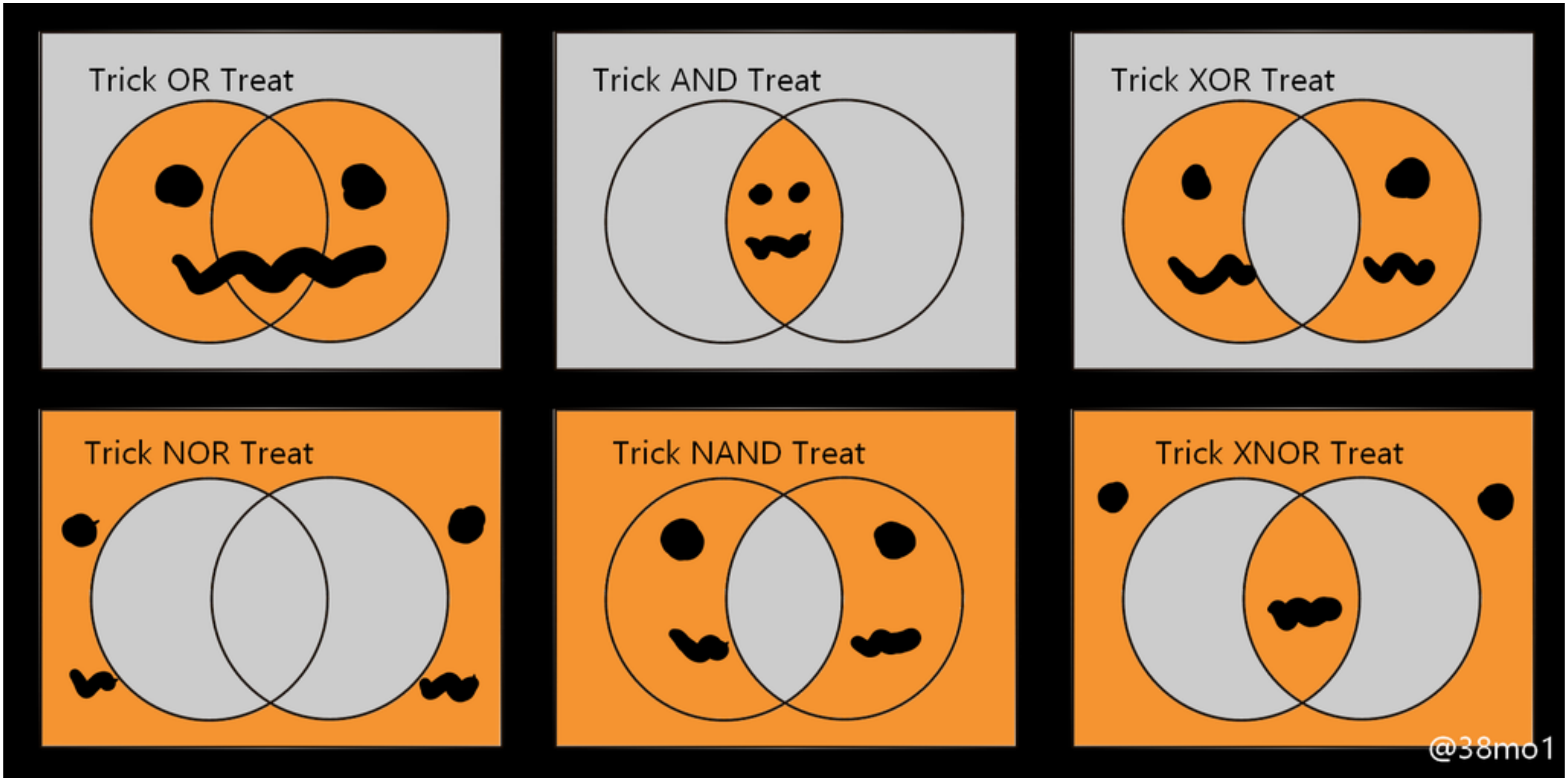
CS 43: Computer Networks

TCP Connections and Flow Control

October 27, 2020

“Boo”lean Logic

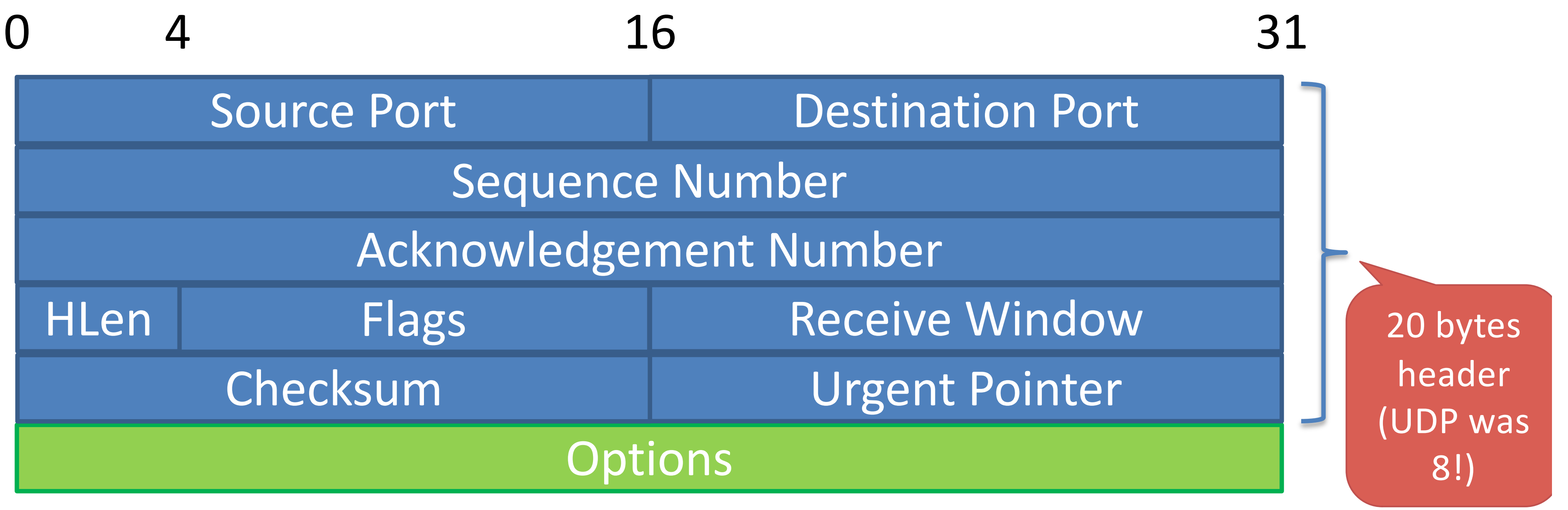


Courtesy: Kevin Webb

Transmission Control Protocol

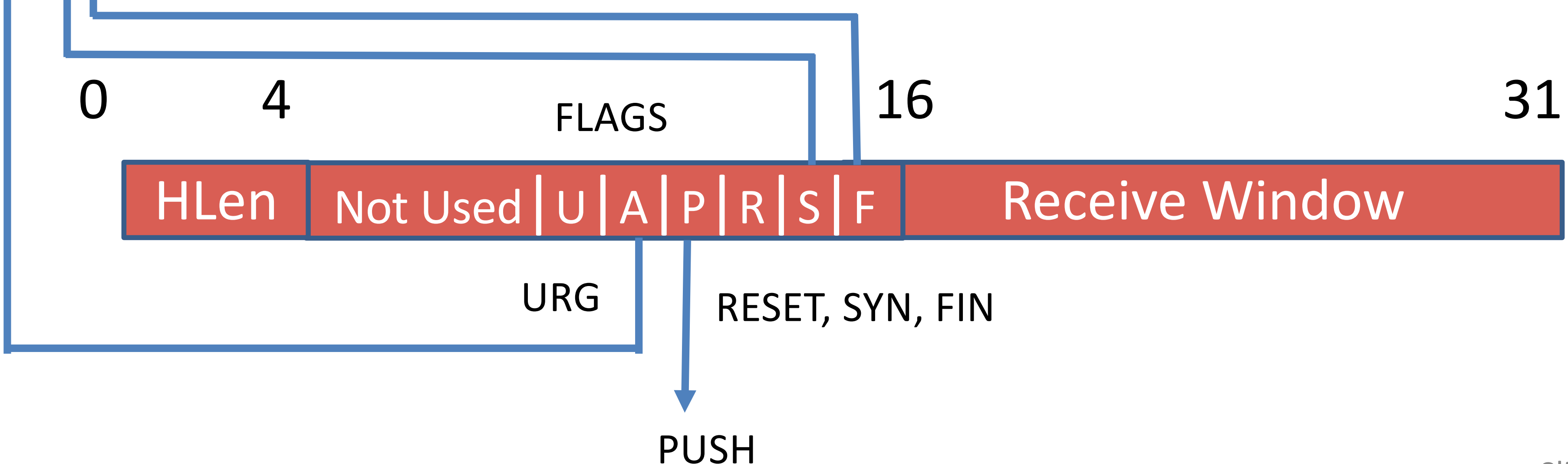
Reliable, in-order, bi-directional byte streams

- Port numbers for demultiplexing
- Flow control
- Congestion control, approximate fairness



Transmission Control Protocol

- Important TCP flags (1 bit each)
  - ACK – acknowledge received data (ACK valid or not)
  - SYN – synchronization, used for connection setup
  - FIN – finish, used to tear down connection



Practical Reliability Questions

- What does connection establishment look like?
- How do we choose sequence numbers?
- How do the sender and receiver keep track of outstanding pipelined segments?
- How should we choose timeout values?
- How many segments should be pipelined?

Practical Reliability Questions

- What does connection establishment look like?
- How should we choose timeout values?
- How do the sender and receiver keep track of outstanding pipelined segments?
- How do we choose sequence numbers?
- How many segments should be pipelined?

A connection…

1. Requires stored state at two hosts.
2. Requires stored state within the network.
3. Establishes a path between two hosts.

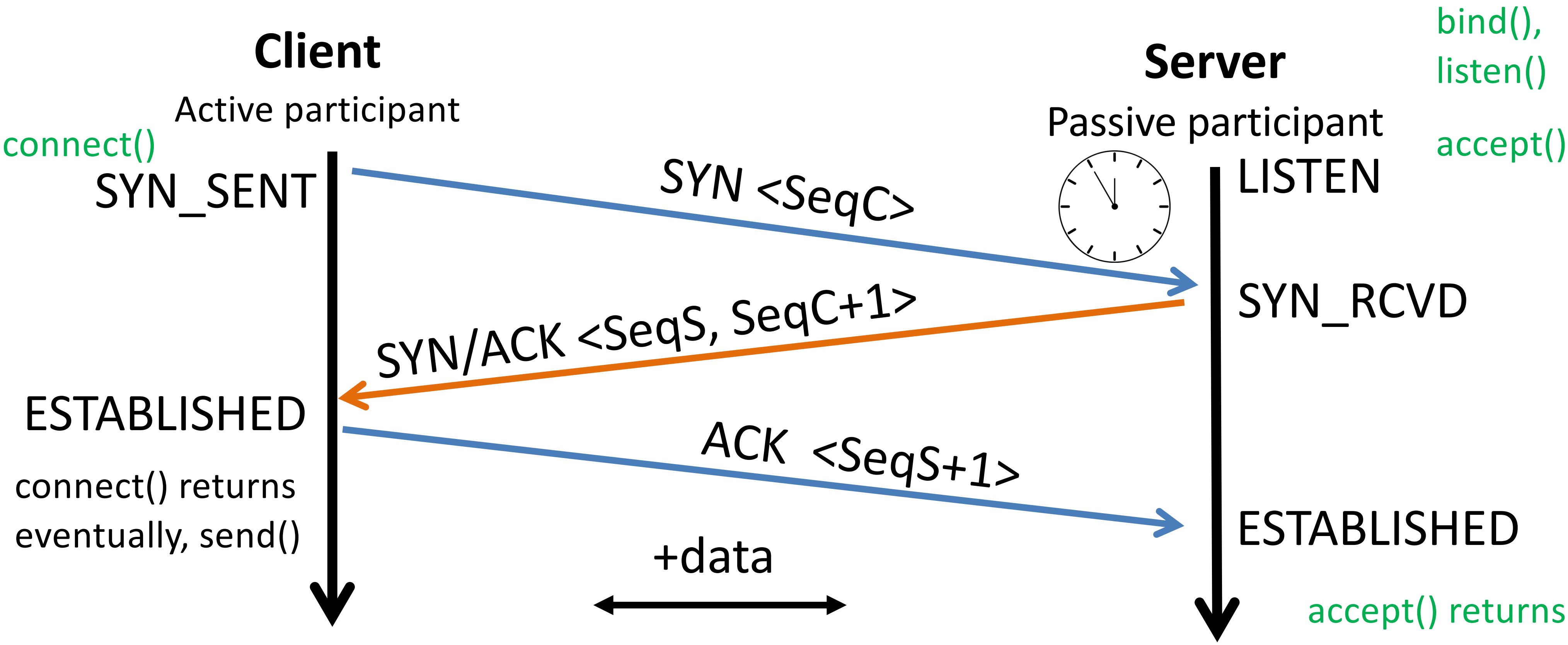
A. 1
B. 1 & 3
C. 1, 2 & 3
D. 2
E. 2 & 3

Connections

- In TCP, hosts must establish a connection prior to communicating.

- Exchange initial protocol state.
  - sequence #s to use.
  - maximum segment size (MSS)
  - Initial window sizes, etc. (several parameters)

Three Way Handshake



Both sides agree on connection.

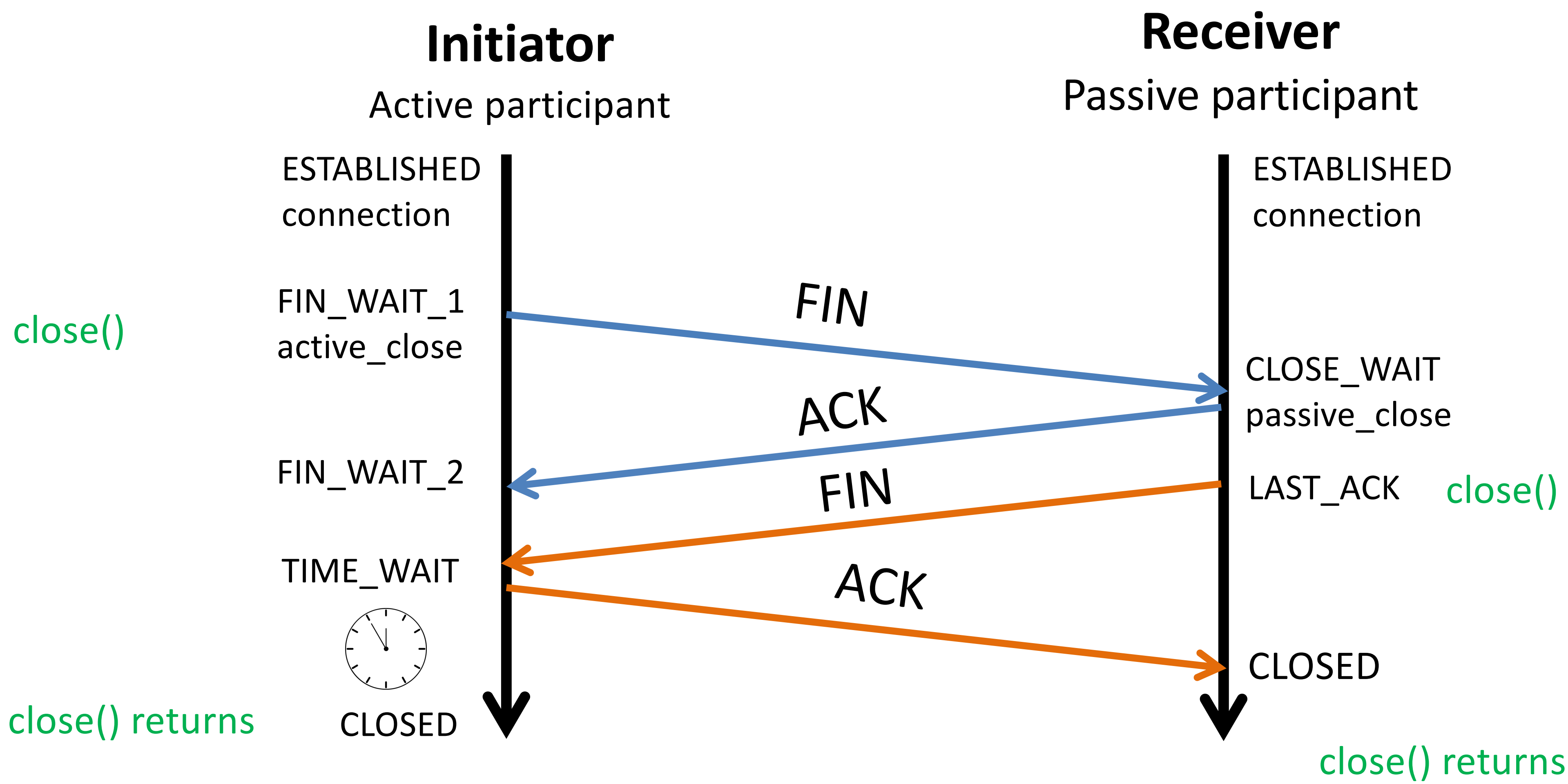
Connection Teardown

- Orderly release by sender and receiver when done
  - Delivers all pending data and “hangs up”

- Cleans up state in sender and receiver

- Each side may terminate independently

TCP Connection Teardown

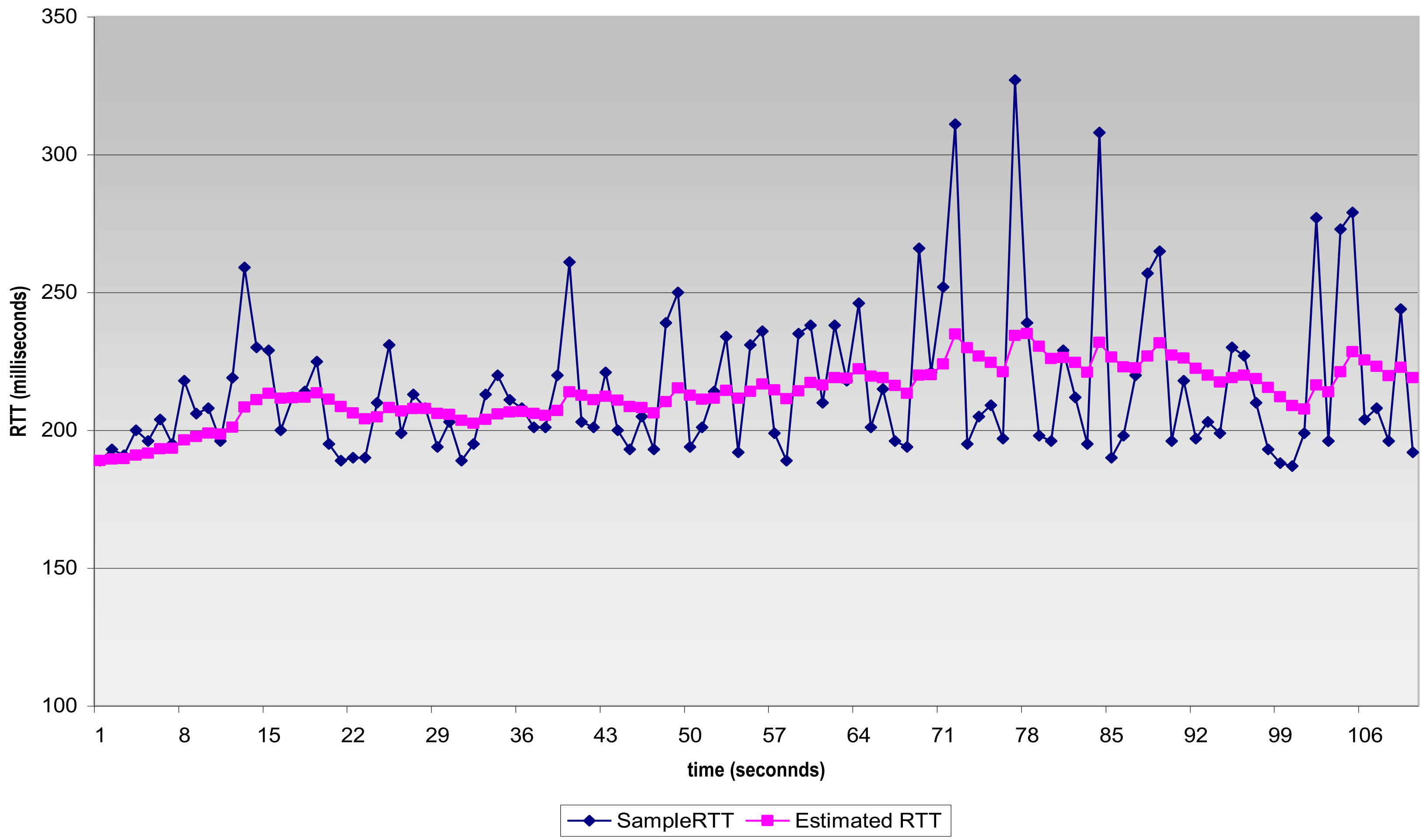


Both sides agree on closing the connection.

Practical Reliability Questions

- What does connection establishment look like?
- How do we choose sequence numbers?
- How should we choose timeout values?
- How do the sender and receiver keep track of outstanding pipelined segments?
- How many segments should be pipelined?

Example RTT Estimation (Smoothing)



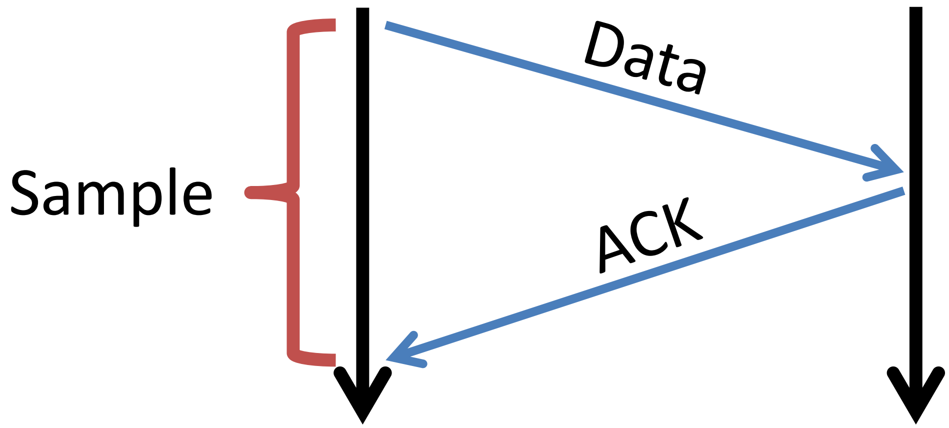
TCP Timeout Value

`TimeoutInterval = EstimatedRTT + 4*DevRTT`

estimated RTT (EstimatedRTT)

"safety margin" (4*DevRTT)

Round Trip Time Estimation: Exponentially Weighted Moving Average (EWMA)



<u>EstimatedRTT = (1 – a) * EstimatedRTT + a * SampleRTT</u>

- a is usually 1/8.

In words current estimate is a blend of:

- 7/8 of the previous estimate
- 1/8 of the new sample.

<u>DevRTT = (1 – B) * DevRTT + B * | SampleRTT – EstimatedRTT |</u>

- B is usually 1/4

Example RTT Estimation

- Suppose EstimateRTT = 64, Dev = 8
- Latest sample: 120

New estimate = 7/8 * 64 + 1/8 * 120 = 56 + 15= 71

New dev = 3/4 * 8 + 1/4 * | 120 - 71 | = 6 + 12 = 18

- Another sample: 400

New estimate = 7/8 * 71 + 1/8 * 400 = 62 + 50 = 112

New dev = 3/4 * 18 + 1/4 * | 400 - 112 | = 13 + 72 = 85

Practical Reliability Questions

- What does connection establishment look like?
- How do we choose sequence numbers?
- How should we choose timeout values?
- How do the sender and receiver keep track of outstanding pipelined segments?
- How many segments should be pipelined?

Sliding window

- How many bytes to pipeline?
- How big do we make that window?
  - Too small: link is under-utilized
  - Too large: congestion, packets dropped
  - Other concerns: fairness

Discussion: Why do we need rate control ?

A. to help the global network (core routers, and other end-hosts)
B. to help the receiver
C. to help the sender
D. some other reason

Shared high-level goal: don't waste capacity by sending something that is likely to be dropped.

Rate Control

Flow Control

- Don’t send so fast that we overload the receiver.
- Rate directly negotiated between one pair of hosts (the sender and receiver).

Congestion Control

- Don’t send so fast that we overload the network.
- Rate inferred by sender in response to “congestion events.”

Shared high-level goal: don’t waste capacity by sending something that is likely to be dropped.

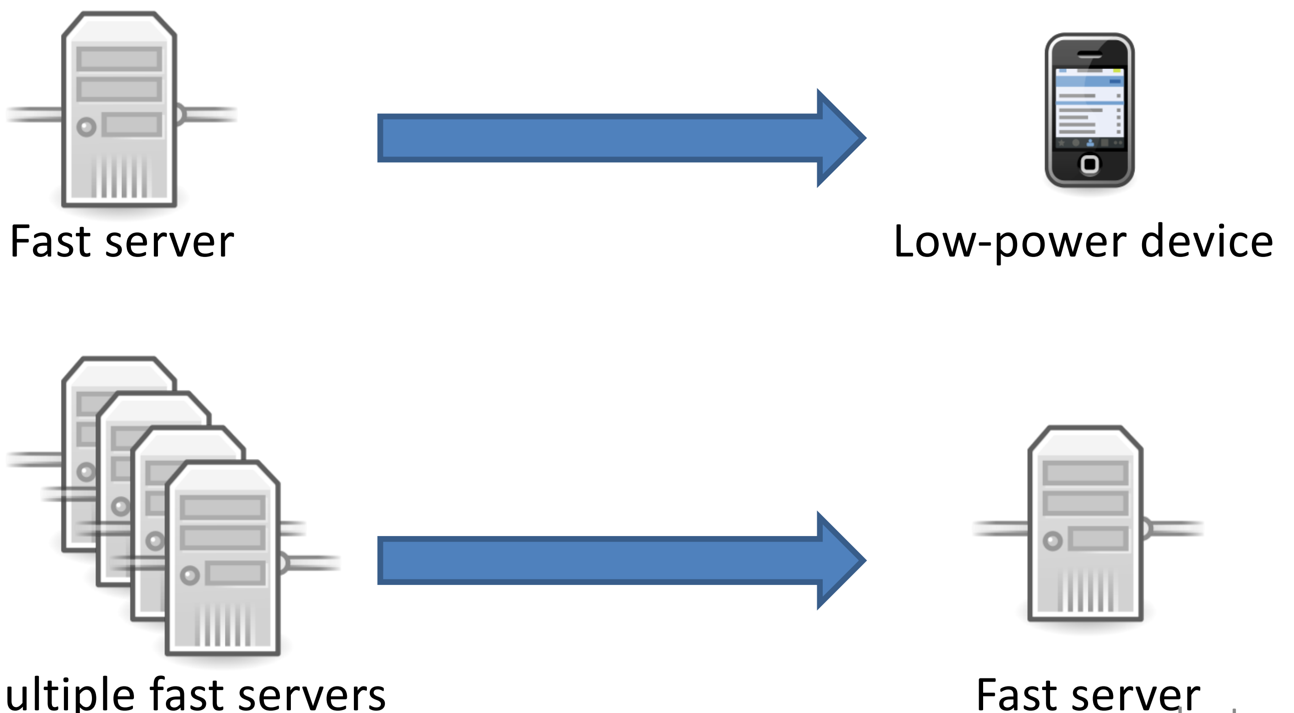
Flow Control

- Don't send so fast that we overload the receiver.
- Rate directly negotiated between one pair of hosts (the sender and receiver).

Flow Control

Problem: Sender can send at a high rate. Network can deliver at a high rate. The receiver is drowning in data.

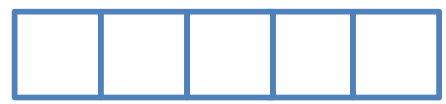
- Example scenarios:



Flow Control

Fast server

Low-power device

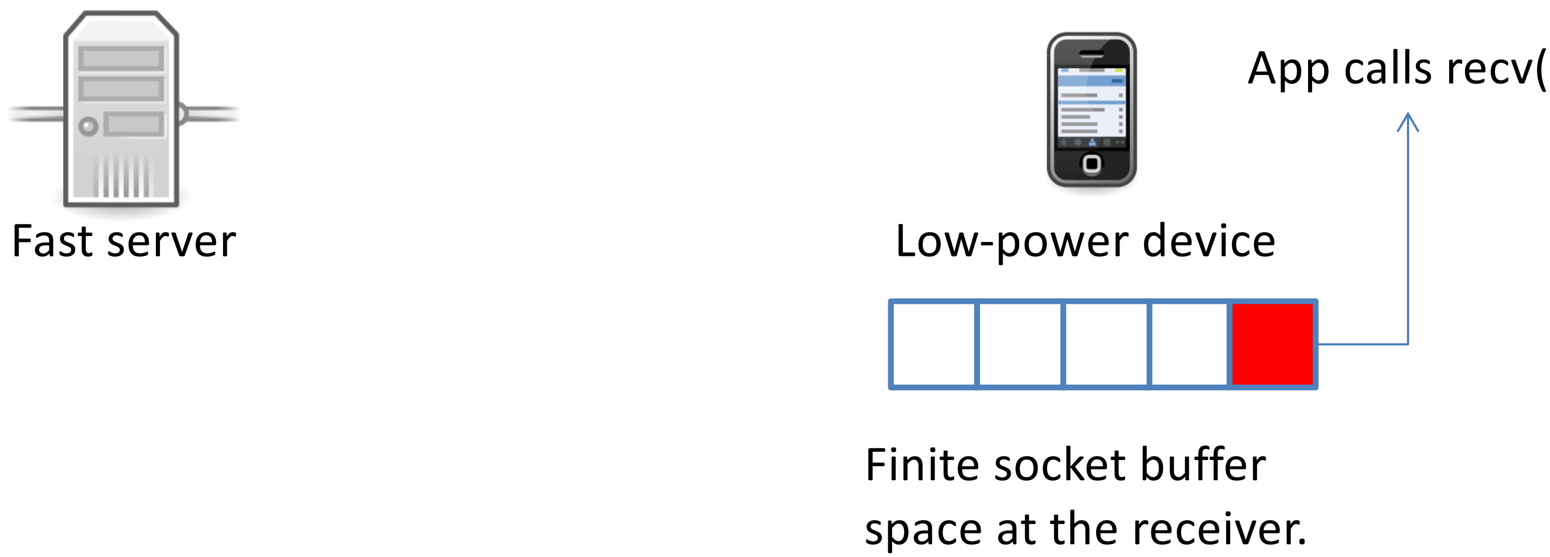


Finite socket buffer
space at the receiver.

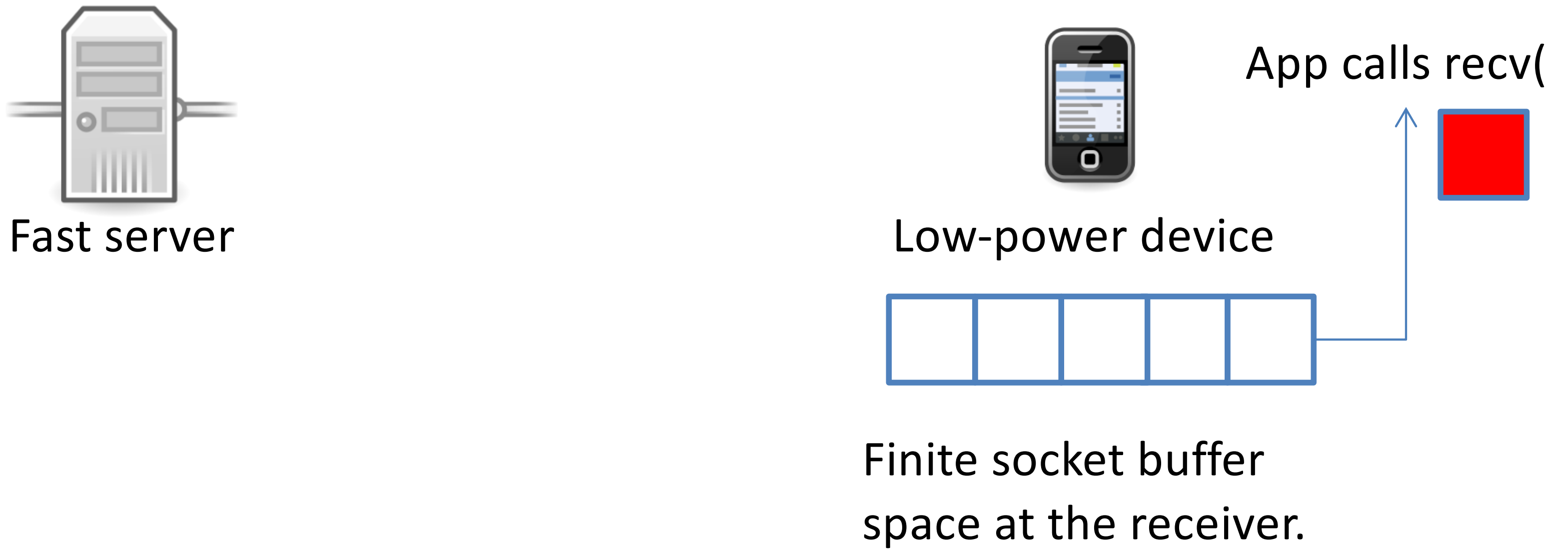
Flow Control

Finite socket buffer space at the receiver.

Flow Control



Flow Control



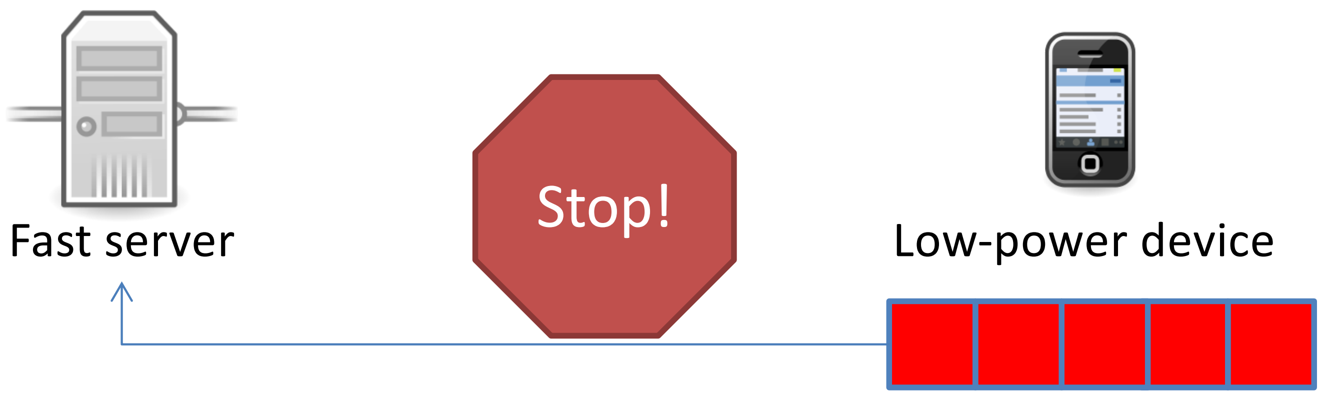
Flow Control

Finite socket buffer space at the receiver.

Flow Control

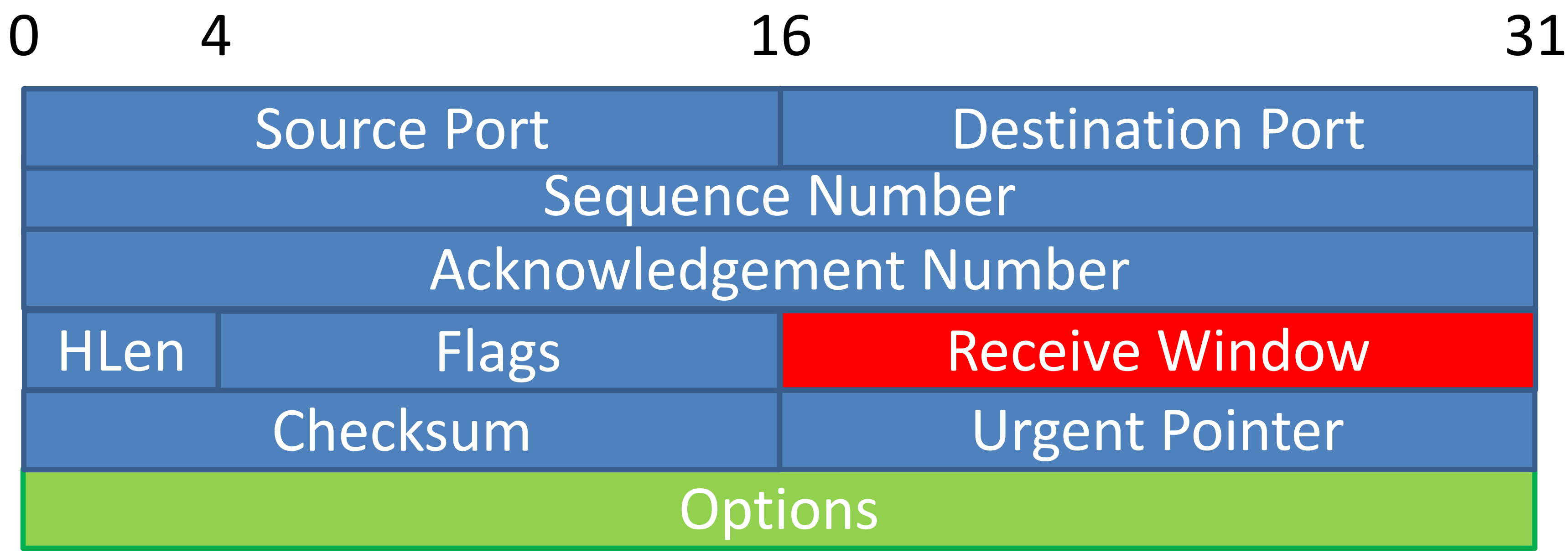
Finite socket buffer space at the receiver.

Flow Control



Finite socket buffer
space at the receiver.

TCP Receive Window (rwnd)



Flow Control

- Sender never sends more than rwnd.

Finite socket buffer
space at the receiver.

Flow Control

- Sender never sends more than rwnd.

Finite socket buffer space at the receiver.

Flow Control

- Sender never sends more than rwnd.