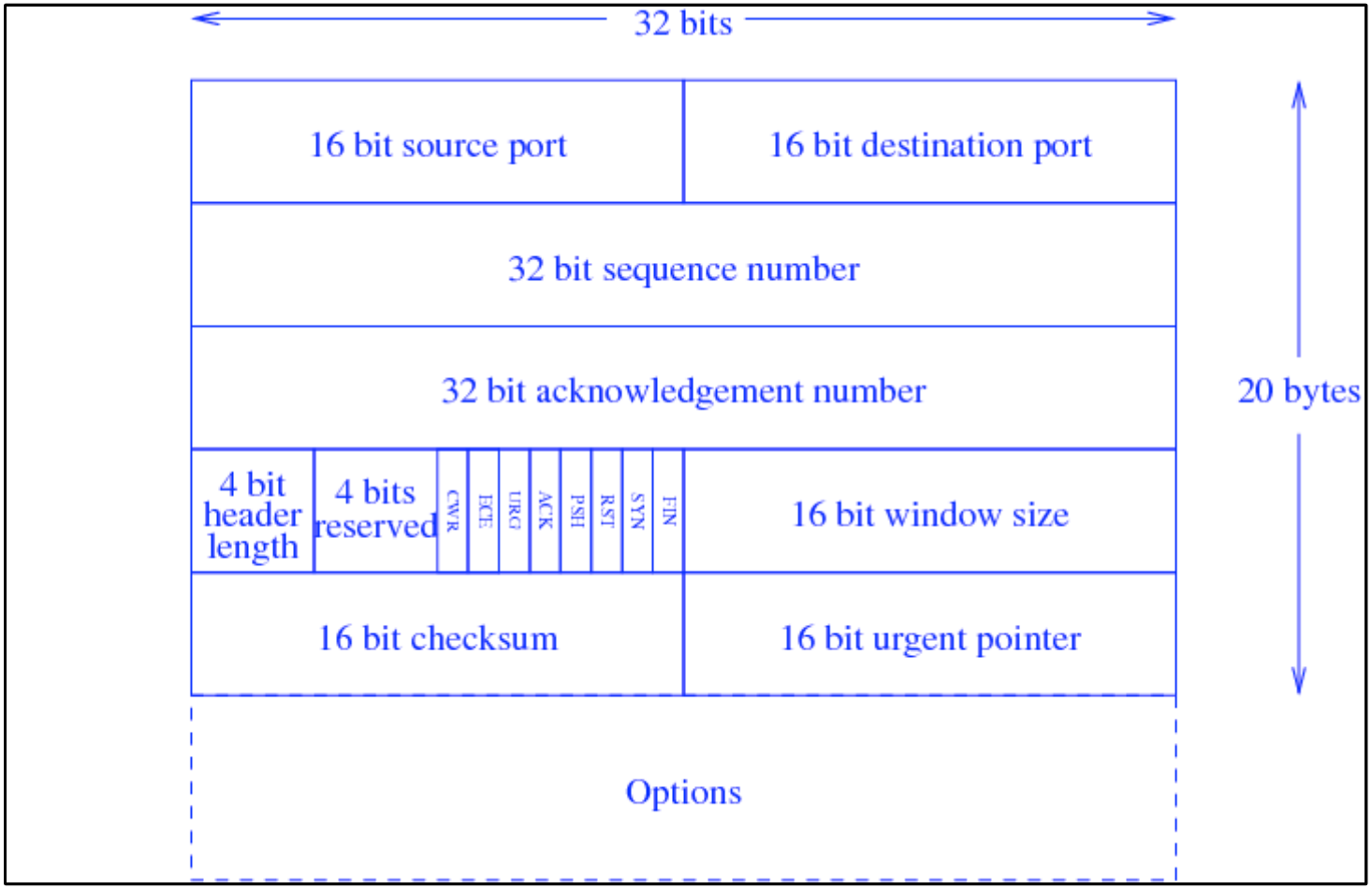
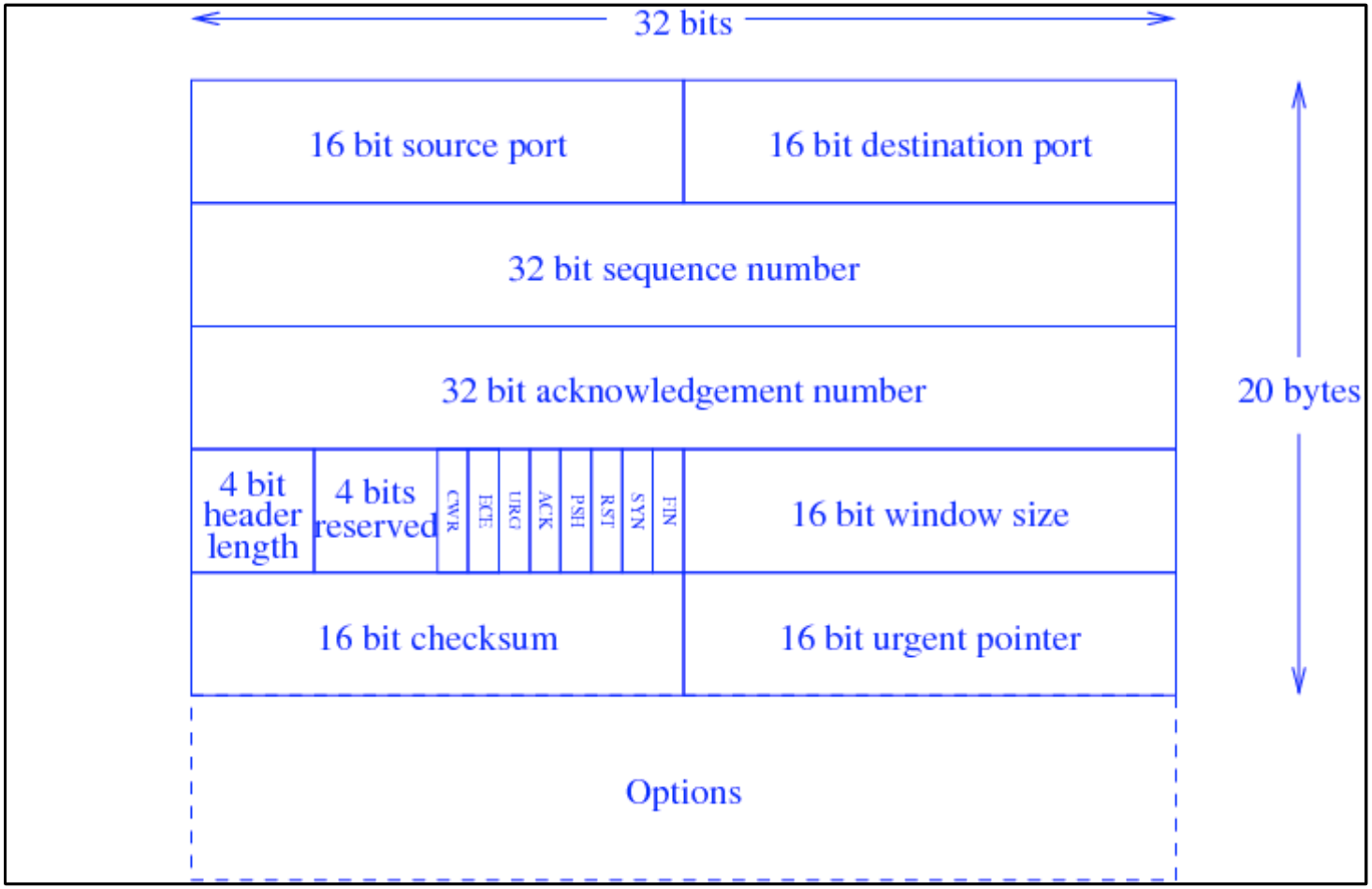
The Transport Layer



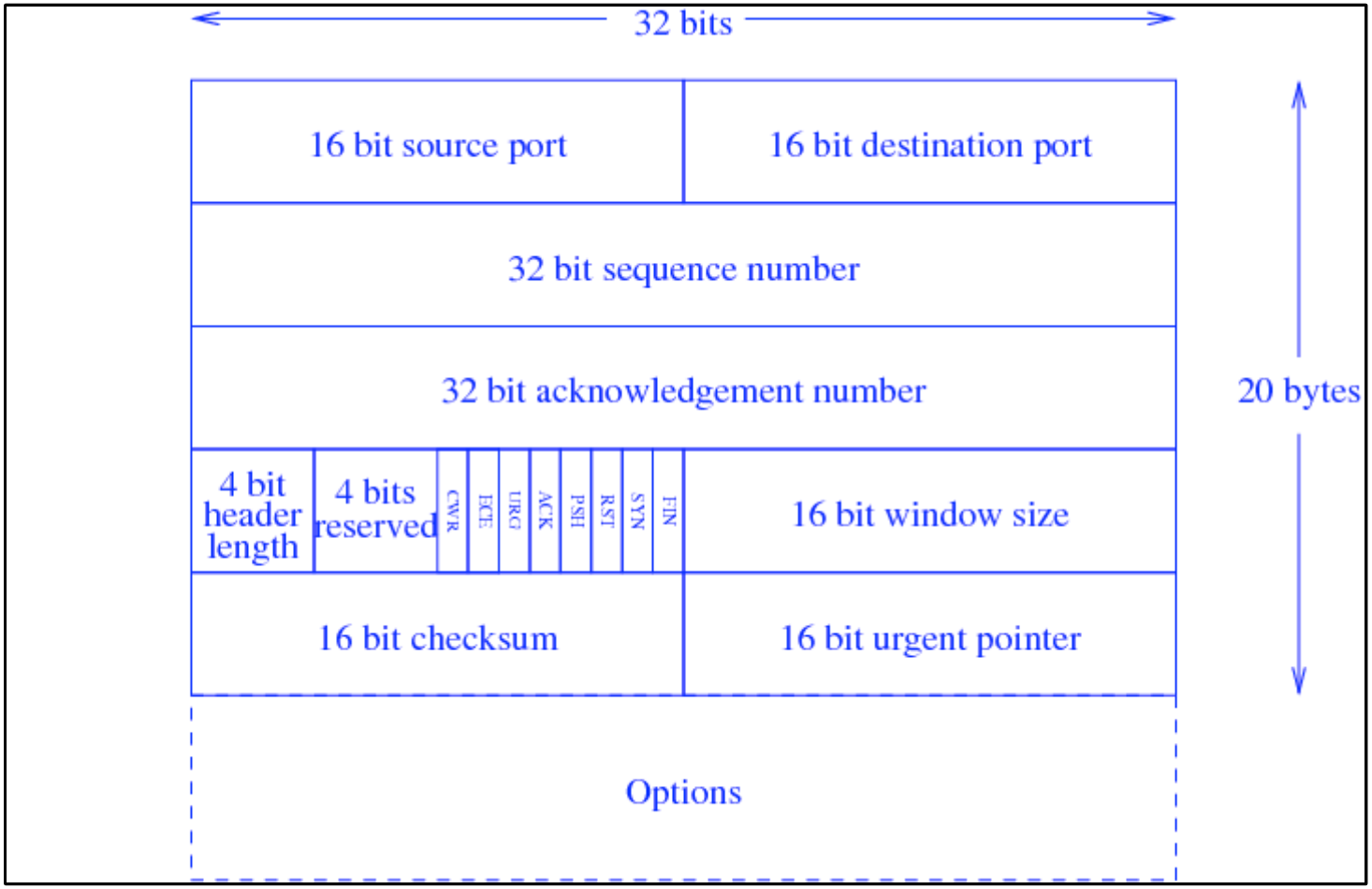
- 4 bits header length in 32 bit words: the header can have options, so is of variable length

The Transport Layer



- So maximum is 60 bytes. Minimum is 20 bytes

The Transport Layer



- Many flags performing various functions

The Transport Layer

TCP Flags

These will be described in more detail later

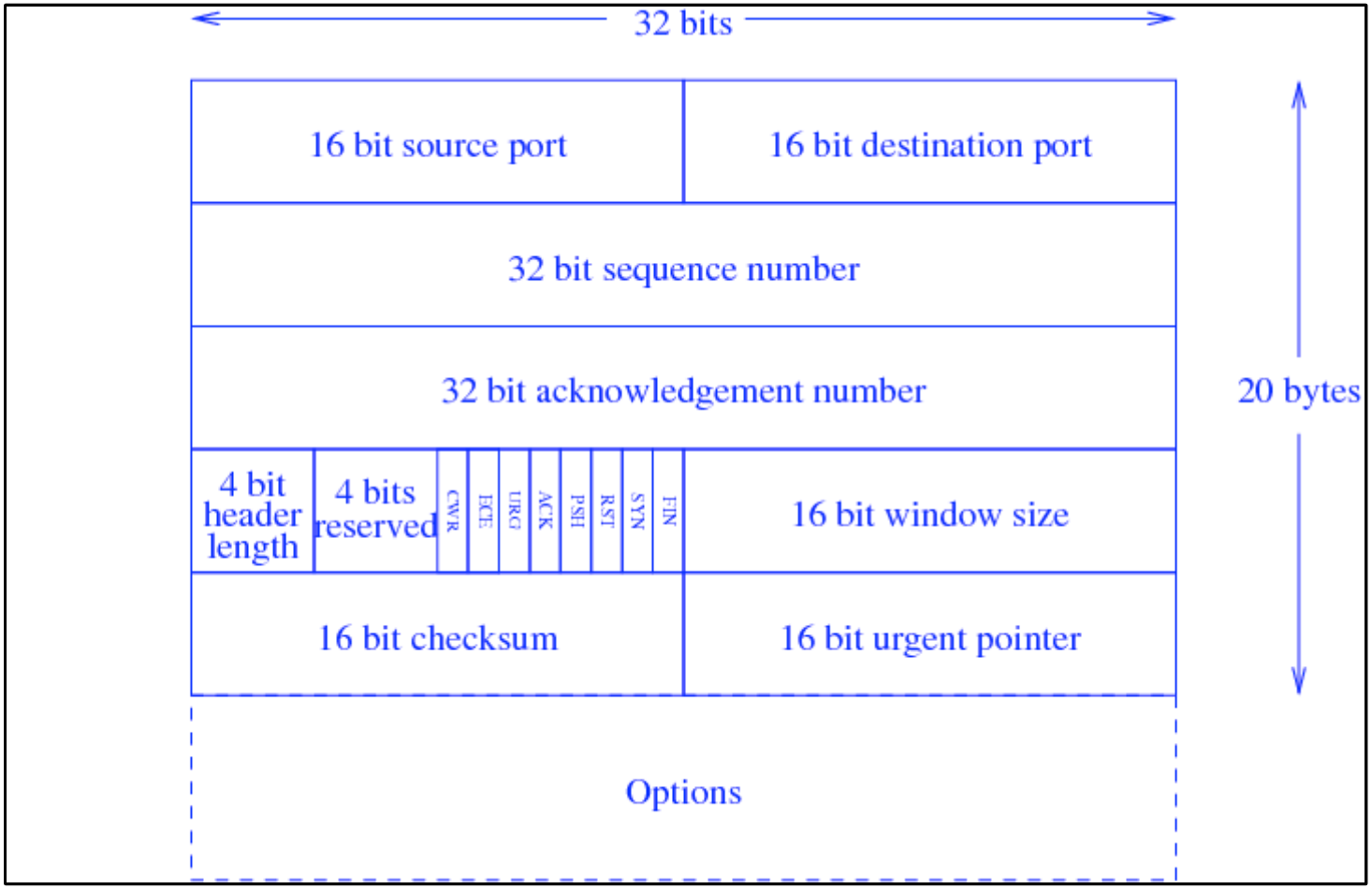
- URG: urgent data
- ACK: the acknowledgement field is active
- PSH: push this data to the application as fast as possible
- RST: reset (break) the connection

The Transport Layer

TCP Flags

- SYN: synchronise a new connection
- FIN: finish a connection
- ECE: congestion notification
- CWR: congestion window reduced
- 4 reserved bits, set to 0

The Transport Layer



- 16 bits of *advertised window size*: for flow control

The Transport Layer

TCP flow control

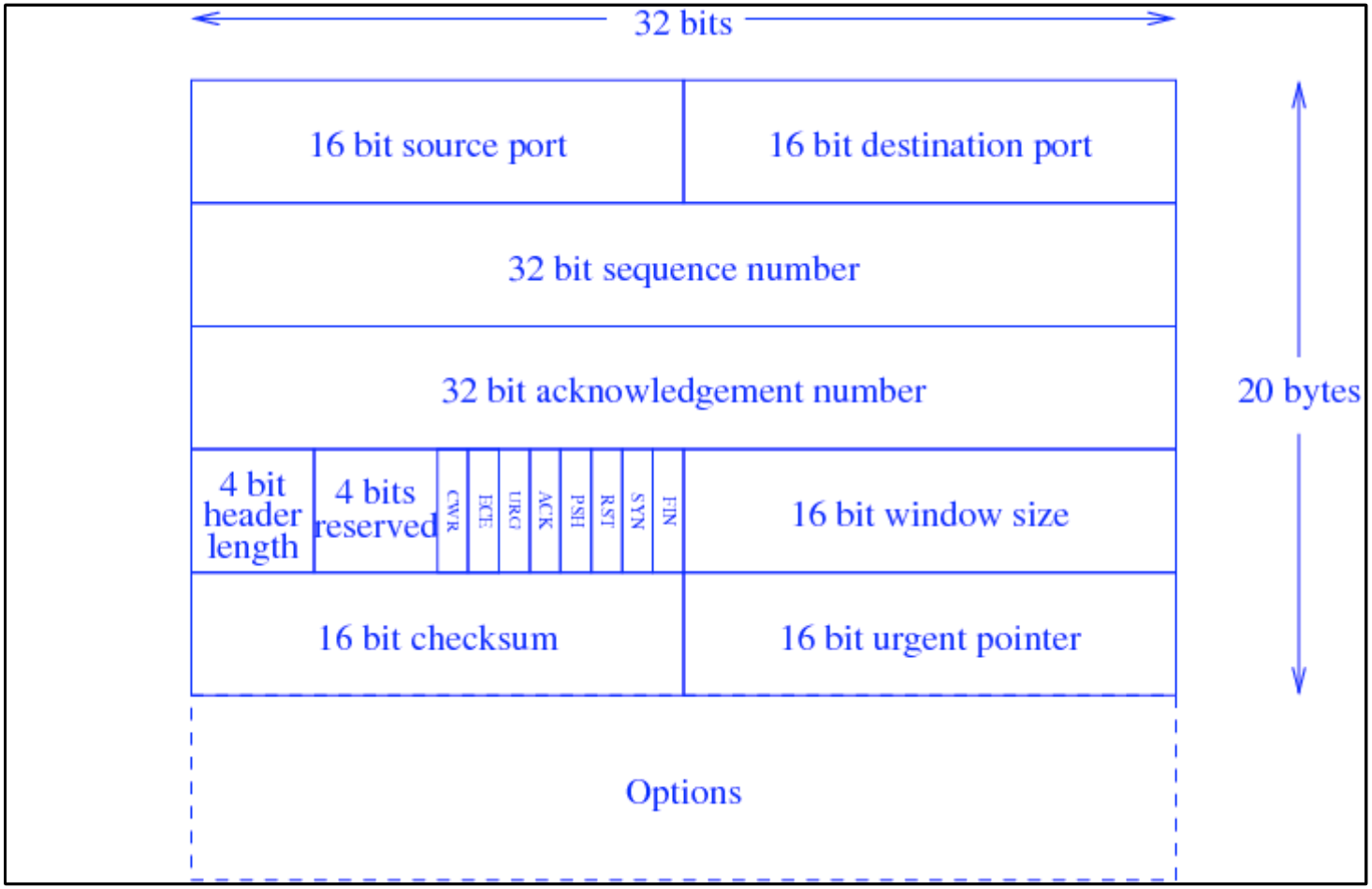
- The destination has only a limited amount of buffer memory it can store new segments in
- If the application is not reading the data as fast as it arrives, the buffer will fill up
- The window size is the amount of buffer left: it sends this in the next segment to the client
- If the amount is very small, the client can slow down sending until space is freed up

The Transport Layer

TCP flow control

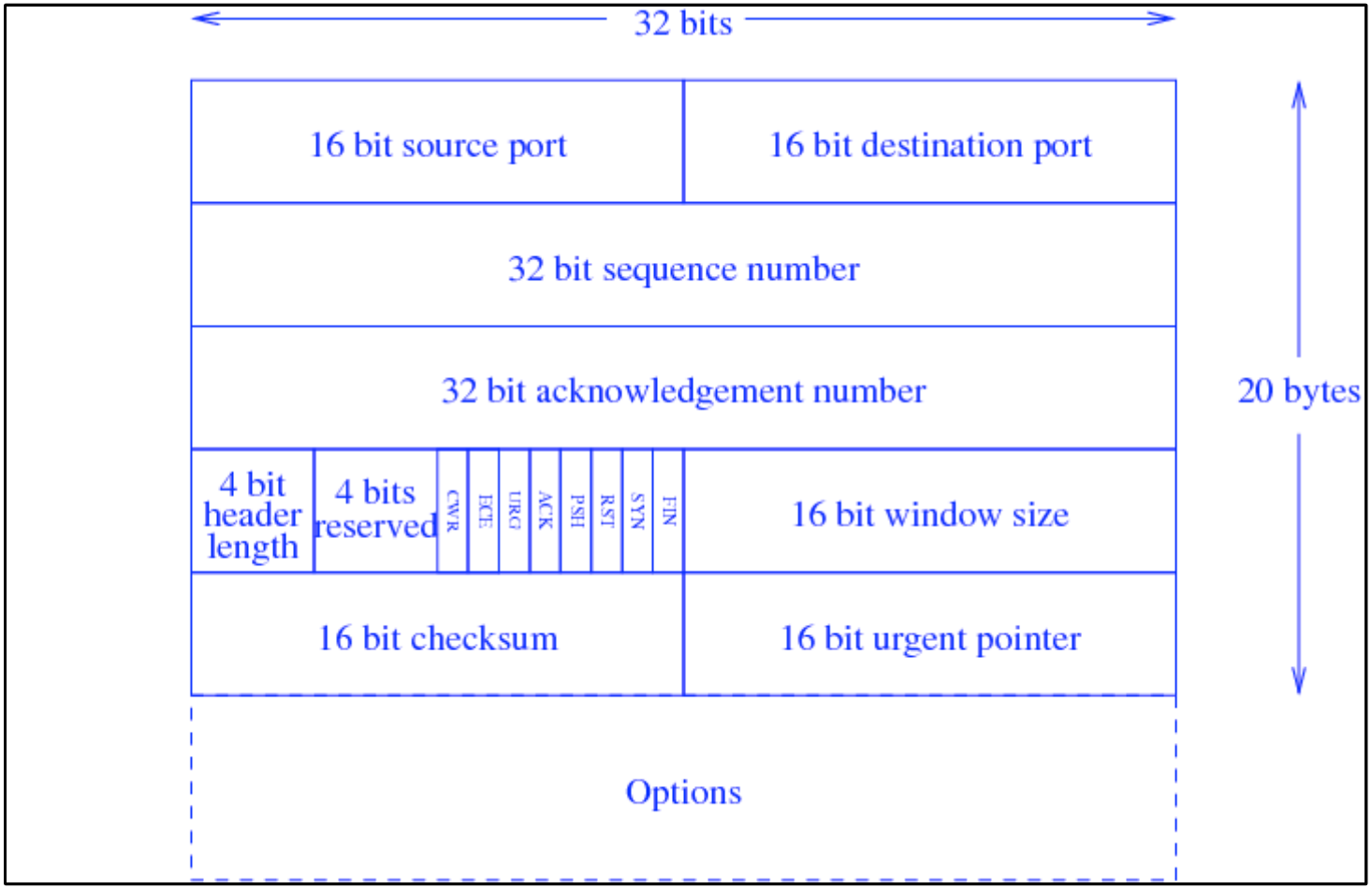
- 16 bits gives a maximum buffer of 65535 bytes: much too small
- There is a header option to scale this up to something reasonable

The Transport Layer



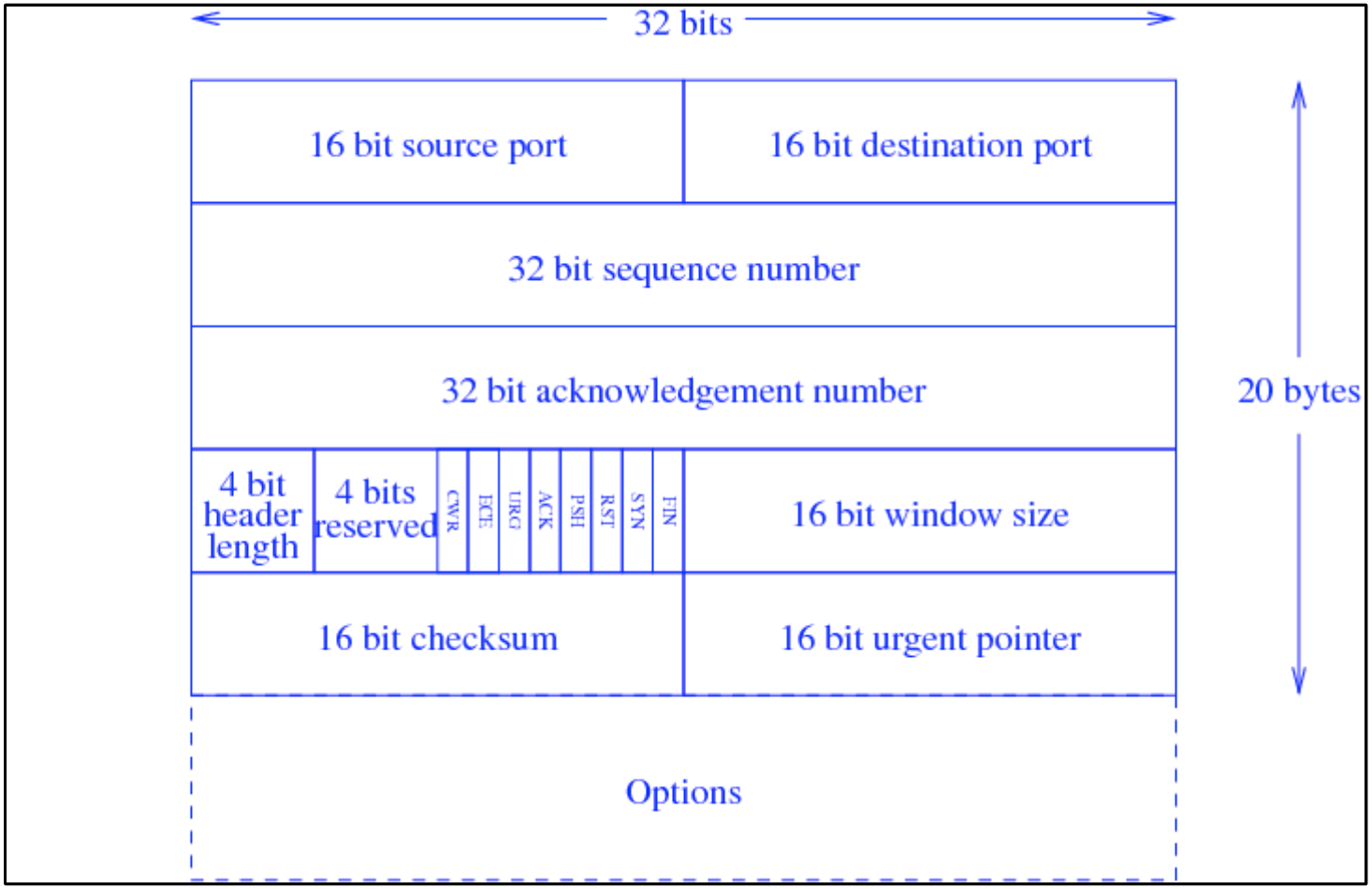
- Checksum of the header, the data, plus some fields of the IP layer

The Transport Layer



- Just as UDP, this breaking of layering is poor design

The Transport Layer



- Urgent pointer: active if the URG flag is set

The Transport Layer

TCP URG

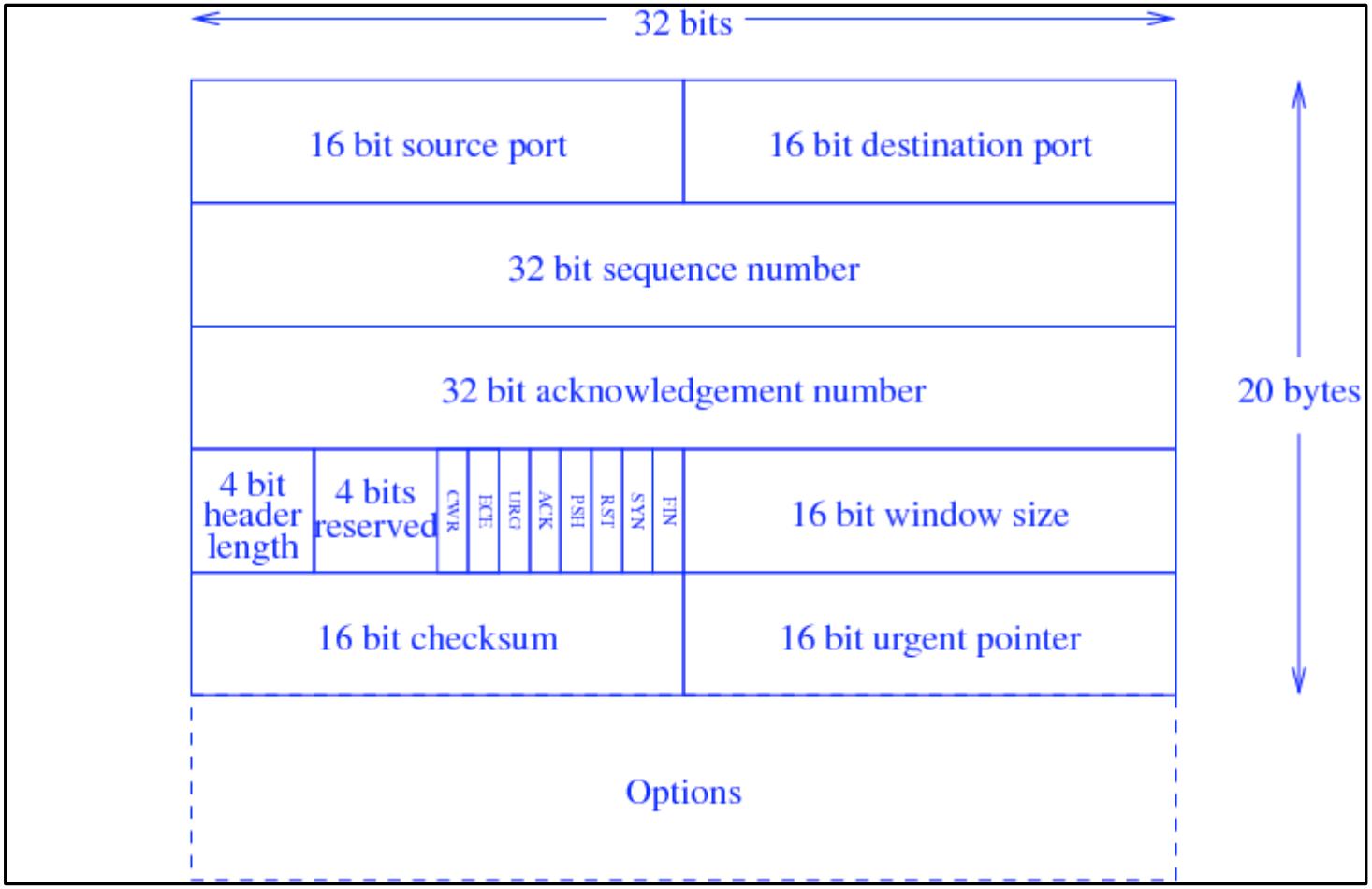
- The urgent pointer is a pointer into the data stream that indicates where the current *urgent data block* ends
- Urgent data includes things like interrupts that need to be processed before any other data that is buffered

The Transport Layer

TCP URG

- The original intent of URG was to mean “somewhere in the data up to this point is something important”
- It has ended up meaning “read and act quickly on the data up to this point”

The Transport Layer



- Options: many and varied, including *window scale* and *maximum segment size*

The Transport Layer

TCP

- After the header is the data: this can be empty, e.g., in a pure ACK

The Transport Layer

TCP Connection Setup & Teardown

- Setting up a TCP connection is complicated, as there is a lot of state that must be set up, e.g., sequence numbers
- Similarly, closing a connection is not trivial: we must ensure all segments in flight have been ACKed properly. Perhaps segments need to be resent. Thus a connection will hang around for a little after closing to ensure everything is tidied up

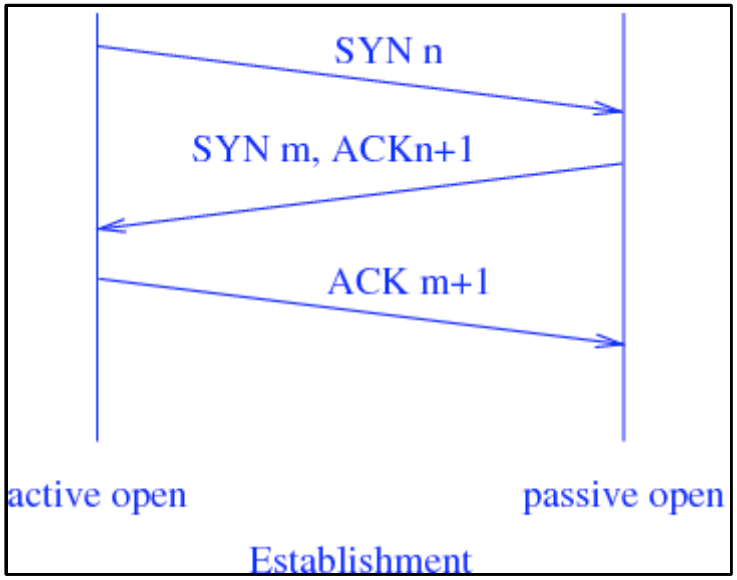
The Transport Layer

TCP Connection Setup & Teardown

- Fortunately, all this detail is taken care of by the TCP layer software in the operating system: though it does have occasional repercussions in the application

The Transport Layer

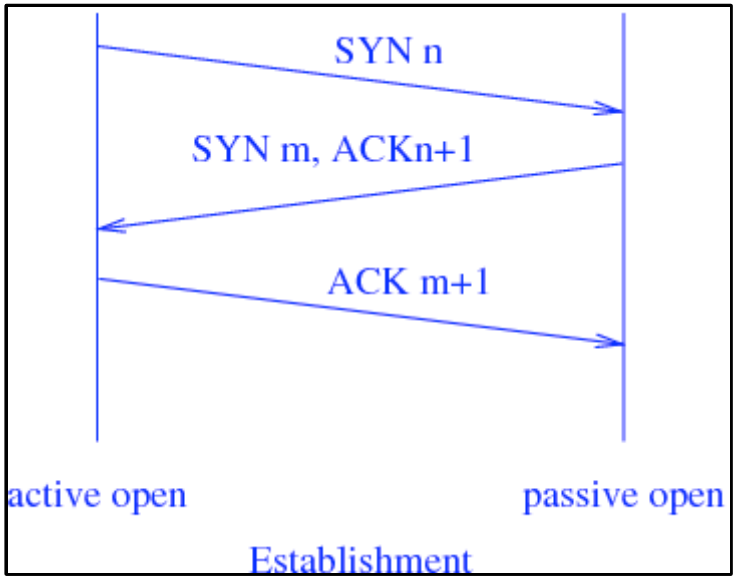
TCP Connection Setup



- Three segments are needed to make a new connection

The Transport Layer

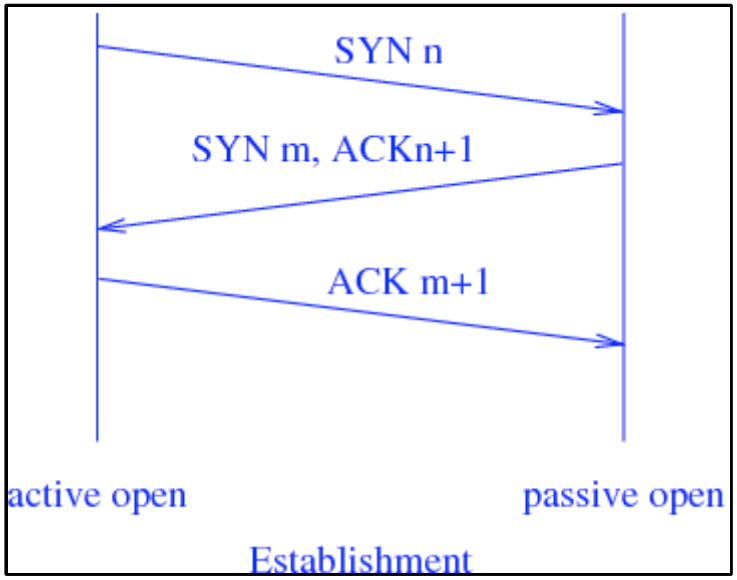
TCP Connection Setup



- The initiator, the *client*, sends a segment with the SYN flag set and an *initial sequence number* (ISN) n, randomly generated

The Transport Layer

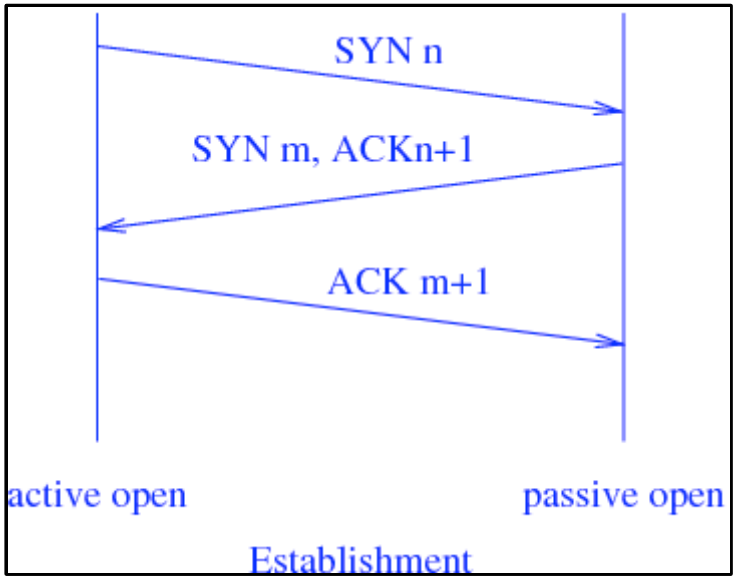
TCP Connection Setup



- The receiver, the *server*, replies with another SYN segment containing its own ISN m

The Transport Layer

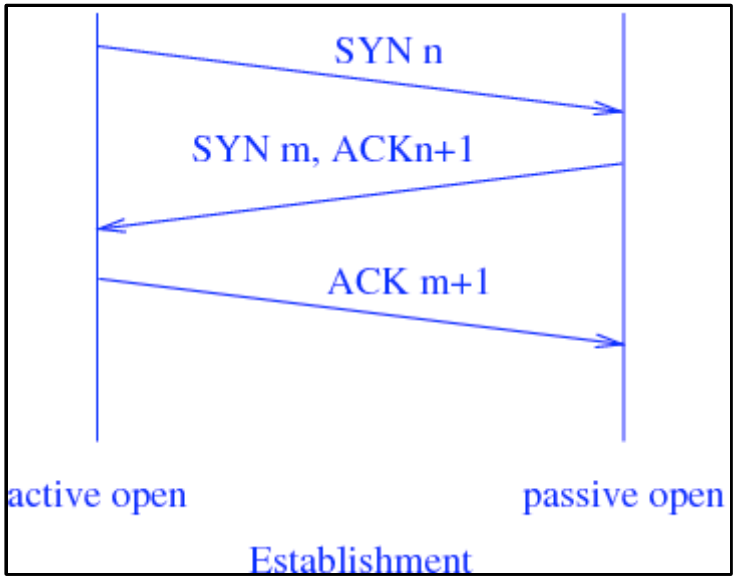
TCP Connection Setup



- It also ACKs the client's ISN with n+1

The Transport Layer

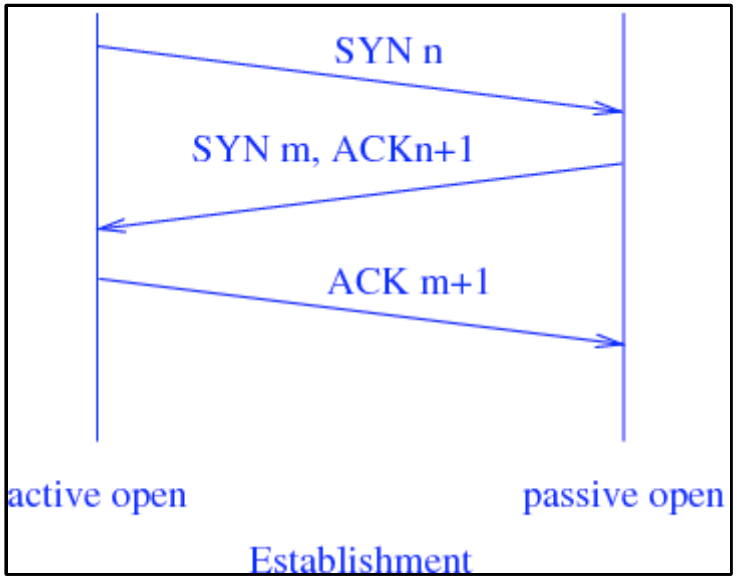
TCP Connection Setup



- The initial SYN can be lost just like any other segment, so we need to ACK it independently of the first data byte, which comes later

The Transport Layer

TCP Connection Setup



- The client ACKs the server's ISN with m+1

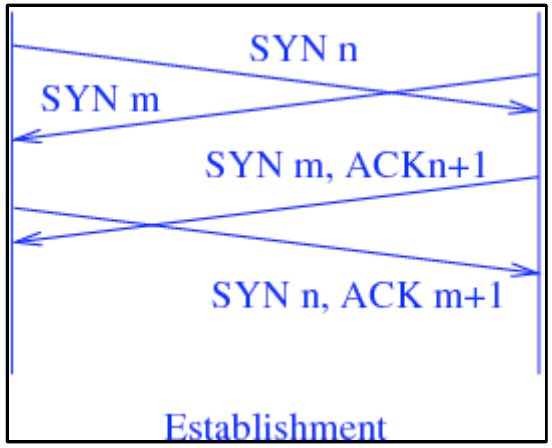
The Transport Layer

TCP Connection Setup

- This is called a *three way handshake*
- These segments contain no data: they are overhead in setting up the connection
- After the handshake we can start sending data
- The client is said to do an *active open*, while the server does a *passive open*

The Transport Layer

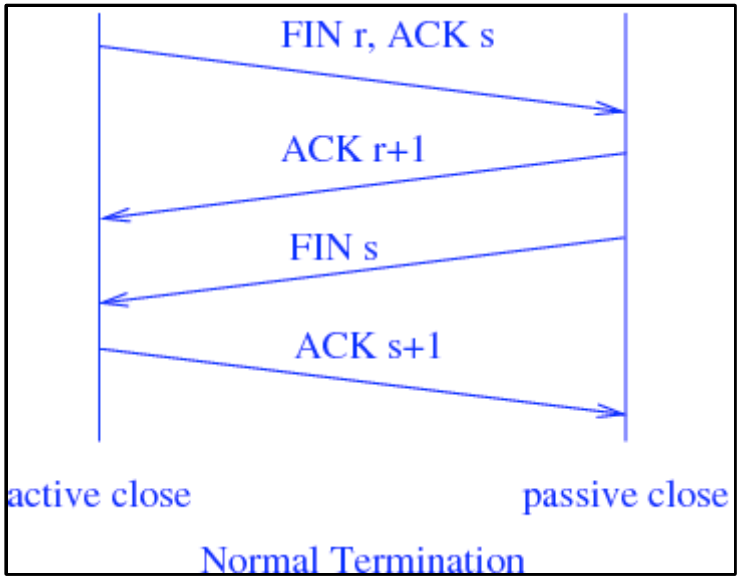
TCP Connection Setup



- It is possible (but rare) for *both* hosts to do an active open, where the SYNs cross each other in flight
- This defined to produce *one* new connection, not two

The Transport Layer

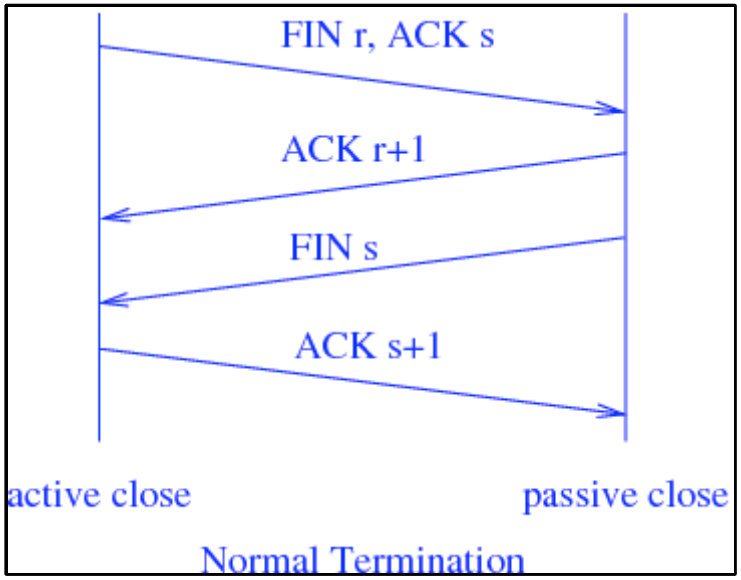
TCP Connection Teardown



- Closing a connection normally takes four segments
- TCP is full duplex, and a connection in one direction may be closed independently of the other

The Transport Layer

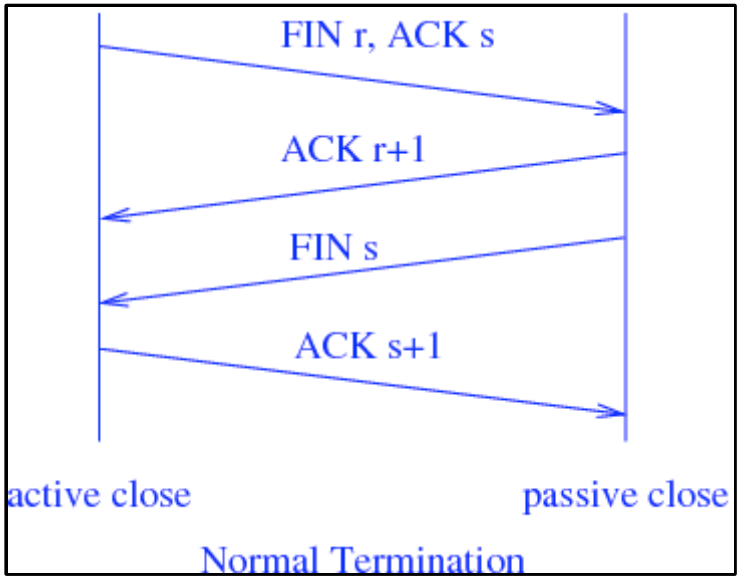
TCP Connection Teardown



- The FIN flag is set to indicate a (half)close: no more data will be sent from this end
- We can still receive data at this end

The Transport Layer

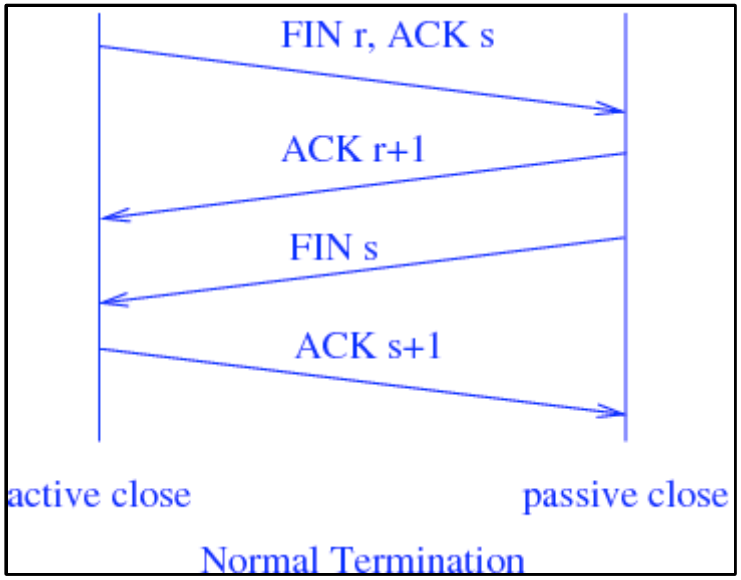
TCP Connection Teardown



- This is ACKed
- When the other end wants to close, it sends a FIN and gets an appropriate ACK

The Transport Layer

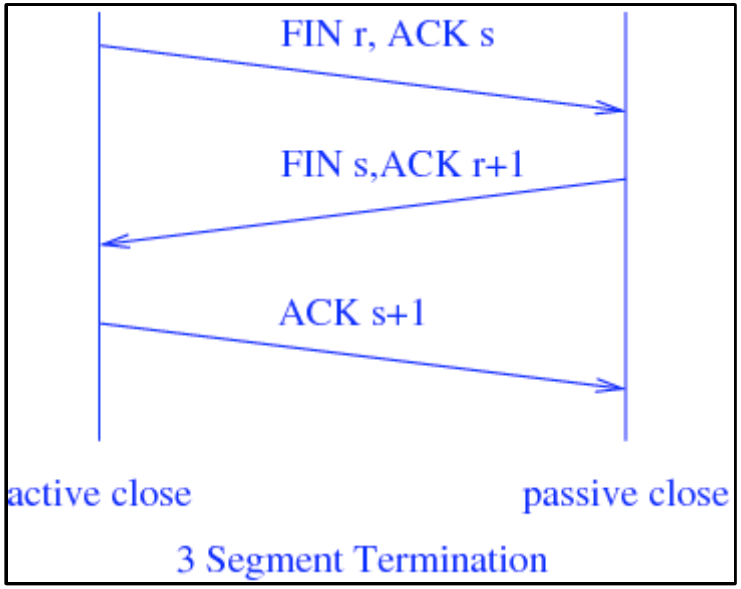
TCP Connection Teardown



- Either end can initiate in an *active close*
- The other end does a *passive close*

The Transport Layer

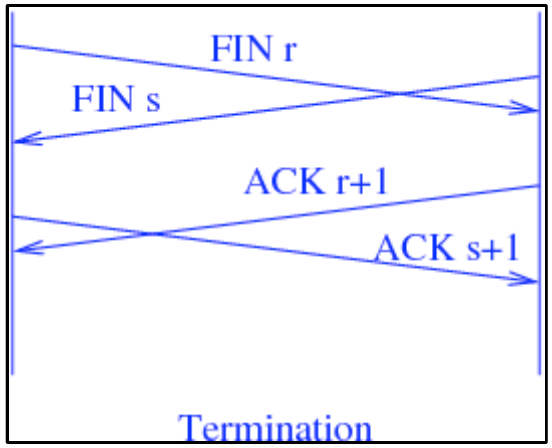
TCP Connection Teardown



- The passive FIN can be piggybacked on the ACK: this then takes only three segments

The Transport Layer

TCP Connection Teardown



- There can (rarely) be a simultaneous active close: this takes four segments again

The Transport Layer

TCP Connection Termination

- Another way to end a connection is to send a *reset* (RST) segment, i.e., with the RST flag set
- This is for error cases, e.g., a segment arrives that doesn't appear to be for a current connection, as might happen after a server crashes and reboots while the client is still sending

The Transport Layer

TCP Connection Termination

- When a host gets a RST it ends the connection immediately, discarding all state and buffered segments
- Often seen by the application as a “connection reset by peer” message
- A connection ended by FINs is called an *orderly release*; if ended by a RST it is an *abortive release*

The Transport Layer

TCP Connection Termination

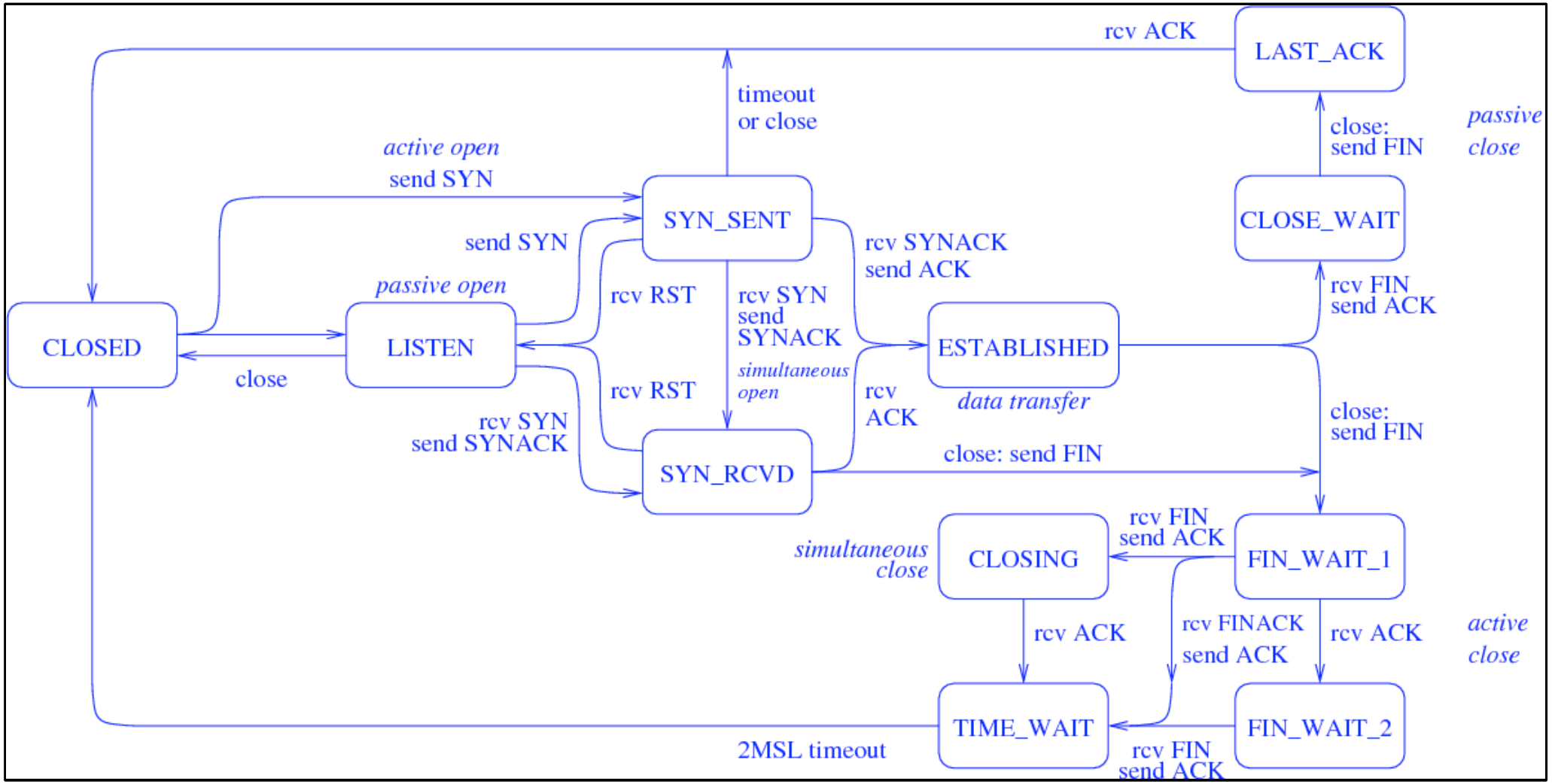
- RSTs are not ACKed: the connection ends right here

The Transport Layer

TCP State Machine

- The transitions between states in TCP are very complicated
- There is a standard TCP state diagram that describes how TCP should act in most cases
- Though it only covers non-error cases: it does not say what to do if, say, a SYNFIN segment arrives

The Transport Layer



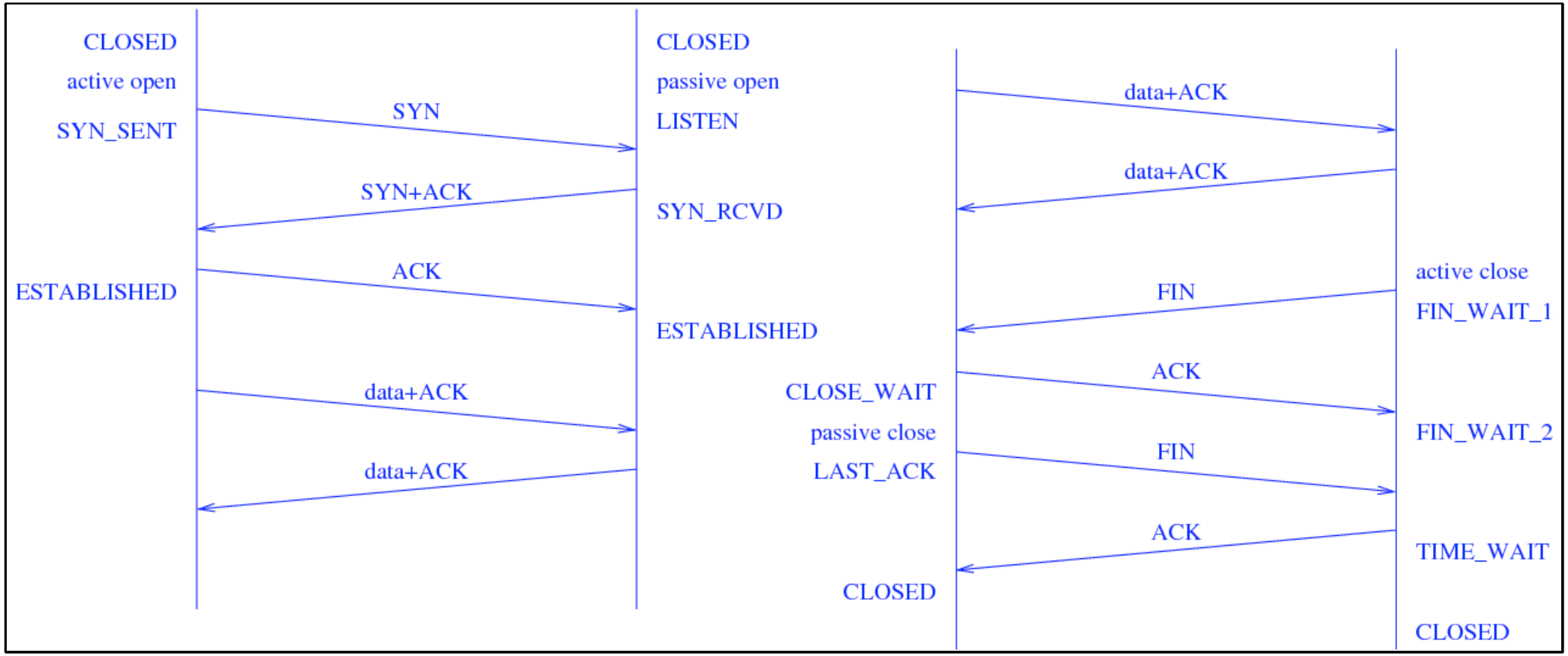
The Transport Layer

TCP State Machine

- We start (and end) in CLOSED
- Data only flows in the ESTABLISHED state
- This state diagram is followed for *each* connection, i.e., each socket pair

The Transport Layer

Typical TCP Timeline



The Transport Layer

TCP States

- The TIME_WAIT state (also called 2MSL state) appears before a final close: the connection must remain non-closed until a time period has passed
- The *maximum segment lifetime* (MSL) is a value that is the longest time a segment can live in the network before being discarded

The Transport Layer

TCP States

- A typical value chosen is 2 minutes
- A TCP connection must stay in TIME_WAIT for twice the MSL
- This is in case the final ACK was lost and gives the other end time to retransmit its FIN
- So this connection (quad) cannot be reused until 2MSL has passed

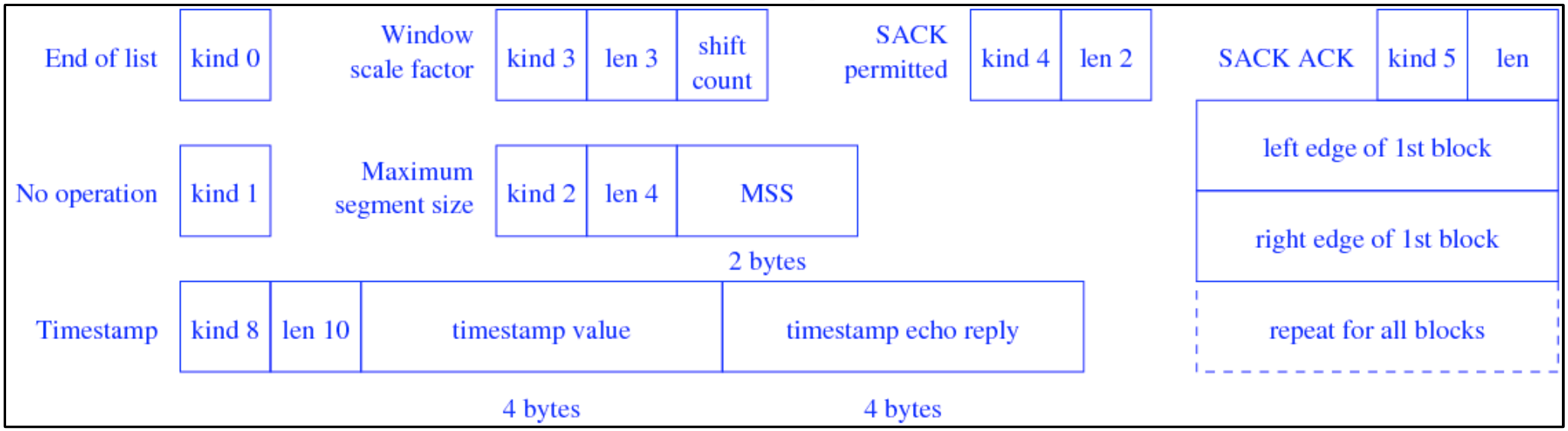
The Transport Layer

TCP States

- Not a problem is the client does an active close, as it probably used an ephemeral port and if it wants another connection is can use another ephemeral port
- If the server, using a well-known port, does the active close, it cannot reuse the well-known port for 2MSL
- Thus the server cannot restart (on the same port) for a few minutes

The Transport Layer

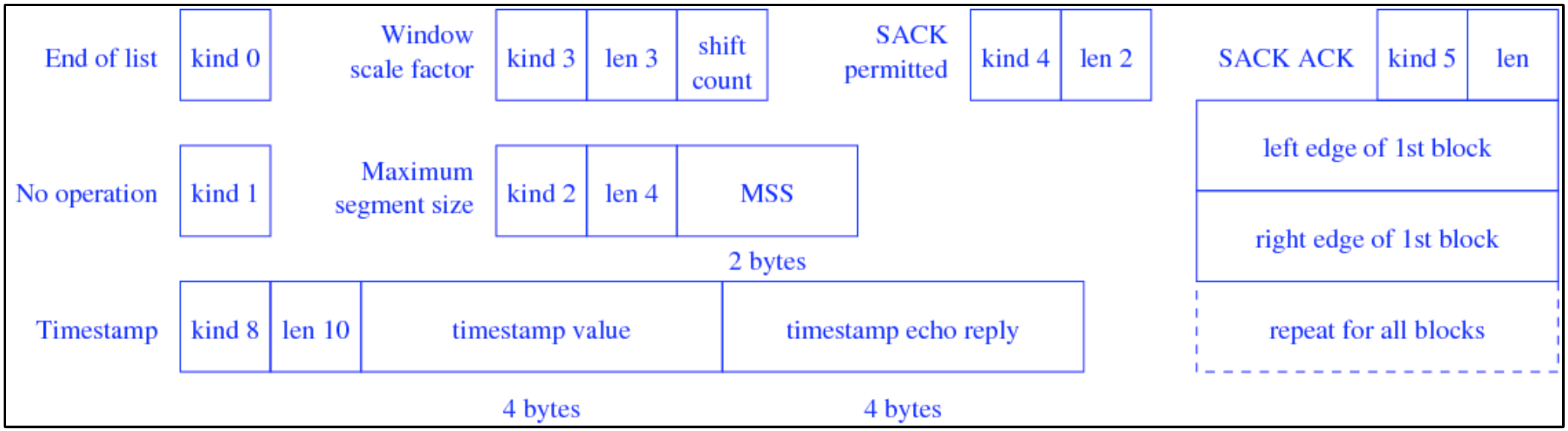
TCP Options



- Options start with a 1 byte *kind* which indicates what the option is to do
- Kinds 0 and 1 are one byte long; others have a length field

The Transport Layer

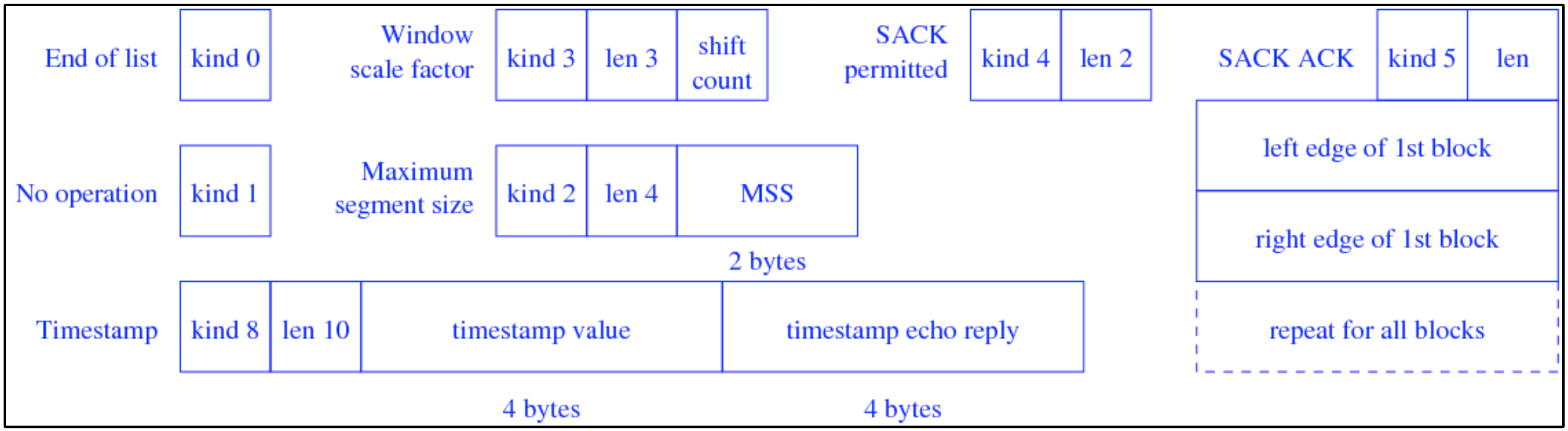
TCP Options



- *No operation* (NOP) is used to pad to align fields to a multiple of 4 bytes

The Transport Layer

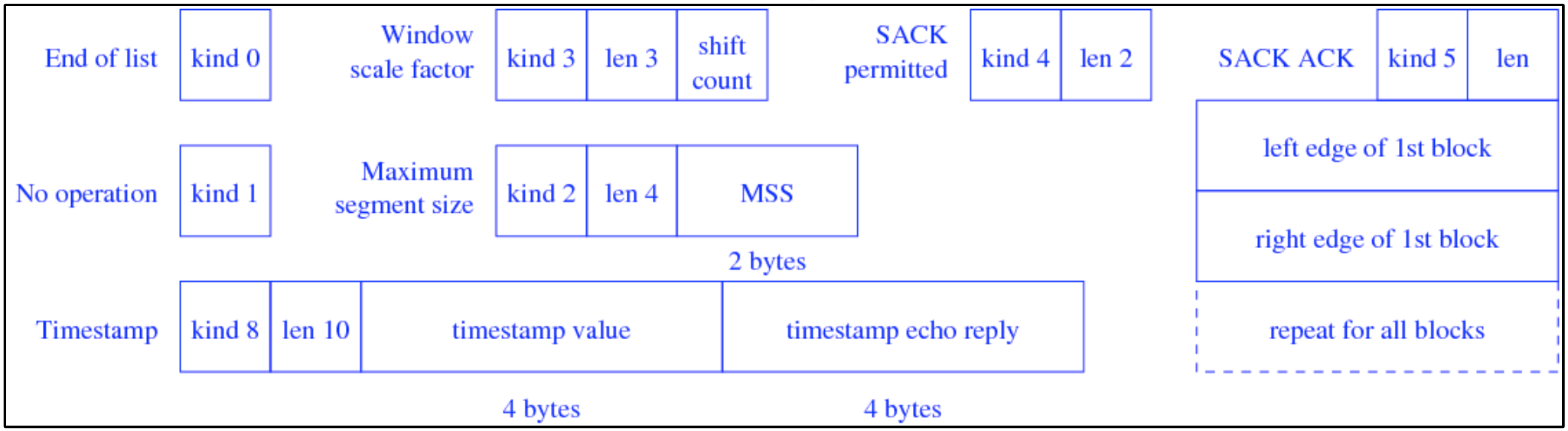
TCP Options



- *Maximum segment size* (MSS) specifies how large a segment we can cope with without fragmentation

The Transport Layer

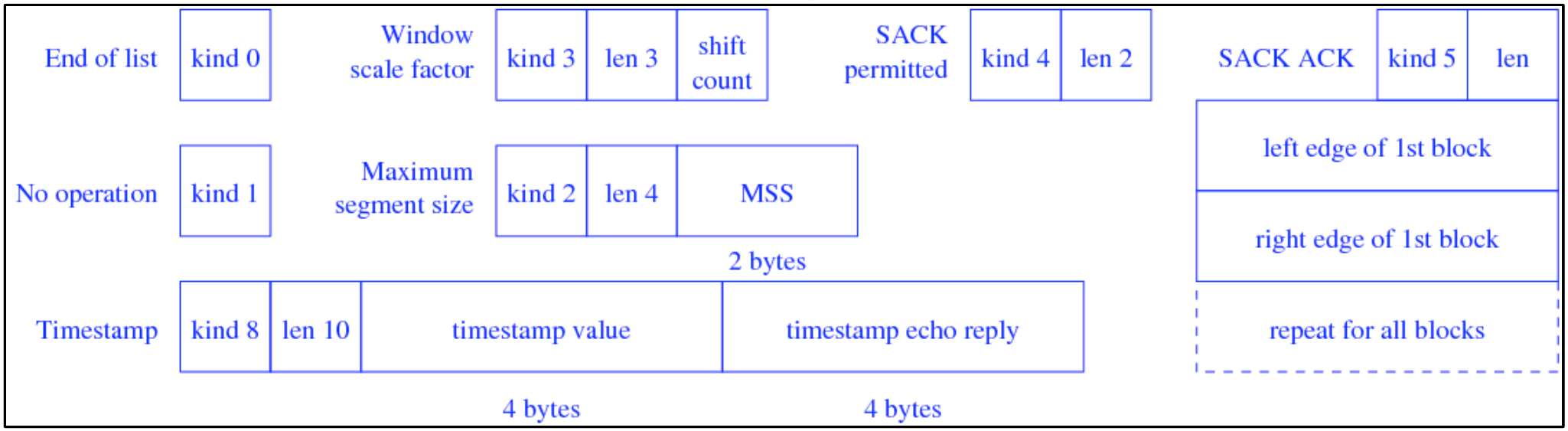
TCP Options



- *Window scale* allows us to multiply up the window size from the 65535 the standard header allows: contains value from 0 to 14. A large window is very important is modern fast networks

The Transport Layer

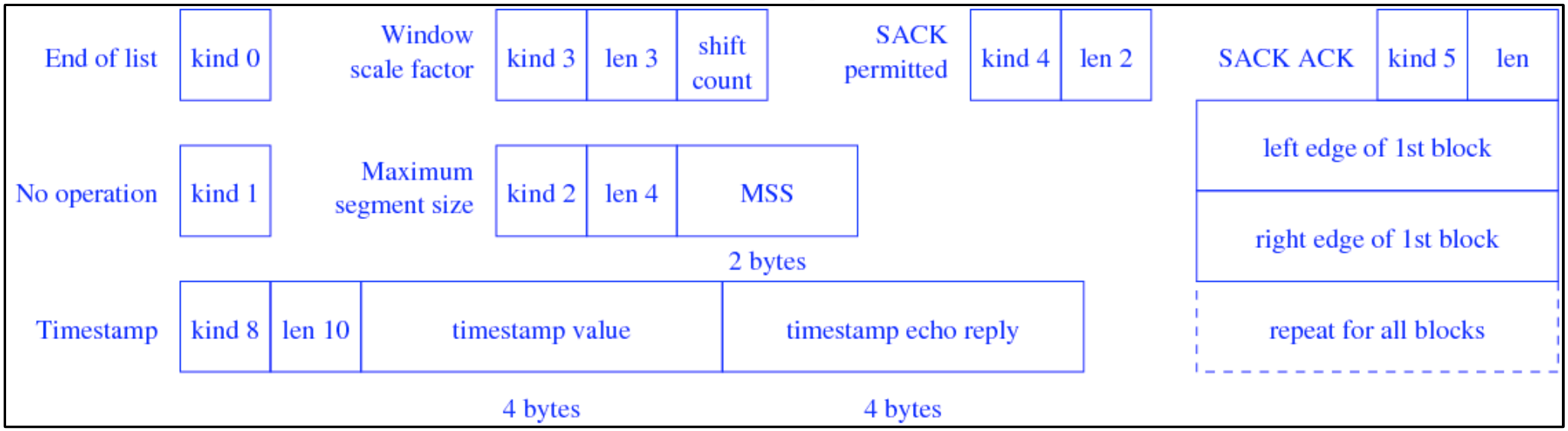
TCP Options



- A value of *n* scales by 2^n: thus a maximum window of 2^{14} x 65535 = 1,073,725,440 bytes (a gigabyte)
- But that's only about a second's worth of data in a 10Gb/s Ethernet

The Transport Layer

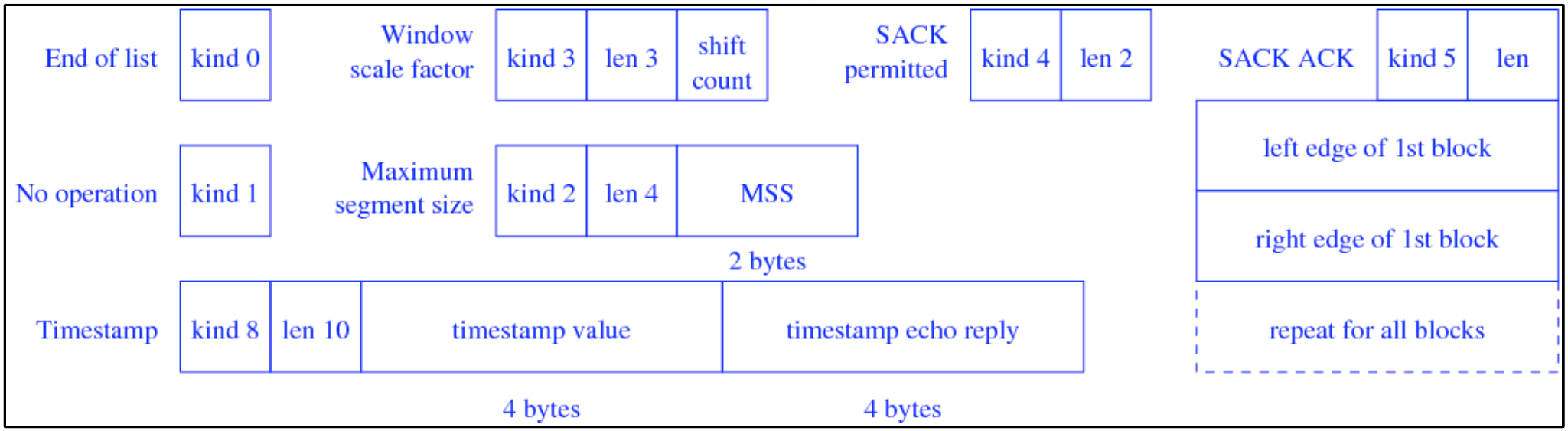
TCP Options



- *Timestamp* puts the time of day into the segment, allowing accurate measurement of the time segments are taking to travel. Useful for computing retransmission times

The Transport Layer

TCP Options



- *Selective acknowledgement* (SACK) is an extension of the ACK mechanism. *SACK permitted* says we are able to use SACK. More on this later

The Transport Layer

TCP Options

- Several options are only allowed in SYN segments, e.g., Window scale, MSS and SACK permitted. This is because some things, e.g., buffer space, need to be set up before a connection and varying them mid-connection make little sense