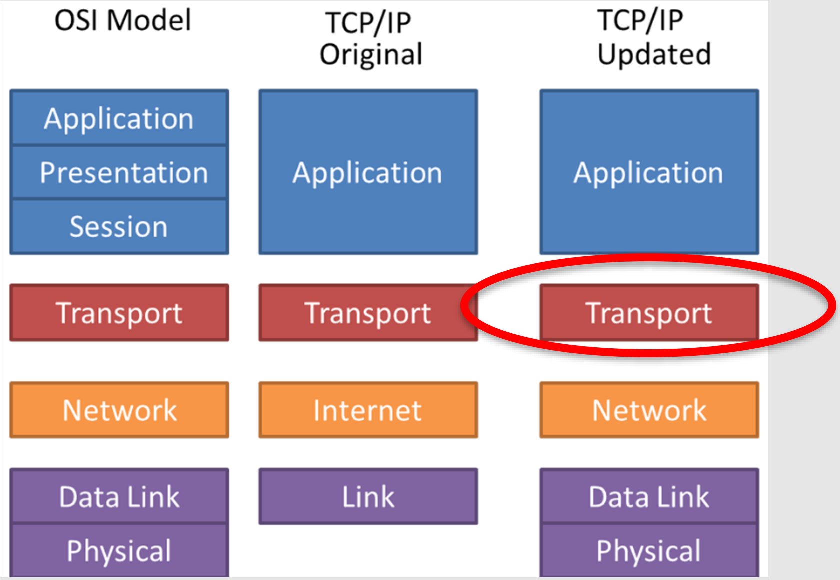
Transport Layer

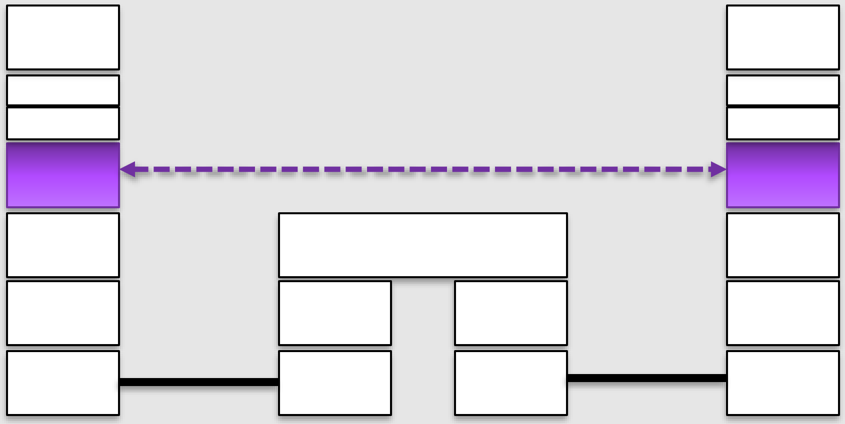
Protocols

- Rules of operation of a network

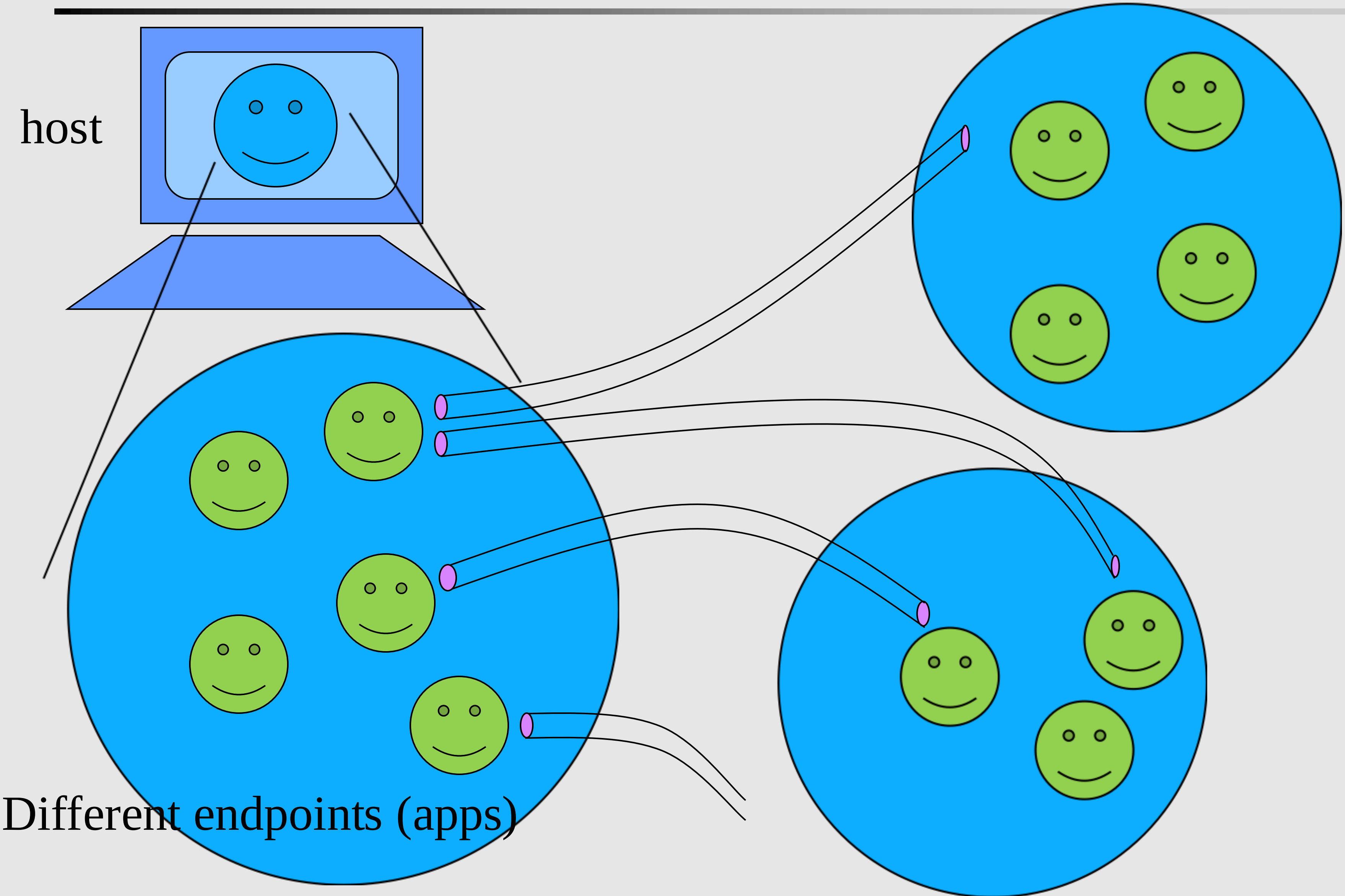


Transport Layer

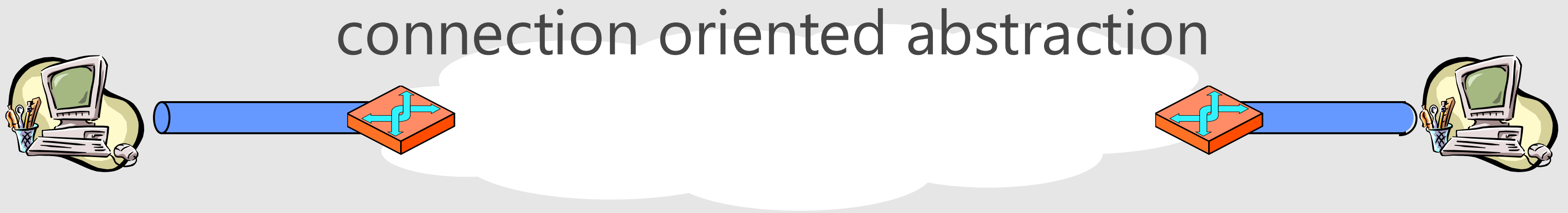
- First end-to-end layer
- Functions
  - Endpoint abstraction
    - Multiplexing on a host
  - Context establishment
  - Enhancements
    - Reliability (ARQ)
    - Flow Control
    - Other services



Communication Endpoints



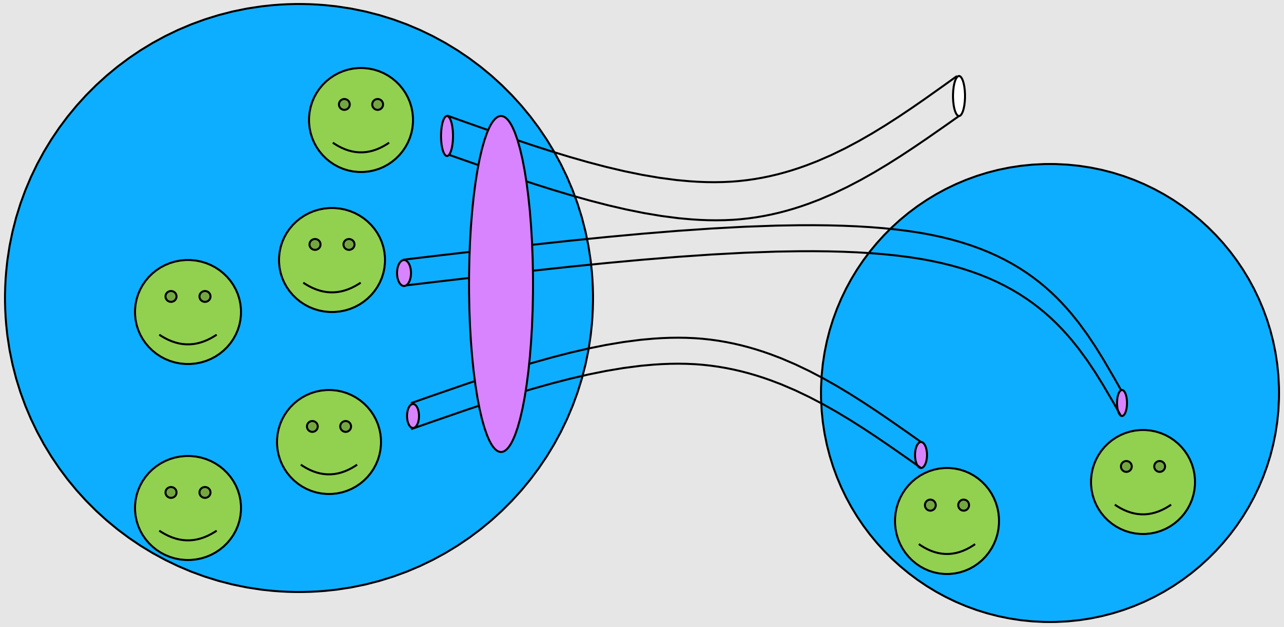
End-to-End Transport



- Peer layer is only at the remote host
  - Point-to-point cooperation, like DLC
  - Hence DLC mechanisms like ARQ can be applied
    - For flow control
    - For reliability (error control - retransmission)
  - However, the challenge is more complex
    - Multiple applications at each endpoint
    - Network may lose, reorder, or duplicate packets
  - Need:
    - Context (the state information of each connection)
    - Explicit addressing, both for destination host and endpoint

Endpoint Access

- Transport software built in two parts
  - Host specific part - multiplexes network layer, global context
  - Application specific part - maintains flow state and provides reliable network
    - Also provides access point for higher layers (sockets in TCP)
    - Socket – TCP or UDP, IP address, port number of an app (these parameters uniquely identify a network connection.)



Transport-Layer Protocol

- User Datagram Protocol (UDP)
  - connection-less
  - no reliability, sequencing, congestion control, flow control, or connection management
  - real-time traffic, e.g., Skype, Zoom, etc.

- Transmission Control Protocol (TCP)
  - connection-oriented
  - reliable (congestion control, flow control, sequencing)
  - e.g., streaming applications, Web

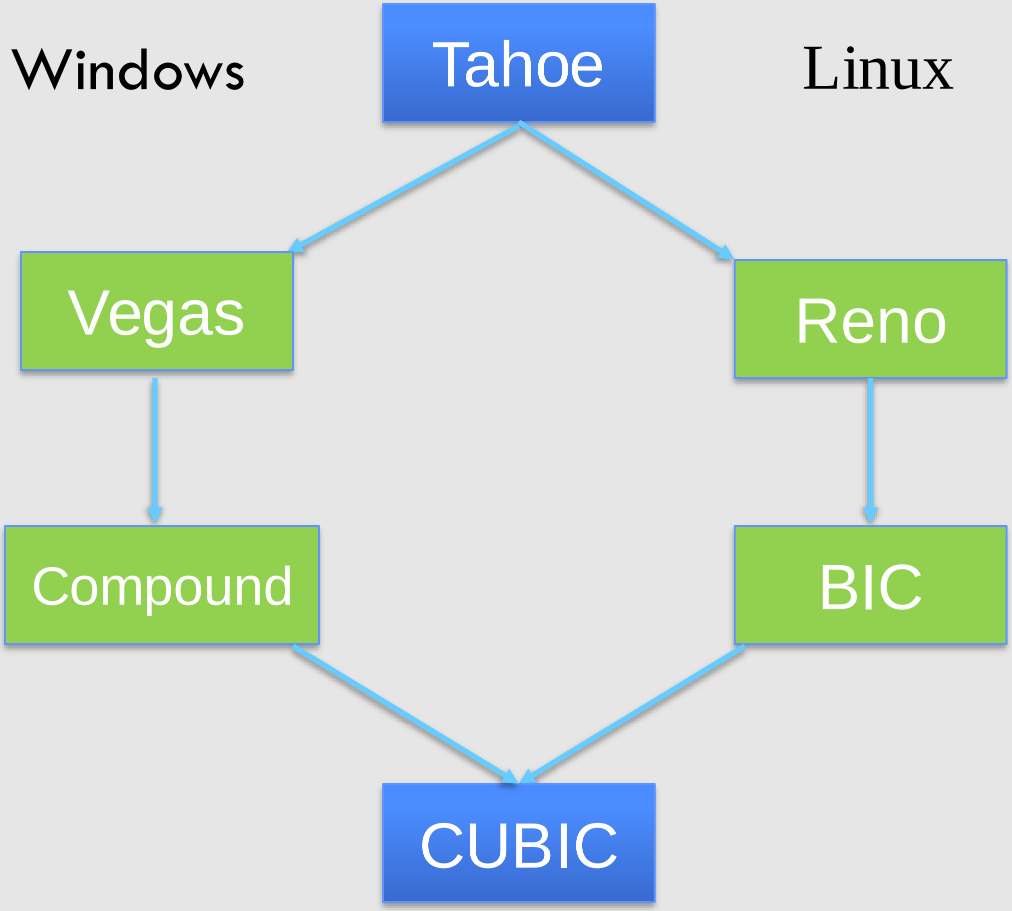
TCP Design Goals

- TCP is a transport protocol
  - Does the process-level mux-demux (ports) of apps (as UDP does)
- TCP attempts to add several functionalities:
  - Reliability
    - No erroneous data, guaranteed in-order delivery
  - Flow control
    - Do not swamp a slow receiver
  - Rate (Congestion) control
    - Do not swamp a congested network
- In order to reach above goals, TCP adopts stream orientation (sequences each byte instead of packet)

TCP Overview

- Transmission Control Protocol (RFCs 793, 1122, 1323)
- Segment: unit of transfer, usually contained in a single IP datagram
- Reliability achieved using
  - Acknowledgments (therefore, seq. no.s)
  - Timeouts, retransmissions
  - Checksums on header and data
- Sliding window mechanism for
  - efficient transmission
  - flow control
  - congestion control

History of TCP



Properties of TCP Reliable Service

- **Stream orientation**: TCP thinks of data as a **stream of bytes**
- Unstructured stream: TCP does not honor structured data
- **Connection orientation**: state maintained at both ends
- Buffered transfer: the software divides data stream into segments independent of application program transfers
- Full duplex connection: concurrent data transfer in both directions

TCP Header Format

2 bytes | 2 bytes

Source port number			Destination port number
Sequence number			
Acknowledgement number			
Header Len	Reserved (6)	Control Field (6)	Window Size
Checksum			Urgent Pointer
Options			

20 bytes

TCP Connections and Endpoints

- TCP uses *destination port* to identify ultimate destination (endpoint address)
- TCP uses connection as fundamental abstraction
  - *Connection*: a pair of endpoints
  - *Endpoint*: (host_IP_address, port #) (socket)
- TCP ports
  - A port # does not correspond to a single object
  - A TCP port number can be shared by multiple connections on the same machine (remote endpoints differ)

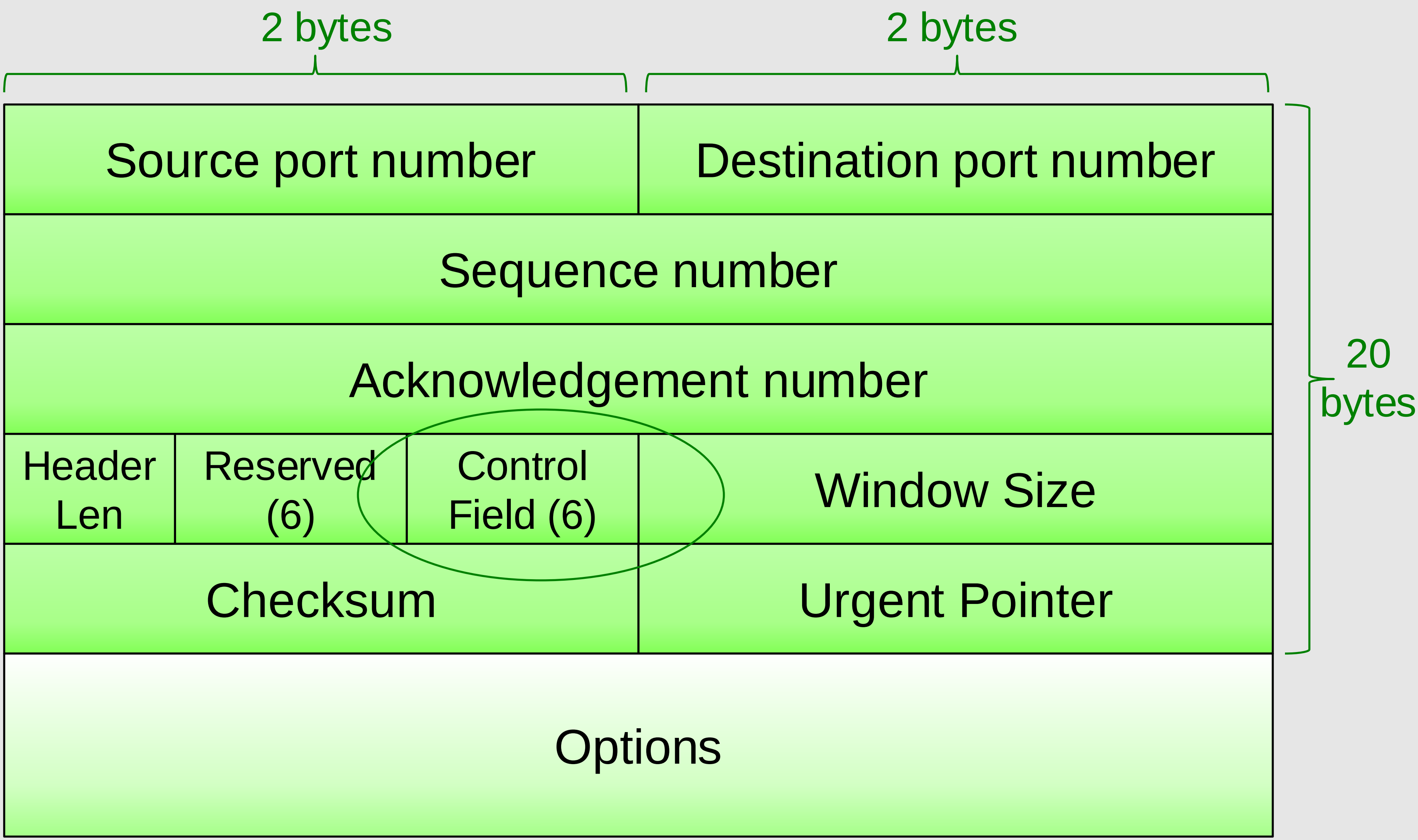
TCP Header Fields

- *Sequence number*
  - Every **byte** in data stream is numbered
  - Sequence number = number (in the sender's byte stream) of the **first data byte** in this segment
- *ACK number*
  - Next sequence number the sender of the acknowledgment expects to receive
  - Valid field only when ACK flag = 1
- *Header length*
  - With maximum value of 15 (2^4-1)

TCP Header Fields

- *Window size*
  - Number of bytes (starting with the one specified in the ACK field) that receiver is willing to accept
  - 16 bits long; max value is 65535
  - Used for flow control
- *Checksum*
  - Similar to IP computation; includes a “pseudo-header”;
  - ***For entire segment (including data)***
  - Use is mandatory
- *Urgent pointer*
  - Sequence number of last byte of urgent data (later)
  - Added to sequence number of segment
- *Options*
  - Most common option is *maximum segment size* (MSS)

TCP Header Format



Description of Flags in Control Field

Flag	Description
URG	The value of the urgent pointer is valid
ACK	The value of the acknowledgment number is valid
PSH	Push the data (pass data to receiver as quickly as possible)
RST	The connection must be reset (e.g., blocked)
SYN	Synchronize the sequence numbers during connection establishment (start a new connection)
FIN	The sender has no more data to transmit (end)

Maximum Segment Size Option

- Code: 2
- Declared during connection establishment phase (in SYN segments)
  - Cannot be re/specified during data transfer
- Defines the maximum segment size of *data* the sender is willing to accept
- MSS must be $\leq$ interface MTU - 40 bytes

Review: DLC ARQ – Window Size

- Idea: to fill the pipe entirely
  - w: window size (to set)
  - B: bandwidth (bps) divided by frame size (bits-per-frame)
    - Expressed in terms of frame rate, or fps (not fps in video gaming :)
  - D: one-way transit time/delay
  - BD: bandwidth-delay product
  - Upper bound on link utilization:

$$\text{link utilization} \leq \frac{w}{1 + 2BD}$$

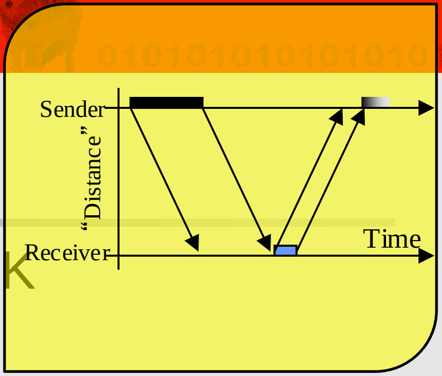
  - w ~ 1 + 2BD

Long Fat Pipes (LFN)

- Bandwidth-delay product (capacity of "pipe")
  - Bandwidth (b/s) X Round-trip-time (RTT, in s)
  - Window size should be proportional to BD

- Problem: TCP cannot handle “Long Fat Pipes” efficiently
  - Small window size (16 bits → maximum of 64 KB)
  - Small sequence number space
  - (Go-back-N ARQ protocol)

Exercise



- Stop-and-Wait discipline – 12 KB packet, 500 Byte ACK
- Link parameters: Speed of light
- 11 Mbps WiFi
- 10 m
- Last bit – 10/c ~ 0.03 µs each
- Transmission ~ 1/11 =0.09 µs/bit
- Busy: 8640 µs (12KB * 8* 0.09)
- Idle: 0.03 + 360 + 0.03
  = 360.06 µs

- 380 Mbps terrestrial microwave
- 20 km
- Last bit – 20000/c ~ 0.07 ms each
- Transmission ~ 2.6 ns/bit
- Busy: 0.2 ms
- Idle: 0.07 + 0.01 + 0.07
  = 0.15 ms

- 1 Gbps TCP endpoint
- Coast-to-coast fiber, forwarding delays
- *Lager bandwidth, longer distance, more idle time*

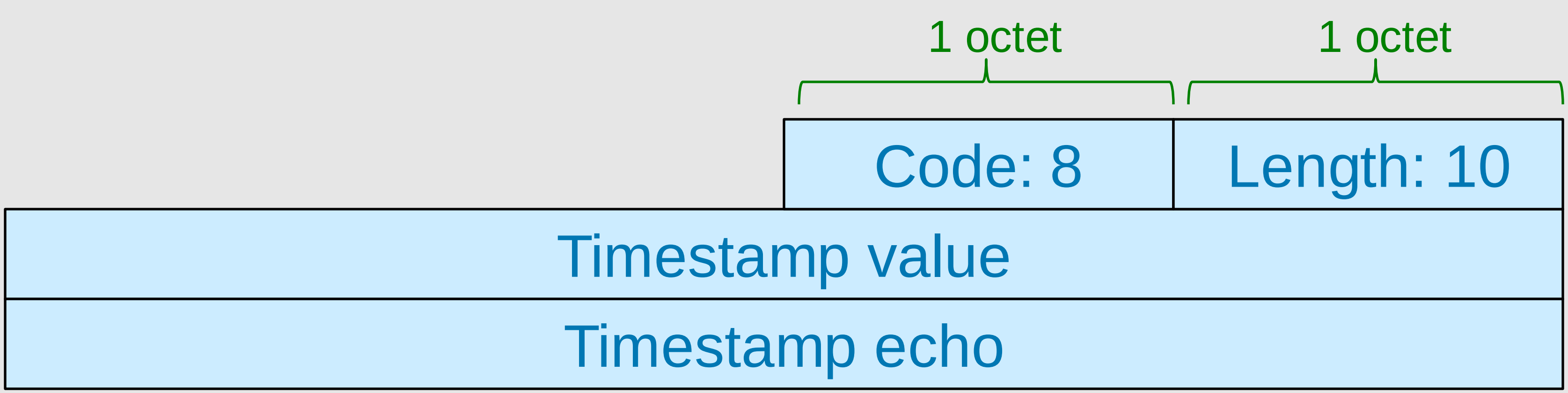
Window Scale Option

1 octet	1 octet	1 octet
Code: 3	Length: 3	Scale Factor

- “Window size” in header only 2 octets ➔ 64K is largest window that can be specified
  - Kernel may be able to allocate much more space for socket – cannot utilize with this window size
- New window size = window size in header times $2^{\text{scale-factor}}$
  - Equivalent to “shift window size in header left by a number of bits equal to the window scale factor”
  - Only carried in segments with SYN flag = 1 (permanent for the connection)

Timestamp Option

- Sender puts timestamp option in segment, reflected by receiver in the acknowledgment
  - Can compute RTT for each received ACK
  - allows sender to compute RTT more accurately
- Without timestamps, RTT calculated once every window
  - OK for 8-segment windows
  - but larger windows require better RTT calculations



Review: ARQ in TCP vs. DLC Layer

Review: ARQ in TCP vs. DLC Layer

Worth **1 participation point**and **0 correctness points**

Multiple answers: Multiple answers are accepted for this question

What are the major difference between ARQ in TCP and ARQ in the DLC layer?

A	TCP models a byte stream while DLC protocols model individual frames
B	TCP uses a different error detection method than Ethernet
C	TCP does not use sequence numbers
D	TCP does not need acknowledgements
E	TCP does not need retransmissions
F	TCP receives reliable service from the networking layer
G	TCP delivers reliable end-to-end delivery to the upper layer

Transmission Control Protocol (TCP)

Connection Establishment and Termination

Connection Establishment Issues

- Naive approach:
  - Connection Request message
  - Connection Accepted message
  - Sequence numbers always start at 0
- Problem: delayed duplicates
  - Packet is replicated at each hop of network
  - Delivering data to the wrong application
  - L2 software of hop may attempt retransmissions
  - Other software operation may cause duplicates to be sent after the first copy is successfully sent
- Connection request itself may be duplicated

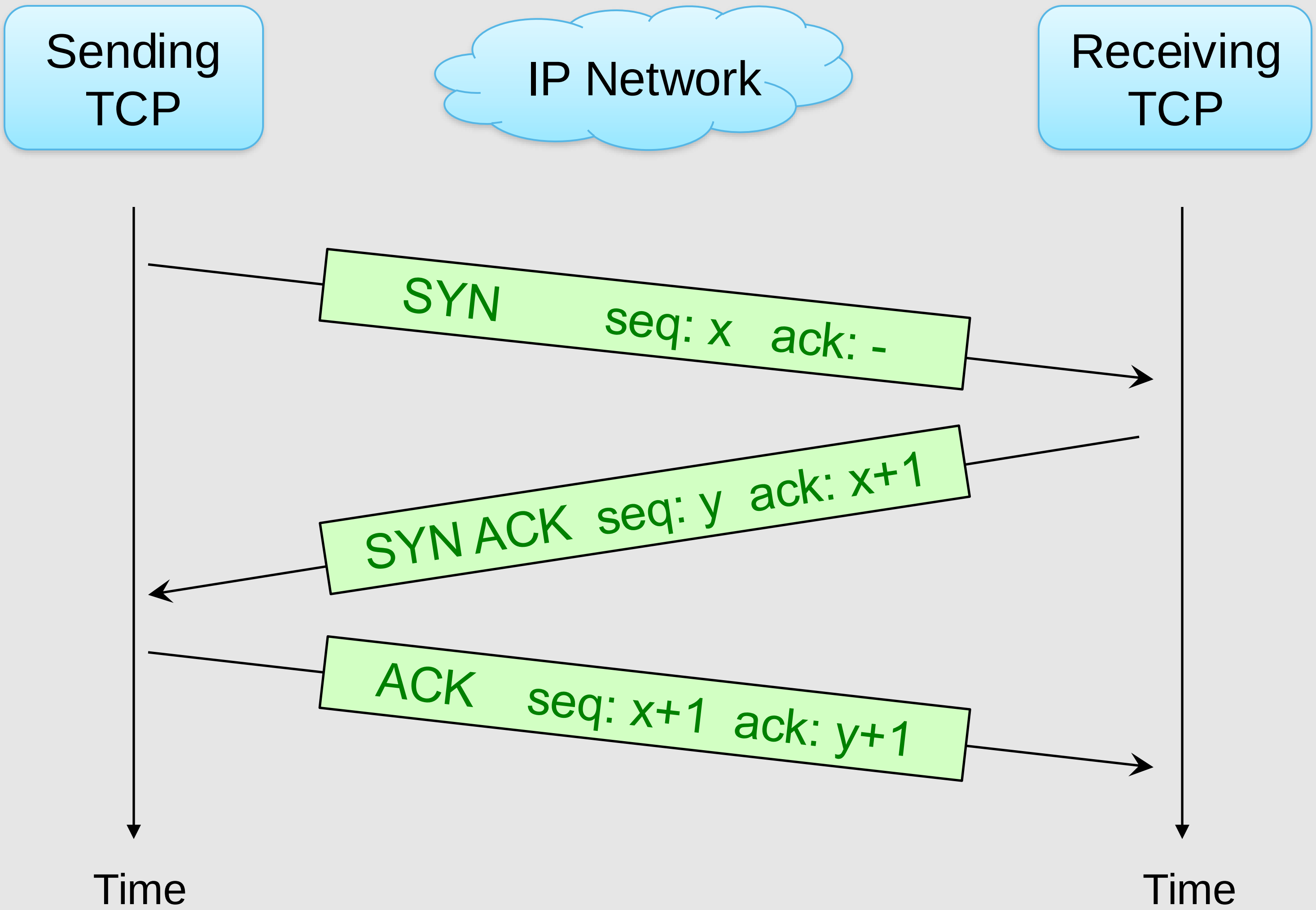
Connection Establishment Solution

- Limit packet lifetime
  - T = Max Segment Lifetime
- Use long sequence numbers
  - Wrap around time > 2T
  - = time for a packet and its ACKs to "die"
- Choose a different initial sequence number (*ISN*) with each connection request
- Ignore additional requests for connection after establishment
- After host crash: wait for time 2T
  - Allow time for old packets to die off

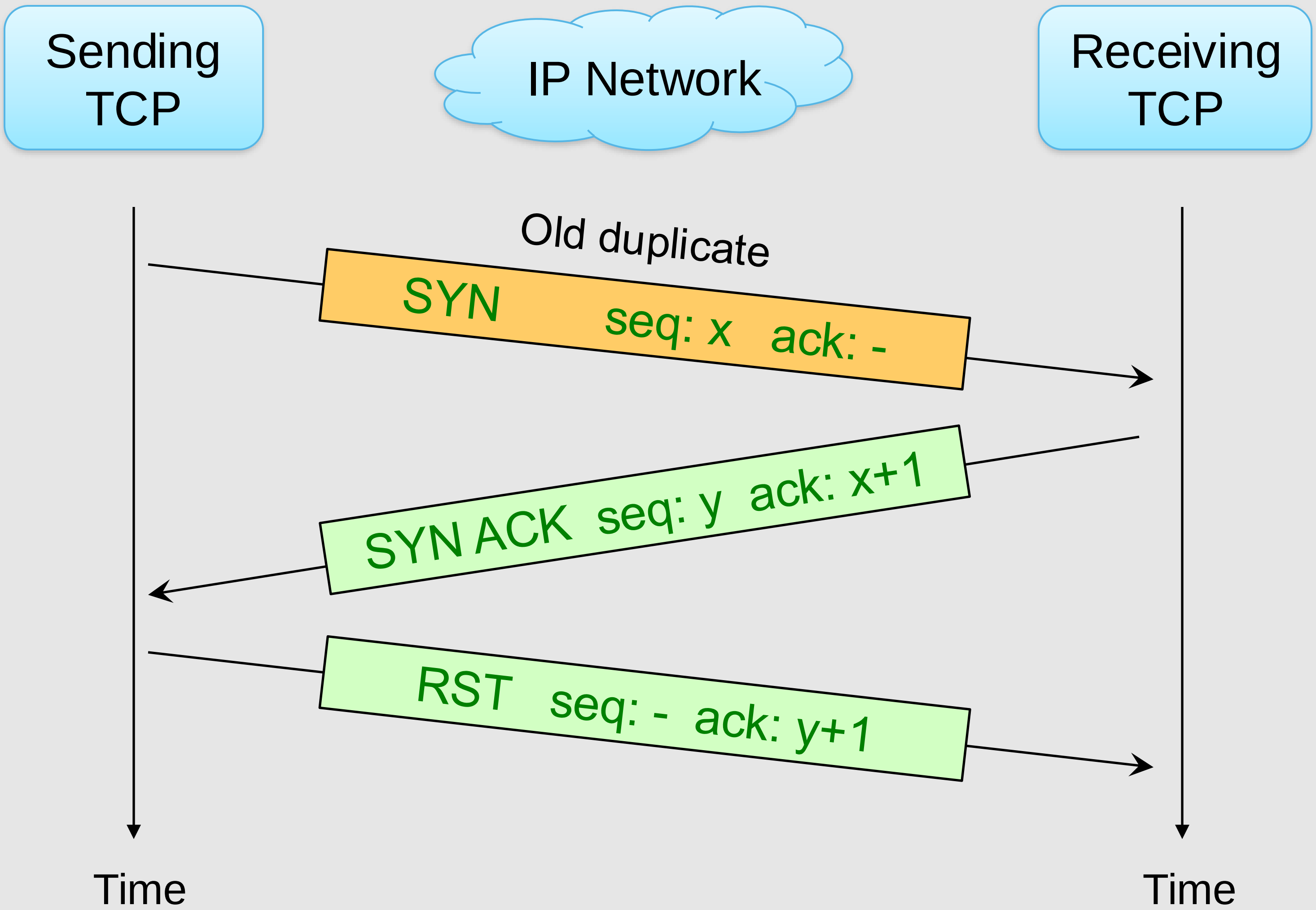
Description of Flags in Control Field

Flag	Description
URG	The value of the urgent pointer is valid
ACK	The value of the acknowledgment number is valid
PSH	Push the data (pass data to receiver as quickly as possible, later)
RST	The connection must be reset (e.g., blocked)
SYN	Synchronize the sequence numbers during connection establishment (start a new connection)
FIN	The sender has no more data to transmit (end)

Three-Way Handshake

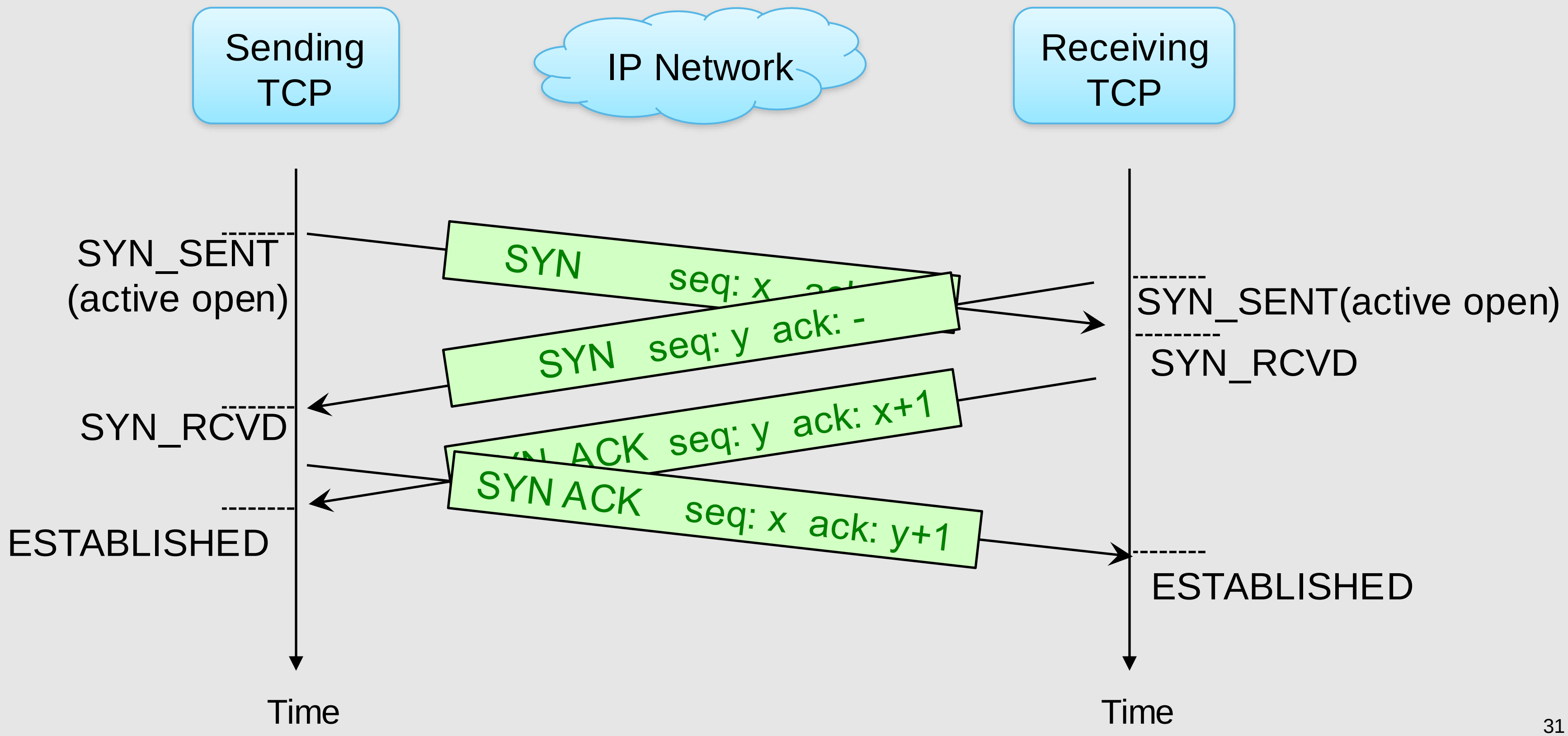


Three-Way Handshake



Simultaneous Open

- A connects with B, B connects with A at same time (pass each other in the network)
  - Only one connection will be established!
  - Using only 2 ports (one on A, one on B)



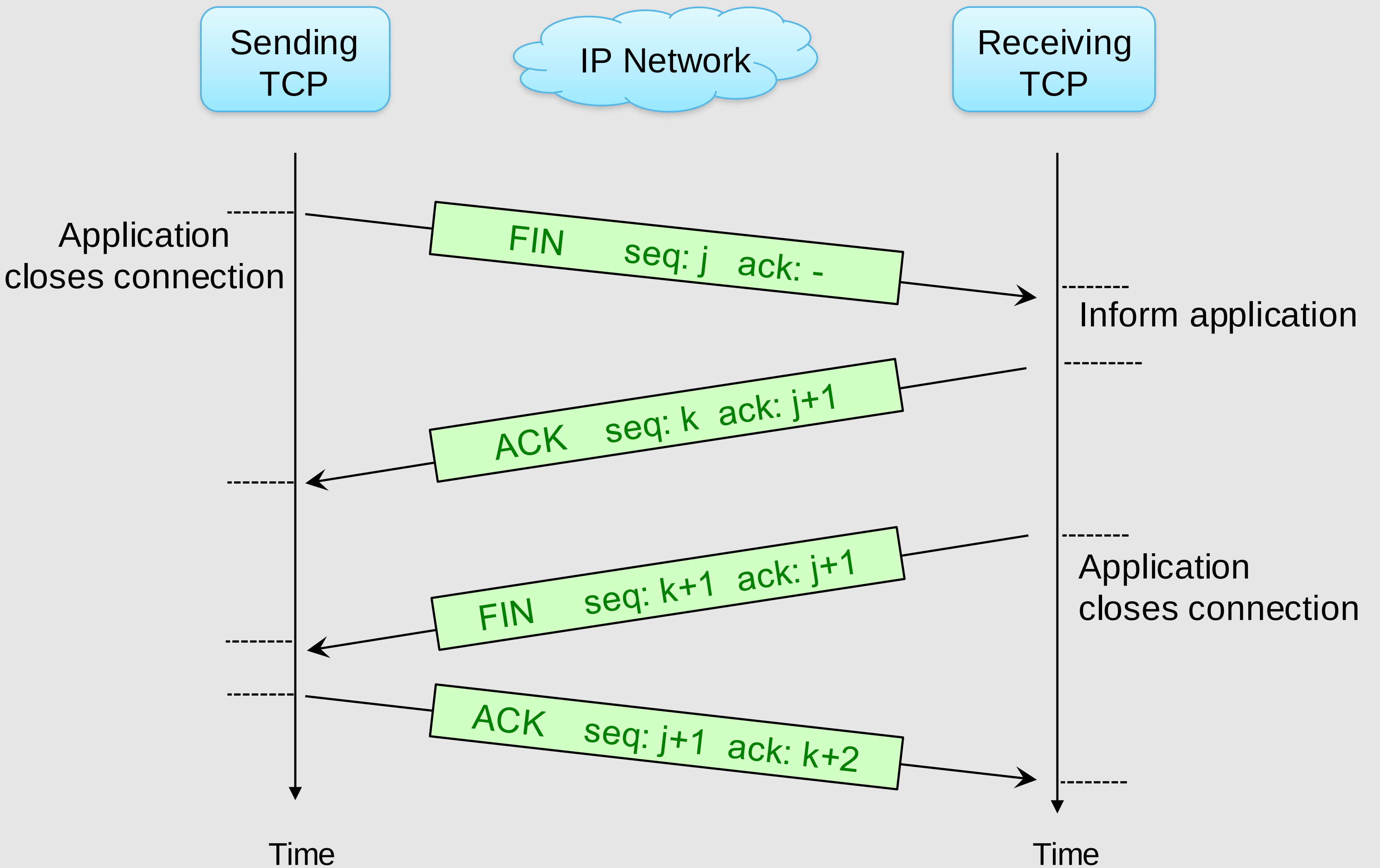
Connection Termination Issues

- Asymmetric (unilateral) release
  - abrupt, data may be lost
- Symmetric (bilateral) release
  - each direction released independently of the other
  - each side can close its transmission
    - "half closed" state is possible
- To avoid data loss in a symmetric release…
  - no side should disconnect until it is convinced that the other side is also prepared to disconnect

TCP Connection Termination

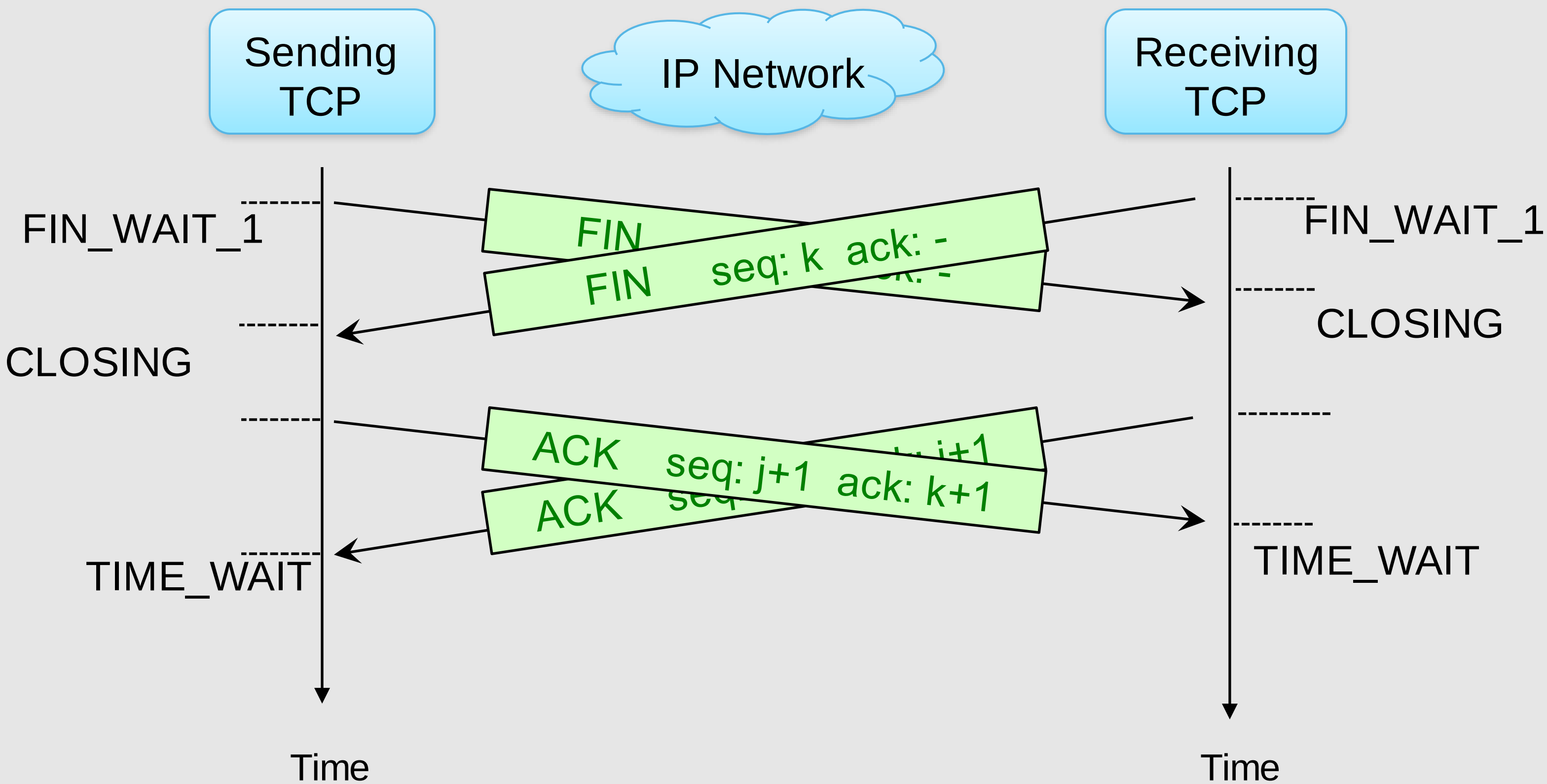
1. A sender closes its part of the connection by sending a FIN segment
2. After ACKing the FIN, the receiver can still send data on its part of the connection (half-close)
3. Finally, the receiver closes its part of the connection by sending a FIN segment (graceful close)

TCP Connection Termination



Simultaneous Close

- A sends FIN to B, B sends FIN to A at same time (pass each other in the network)



Connection Reset

- A connection can also be aborted with a RST segment (hard reset)
  - normally reserved for error conditions, not normal termination

Half Closed Connection

- One end of connection (e.g. client→server) terminates (sends FIN and receives ACK of FIN)
- Other end (server→client) remains open (sending data)
- Other end (server) later terminates (sends FIN and receives ACK of FIN), and connection is then completely closed

TCP Connection Termination