dr. heap

Netherlands Defence Academy NLDA

The Transport Layer

Lecture 6

Computer Networks

Hans-Dieter A. Hiep

17 September, 2026

CC BY 4.0

The content reflects solely the findings of the presenter and should not be interpreted as representing the views of the Ministry of Defence, the Netherlands Defence Academy (NLDA), or any other governmental institution.

Examination

- **Thursday 15 October, 8:30 – 10:30**. Two hours, written examination.
- It covers all ten lecture topics and the corresponding book chapters.
- Combination of multiple choice and open questions.
- **Open book**, no electronic devices, no calculator, no notes, no slides.
- Multiple choice questions test your **comprehension** ability.
- Open questions focus on your **explaination** ability.
- End of each book chapter: **example problems**.

Examination

- **Thursday 15 October, 8:30 – 10:30**. Two hours, written examination.
- It covers all ten lecture topics and the corresponding book chapters.
- Combination of multiple choice and open questions.
- **Open book**, no electronic devices, no calculator, no notes, no slides.
- Multiple choice questions test your **comprehension** ability.
- Open questions focus on your **explaination** ability.
- End of each book chapter: **example problems**.

Examination

- **Thursday 15 October, 8:30 – 10:30**. Two hours, written examination.
- It covers all ten lecture topics and the corresponding book chapters.
- Combination of multiple choice and open questions.
- **Open book**, no electronic devices, no calculator, no notes, no slides.
- Multiple choice questions test your **comprehension** ability.
- Open questions focus on your **explaination** ability.
- End of each book chapter: **example problems**.

Examination

- **Thursday 15 October, 8:30 – 10:30**. Two hours, written examination.
- It covers all ten lecture topics and the corresponding book chapters.
- Combination of multiple choice and open questions.
- **Open book**, no electronic devices, no calculator, no notes, no slides.
- Multiple choice questions test your **comprehension** ability.
- Open questions focus on your **explaination** ability.
- End of each book chapter: **example problems**.

What a good answer looks like

- Name all the **mechanisms**, **protocols**, etc. involved.
- Hint: **which layer are you on**, and how it relates to the layers above/below.

- When computing an outcome, always show the computation steps.
- When a picture helps your case, draw it.

What a good answer looks like

- Name all the **mechanisms**, **protocols**, etc. involved.
- Hint: **which layer are you on**, and how it relates to the layers above/below.

- When computing an outcome, always show the computation steps.
- When a picture helps your case, draw it.

Example: Lecture 1

- An autonomous system unexpectedly loses its **upstream transit provider**, but keeps two peering sessions with neighbouring ASes at a local internet exchange point. Which destinations can it still reach, which can it not, and why? Answer in technical terms involving forwarding tables and BGP announcements.

- You are part of a **European defence network task force**, to design and roll out a computer network suitable for border protection, air defence, and command and control functions. What network architecture do you choose, and what are your requirements? How can you ensure survivability while facing external threats *and* internal sabotage?

Example: Lecture 1

- An autonomous system unexpectedly loses its **upstream transit provider**, but keeps two peering sessions with neighbouring ASes at a local internet exchange point. Which destinations can it still reach, which can it not, and why? Answer in technical terms involving forwarding tables and BGP announcements.

- You are part of a **European defence network task force**, to design and roll out a computer network suitable for border protection, air defence, and command and control functions. What network architecture do you choose, and what are your requirements? How can you ensure survivability while facing external threats *and* internal sabotage?

Example: Lecture 2

- One computer system uses **two** distinguishable values per symbol (bits), while another system uses **three** distinguishable values per symbol (trits). In each case, how many symbols do you need to discriminate five situations, i.e. how do you encode 5 values? Moreover, explain the difficulty of communicating between bits-based and trits-based computers.
- (Given some network as a graph with capacities on the edges. Two locations are marked: these are headquarters.) Design an operation that results in the **cheapest cut** separating the two headquarters. Where would you put extra effort in surveilling the network, to ensure an adversary cannot separate them?

Example: Lecture 2

- One computer system uses **two** distinguishable values per symbol (bits), while another system uses **three** distinguishable values per symbol (trits). In each case, how many symbols do you need to discriminate five situations, i.e. how do you encode 5 values? Moreover, explain the difficulty of communicating between bits-based and trits-based computers.
- (Given some network as a graph with capacities on the edges. Two locations are marked: these are headquarters.) Design an operation that results in the **cheapest cut** separating the two headquarters. Where would you put extra effort in surveilling the network, to ensure an adversary cannot separate them?

Example: Lecture 3

- Recall: $C = B \log_2(1 + S/N)$, and 10 dB is a ratio of $10 : 1$. A channel is 10 MHz wide with a signal-to-noise ratio of 30 dB. Give an upper bound on its capacity. Explain why **bandwidth is not capacity** in terms of widening the channel to 20 MHz or narrowing it to 5 MHz.
- We want one fibre to carry eight different channels. Explain how **wavelength division** differs from **time division** multiplexing, and name what (physically) limits each.

Example: Lecture 3

- Recall: $C = B \log_2(1 + S/N)$, and 10 dB is a ratio of $10 : 1$. A channel is 10 MHz wide with a signal-to-noise ratio of 30 dB. Give an upper bound on its capacity. Explain why **bandwidth is not capacity** in terms of widening the channel to 20 MHz or narrowing it to 5 MHz.
- We want one fibre to carry eight different channels. Explain how **wavelength division** differs from **time division** multiplexing, and name what (physically) limits each.

Example: Lecture 4

- Can you argue why Ethernet has a **minimum frame size** of 64 bytes? What are possible issues with smaller frames?
- An Ethernet **bridge** that learns MAC addresses receives a frame for a destination it has never seen before. Explain: what does it do next? What happens to the reply? (You can use **ARP** as an example protocol.)

Example: Lecture 4

- Can you argue why Ethernet has a **minimum frame size** of 64 bytes? What are possible issues with smaller frames?
- An Ethernet **bridge** that learns MAC addresses receives a frame for a destination it has never seen before. Explain: what does it do next? What happens to the reply? (You can use **ARP** as an example protocol.)

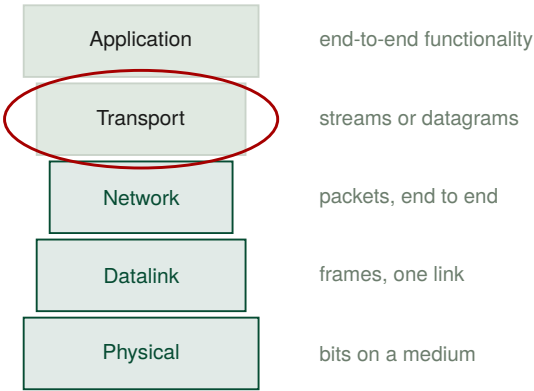
Example: Lecture 5

- A router's **forwarding table** holds `10.0.0.0/8`, `10.1.0.0/16` and `0.0.0.0/0` with each a different next hop. A packet arrives for `10.1.2.3`. Which entry forwards it, and what is the mechanism called?
- A 4000-byte datagram must cross a network where there is a link with an **MTU** of 1500. Explain what happens in IPv4. How can this mechanism be abused?

Example: Lecture 5

- A router's **forwarding table** holds `10.0.0.0/8`, `10.1.0.0/16` and `0.0.0.0/0` with each a different next hop. A packet arrives for `10.1.2.3`. Which entry forwards it, and what is the mechanism called?
- A 4000-byte datagram must cross a network where there is a link with an **MTU** of 1500. Explain what happens in IPv4. How can this mechanism be abused?

Where we are


Application
end-to-end functionality
Transport
streams or datagrams
Network
packets, end to end
Datalink
frames, one link
Physical
bits on a medium

Recall: what the network layer does not do

packets arrive out of order	→ sequence numbers
packets do not arrive at all	→ acknowledgement, retransmission
capacity utilisation and congestion	→ flow and congestion control
one address, many applications	→ ports

The network layer delivers **some** packets, in **some** order, to **one** computer. The transport layer is what you build on that.

Overview

1. Ports
2. Transport protocols
3. Congestion control
4. Quality of service
5. Towards the application layer
6. Problems

Overview

1. Ports

2. Transport protocols

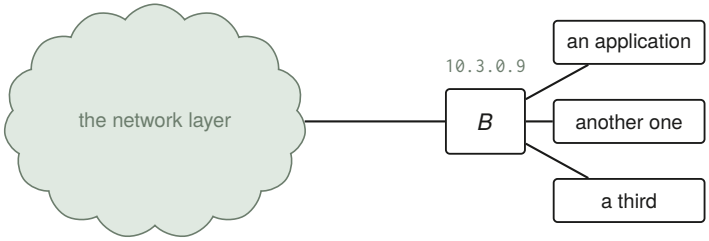
3. Congestion control

4. Quality of service

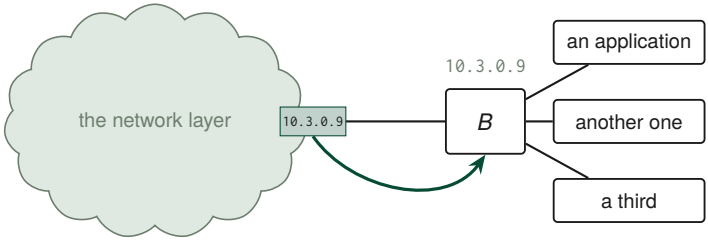
5. Towards the application layer

6. Problems

One address, many applications

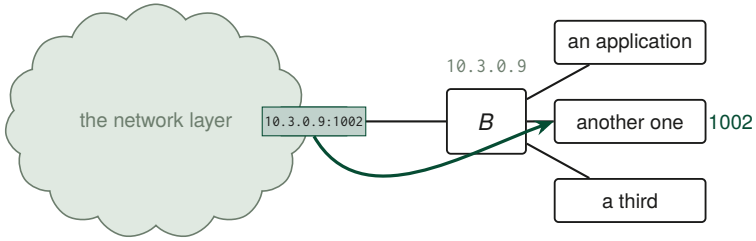


One address, many applications



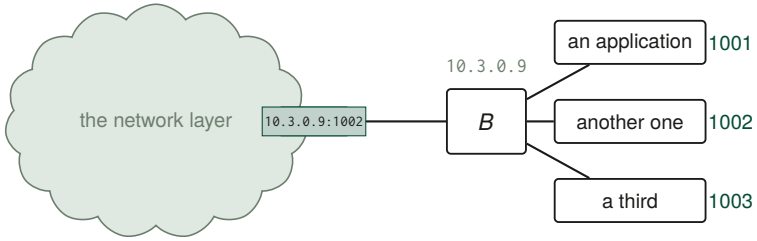
The address picks the computer, not
the particular application running atop.

One address, many applications



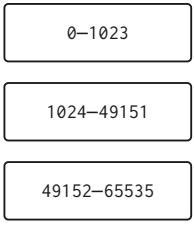
A second number, the **port**, picks the application.
A colon separates it from the address.

One address, many applications



One computer (= **host**), many applications (= **processes**).

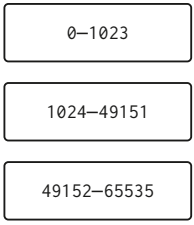
The port numbers themselves



Two bytes in the header.

- Well-known and registered ports have a specific meaning, agreed in advance.
- A port number and service registry is maintained by **IANA**, the same body that manages the IP address space.

The port numbers themselves



Two bytes in the header.

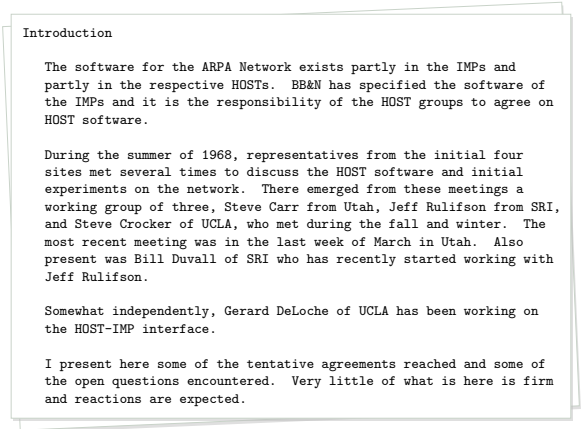
- Well-known and registered ports have a specific meaning, agreed in advance.
- A port number and service registry is maintained by **IANA**, the same body that manages the IP address space.

Some well-known port numbers

port	transport protocol	service
7	TCP, UDP	echo: send it back
9	TCP, UDP, DCCP, SCTP	discard: throw it away
21	TCP, SCTP	FTP: file transfer
22	TCP, SCTP	SSH: secure shell
53	TCP, UDP	DNS: domain names
80	TCP, SCTP	HTTP: the web
...	...	...

Service Name and Transport Protocol Port Number Registry (IANA)

Request For Comments (RFCs)



Introduction

The software for the ARPA Network exists partly in the IMPs and partly in the respective HOSTs. BB&N has specified the software of the IMPs and it is the responsibility of the HOST groups to agree on HOST software.

During the summer of 1968, representatives from the initial four sites met several times to discuss the HOST software and initial experiments on the network. There emerged from these meetings a working group of three, Steve Carr from Utah, Jeff Rulifson from SRI, and Steve Crocker of UCLA, who met during the fall and winter. The most recent meeting was in the last week of March in Utah. Also present was Bill Duvall of SRI who has recently started working with Jeff Rulifson.

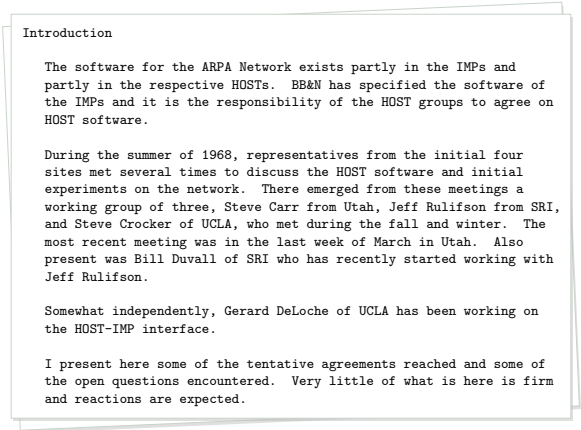
Somewhat independently, Gerard DeLoche of UCLA has been working on the HOST-IMP interface.

I present here some of the tentative agreements reached and some of the open questions encountered. Very little of what is here is firm and reactions are expected.

Host Software Crocker (1969)

- **April 1969, RFC 1**. The first of the series, written by a graduate student.
- *Very little of what is here is firm and reactions are expected.*
- Hence the name the series still carries: **Request for Comments**.

Request For Comments (RFCs)



Introduction

The software for the ARPA Network exists partly in the IMPs and partly in the respective HOSTs. BB&N has specified the software of the IMPs and it is the responsibility of the HOST groups to agree on HOST software.

During the summer of 1968, representatives from the initial four sites met several times to discuss the HOST software and initial experiments on the network. There emerged from these meetings a working group of three, Steve Carr from Utah, Jeff Rulifson from SRI, and Steve Crocker of UCLA, who met during the fall and winter. The most recent meeting was in the last week of March in Utah. Also present was Bill Duvall of SRI who has recently started working with Jeff Rulifson.

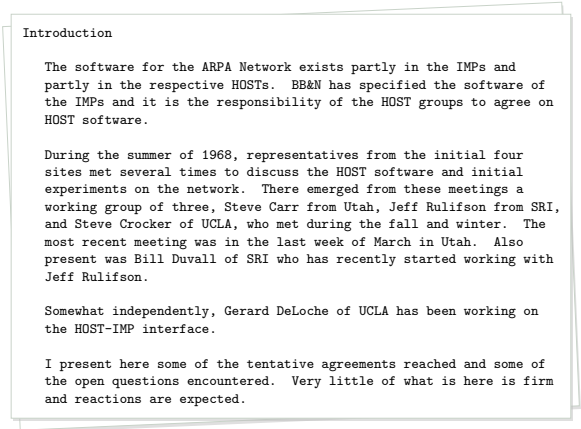
Somewhat independently, Gerard DeLoche of UCLA has been working on the HOST-IMP interface.

I present here some of the tentative agreements reached and some of the open questions encountered. Very little of what is here is firm and reactions are expected.

Host Software Crocker (1969)

- **April 1969, RFC 1**. The first of the series, written by a graduate student.
- *Very little of what is here is firm and reactions are expected.*
- Hence the name the series still carries: **Request for Comments**.

Request For Comments (RFCs)



Introduction

The software for the ARPA Network exists partly in the IMPs and partly in the respective HOSTs. BB&N has specified the software of the IMPs and it is the responsibility of the HOST groups to agree on HOST software.

During the summer of 1968, representatives from the initial four sites met several times to discuss the HOST software and initial experiments on the network. There emerged from these meetings a working group of three, Steve Carr from Utah, Jeff Rulifson from SRI, and Steve Crocker of UCLA, who met during the fall and winter. The most recent meeting was in the last week of March in Utah. Also present was Bill Duvall of SRI who has recently started working with Jeff Rulifson.

Somewhat independently, Gerard DeLoche of UCLA has been working on the HOST-IMP interface.

I present here some of the tentative agreements reached and some of the open questions encountered. Very little of what is here is firm and reactions are expected.

Host Software Crocker (1969)

- **April 1969, RFC 1**. The first of the series, written by a graduate student.
- *Very little of what is here is firm and reactions are expected.*
- Hence the name the series still carries: **Request for Comments**.

The first specification of TCP

RFC 675 Specification of Internet TCP December 1974

2.2 Sockets and Addressing

We have borrowed the term SOCKET from the ARPANET terminology [CACR70, MCKE73]. In general, a socket is the concatenation of a NETWORK identifier, TCP identifier, and PORT identifier. A CONNECTION is fully specified by the pair of SOCKETS at each end since the same local socket may participate in many connections to different foreign sockets.

Specification of Internet Transmission Control Program Cerf, Dalal and Sunshine (1974)

- **December 1974, RFC 675**.
- A **socket** is a triple: (network address, transport protocol, port).

The first specification of TCP

RFC 675 Specification of Internet TCP December 1974

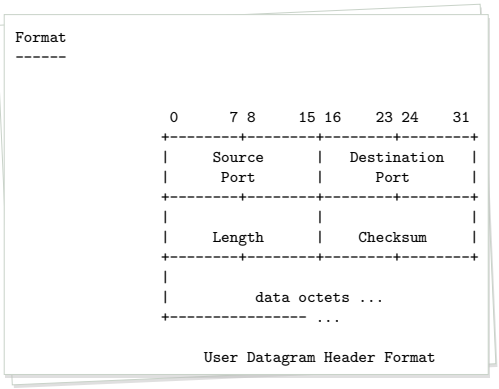
2.2 Sockets and Addressing

We have borrowed the term SOCKET from the ARPANET terminology [CACR70, MCKE73]. In general, a socket is the concatenation of a NETWORK identifier, TCP identifier, and PORT identifier. A CONNECTION is fully specified by the pair of SOCKETS at each end since the same local socket may participate in many connections to different foreign sockets.

Specification of Internet Transmission Control Program Cerf, Dalal and Sunshine (1974)

- **December 1974, RFC 675**.
- A **socket** is a triple: (network address, transport protocol, port).

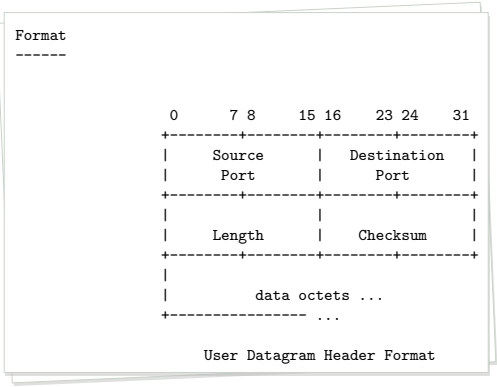
The first specification of UDP



User Datagram Protocol Postel (1980)

- **August 1980, RFC 768**. Three pages, and unchanged since.
- Two ports, a length and a checksum.
- *delivery and duplicate protection are not guaranteed*, in its own words.

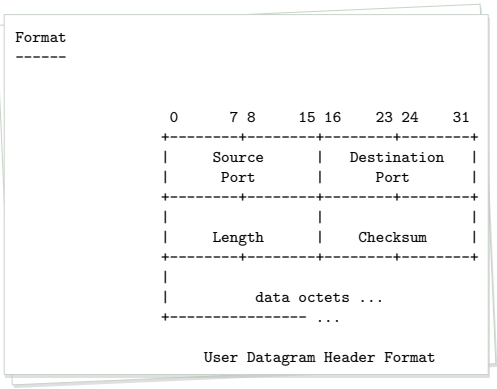
The first specification of UDP



User Datagram Protocol Postel (1980)

- **August 1980, RFC 768**. Three pages, and unchanged since.
- Two ports, a length and a checksum.
- *delivery and duplicate protection are not guaranteed*, in its own words.

The first specification of UDP



User Datagram Protocol Postel (1980)

- **August 1980, RFC 768**. Three pages, and unchanged since.
- Two ports, a length and a checksum.
- *delivery and duplicate protection are not guaranteed*, in its own words.

Overview

1. Ports

2. Transport protocols

3. Congestion control

4. Quality of service

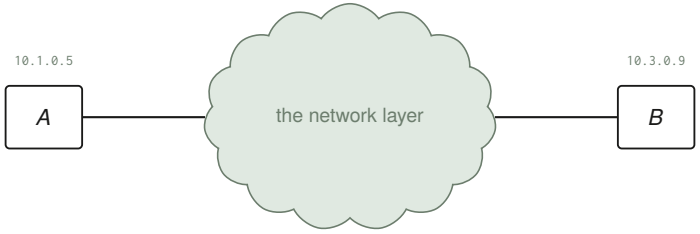
5. Towards the application layer

6. Problems

Four transport protocols

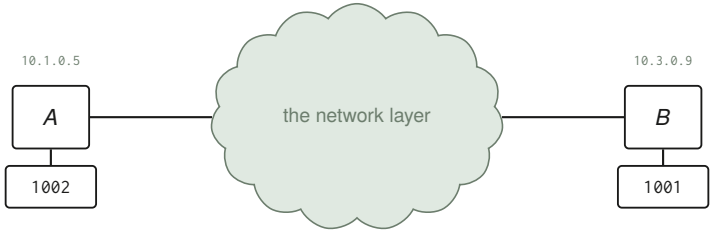
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
DCCP	Datagram Congestion Control Protocol
SCTP	Stream Control Transmission Protocol

An association of sockets



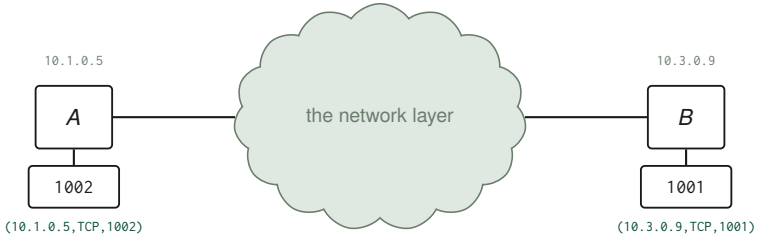
Two hosts, each with an address.

An association of sockets



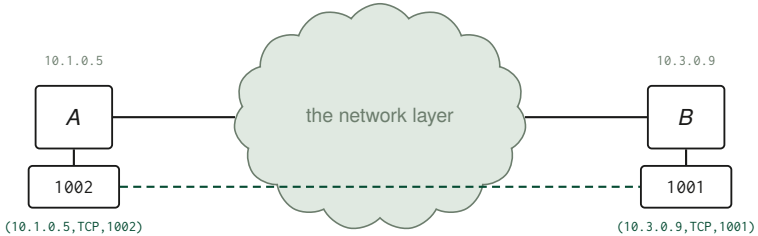
Applications at a host are assigned a **port**.

An association of sockets



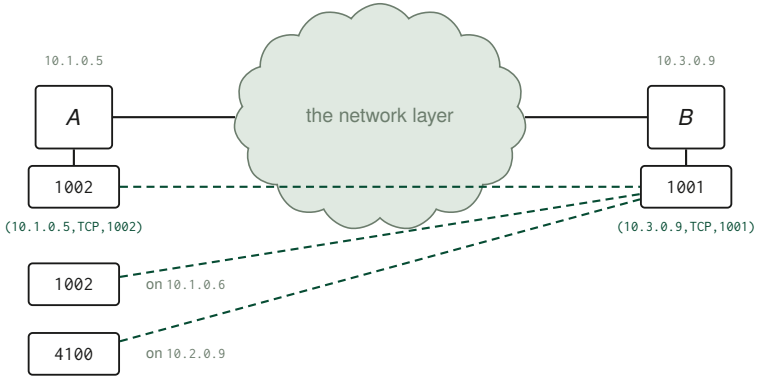
A **socket** is the triple: (address, transport protocol, port).

An association of sockets



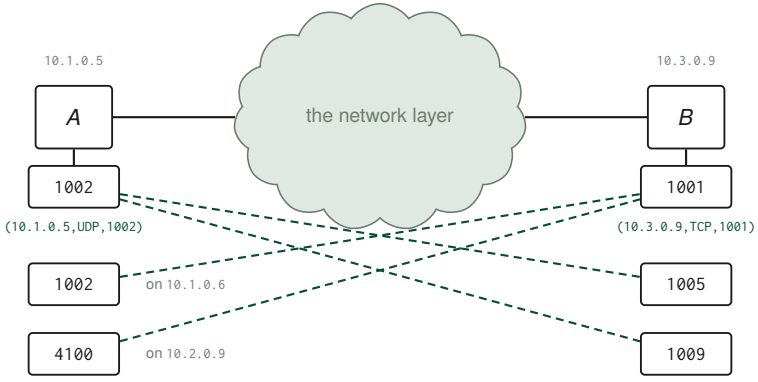
Two sockets are in an **association**.

An association of sockets



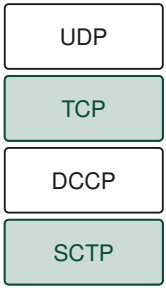
One socket may be in many associations
at the same time: **many-to-one**.

An association of sockets



And **one-to-many**, which **UDP** allows.
Each association is a **pair** of sockets.

Four transport protocols



datagrams, and nothing else

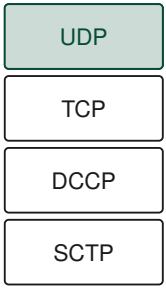
a stream of bytes, arriving in order

datagrams, with congestion control

messages, in several streams, with multihoming

- TCP and SCTP agree the association first: that is a **connection**.
- UDP is **connectionless**: no set-up, and no teardown.
- DCCP supports both.
- TCP supports **one-to-one** and **many-to-one** associations.
- UDP supports all of them: **one-to-one**, **many-to-one**, **one-to-many** and **many-to-many**.

Four transport protocols



datagrams, and nothing else

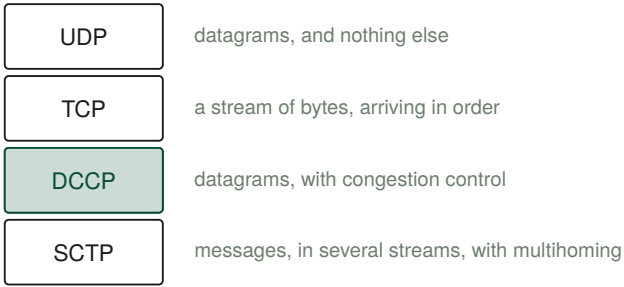
a stream of bytes, arriving in order

datagrams, with congestion control

messages, in several streams, with multihoming

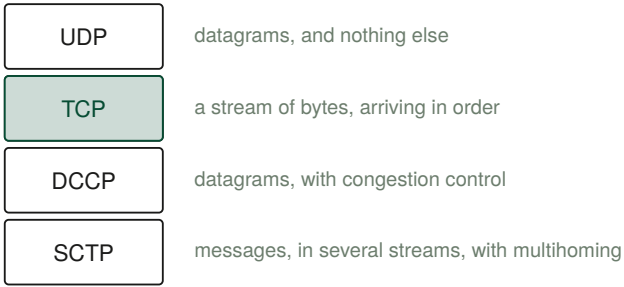
- TCP and SCTP agree the association first: that is a **connection**.
- UDP is **connectionless**: no set-up, and no teardown.
- DCCP supports both.
- TCP supports **one-to-one** and **many-to-one** associations.
- UDP supports all of them: **one-to-one**, **many-to-one**, **one-to-many** and **many-to-many**.

Four transport protocols



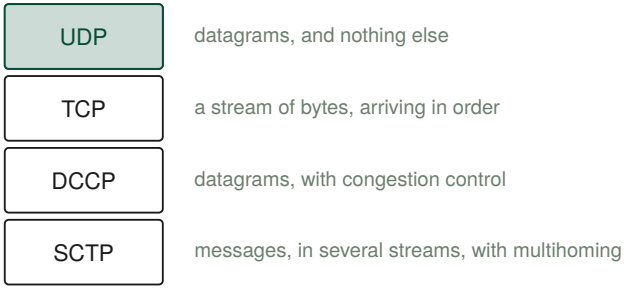
- TCP and SCTP agree the association first: that is a **connection**.
- UDP is **connectionless**: no set-up, and no teardown.
- DCCP supports both.
- TCP supports **one-to-one** and **many-to-one** associations.
- UDP supports all of them: **one-to-one**, **many-to-one**, **one-to-many** and **many-to-many**.

Four transport protocols



- TCP and SCTP agree the association first: that is a **connection**.
- UDP is **connectionless**: no set-up, and no teardown.
- DCCP supports both.
- TCP supports **one-to-one** and **many-to-one** associations.
- UDP supports all of them: **one-to-one**, **many-to-one**, **one-to-many** and **many-to-many**.

Four transport protocols



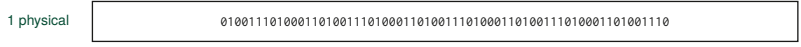
- TCP and SCTP agree the association first: that is a **connection**.
- UDP is **connectionless**: no set-up, and no teardown.
- DCCP supports both.
- TCP supports **one-to-one** and **many-to-one** associations.
- UDP supports all of them: **one-to-one**, **many-to-one**, **one-to-many** and **many-to-many**.

Protocol stack

1 physical

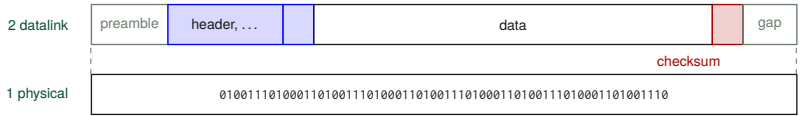
A signal on a medium.

Protocol stack



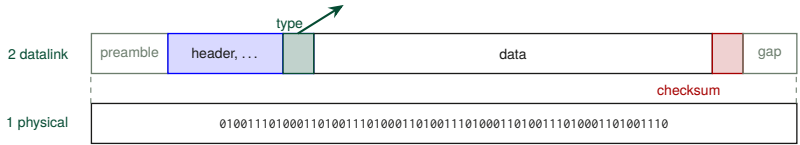
Bits. No boundaries, no meaning.

Protocol stack



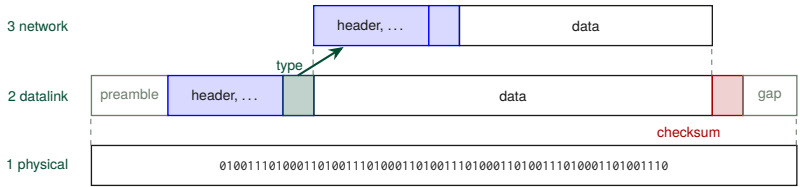
The datalink layer finds a **frame** in them,
and its own data is a run of bits again.

Protocol stack



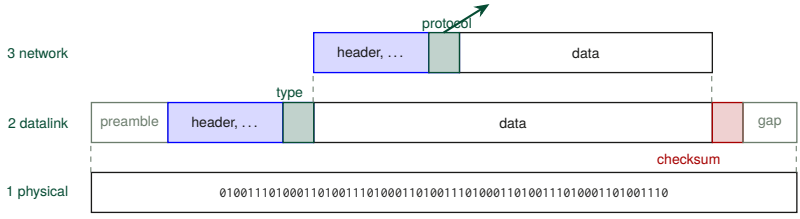
One field says what kind: the **type**, here IPv4.

Protocol stack



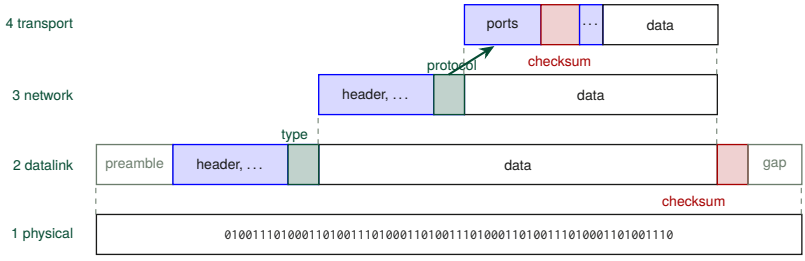
So the next layer reads a **packet**. A router gets this far and no further.

Protocol stack



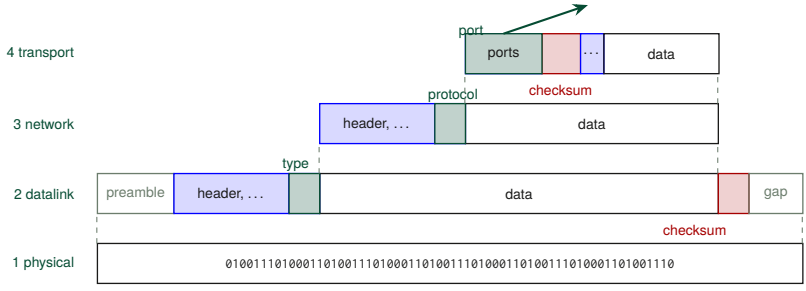
And one field says what is inside: the **transport protocol**, e.g. TCP.

Protocol stack



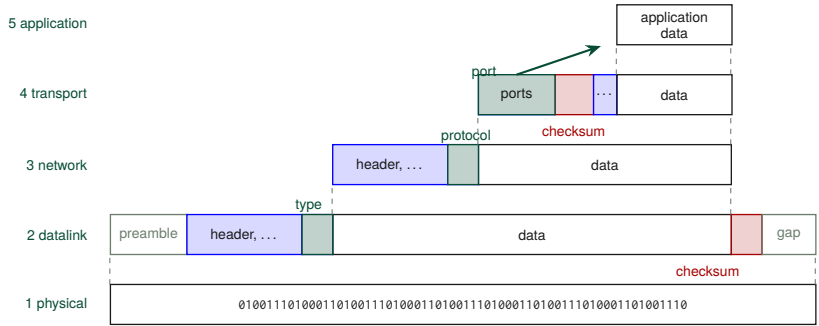
So the transport layer reads a **datagram** or a **segment**.

Protocol stack



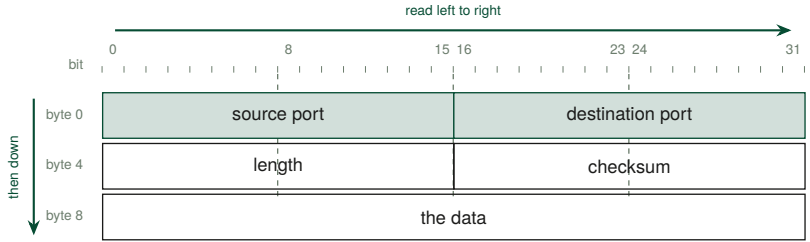
And its **port** says which application.

Protocol stack



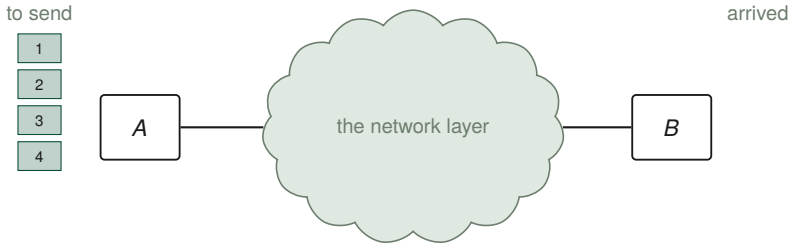
Five layers, each one fitting inside the data of the one below it.

The UDP header



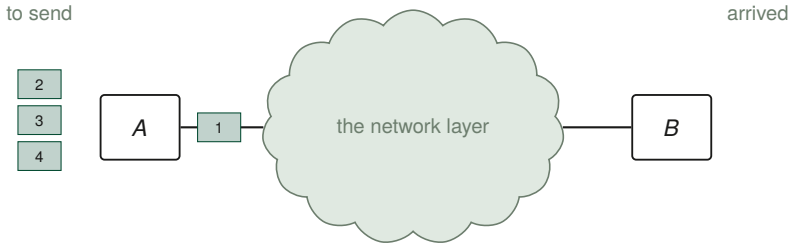
32 bits to a row, read left to right and then down. 8 bytes of header. **UDP**: User Datagram Protocol

What UDP promises



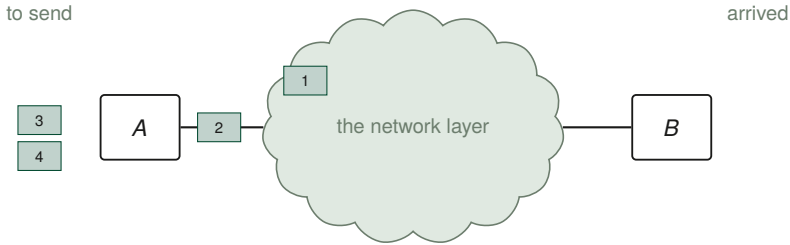
Four datagrams, in the order the application gave them.

What UDP promises



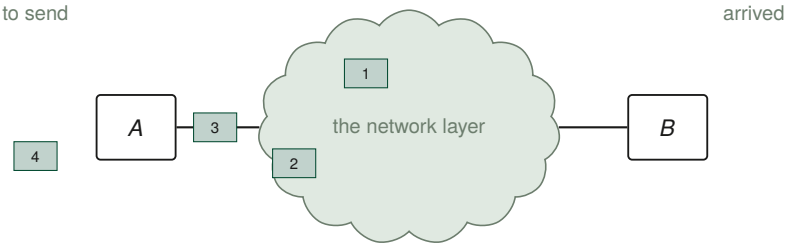
One by one they are handed to the network layer, and travel it on their own.

What UDP promises



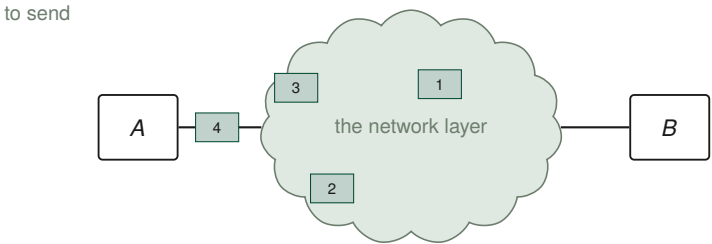
One by one they are handed to the network layer, and travel it on their own.

What UDP promises



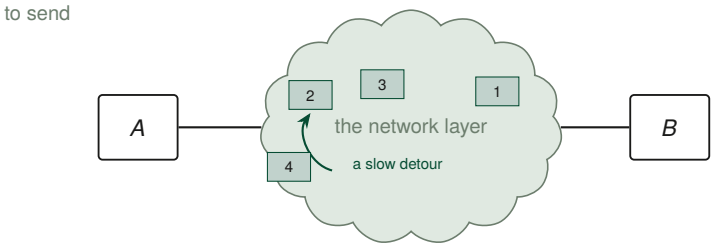
One by one they are handed to the network layer, and travel it on their own.

What UDP promises



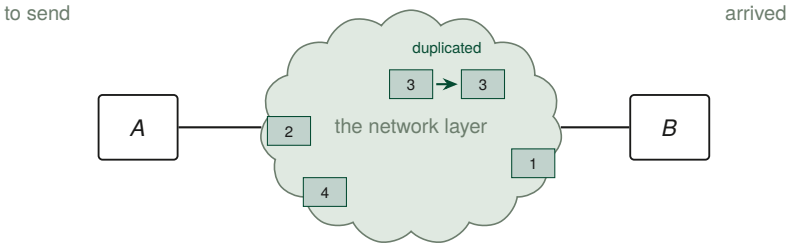
One by one they are handed to the network layer, and travel it on their own.

What UDP promises



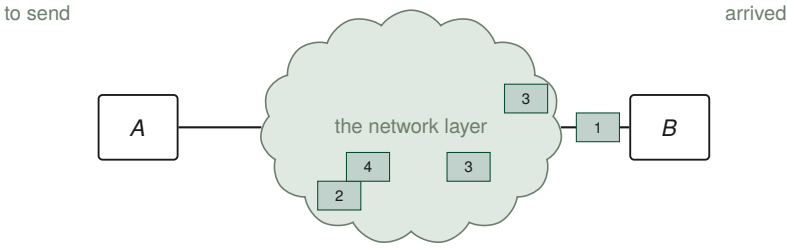
The second takes a slow detour.

What UDP promises



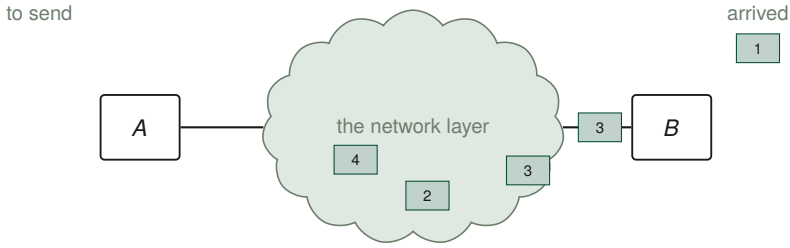
The third is duplicated on the way.

What UDP promises



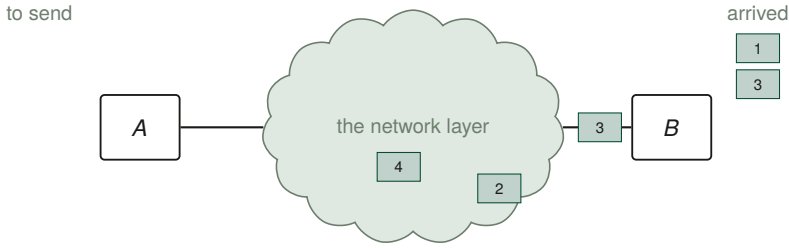
The first arrives.

What UDP promises



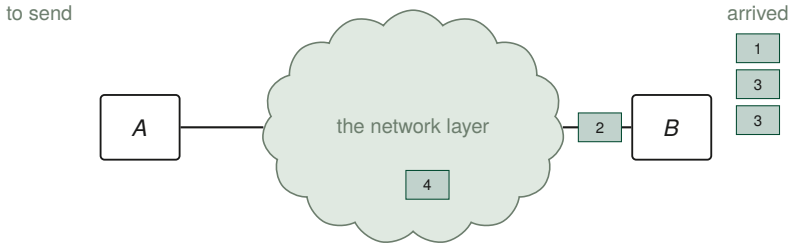
Then the third.

What UDP promises



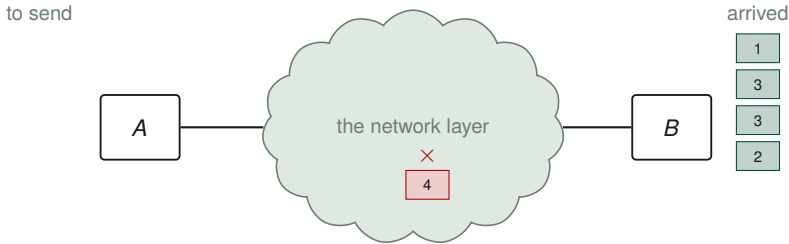
Then the third again.

What UDP promises



Then the second, late.

What UDP promises



The fourth is dropped, also called **lost**.

Applications suitable for UDP

One question, one answer

- Question lost: no answer comes, so ask again.
- Question duplicated: it is answered twice.
- Answer lost: the same silence, so ask again.
- Answer duplicated: take the first, drop the rest.

Real-time applications

- Video and audio, using special codecs.
- Voice over IP (VoIP).
- Supports broadcast and multicast.
- Real-time interaction.

Applications suitable for UDP

One question, one answer

- Question lost: no answer comes, so ask again.
- Question duplicated: it is answered twice.
- Answer lost: the same silence, so ask again.
- Answer duplicated: take the first, drop the rest.

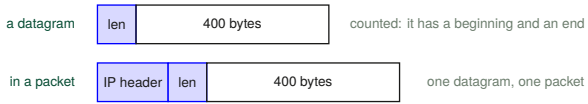
Real-time applications

- Video and audio, using special codecs.
- Voice over IP (VoIP).
- Supports broadcast and multicast.
- Real-time interaction.

A datagram versus a stream segment

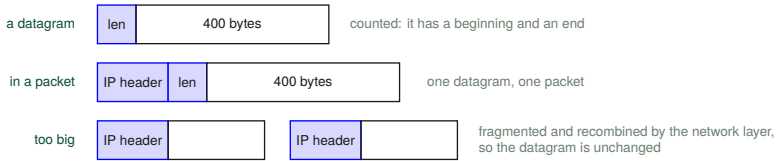
A **datagram** carries its own length.

A datagram versus a stream segment



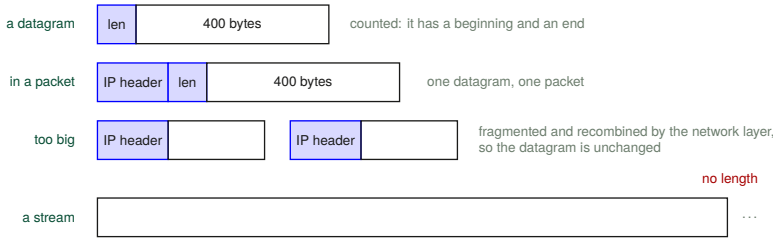
So it goes into one packet as it stands.

A datagram versus a stream segment



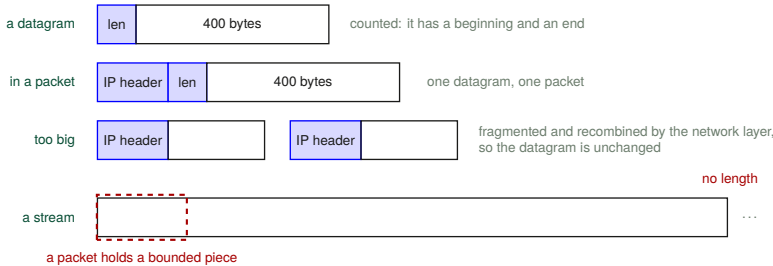
And if that packet is too large for a link, the layer below splits the packet, not the datagram.

A datagram versus a stream segment



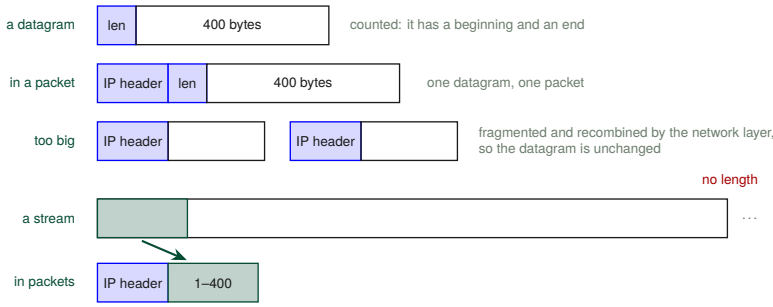
A stream has no length. The sender has not decided how much it will send, so no field could hold that number.

A datagram versus a stream segment



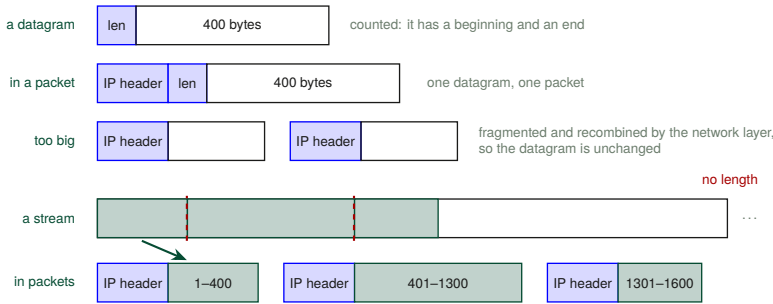
A packet must be bounded. This one is not.

A datagram versus a stream segment



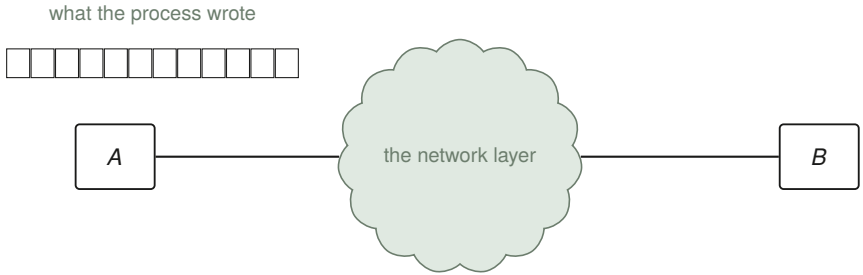
So the transport layer cuts a piece off: a **segment**.

A datagram versus a stream segment



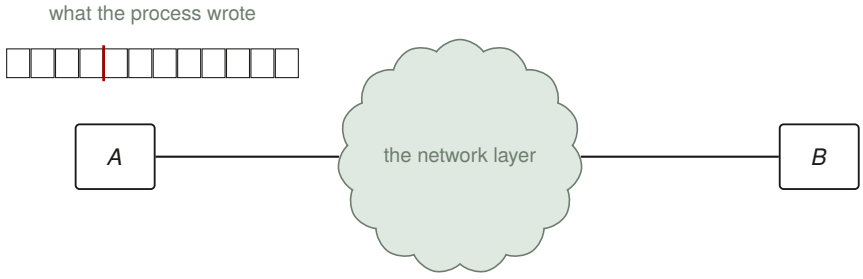
Segment after segment, each carrying where it sits in the stream rather than how long it is. The transport layer determines the cuts, which can differ in length for each segment.

A stream, not datagrams



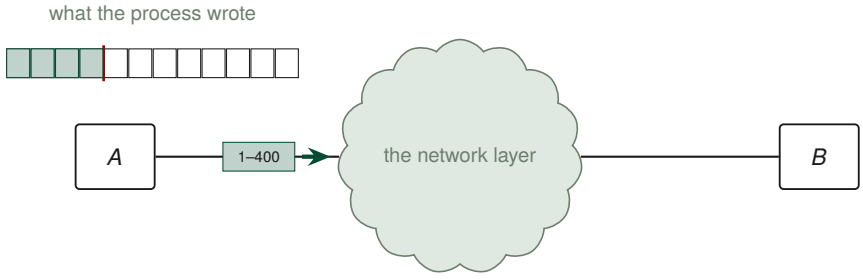
Bytes, with no end marked.

A stream, not datagrams



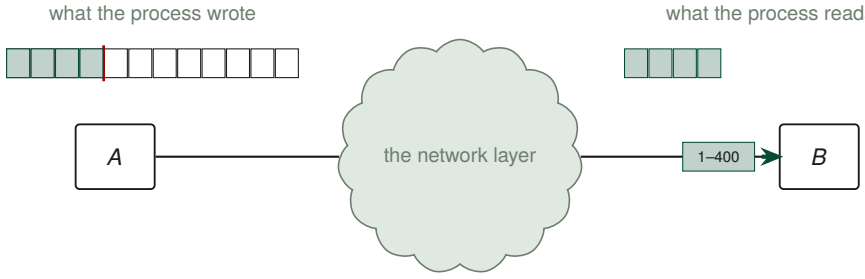
Cut into **segments**, the bytes numbered.

A stream, not datagrams



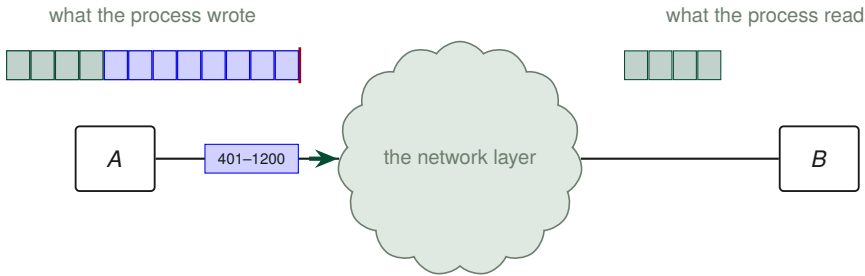
One segment, one packet.

A stream, not datagrams



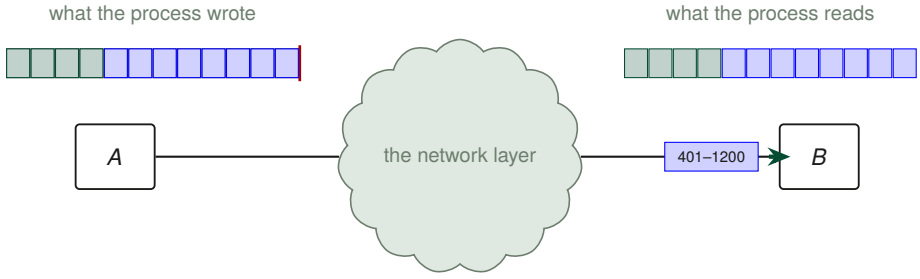
It is received, and those same bytes are readable at *B*.

A stream, not datagrams



The next one follows, and it is longer. The transport protocol decides where to cut the stream.

A stream, not datagrams

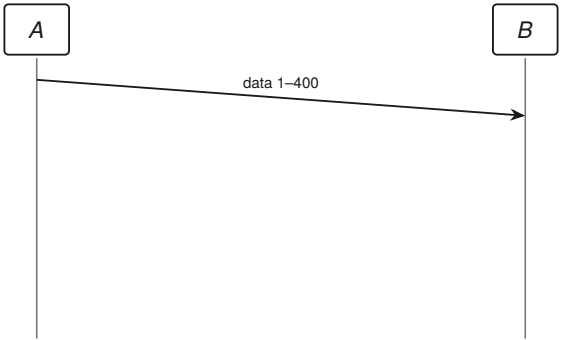


Put back in order, and read as bytes.

Stop and wait

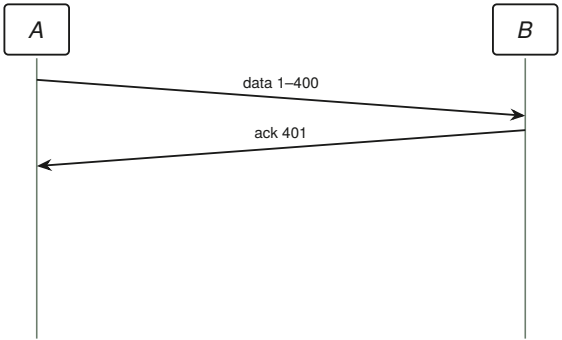
Time runs downwards.

Stop and wait



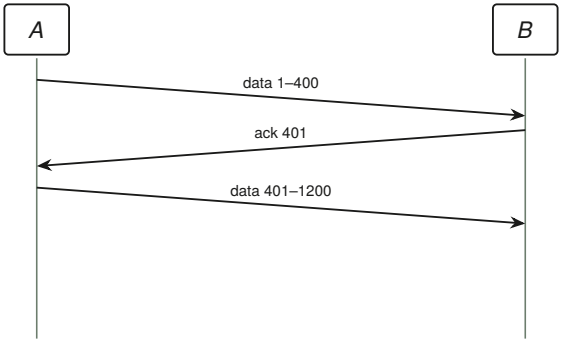
One segment, carrying bytes 1 to 400 of the stream.

Stop and wait



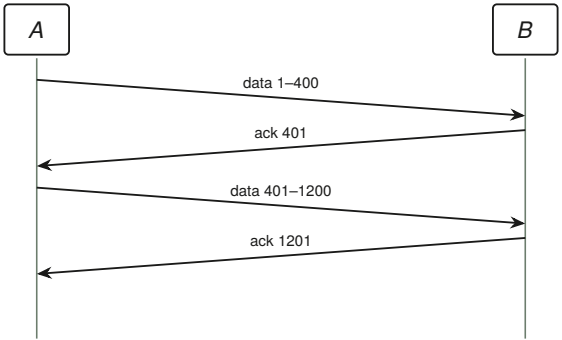
An **acknowledgement**. Its number is a **sequence number**:
the next byte *B* expects, so everything before 401 has arrived.

Stop and wait



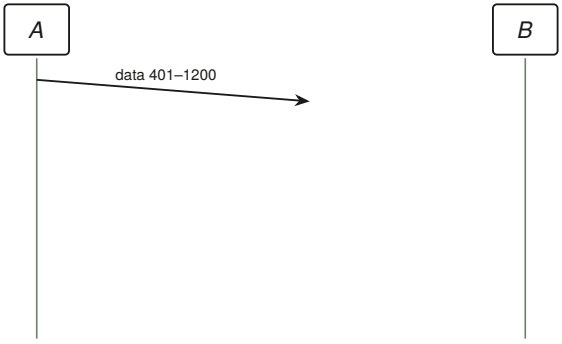
The next segment carries bytes 401 to 1200.

Stop and wait



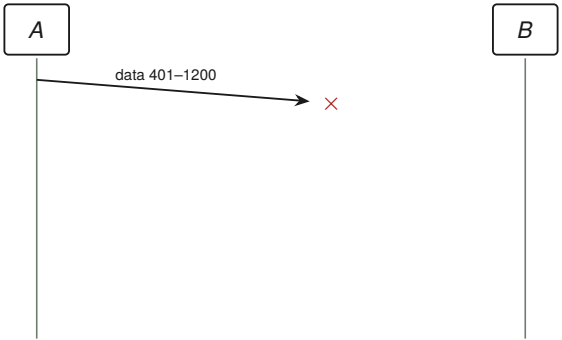
Acknowledged by 1201. One segment in flight, one round trip each.

When the segment is lost



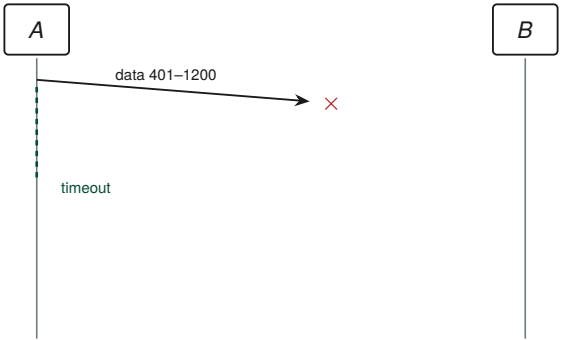
A sends, and waits.

When the segment is lost



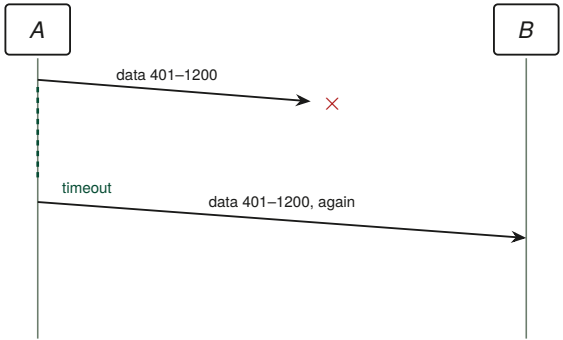
Dropped in a queue. Nobody is told.

When the segment is lost



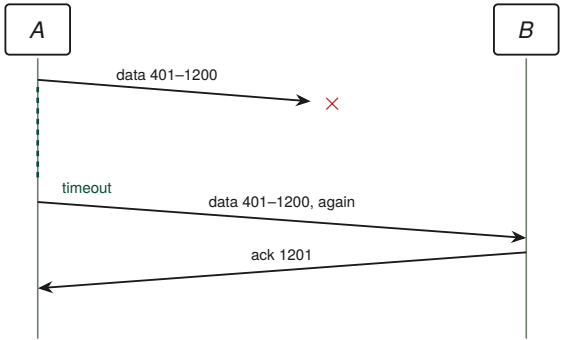
The timer runs out.

When the segment is lost



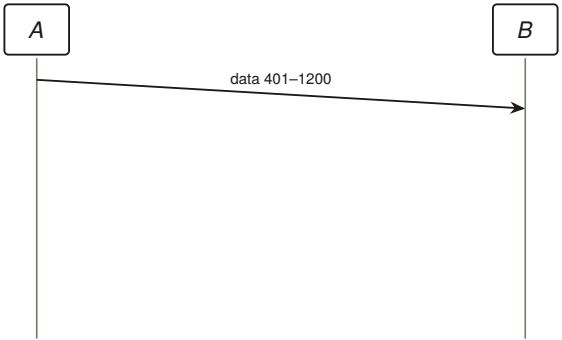
Loss is **inferred**, never reported.

When the segment is lost



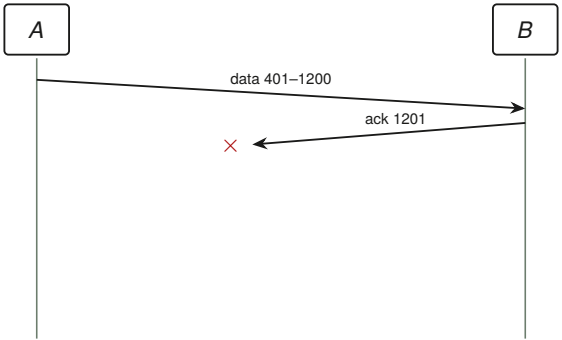
Loss is **inferred**, never reported.

When the acknowledgement is lost



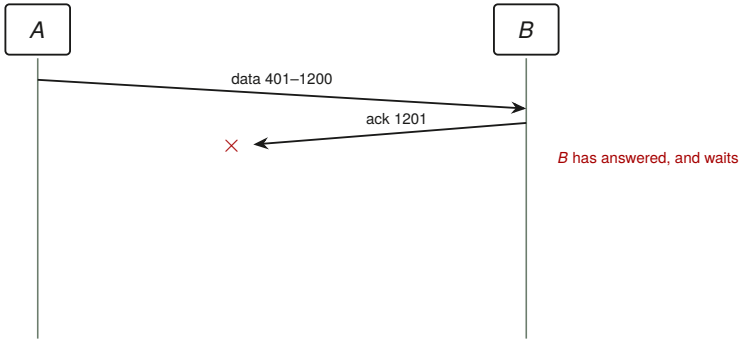
The segment arrives.

When the acknowledgement is lost



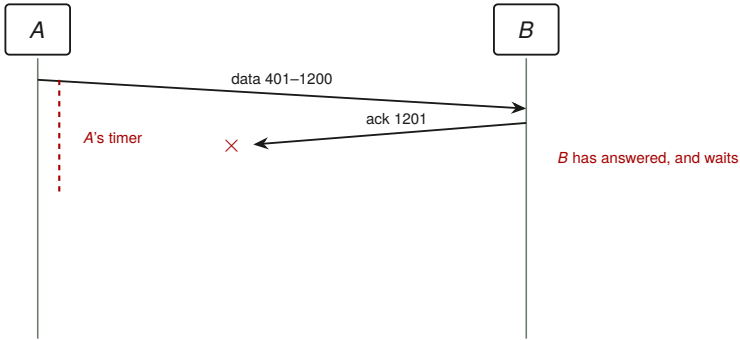
The acknowledgement does not.

When the acknowledgement is lost



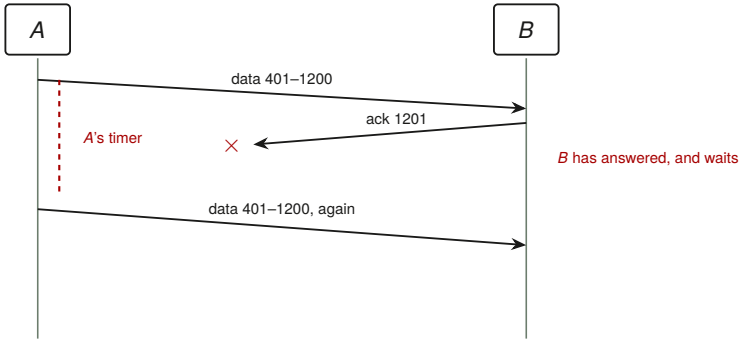
A has nothing more to send, so nothing else goes over the wire. Each end is waiting for the other.

When the acknowledgement is lost



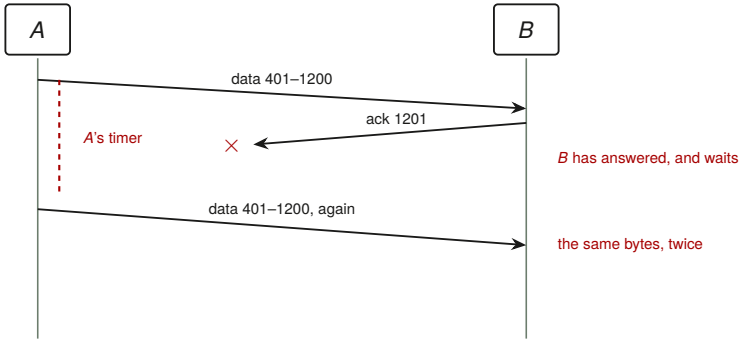
Only a timer can break it, and it is *A*'s: the receiver does not re-send an acknowledgement on its own.

When the acknowledgement is lost



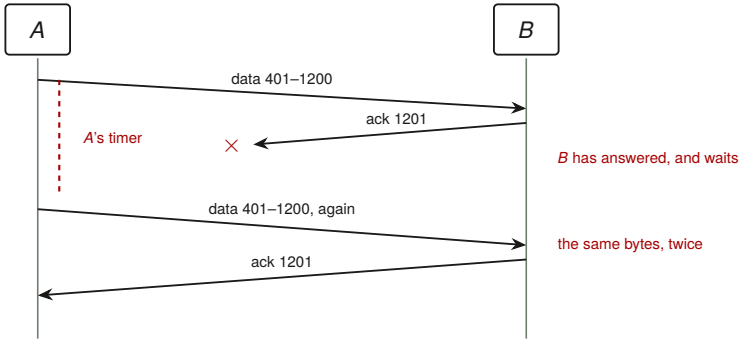
So the segment goes again.

When the acknowledgement is lost



The bytes are numbered, so the copy is dropped.

When the acknowledgement is lost



And it is acknowledged again. Retrying is safe because repeating is harmless.

How long is the timer?

At the sender

- The **retransmission** timer.
- Send a segment, start the timer.
- Acknowledgement arrives: stop it.
- Timer runs out: send the segment again, with the timeout doubled.

At the receiver

- The **delayed acknowledgement** timer.
- Segment arrives, start the timer: 500 ms.
- Data to send back: acknowledge with it, stop the timer.
- Timer runs out: acknowledge on its own.

- Too short, and the segment is sent twice. Too long, and a loss is found late. So the sender measures the round trip rather than assuming it.

How long is the timer?

At the sender

- The **retransmission** timer.
- Send a segment, start the timer.
- Acknowledgement arrives: stop it.
- Timer runs out: send the segment again, with the timeout doubled.

At the receiver

- The **delayed acknowledgement** timer.
- Segment arrives, start the timer: 500 ms.
- Data to send back: acknowledge with it, stop the timer.
- Timer runs out: acknowledge on its own.

- Too short, and the segment is sent twice. Too long, and a loss is found late. So the sender measures the round trip rather than assuming it.

How long is the timer?

At the sender

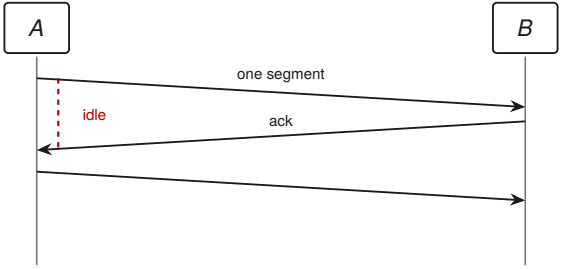
- The **retransmission** timer.
- Send a segment, start the timer.
- Acknowledgement arrives: stop it.
- Timer runs out: send the segment again, with the timeout doubled.

At the receiver

- The **delayed acknowledgement** timer.
- Segment arrives, start the timer: 500 ms.
- Data to send back: acknowledge with it, stop the timer.
- Timer runs out: acknowledge on its own.

- Too short, and the segment is sent twice. Too long, and a loss is found late. So the sender measures the round trip rather than assuming it.

One at a time is slow

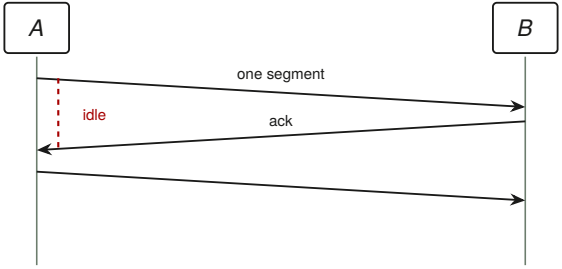


- One segment, then a whole round trip of waiting.

$$\text{max. data rate} = \frac{\text{segment size}}{\text{round trip time}}$$

The link's own speed does not appear in it.

One at a time is slow

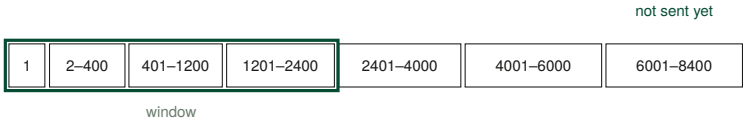


- One segment, then a whole round trip of waiting.

$$\text{max. data rate} = \frac{\text{segment size}}{\text{round trip time}}$$

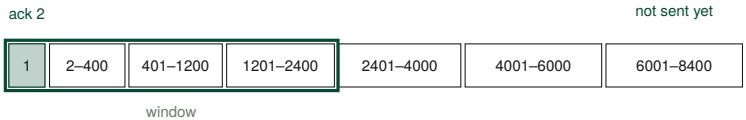
The link's own speed does not appear in it.

Sliding window



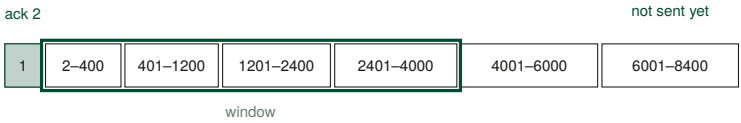
Several segments may be unacknowledged at once, not one. That is the **window**, and it is measured in bytes: the segments under it need not be the same size.

Sliding window



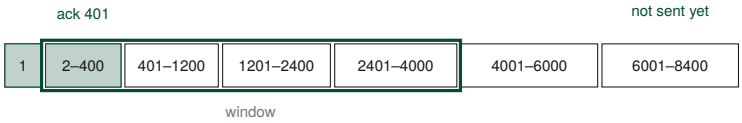
One byte is sent, and acknowledged. The acknowledgement names the next byte expected.

Sliding window



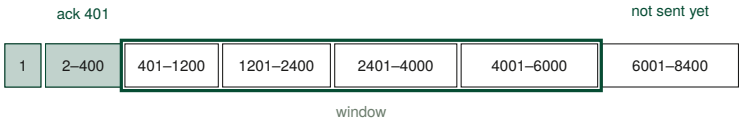
Only now does the left edge move. The window **slides**, and more may be sent.

Sliding window



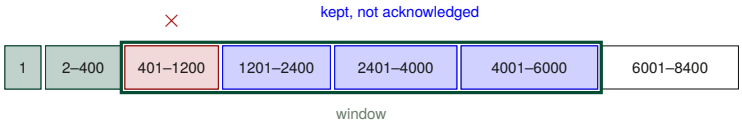
The next segment is acknowledged: 399 bytes at once.

Sliding window



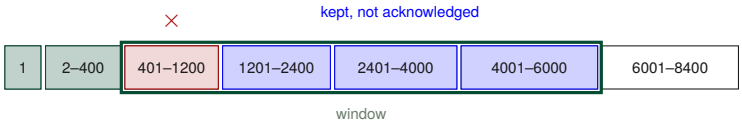
And the window slides again. The line stays busy as long as it keeps moving.

Sliding window



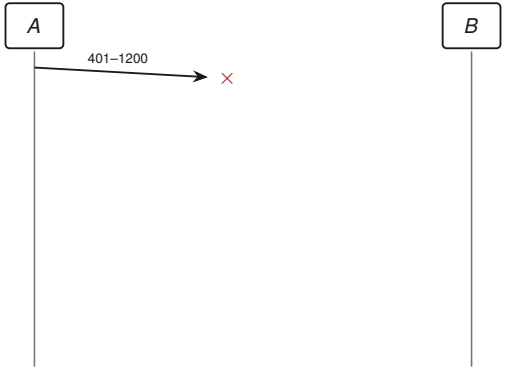
Now 401–1200 is lost, and the three behind it arrive anyway. The receiver keeps them.

Sliding window



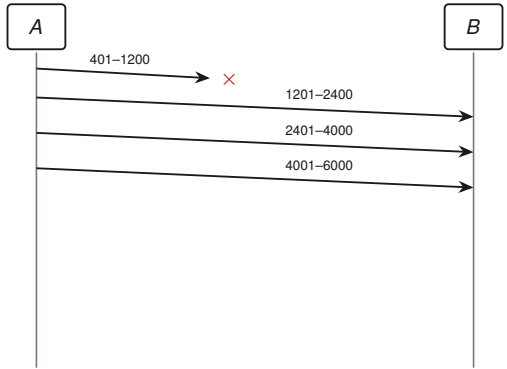
But it can only acknowledge 401: a cumulative acknowledgement does not go past a gap. Until 401–1200 is resent the window cannot slide, and every arrival repeats the same acknowledgement.

Fast retransmission



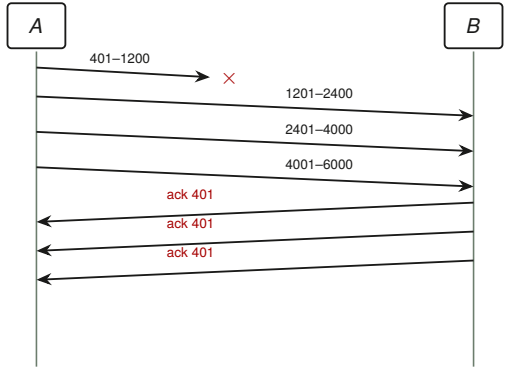
One is lost. The window is open, so more are already on their way, and each is larger than the last.

Fast retransmission



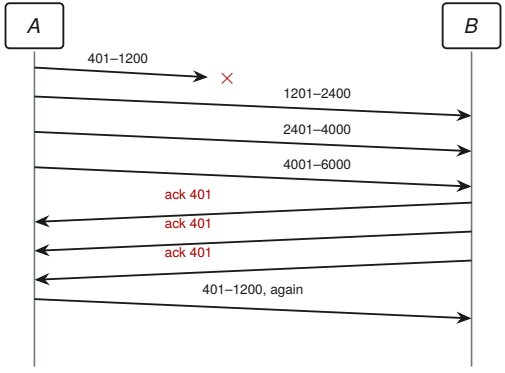
The rest arrive.

Fast retransmission



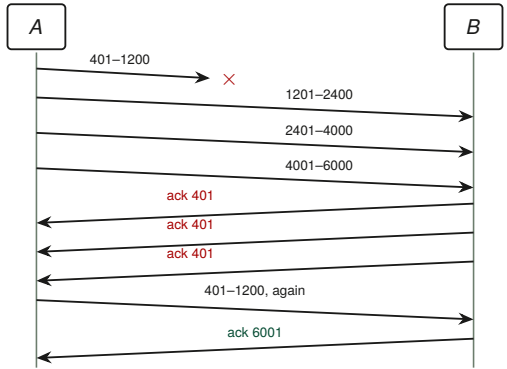
Each repeats the same acknowledgement: *still waiting for 401*.

Fast retransmission



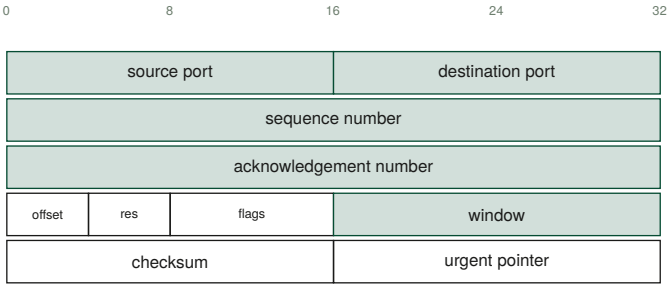
Fast retransmission: three the same means a gap, so it is resent without waiting for the timer.

Fast retransmission



The gap is closed, and one acknowledgement covers everything that was kept behind it: 6001.

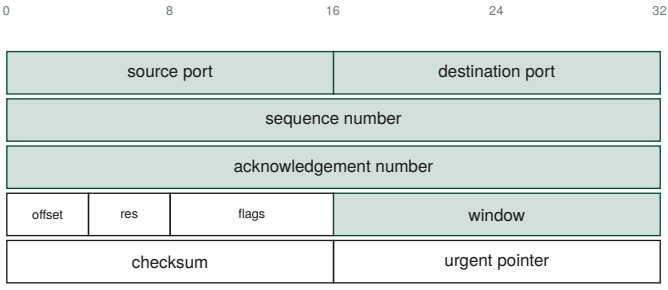
The TCP header



20 bytes, before any options. **TCP**: Transmission Control Protocol

- **Sequence** and **acknowledgement** are positions in the byte stream: this segment's first byte, and the next byte expected. Neither starts at zero; each side picks its own first number as the connection opens.
- **Window**: how many bytes the other side may have outstanding. 16 bits, so at most 65,535. The **window scale** option multiplies it.
- **Flags**: `SYN` opens, `FIN` closes, `RST` aborts.

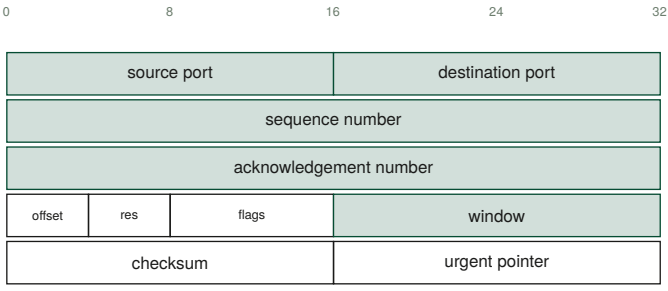
The TCP header



20 bytes, before any options. **TCP**: Transmission Control Protocol

- **Sequence** and **acknowledgement** are positions in the byte stream: this segment's first byte, and the next byte expected. Neither starts at zero; each side picks its own first number as the connection opens.
- **Window**: how many bytes the other side may have outstanding. 16 bits, so at most 65,535. The **window scale** option multiplies it.
- **Flags**: `SYN` opens, `FIN` closes, `RST` aborts.

The TCP header



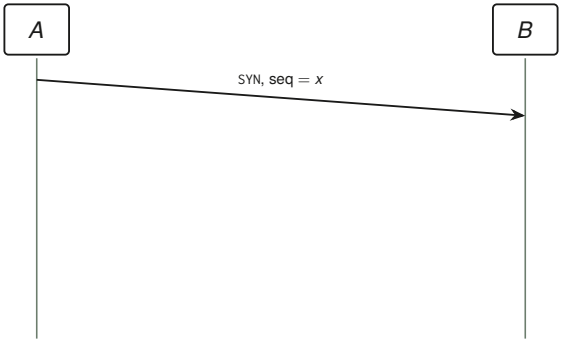
20 bytes, before any options. **TCP**: Transmission Control Protocol

- **Sequence** and **acknowledgement** are positions in the byte stream: this segment's first byte, and the next byte expected. Neither starts at zero; each side picks its own first number as the connection opens.
- **Window**: how many bytes the other side may have outstanding. 16 bits, so at most 65,535. The **window scale** option multiplies it.
- **Flags**: `SYN` opens, `FIN` closes, `RST` aborts.

Opening a connection

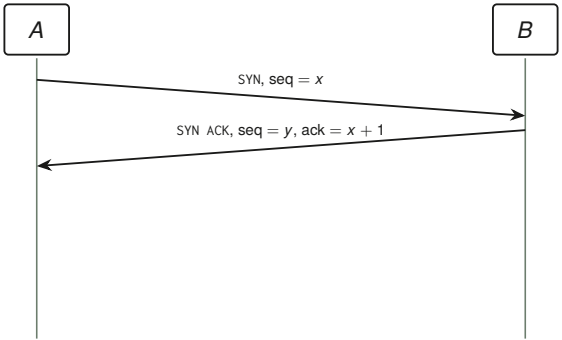
No state in the network. Only at the two ends.

Opening a connection



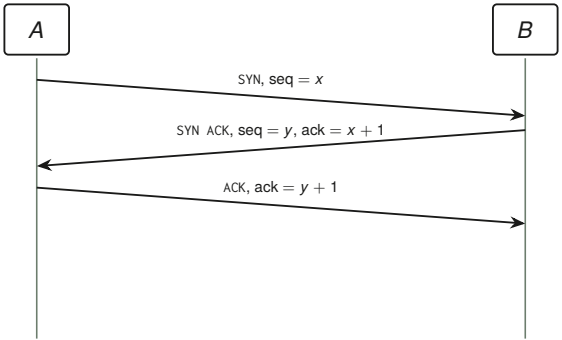
A gives where its stream starts: an index x, of no particular value.

Opening a connection



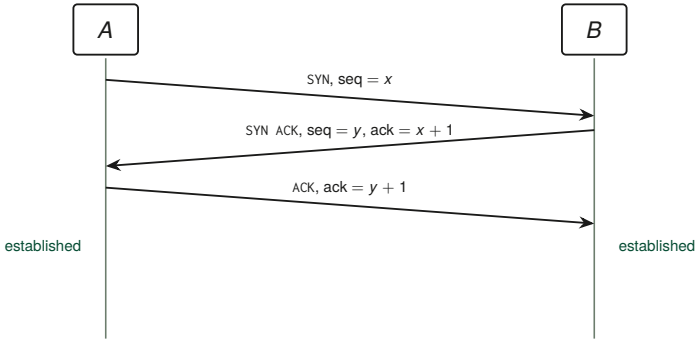
B gives where its own starts, and confirms *A*'s. The SYN itself takes one place in the numbering, so what is expected next is $x + 1$.

Opening a connection



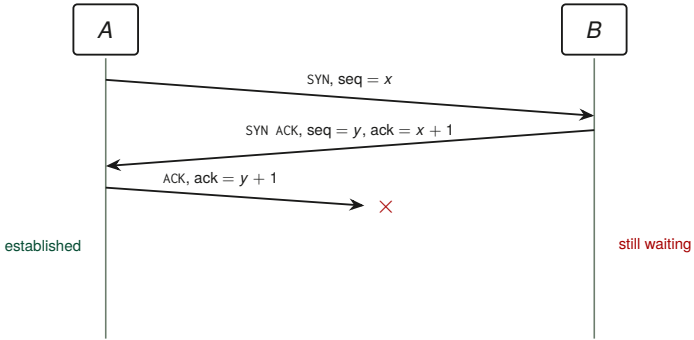
A confirms *B*'s, the same way. A number that is used up can be acknowledged; a number that is not could mean anything.

Opening a connection



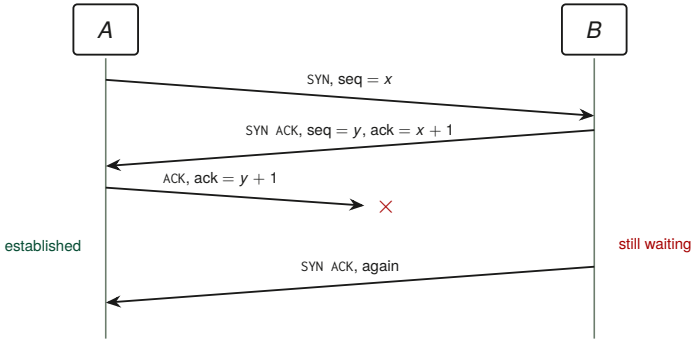
Two numbers, each confirmed, and both ends open.

Opening a connection



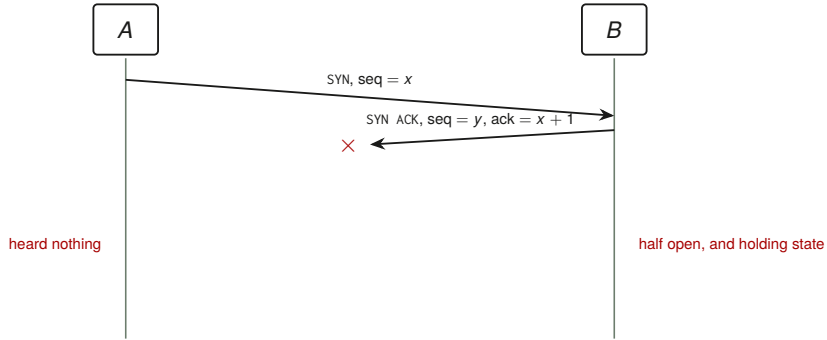
Or not. The third message is lost: *A* has all it needs and opens, *B* never hears it and keeps waiting.
Half open: a connection that exists at one end.

Opening a connection



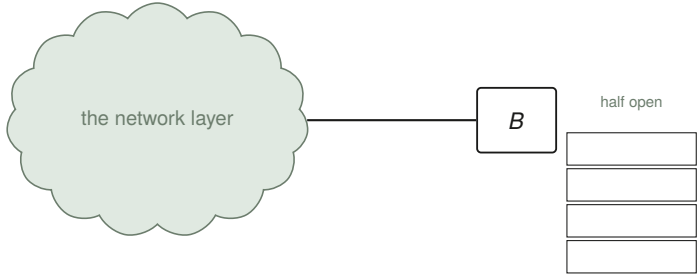
B sends its SYN ACK again, and gives up if nothing answers. The first data *A* sends would settle it too.

Opening a connection



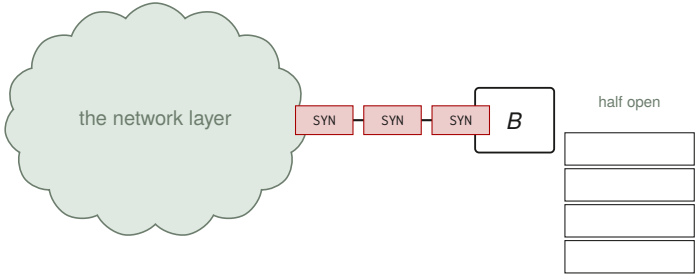
The second message can go missing as well.

SYN flooding



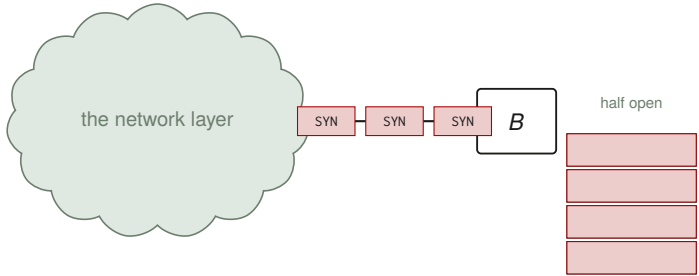
Opening costs memory, before anyone is known.

SYN flooding



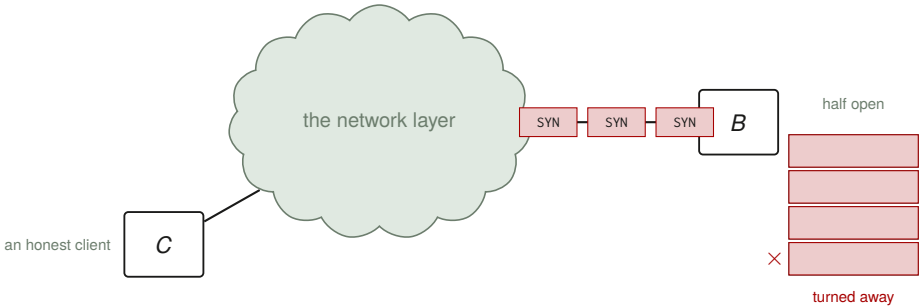
One request, then another, then another,
from addresses that never answer.

SYN flooding



Each holds a slot until it times out.

SYN flooding



The table is full. **SYN cookies**: put the state in the sequence number, and allocate nothing until it comes back.

Session hijacking

- Snoop the traffic on a shared medium, or anywhere on the path, and the sessions are there to be read: the addresses, the ports, and the numbers in flight.
- A segment is accepted if the numbers fit. Nothing proves who sent it.
- Guess the sequence number and you can write into the stream, or tear it down with one `RST`.
- Random initial numbers make guessing harder. They are not authentication.

Session hijacking

- Snoop the traffic on a shared medium, or anywhere on the path, and the sessions are there to be read: the addresses, the ports, and the numbers in flight.
- A segment is accepted if the numbers fit. Nothing proves who sent it.
- Guess the sequence number and you can write into the stream, or tear it down with one `RST`.
- Random initial numbers make guessing harder. They are not authentication.

Session hijacking

- Snoop the traffic on a shared medium, or anywhere on the path, and the sessions are there to be read: the addresses, the ports, and the numbers in flight.
- A segment is accepted if the numbers fit. Nothing proves who sent it.
- Guess the sequence number and you can write into the stream, or tear it down with one `RST`.
- Random initial numbers make guessing harder. They are not authentication.

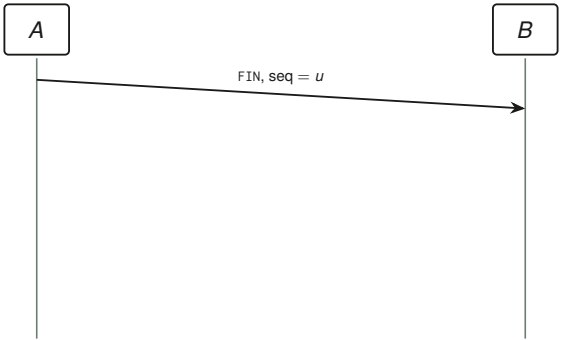
Session hijacking

- Snoop the traffic on a shared medium, or anywhere on the path, and the sessions are there to be read: the addresses, the ports, and the numbers in flight.
- A segment is accepted if the numbers fit. Nothing proves who sent it.
- Guess the sequence number and you can write into the stream, or tear it down with one `RST`.
- Random initial numbers make guessing harder. They are not authentication.

Closing a connection

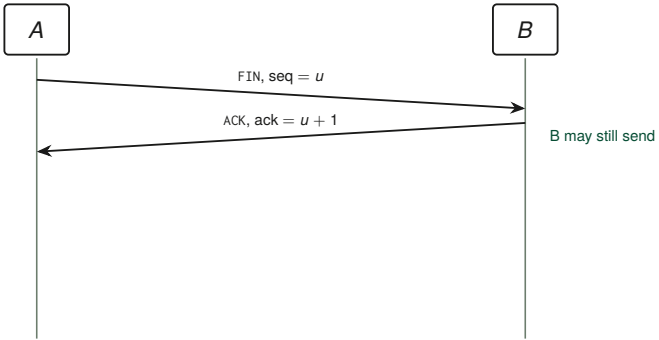
One direction at a time.

Closing a connection



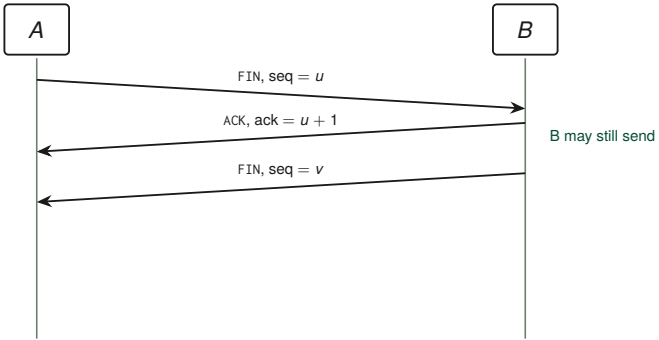
A has no more to send. *u* is where its stream ended.

Closing a connection



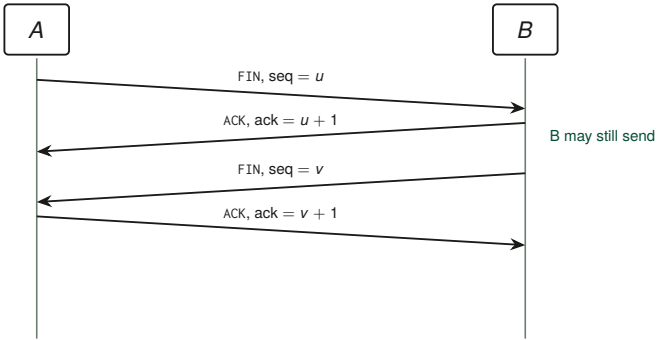
B may still send. A `FIN` takes one place in the numbering, as a `SYN` does, so it is acknowledged with $u + 1$.

Closing a connection



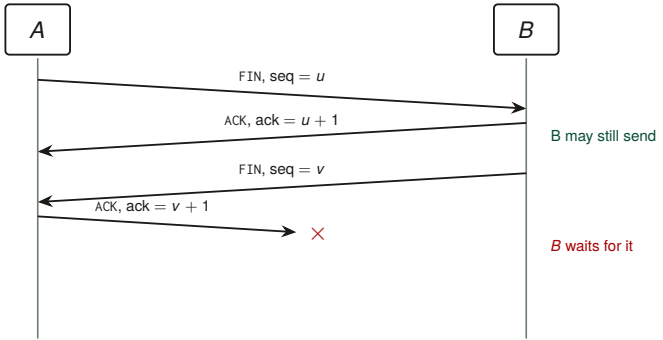
And then B is done, at v.

Closing a connection



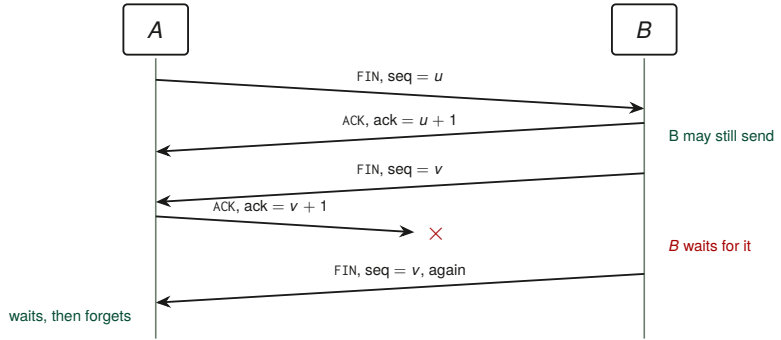
Acknowledged.

Closing a connection



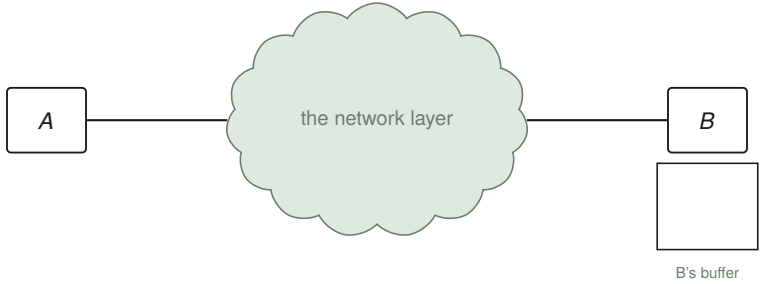
Unless that one is lost. Nobody acknowledges an acknowledgement, so *A* cannot know.

Closing a connection



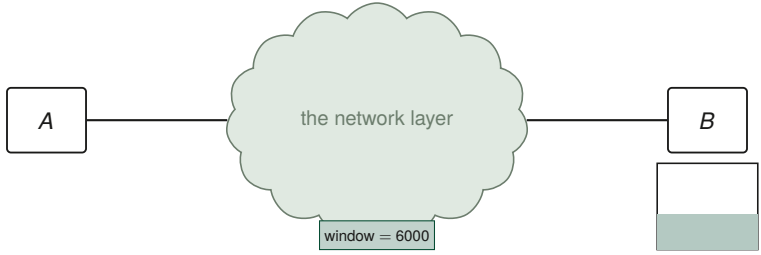
B repeats its FIN, and *A* must still be there to answer it: that is what the wait is for. **RST** is the other ending: no agreement, just stop.

Flow control: the receiver sets the pace



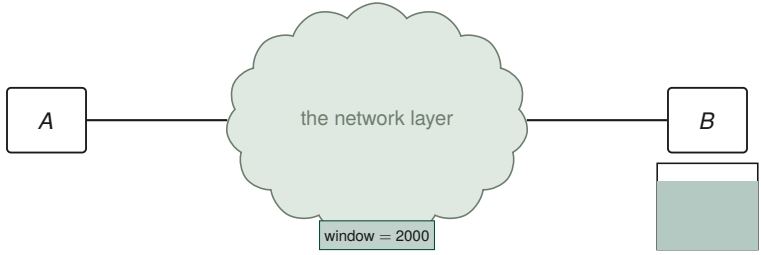
A buffer, and a process that reads from it.

Flow control: the receiver sets the pace



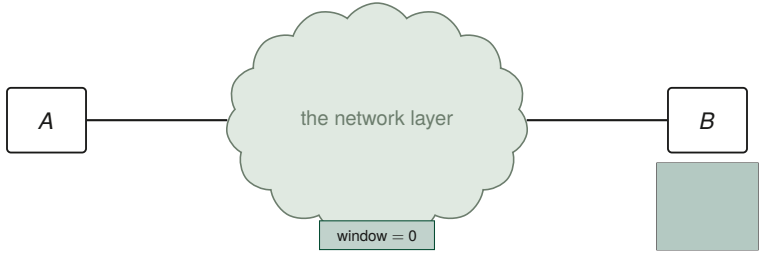
Every acknowledgement carries the room left.

Flow control: the receiver sets the pace



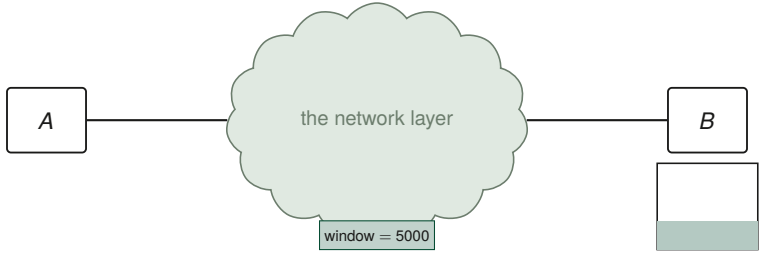
It falls.

Flow control: the receiver sets the pace



At zero the sender stops. Nothing is lost.

Flow control: the receiver sets the pace



A read reopens it. **Flow control** protects the receiver, not the network.

Small pieces, and the silly window

- One keystroke is one byte of payload under forty bytes of header. A command line session can spend the line on envelopes.
- **Nagle's algorithm**: send the first piece, buffer the rest until it is acknowledged, then send them as one. Only one small segment is ever in flight.
- The receiver can be as wasteful: it reads one byte and announces room for one byte. **Clark's solution**: do not offer a window until a whole segment, or half the buffer, is free.
- Nagle's algorithm can be switched off. A real-time session wants small packets as soon as possible.

Small pieces, and the silly window

- One keystroke is one byte of payload under forty bytes of header. A command line session can spend the line on envelopes.
- **Nagle's algorithm**: send the first piece, buffer the rest until it is acknowledged, then send them as one. Only one small segment is ever in flight.
- The receiver can be as wasteful: it reads one byte and announces room for one byte. **Clark's solution**: do not offer a window until a whole segment, or half the buffer, is free.
- Nagle's algorithm can be switched off. A real-time session wants small packets as soon as possible.

Small pieces, and the silly window

- One keystroke is one byte of payload under forty bytes of header. A command line session can spend the line on envelopes.
- **Nagle's algorithm**: send the first piece, buffer the rest until it is acknowledged, then send them as one. Only one small segment is ever in flight.

- The receiver can be as wasteful: it reads one byte and announces room for one byte. **Clark's solution**: do not offer a window until a whole segment, or half the buffer, is free.

- Nagle's algorithm can be switched off. A real-time session wants small packets as soon as possible.

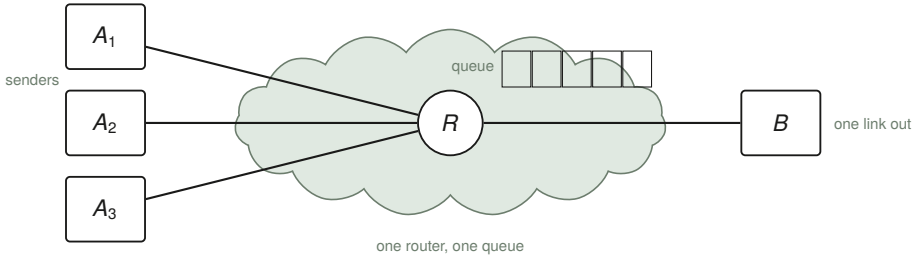
Small pieces, and the silly window

- One keystroke is one byte of payload under forty bytes of header. A command line session can spend the line on envelopes.
- **Nagle's algorithm**: send the first piece, buffer the rest until it is acknowledged, then send them as one. Only one small segment is ever in flight.
- The receiver can be as wasteful: it reads one byte and announces room for one byte. **Clark's solution**: do not offer a window until a whole segment, or half the buffer, is free.
- Nagle's algorithm can be switched off. A real-time session wants small packets as soon as possible.

Overview

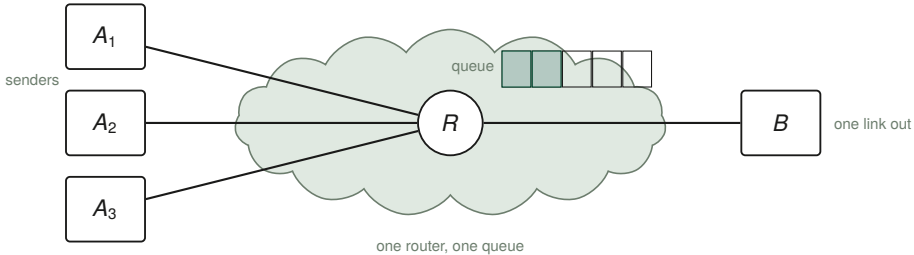
1. Ports
2. Transport protocols
3. **Congestion control**
4. Quality of service
5. Towards the application layer
6. Problems

Where the congestion is



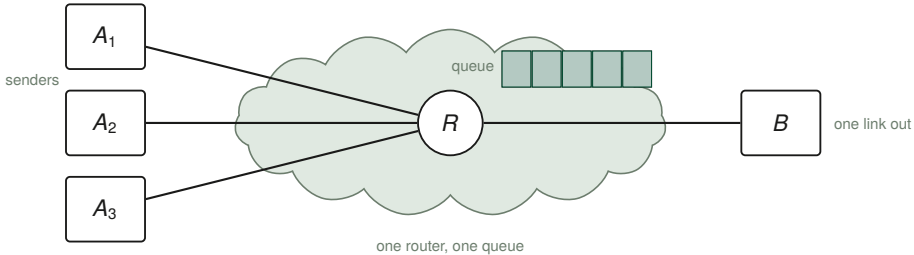
Three senders, one link out.

Where the congestion is



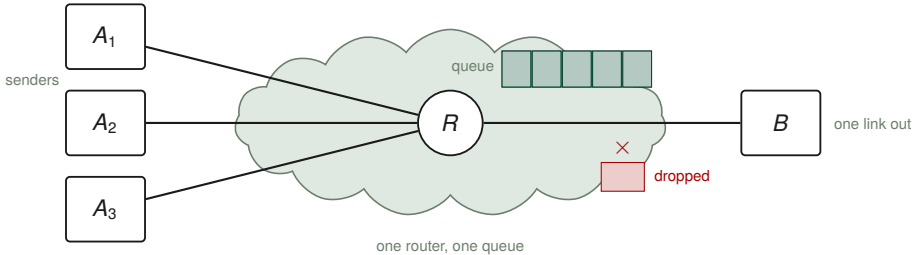
The queue fills when the link to B is slower.

Where the congestion is



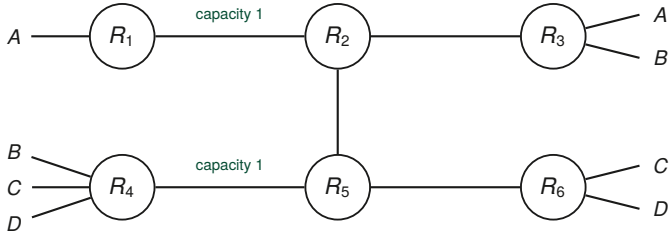
And then there is no room.

Where the congestion is



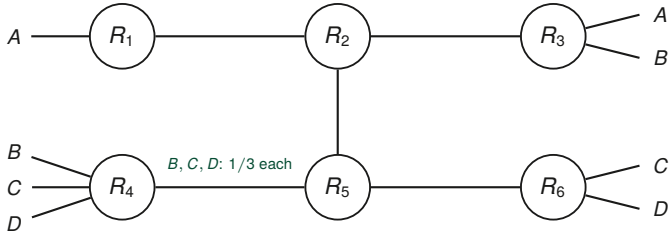
Then it drops, and tells nobody. Fixing this is the senders' job.

Network flows



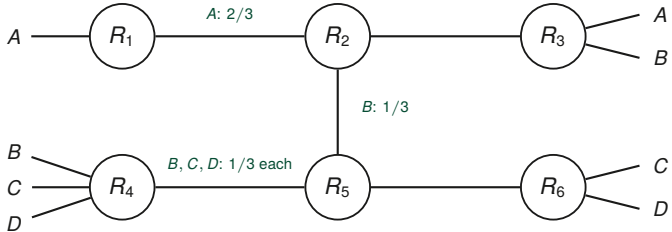
Every link the same capacity. Four end-to-end flows.

Network flows



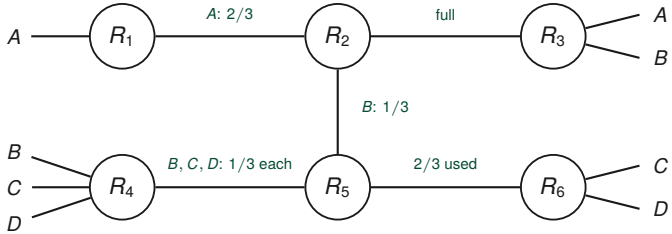
Three of them meet on one link, and
that link is the bottleneck: a third each.

Network flows



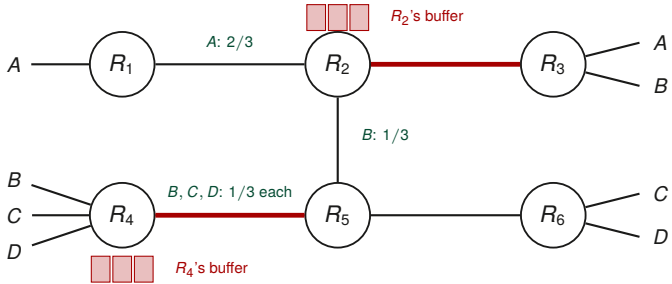
A meets only *B* upstream, and *B* is already held to a third, so *A* takes the rest.

Network flows



Max-min fair: no flow can be given more without taking from one that has less. Spare capacity downstream stays spare.

Network flows



When a link is at full capacity, buffers fill up and start to overflow.

Packet loss is the whole signal

- The network says nothing. A missing acknowledgement is the only news.
- Flow control protects the receiver, congestion control the network.
- Nobody is obliged to obey. It works because the common implementations behave.

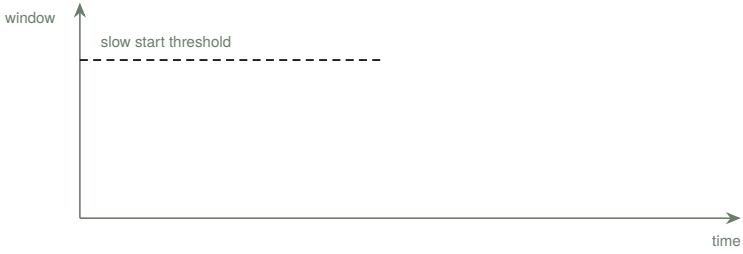
Packet loss is the whole signal

- The network says nothing. A missing acknowledgement is the only news.
- Flow control protects the receiver, congestion control the network.
- Nobody is obliged to obey. It works because the common implementations behave.

Packet loss is the whole signal

- The network says nothing. A missing acknowledgement is the only news.
- Flow control protects the receiver, congestion control the network.
- Nobody is obliged to obey. It works because the common implementations behave.

Slow start, then careful

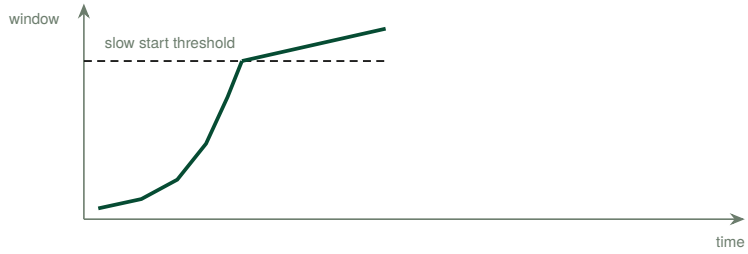


The path's capacity is unknown. What is carried instead is a **threshold**: a level the sender is willing to reach quickly.

Slow start, then careful

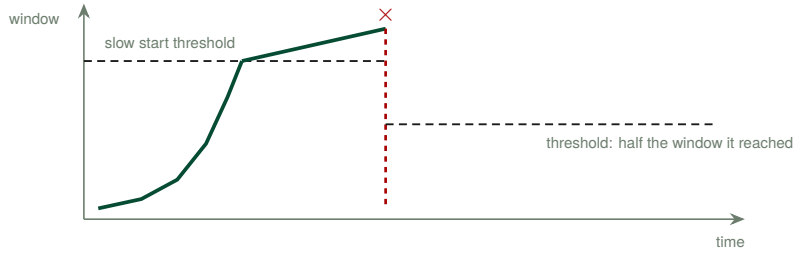
Slow start: double every round trip, up to that line.

Slow start, then careful



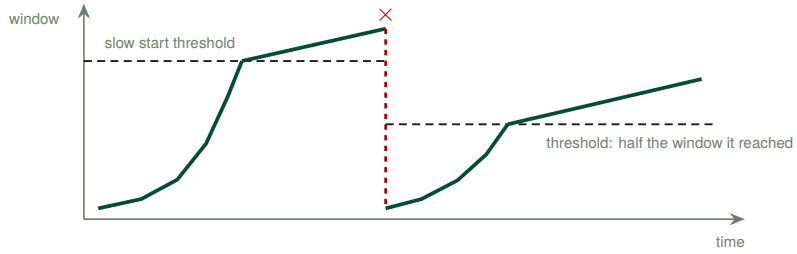
Above it, **congestion avoidance**: one more segment per round trip.

Slow start, then careful



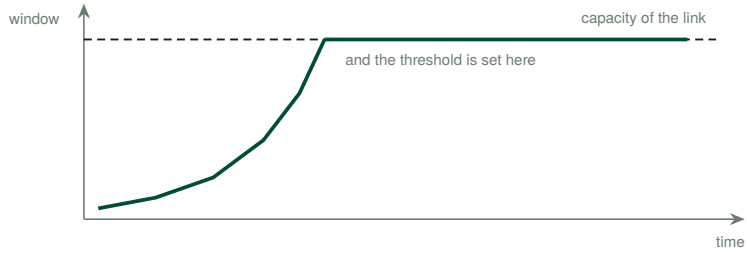
Loss. The threshold drops to half the window, and the window starts again from the bottom. **Exponential backoff**: halve, and halve again if it happens again.

Slow start, then careful



The same two growths under a lower line. Up gently, down by half: that is how a moving limit is tracked.

Slow start, then careful



A threshold can be kept and given to the next connection over the same uplink.

When loss is not congestion

- Wireless loses packets to interference, and the sender halves its window for nothing.
- **Bufferbloat**: a queue too large to drop, so the signal never comes and delay grows to seconds.
- **Explicit congestion notification**: a router that is filling up sets a bit in the packet instead of discarding it. The receiver echoes it back, and the sender slows down with nothing lost.
- Or measure the delay instead of the loss: a queue that is growing shows up as a round trip that is getting longer. Both need every router on the path to agree.

When loss is not congestion

- Wireless loses packets to interference, and the sender halves its window for nothing.
- **Bufferbloat**: a queue too large to drop, so the signal never comes and delay grows to seconds.
- **Explicit congestion notification**: a router that is filling up sets a bit in the packet instead of discarding it. The receiver echoes it back, and the sender slows down with nothing lost.
- Or measure the delay instead of the loss: a queue that is growing shows up as a round trip that is getting longer. Both need every router on the path to agree.

When loss is not congestion

- Wireless loses packets to interference, and the sender halves its window for nothing.
- **Bufferbloat**: a queue too large to drop, so the signal never comes and delay grows to seconds.
- **Explicit congestion notification**: a router that is filling up sets a bit in the packet instead of discarding it. The receiver echoes it back, and the sender slows down with nothing lost.
- Or measure the delay instead of the loss: a queue that is growing shows up as a round trip that is getting longer. Both need every router on the path to agree.

When loss is not congestion

- Wireless loses packets to interference, and the sender halves its window for nothing.
- **Bufferbloat**: a queue too large to drop, so the signal never comes and delay grows to seconds.
- **Explicit congestion notification**: a router that is filling up sets a bit in the packet instead of discarding it. The receiver echoes it back, and the sender slows down with nothing lost.
- Or measure the delay instead of the loss: a queue that is growing shows up as a round trip that is getting longer. Both need every router on the path to agree.

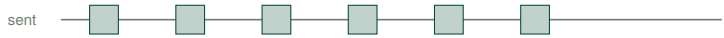
Overview

1. Ports
2. Transport protocols
3. Congestion control
4. **Quality of service**
5. Towards the application layer
6. Problems

Four things to measure

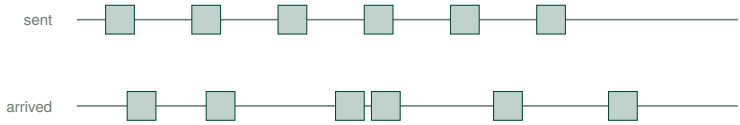
ordering	do the packets arrive in the order they were sent
throughput	how many bits per second get through
latency	how long one packet takes to cross
jitter	how much that time varies from packet to packet

Jitter, and the buffer that hides it



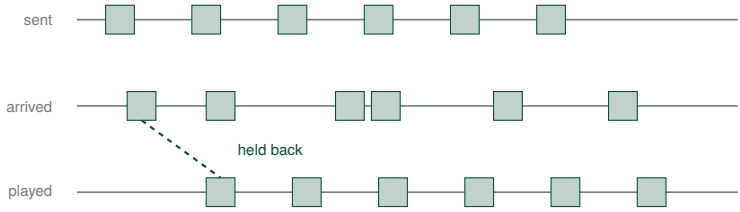
Sent evenly, one every 20 ms.

Jitter, and the buffer that hides it



Not received evenly. **Jitter** is the unpredictable variation in latency:
every buffer on the path adds its own, and none of them says how much.

Jitter, and the buffer that hides it



A **jitter buffer** plays late, and evenly. Smoothness bought with latency.

Overview

1. Ports
2. Transport protocols
3. Congestion control
4. Quality of service
5. **Towards the application layer**
6. Problems

What the transport layer does not do

an address is not a name → the domain name system

The transport layer delivers **all** the bytes, in **order**, to **one** process. What they say, and who said them, is still open.

What the transport layer does not do

an address is not a name	→ the domain name system
bytes with no agreed meaning	→ formats and protocols

The transport layer delivers **all** the bytes, in **order**, to **one** process. What they say, and who said them, is still open.

What the transport layer does not do

an address is not a name	→ the domain name system
bytes with no agreed meaning	→ formats and protocols
no idea who is at the other end	→ authentication, certificates

The transport layer delivers **all** the bytes, in **order**, to **one** process. What they say, and who said them, is still open.

What the transport layer does not do

an address is not a name	→ the domain name system
bytes with no agreed meaning	→ formats and protocols
no idea who is at the other end	→ authentication, certificates
anyone on the path can read it	→ encryption

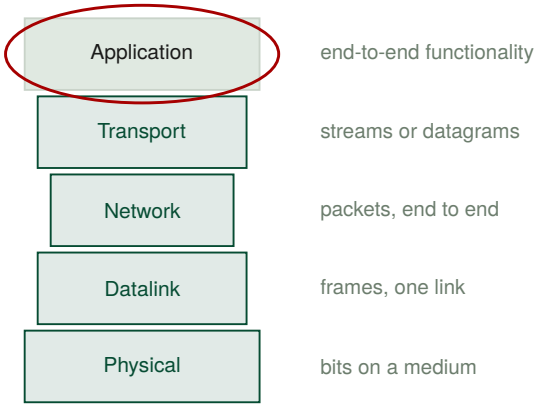
The transport layer delivers **all** the bytes, in **order**, to **one** process. What they say, and who said them, is still open.

What the transport layer does not do

an address is not a name	→ the domain name system
bytes with no agreed meaning	→ formats and protocols
no idea who is at the other end	→ authentication, certificates
anyone on the path can read it	→ encryption

The transport layer delivers **all** the bytes, in **order**, to **one** process. What they say, and who said them, is still open.

Where we are


Application
Transport
Network
Datalink
Physical
end-to-end functionality
streams or datagrams
packets, end to end
frames, one link
bits on a medium

Overview

1. Ports
2. Transport protocols
3. Congestion control
4. Quality of service
5. Towards the application layer
6. **Problems**

Problems: ports

- **19.** Both UDP and TCP use port numbers to identify the destination entity when delivering a message. Give two reasons why these protocols invented a new abstract identifier – the port number – instead of using process identifiers, which already existed when they were designed.
- **26.** A process on host 1 has been assigned port p, and a process on host 2 has been assigned port q. Is it possible for there to be two or more TCP connections between these two ports at the same time?

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.2.1, chapter 6 problems 19 and 26.

Problems: ports

- **19.** Both UDP and TCP use port numbers to identify the destination entity when delivering a message. Give two reasons why these protocols invented a new abstract identifier – the port number – instead of using process identifiers, which already existed when they were designed.
- **26.** A process on host 1 has been assigned port p, and a process on host 2 has been assigned port q. Is it possible for there to be two or more TCP connections between these two ports at the same time?

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.2.1, chapter 6 problems 19 and 26.

Problems: UDP

- **15.** Why does UDP exist? Would it not have been enough to just let user processes send raw IP packets?
- 17. A client sends a 128-byte request to a server 100 km away over a 1-gigabit optical fibre. What is the efficiency of the line during that one exchange?

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.4, chapter 6 problems 15 and 17.

Problems: UDP

- **15.** Why does UDP exist? Would it not have been enough to just let user processes send raw IP packets?
- **17.** A client sends a 128-byte request to a server 100 km away over a 1-gigabit optical fibre. What is the efficiency of the line during that one exchange?

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.4, chapter 6 problems 15 and 17.

Problems: sequence numbers

- **23.** Datagram fragmentation and reassembly are handled by IP, and are invisible to TCP. Does this mean that TCP does not have to worry about data arriving in the wrong order?

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.5.3 and §6.5.4, chapter 6 problem 23.

Problems: the data rate

- **36.** In a network whose maximum segment is 128 bytes, whose maximum segment lifetime is 30 seconds, and which has 8-bit sequence numbers, what is the maximum data rate per connection?

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.5.3, chapter 6 problem 36.

Problems: the window

- **41.** A 1-Gbps network over 4000 km is limited by delay rather than by bandwidth. Now take a city network, source and destination 20 km apart. At what data rate does the round trip delay, at the speed of light, equal the time it takes to send a 1-KB packet? (Signals travel at about 200,000 km/s in fibre, so 20 km takes roughly 100 μs one way.)
- **3.** The sliding window protocol assumes the network layer may lose packets, so each has to be acknowledged. Suppose the network layer never lost one. What, if anything, would still be needed?

Tanenbaum & Wetherall, Computer Networks, 5th edition, §6.2.4 and §6.6, chapter 6 problems 41 and 3.

Problems: the window

- **41.** A 1-Gbps network over 4000 km is limited by delay rather than by bandwidth. Now take a city network, source and destination 20 km apart. At what data rate does the round trip delay, at the speed of light, equal the time it takes to send a 1-KB packet? (Signals travel at about 200,000 km/s in fibre, so 20 km takes roughly 100 μs one way.)
- **3.** The sliding window protocol assumes the network layer may lose packets, so each has to be acknowledged. Suppose the network layer never lost one. What, if anything, would still be needed?

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.2.4 and §6.6, chapter 6 problems 41 and 3.

Problems: TCP at speed

- **34.** What is the fastest line speed at which a host can blast out 1500-byte TCP payloads with a 120-second maximum packet lifetime, without the sequence numbers wrapping around? Take TCP, IP and Ethernet overhead into account.
- **35.** Addressing the limitations of IPv4 needed a whole new version of the protocol, and the adoption of it is still reluctant. No such effort is needed to address the limitations of TCP. Explain why.

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.5.11, chapter 6 problems 34 and 35

Problems: TCP at speed

- **34.** What is the fastest line speed at which a host can blast out 1500-byte TCP payloads with a 120-second maximum packet lifetime, without the sequence numbers wrapping around? Take TCP, IP and Ethernet overhead into account.
- **35.** Addressing the limitations of IPv4 needed a whole new version of the protocol, and the adoption of it is still reluctant. No such effort is needed to address the limitations of TCP. Explain why.

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.5.11, chapter 6 problems 34 and 35.

Problems: TCP

- **9.** Imagine that a two-way handshake rather than a three-way handshake were used to set up connections – that is, the third message is not required. Are deadlocks now possible? Give an example, or show that none exist.
- **28.** The maximum payload of a TCP segment is 65,495 bytes. Why was such a strange number chosen? ($2^{16} = 65{,}536$, and a header is not payload.)

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.5, chapter 6 problems 9 and 28.

Problems: TCP

- **9.** Imagine that a two-way handshake rather than a three-way handshake were used to set up connections – that is, the third message is not required. Are deadlocks now possible? Give an example, or show that none exist.
- **28.** The maximum payload of a TCP segment is 65,495 bytes. Why was such a strange number chosen? ($2^{16} = 65{,}536$, and a header is not payload.)

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.5, chapter 6 problems 9 and 28.

Problems: congestion control

- **30.** Consider the effect of slow start on a line with a 10 ms round trip time and no congestion. The receive window is 24 KB and the maximum segment size is 2 KB. How long does it take before the first full window can be sent?
- **31.** Suppose the TCP congestion window is set to 18 KB and a timeout occurs. How big will the window be if the next four transmission bursts are all successful? Assume a maximum segment size of 1 KB.

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.3 and §6.5.10, chapter 6 problems 30 and 31.

Problems: congestion control

- **30.** Consider the effect of slow start on a line with a 10 ms round trip time and no congestion. The receive window is 24 KB and the maximum segment size is 2 KB. How long does it take before the first full window can be sent?
- **31.** Suppose the TCP congestion window is set to 18 KB and a timeout occurs. How big will the window be if the next four transmission bursts are all successful? Assume a maximum segment size of 1 KB.

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.3 and §6.5.10, chapter 6 problems 30 and 31.

Problems: quality of service

- **21.** Two networks have the same average delay between a source and a destination. In the first, the delay is spread evenly, and never exceeds 10 seconds. In the second, 99% of packets take less than a second, but there is no limit at all on the worst case. Discuss how a real-time protocol could carry live audio or video over each of them.
- **24.** A real-time protocol carries CD-quality audio: a pair of 16-bit samples, 44,100 times a second. How many packets per second must it send?

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §5.4.3, chapter 5 problems 21 and 24

Problems: quality of service

- **21.** Two networks have the same average delay between a source and a destination. In the first, the delay is spread evenly, and never exceeds 10 seconds. In the second, 99% of packets take less than a second, but there is no limit at all on the worst case. Discuss how a real-time protocol could carry live audio or video over each of them.
- **24.** A real-time protocol carries CD-quality audio: a pair of 16-bit samples, 44,100 times a second. How many packets per second must it send?

Tanenbaum & Wetherall, *Computer Networks*, 5th edition, §6.4.3, chapter 6 problems 21 and 24.