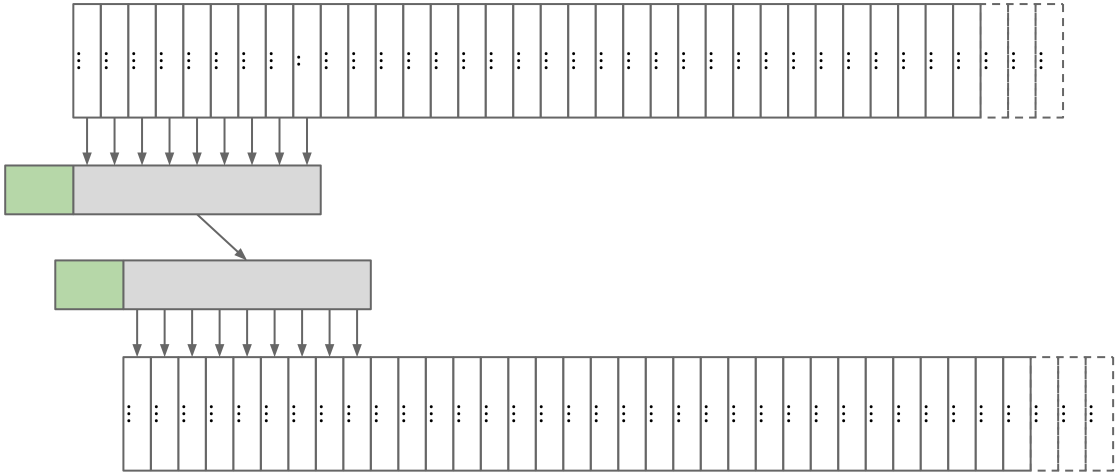


Lecture 12 (Transport 2)

TCP Implementation

CS 168, Spring 2025 @ UC Berkeley

Slides credit: Sylvia Ratnasamy, Rob Shakir, Peyrin Kao

Implementing TCP: Byte Notation (Segments, Sequence Numbers)

Lecture 12, CS 168, Spring 2025

Implementing TCP

- **Byte Notation (Segments, Sequence Numbers)**
- Maintaining State (Full Duplex, Connection Setup and Teardown)
- Sliding Window
- Header

Notation Change: Bytes vs. Packets

So far, we've used *packets* as the primary unit of data.

- Each packet has a number.
- Acks reference packet numbers.
- Window size expressed in terms of number of packets.

TCP is implemented with *bytes* as the primary unit of data.

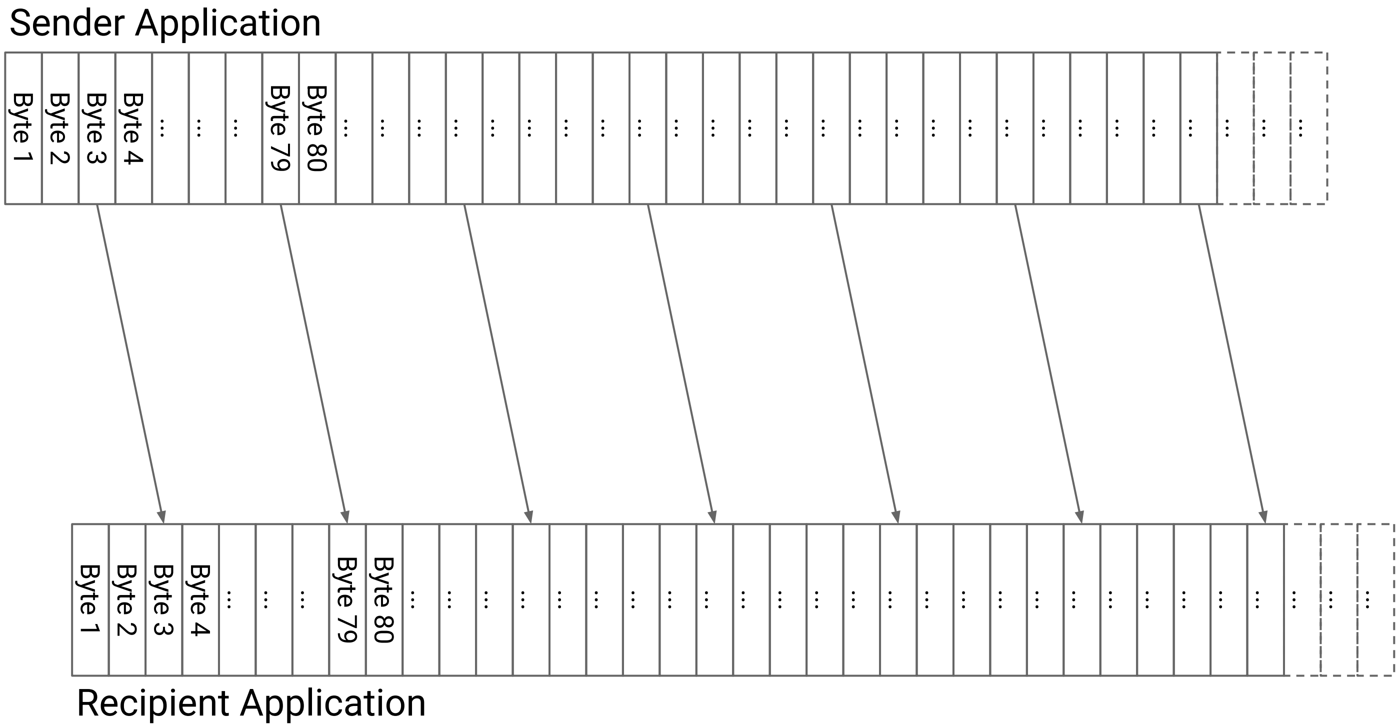
- Each byte has a number.
  Packets are defined by the number of the first byte inside.
- ACKs reference byte numbers.
- Window size expressed in terms of number of bytes.

You should be prepared to reason in terms of either.

TCP Segments

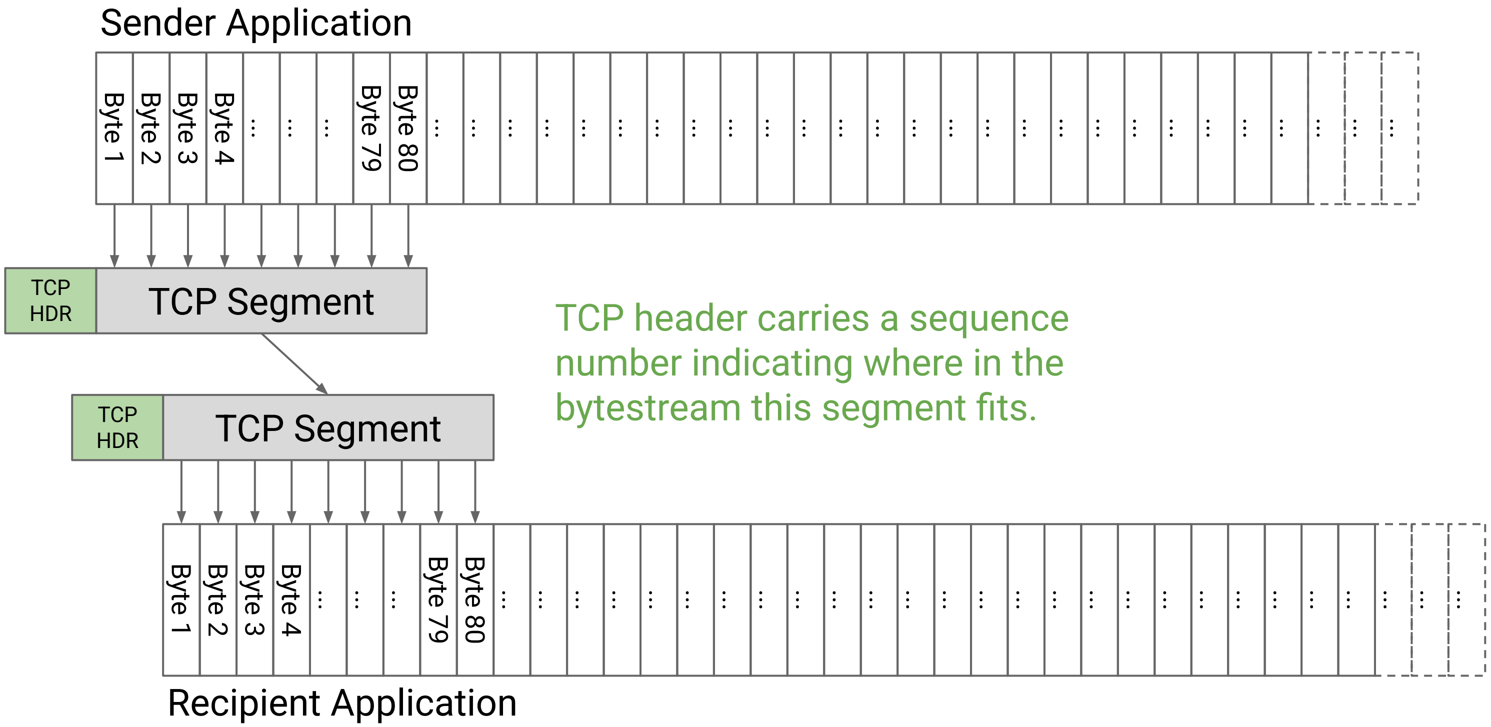
TCP provides a reliable, in-order bytestream.

We have to split this bytestream into packets.



TCP Segments

A segment is sent when the segment is full (max segment size), or when the segment is not full, but times out waiting for more data.



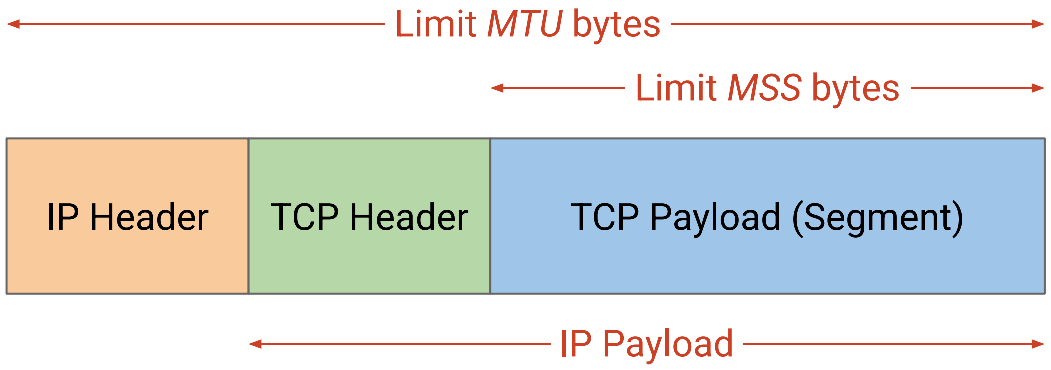
TCP header carries a sequence number indicating where in the bytestream this segment fits.

TCP Segments

TCP/IP packet: IP packet with TCP header and TCP data inside.

Size limits:

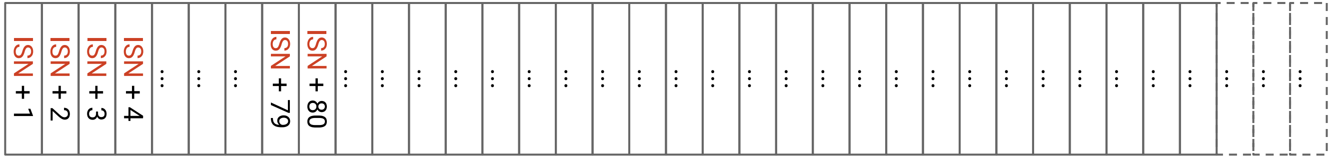
- IP packet: Maximum transmission unit (MTU).
- TCP segment: Maximum segment size (MSS).
- MSS = MTU – (IP header) – (TCP header).



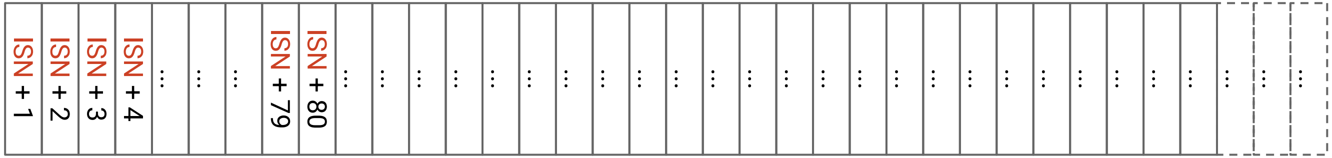
TCP Sequence Numbers

Numbering starts at a randomly-generated **Initial Sequence Number** (ISN).

- First byte is *ISN*+1, then *ISN*+2, etc.
- Starting at a randomly-chosen ISN is very important for security!



Sender Application

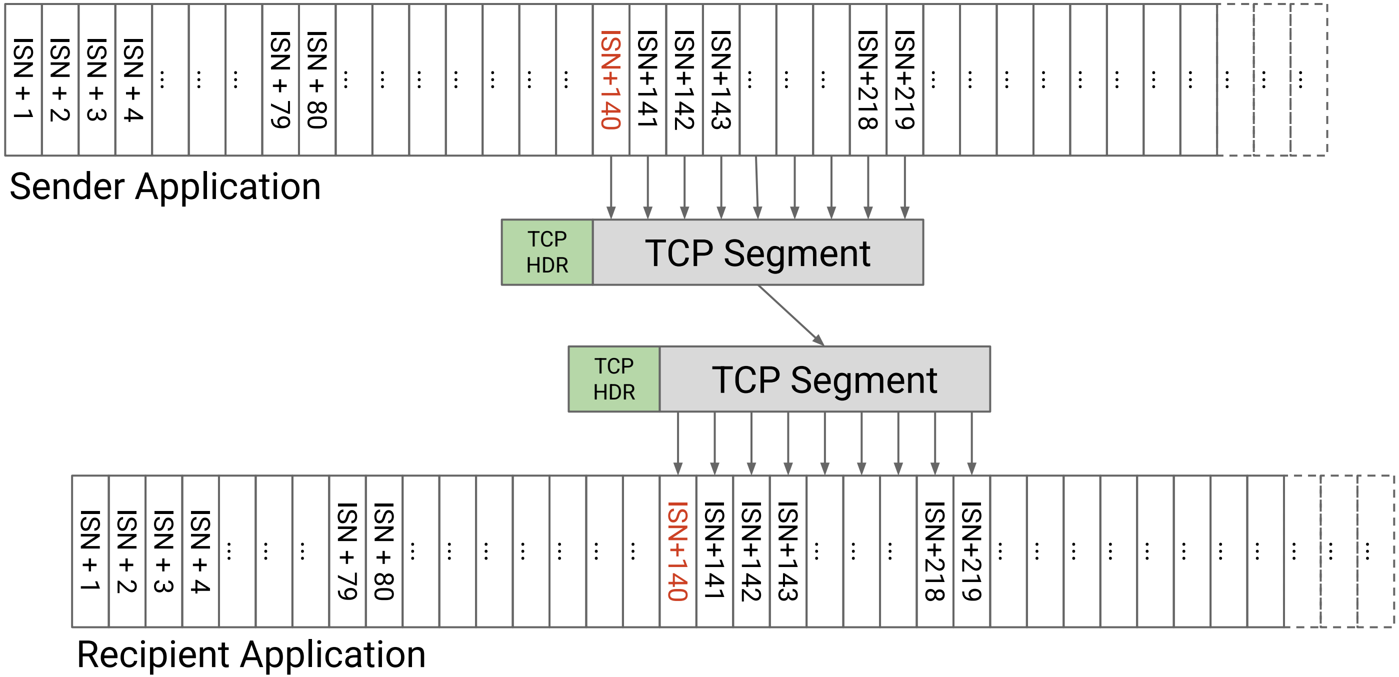


Recipient Application

TCP Sequence Numbers

The sequence number of a segment is the number of the *first byte* in the segment.

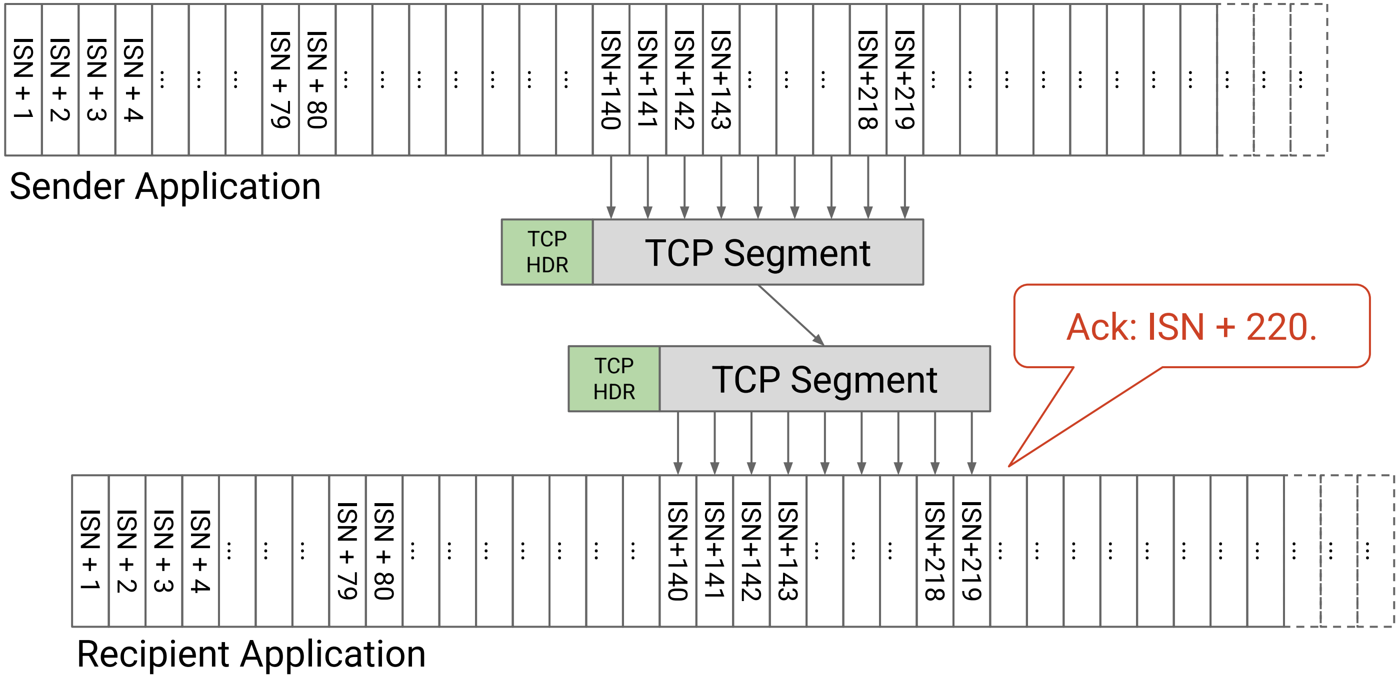
Example: In the segment below, the sequence number is *ISN* + 140.



TCP Ack Numbers

The ack number indicates the next expected byte (i.e. the first unreceived byte).

Example: All bytes up to (and including) *ISN* + 219 have been received, so the next unreceived byte is *ISN* + 220.



TCP Sequence and Ack Numbers

The sequence number of a segment is the number of the *first byte* in the segment.

- Sender sends a packet with sequence number j.
- The packet contains B bytes.
- Bytes in the packet are numbered: j, $j+1$, $j+2$, $j+3$, ..., , $j+B-1$.

Recipient sends a cumulative ack (number of highest byte received, plus one).

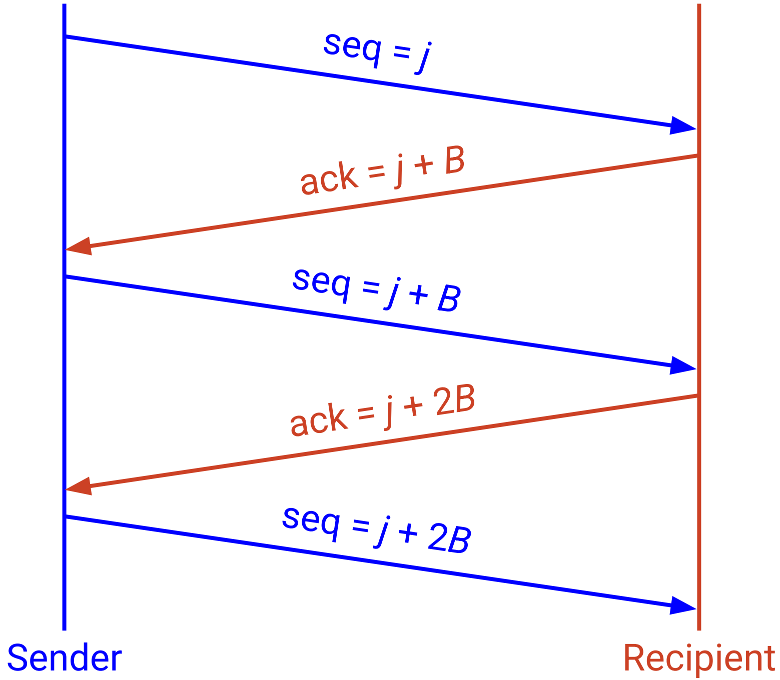
- If all prior data is received, ack number is $j+B$.
- Think of this as the next expected byte, or the first unreceived byte.
- If earlier data before this packet is missing, the ack number will be lower.

TCP Sequence and Ack Numbers

Assuming only one packet in flight, all packets length *B*, and no loss:

The last ack number is equal to the next sequence number.

- "I expect $j + B$ next." → "I'm sending $j + B$."



Maintaining State (Full Duplex)

Lecture 12, CS 168, Spring 2025

Implementing TCP

- Byte Notation (Segments, Sequence Numbers)
- **Maintaining State (Full Duplex, Connection Setup and Teardown)**
- Sliding Window
- Header

TCP State

Reliability requires maintaining *state* at the end hosts.

- Sender has to remember:
  - Which packets have been sent and not acked?
  - How much longer on the timer before I resend a packet?
- Receiver has to buffer the out-of-order packets.
- State is maintained at the end hosts, not in the network.

In each separate connection, both end hosts need to maintain state.

TCP is Full-Duplex

So far, we defined a sender and a recipient in every connection.

Connections in TCP are **full-duplex**.

- Both hosts can send data, and both hosts can receive data.
- A can send to B, and B can send to A, simultaneously, in the same connection.

To support full-duplex connections:

- Two sets of sequence numbers: One for A→B bytes, and one for B→A bytes.
- Each packet carries both data and ack information.
  - "Here are some bytes starting at 15..."
  - "...Also, I ack receiving all your bytes up to (not including) 84."

15	16	17	18	19	20	21
H	e	l	l	o	,	B

A to B bytestream

77	78	79	80	81	82	83
H	e	l	l	o	,	A

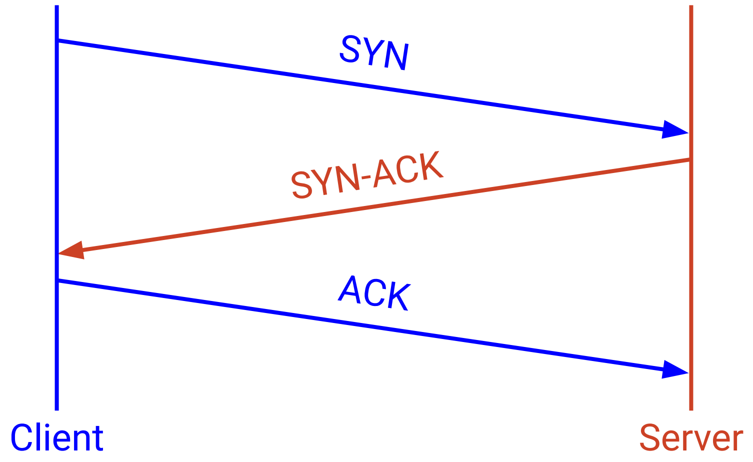
B to A bytestream

TCP Connection Setup: Three-Way Handshake

Goal: Each host tells its ISN to the other host.

1. SYN. Client says: "Here's my ISN."
2. SYN-ACK. Server says: "I received your ISN. Also, here's my ISN."
3. ACK. Client says: "I received your ISN."

After the three-way handshake, both sides can start sending data.

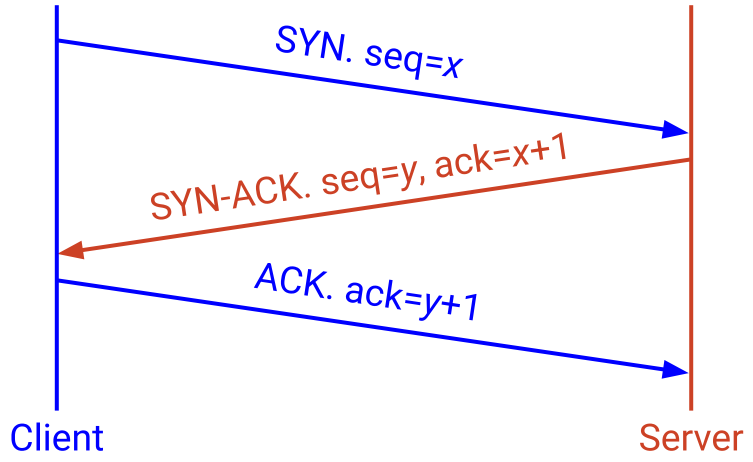


TCP Connection Setup: Three-Way Handshake

Goal: Each host tells its ISN to the other host.

1. SYN. Client says: "My ISN is *x*."
2. SYN-ACK. Server says: "I received *x* (expecting *x*+1 next). Also, my ISN is *y*."
3. ACK. Client says: "I received *y* (expecting *y*+1 next)."

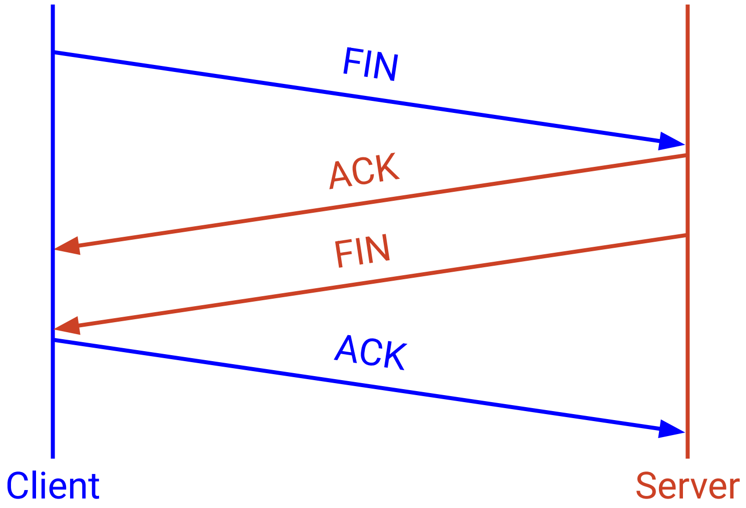
After the three-way handshake, both sides can start sending data.



TCP Connection Teardown

Normal termination:

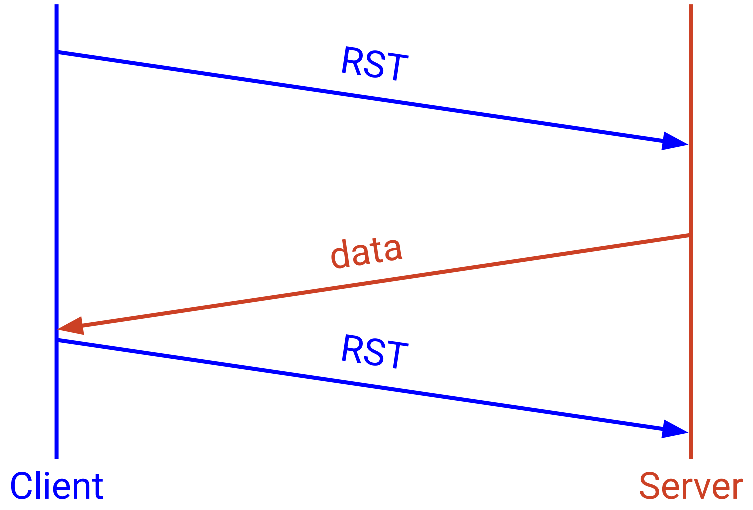
- Each side sends a FIN packet to say: "I'm done sending, but will keep receiving."
- FIN packets must be acked, just like any other data.
- When only one side has sent FIN, the connection is *half-closed*.
- When both sides have sent FIN (both done sending), the connection is closed.



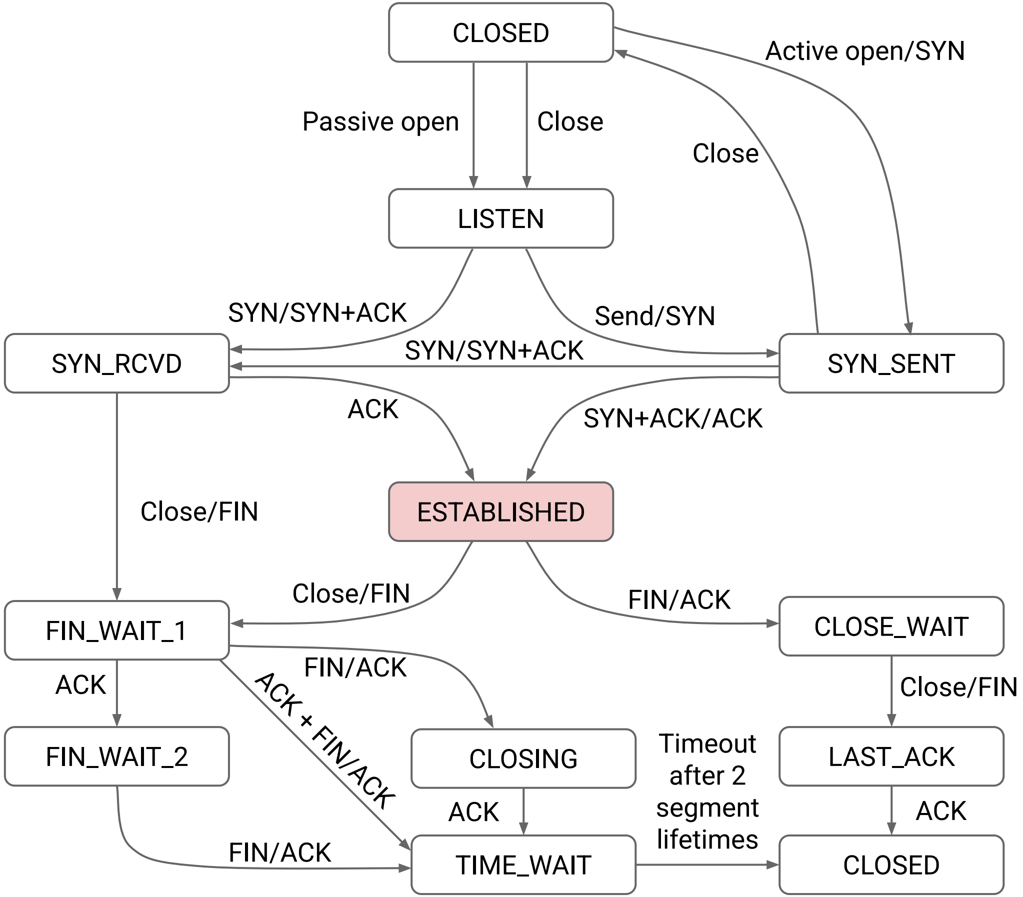
TCP Connection Teardown

Abrupt termination can be used instead (e.g. in case of error).

- Send a RST (reset) to say: "I will no longer send or receive data."
- RST packets do not need to be acked.
- Any data in flight is lost.
- If the RST sender receives more data later, send another RST.

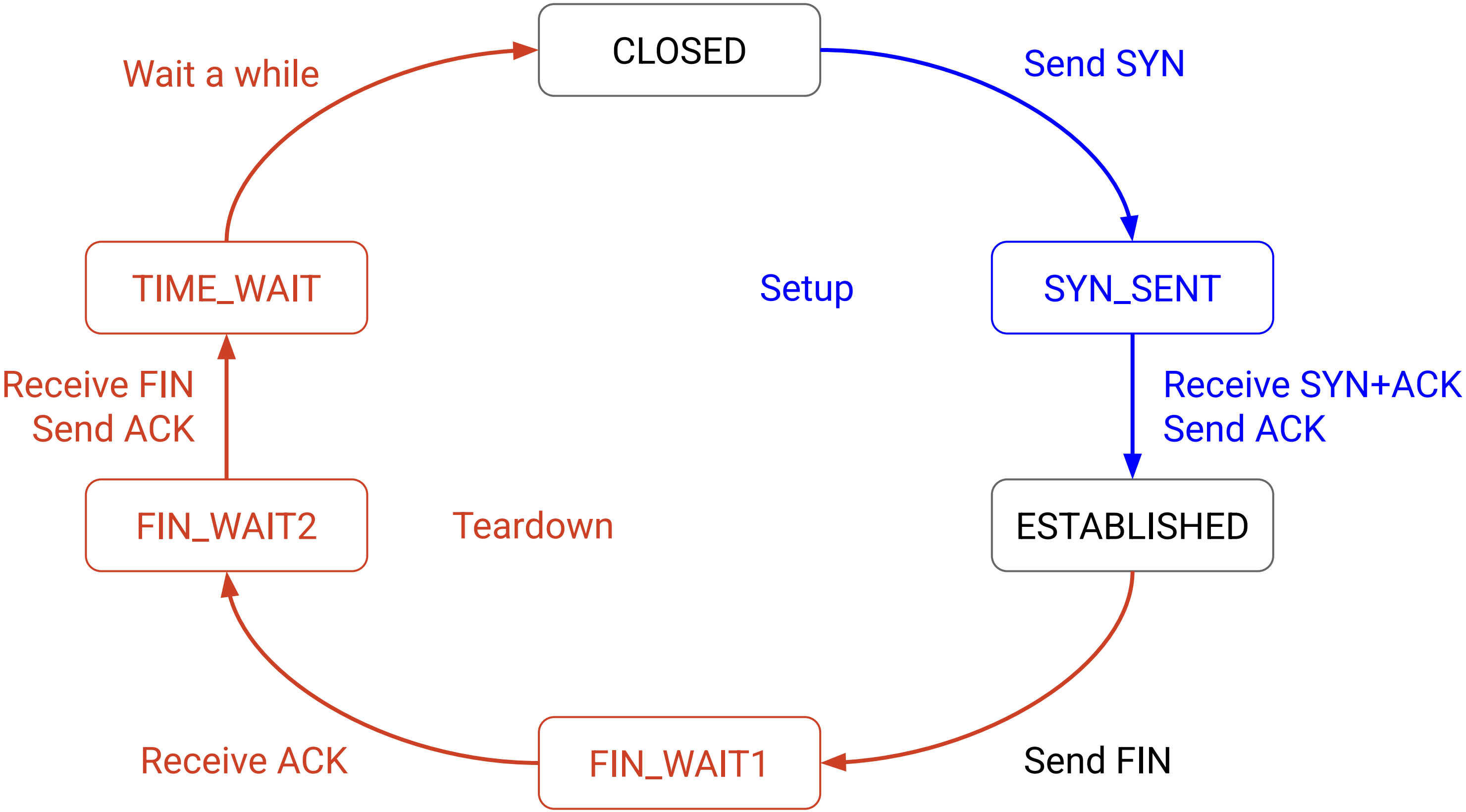


TCP Setup/Teardown Transition Diagram



The ESTABLISHED state is where all data is sent and acked.

TCP Setup/Teardown Transition Diagram (Simplified)



Piggybacking

With full-duplex, if we get a packet but have no data to send, we have two choices:

- Send the ack, with no data.
- **Piggybacking**: Wait for some data, and send the ack with the data.

Piggybacking can be tricky because TCP is in the OS, separate from the application.

- OS doesn't know when application will have more data.
- Application isn't thinking about packets and acks.

SYN-ACKs are always piggybacked.

- Ack and initial sequence number are sent together.
- Not tricky, because OS is doing the handshake, not the application.

Sliding Window

Lecture 12, CS 168, Spring 2025

Implementing TCP

- Byte Notation (Segments, Sequence Numbers)
- Maintaining State (Full Duplex, Connection Setup and Teardown)
- **Sliding Window**
- Header

TCP Sliding Window

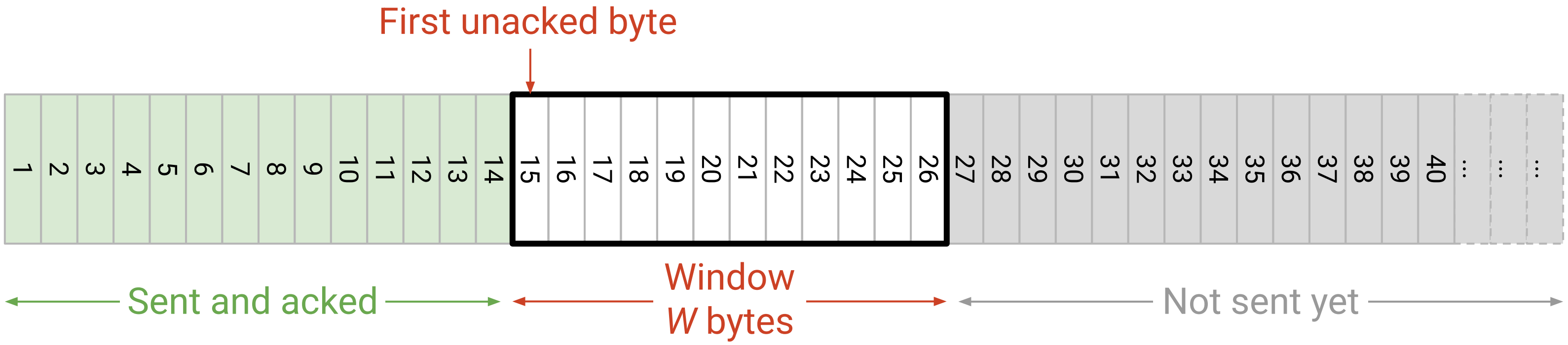
When we measured in packets, *W* was the maximum number of packets in flight.

When we measure in bytes, *W* is the maximum number of *contiguous* bytes in flight.

- The window is a range of *W* contiguous bytes, starting at the first unacked byte.
- Only these *W* bytes are allowed to be in flight.

The window slides right if and only if its *leftmost* bytes are acked.

- Example: When 15–18 arrive, we can send 27–30.



TCP Sliding Window

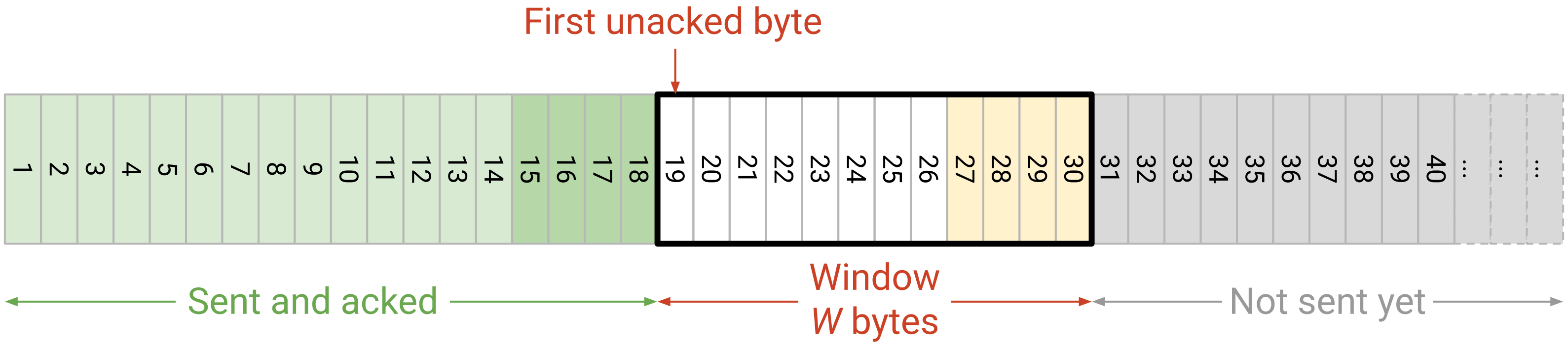
When we measured in packets, *W* was the maximum number of packets in flight.

When we measure in bytes, *W* is the maximum number of *contiguous* bytes in flight.

- The window is a range of *W* contiguous bytes, starting at the first unacked byte.
- Only these *W* bytes are allowed to be in flight.

The window slides right if and only if its *leftmost* bytes are acked.

- Example: When 15–18 arrive, we can send 27–30.



TCP Sliding Window

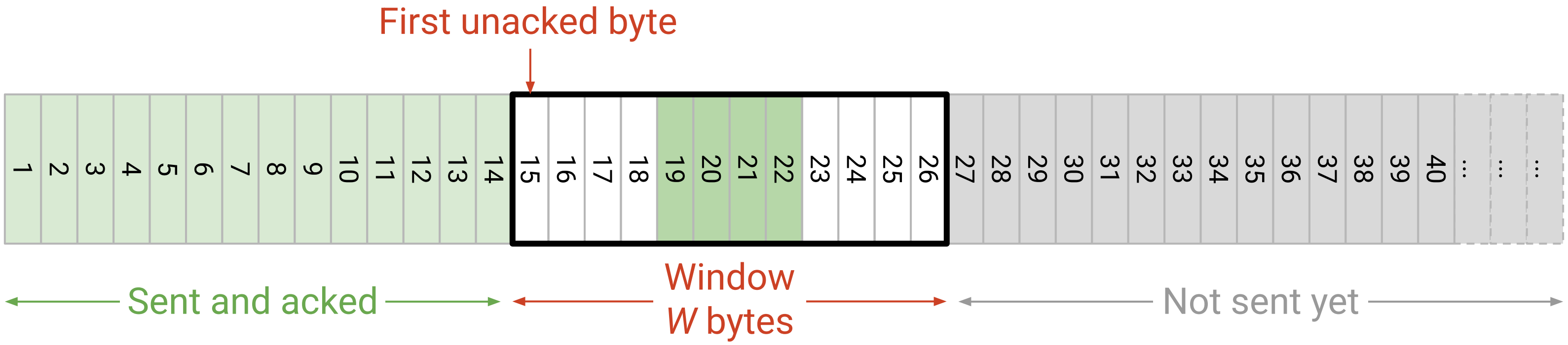
When we measured in packets, *W* was the maximum number of packets in flight.

When we measure in bytes, *W* is the maximum number of *contiguous* bytes in flight.

- The window is a range of *W* contiguous bytes, starting at the first unacked byte.
- Only these *W* bytes are allowed to be in flight.

The window slides right if and only if its *leftmost* bytes are acked.

- Acking non-leftmost bytes in the window (e.g. 19–22) does not slide the window.
- The window is determined by the first unacked byte (e.g. still 15).



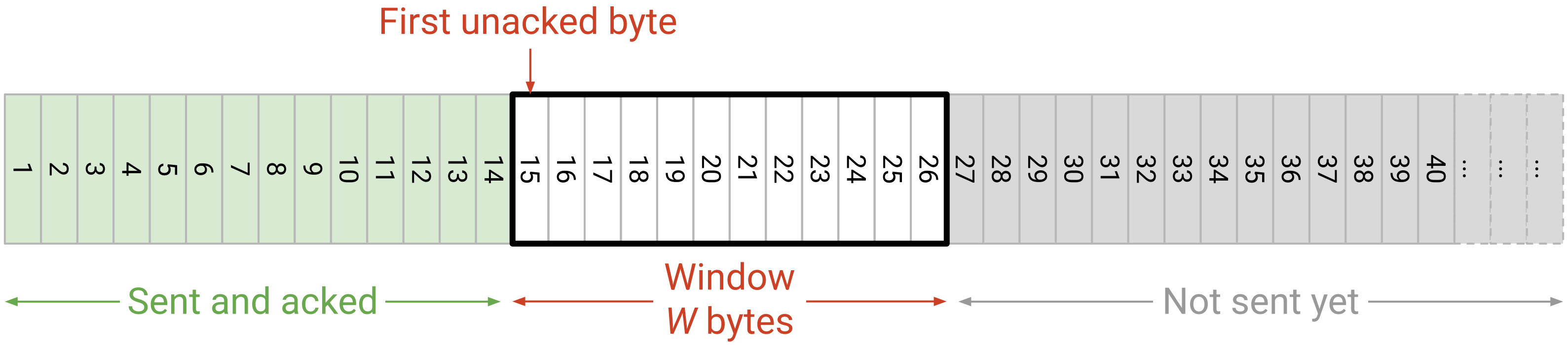
TCP Sliding Window

TCP uses cumulative acks and a sliding window.

- Cumulative ack: "I have received everything up to (not including) 15."
- Thus, first unacked byte is 15, and the sliding window is [15, 15 + *W*].

W is set as the minimum of two values:

- Advertised window (recipient reports their remaining buffer space).
- Congestion window (sender magically finds value to avoid network overload).

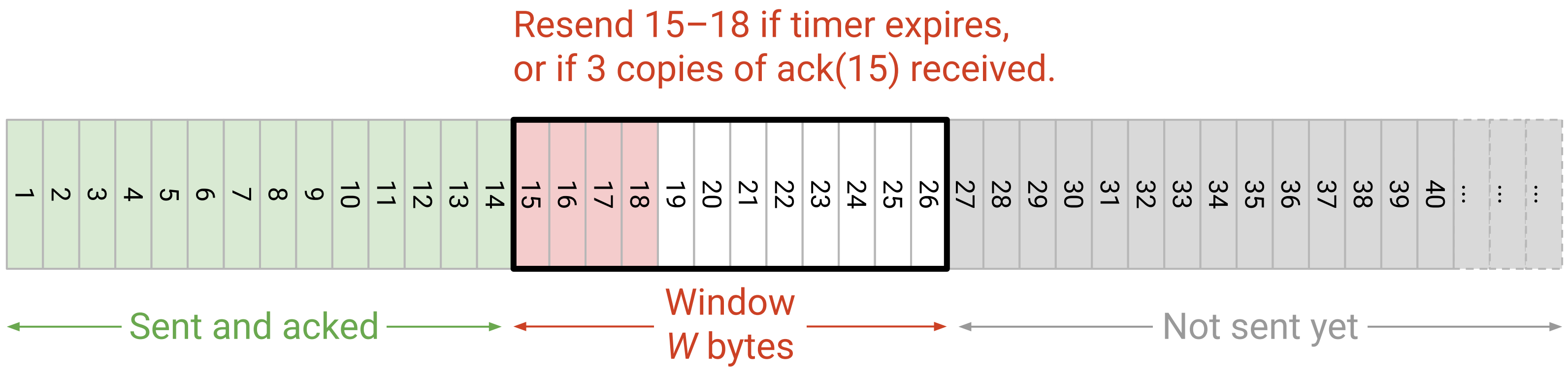


Detecting Loss

Two ways to detect loss and resend. We resend if either condition is true.

In both cases, we always resend the *first unacked packet* (leftmost part of window).

1. Ack-based: If 3 duplicate acks are received, resend.
2. Timer-based: Keep a single timer. If the timer expires, resend.
   - Different from packet-based TCP, where we kept one timer per packet.
   - Set timer by estimating RTT (e.g. measure times between packets and acks, and take a moving average).



TCP Header

Lecture 12, CS 168, Spring 2025

Implementing TCP

- Byte Notation
  (Segments, Sequence Numbers)
- Maintaining State (Full Duplex, Connection Setup and Teardown)
- Sliding Window
- **Header**

TCP Header

What functions does TCP implement?

1. Demultiplexing *(ports)*
2. Reliability *(checksum, sequence and ack numbers)*
3. Connection setup and teardown *(flags)*
4. Flow control *(advertised window)*

Source Port (16)		Destination Port (16)	
Sequence Number (32)			
Acknowledgment Number (32)			
Hdr Len (4)	0000	Flags (8)	Advertised Window (16)
Checksum (8)		Urgent Pointer (8)	
Options (variable-length)			
Payload			

TCP Header

There are 8 flags that can be set in TCP. We care about 4 of them:

- SYN: I'm sending my initial sequence number.
- ACK: I'm acking data (please look at the ack number).
- FIN: I'm done sending data, but will keep receiving data.
- RST: I'm done sending and receiving data.

We won't look at the other four: CWR, ECE, URG, PSH.

Source Port (16)			Destination Port (16)
Sequence Number (32)			
Acknowledgment Number (32)			
Hdr Len (4)	0000	Flags (8)	Advertised Window (16)
Checksum (8)			Urgent Pointer (8)
Options (variable-length)			
Payload			

TCP Header

Remaining fields:

- Header length: Measured in 4-byte words.
  - If no options, this is 5 (header is 20 bytes long).
- `0000`: Reserved bits (always set to 0).
- Urgent pointer: Used with the URG flag to indicate urgent data. Won't discuss.
- Options: Extra functionality.
  - Example: SACK uses the options field to implement full-information acks.

Source Port (16)			Destination Port (16)
Sequence Number (32)			
Acknowledgment Number (32)			
Hdr Len (4)	`0000`	Flags (8)	Advertised Window (16)
Checksum (8)			Urgent Pointer (8)
Options (variable-length)			
Payload			

TCP: Summary

An elegant (though not perfect) piece of engineering that has stood the test of time.

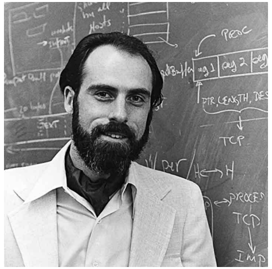
- Thought experiment: Will TCP continue to be a good solution?

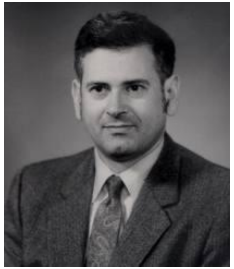
Plenty of evolution in individual pieces:

- Congestion control was added after-the-fact.
- Better acknowledgments, ISN selection, timer estimation, etc.

But the core architectural decisions and abstractions remain:

- Bytestreams, connection-oriented, windows, etc.





Vint Cerf and Bob Kahn have won basically every award ever for their work on TCP.