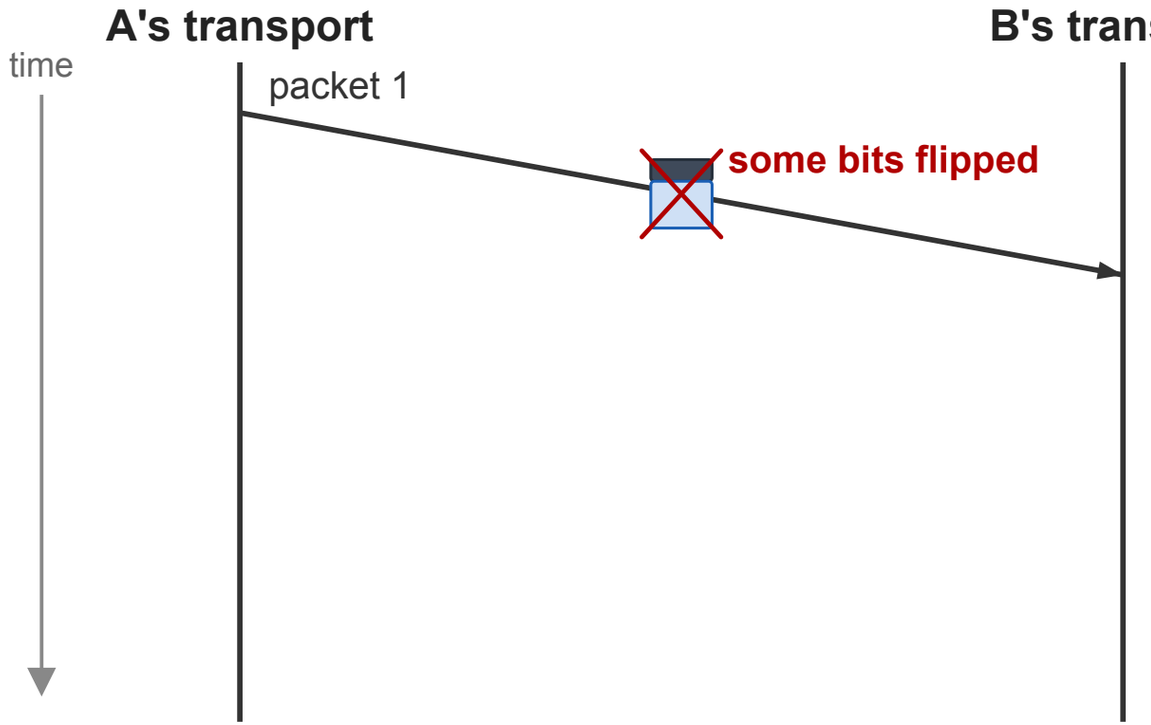
How does B tell that the packet is corrupted?

When the data packet is corrupted



How does B tell that the packet is corrupted?

When the data packet is corrupted



How does B tell that the packet is corrupted?

What should B do?