CS 716: Introduction to communication networks

- 18th class; 7th Oct 2011

Instructor: Sridhar Iyer
IIT Bombay

Reliable Transport

- We have already designed a reliable communication protocol for an analogy scenario.
  - Recall the functioning of secretaries in the CEO example. What did they have to do to reliably transfer the document using weak and unreliable messenger boys?
- From the analogy, we have learnt some ideas about how to ensure reliable transport.
  - Use buffers, timeouts, acknowledgments, retransmission ..
- We need to apply these ideas to an IP network to get details of the protocol between a source (S) and destination (D).

More specifically ...

The actions of the transport layer:

- Before beginning the data transfer:
  - What actions does (transport layer at) S need to do?
  - What messages does S need to send to D?
  - What responses does D need to give to S?
  - What actions does D need to do at its end?

- During the data transfer:
  - Consider above questions again in this context.
- Upon completion of the data transfer:
  - Consider above questions again in this context.

More questions ...

- How does the transport layer at S distinguish between packets coming down to it from multiple applications (ex: http and ssh)?
  - Their destination (D) may be the same or different.
- How to determine the number bits to allocate for the sequence number (unique packet id)?
  - Suppose you use 3 bits, the 1st packet and 8th packet will both have [000] as the sequence number.
- How to decide the number of packets that S could transmit before it waits for the 1st acknowledgment?
  - Suppose you transmit one packet, wait for its ack, then the next packet and so on, what is the drawback?

TCP: Transmission Control Protocol [RFC 793, …]

Guaranteed service protocol

- Ensures that a packet has been received by the destination by using timeouts, acknowledgements and retransmission

Connection-oriented protocol

- Applications need to establish a TCP connection prior to transfer, to fix initial sequence numbers
  - Done using a 3-way handshake

Full duplex protocol

- Both ends can simultaneously read and write

More TCP features

Flow and congestion control:

- Source uses feedback (ack) to adjust transmission rate.

Byte stream:

- Ignores message boundaries.
- Source may send two messages of length 20 and 50 bytes, but destination may simply receive 70 bytes.

Multiplexed:

- many applications can share access to a single TCP layer.

TCP functioning

Application data is broken into Segments (what TCP considers the best sized units to send)

- Segment: unit of data passed from TCP to IP
- MSS: Maximum segment size

TCP sequences data by associating a sequence number with every **byte** it sends

Sending TCP maintains a timer for each segment sent

- waiting for acknowledgement (ACK)
- If ACK doesn’t come in time, segment is retransmitted

TCP functioning

Receiving TCP

- Sends ACK: ACK number is the sequence number of the next byte expected
- Re-sequences the data
- Discards duplicates

- Congestion and Flow control
  - Sending TCP regulates amount of data to avoid network congestion
  - Receiving TCP prevents fast senders from swamping it

TCP connections and sockets

connections are the fundamental abstraction of communication.

- Port: A number on a host assigned to an application to allow multiple destinations.
- Endpoint: A pair, a destination host and a port number on that host.
- Connection: A pair of end points.
- Socket: An abstract address formed by the IP address and port number (characterizes an endpoint)

Unique identifier for a connection: [<Source IP address, port number>, <Destination IP address, port number>]

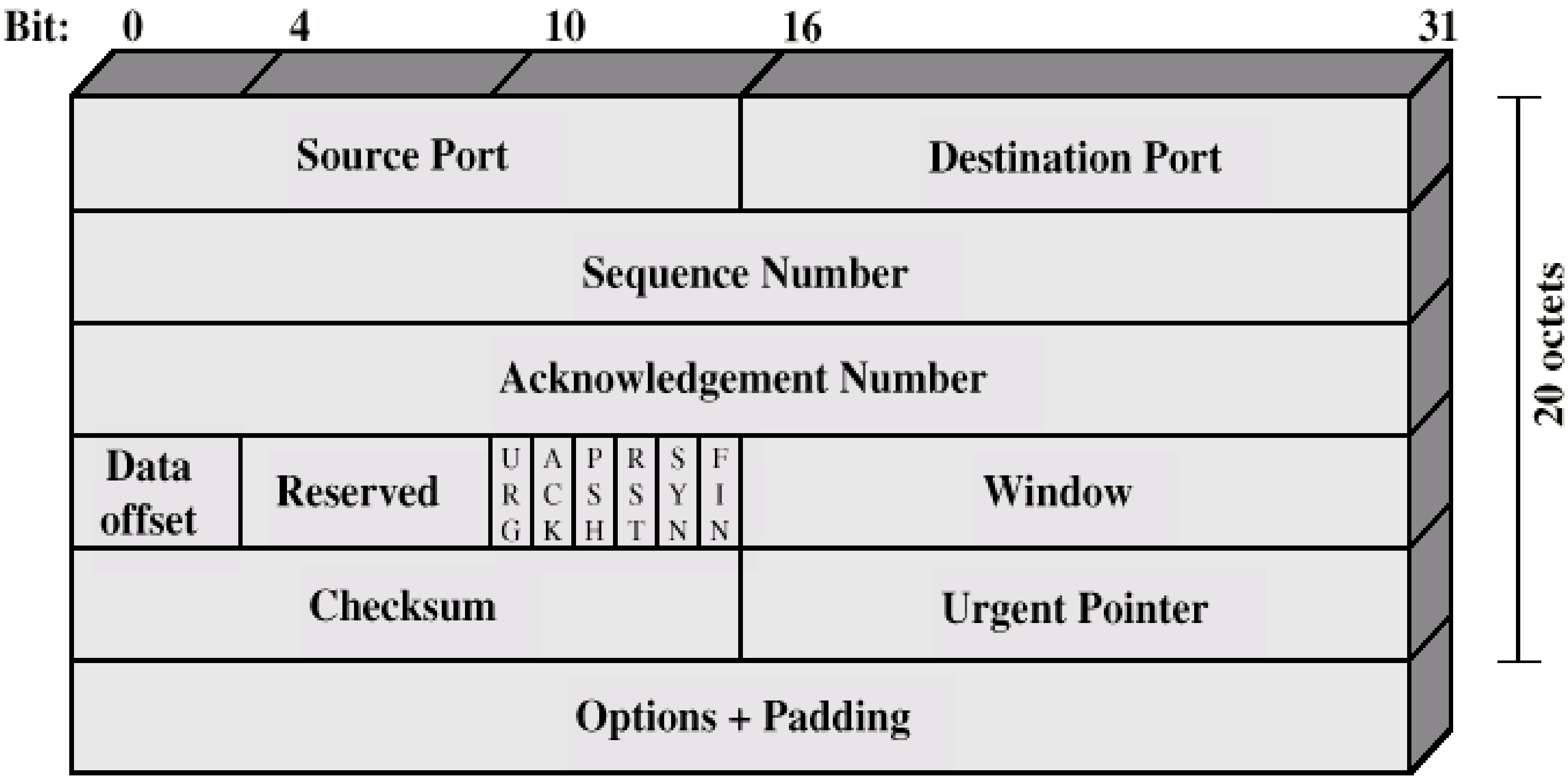
There are exactly two **end-points** communicating with each other in a TCP connection; Broadcast and multicast aren't applicable to TCP.

MSS: Maximum segment size

- Largest size of segment that TCP will send to the other end
  - Each end announces its MSS at connection establishment time
  - default size is 536 bytes (576 – 40)

- Done to avoid fragmentation
  - MSS related to outgoing link's MTU

TCP header


Bit: 0 4 10 16 31
Source Port
Destination Port
Sequence Number
Acknowledgement Number
Data offset
Reserved
URG
ACK
PSH
RST
SYN
FIN
Window
Checksum
Urgent Pointer
Options + Padding
20 octets

Port numbers

- 16-bit port numbers- 0 to 65535
- 0 to 1023 are *well-known* ports
  - assigned to common applications
  - telnet uses 23, SMTP 25, HTTP 80 etc.
- 1024 to 49151 are registered ports
  - 6000 through 6063 for X-win server
- 49152 to 65535 are *dynamic* or *private* ports.

Sequence number size

Sequence number identifies the byte in the stream between sender & receiver; Sequence number wraps around to 0 after reaching $2^{32} - 1$

Should be long enough so that sender does not confuse the sequence numbers on acks

- sending at < 100 packets/sec (R)
- wait for 200 sec before giving up (T)
- receiver may delay up to 100 sec (A)
- packet can be in network up to 300 sec (MSL: Max Segment Lifetime)
- Sender may send 900*100 packets before ack

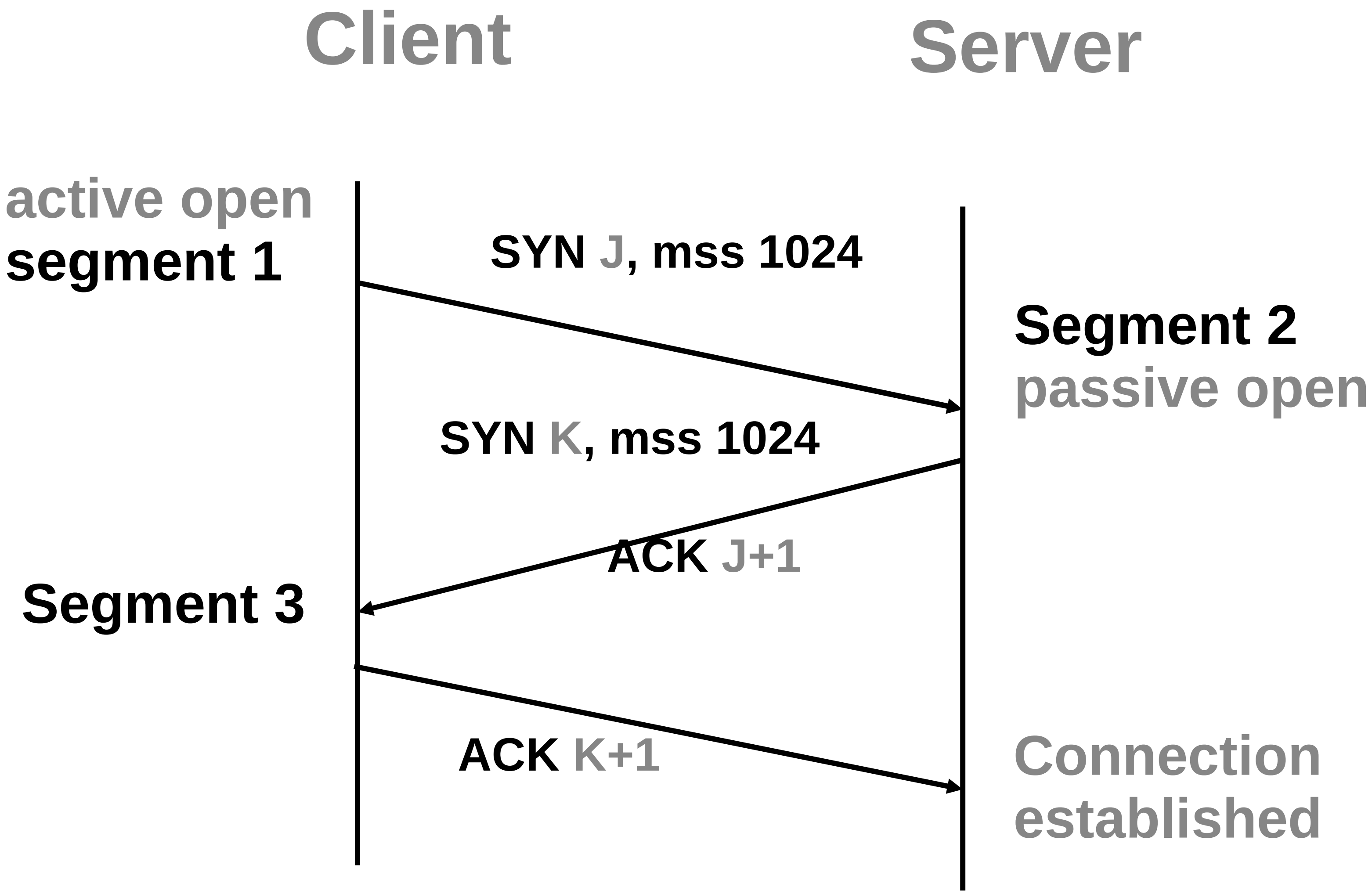
2^seq_size > R (2 MSL + T + A)

Initial sequence number

- Sequence numbers: another reason
  - host A opens connection to B, source port 123, destination port 456
  - Suppose connection terminates, a new connection opens, A and B assign the same port numbers
  - delayed pkt arrives from old connection
- New connection will have different initial sequence number (ISN)

- Tutorial Question: Confirm that Sequence Number Wrap Around Time is around 57 minutes for 10 Mbps Ethernet, while using 32 bits for Sequence Number.

TCP connection establishment



TCP 3-way handshake

1: Client sends SYN segment specifying the port number of *Server* and its initial sequence number (ISN)

2: Server responds with its own SYN containing its ISN and also acknowledges the client's SYN

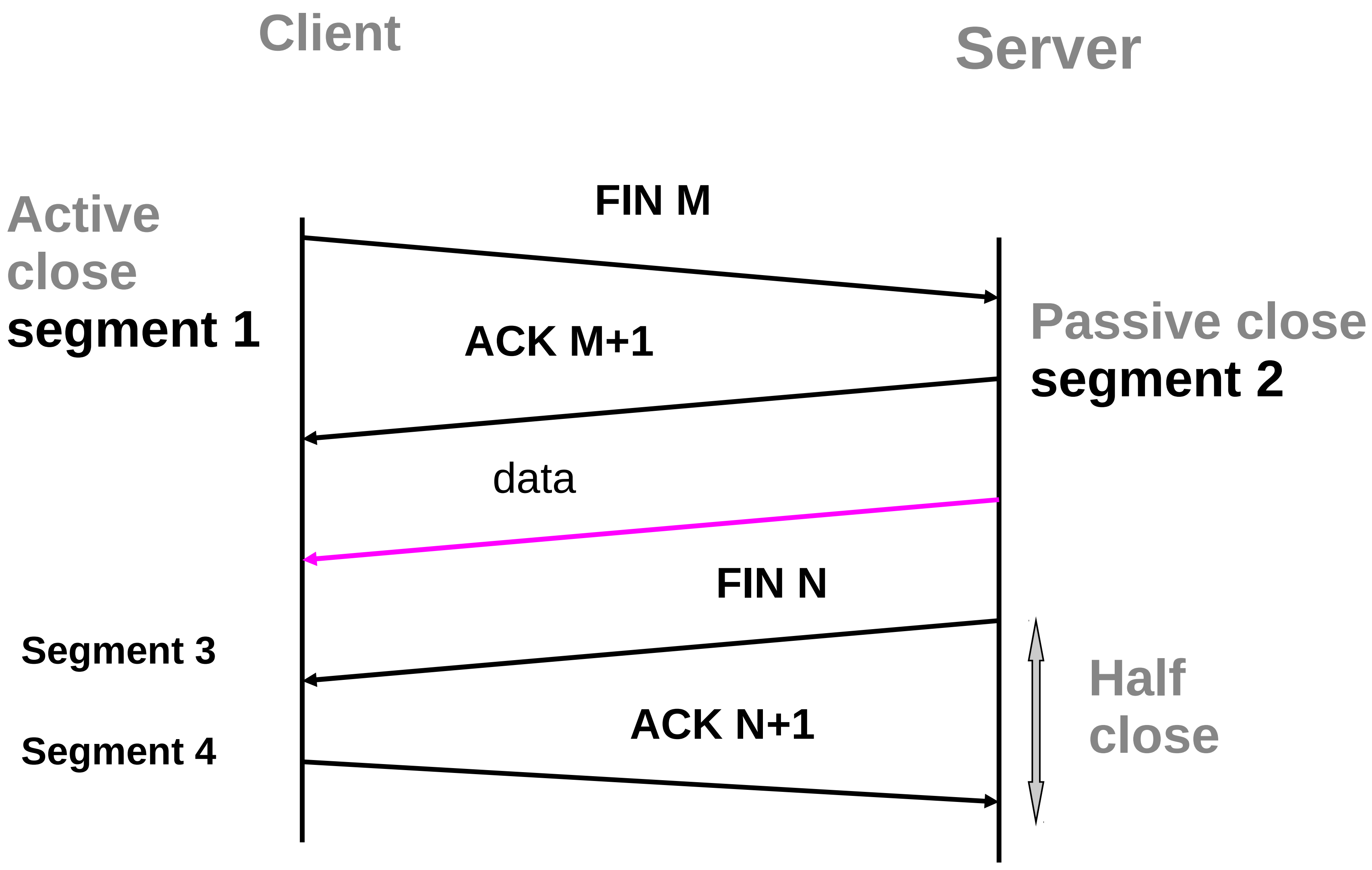
3: client responds to the SYN from the server by ACKing its ISN plus one

- Why 3-way handshake?

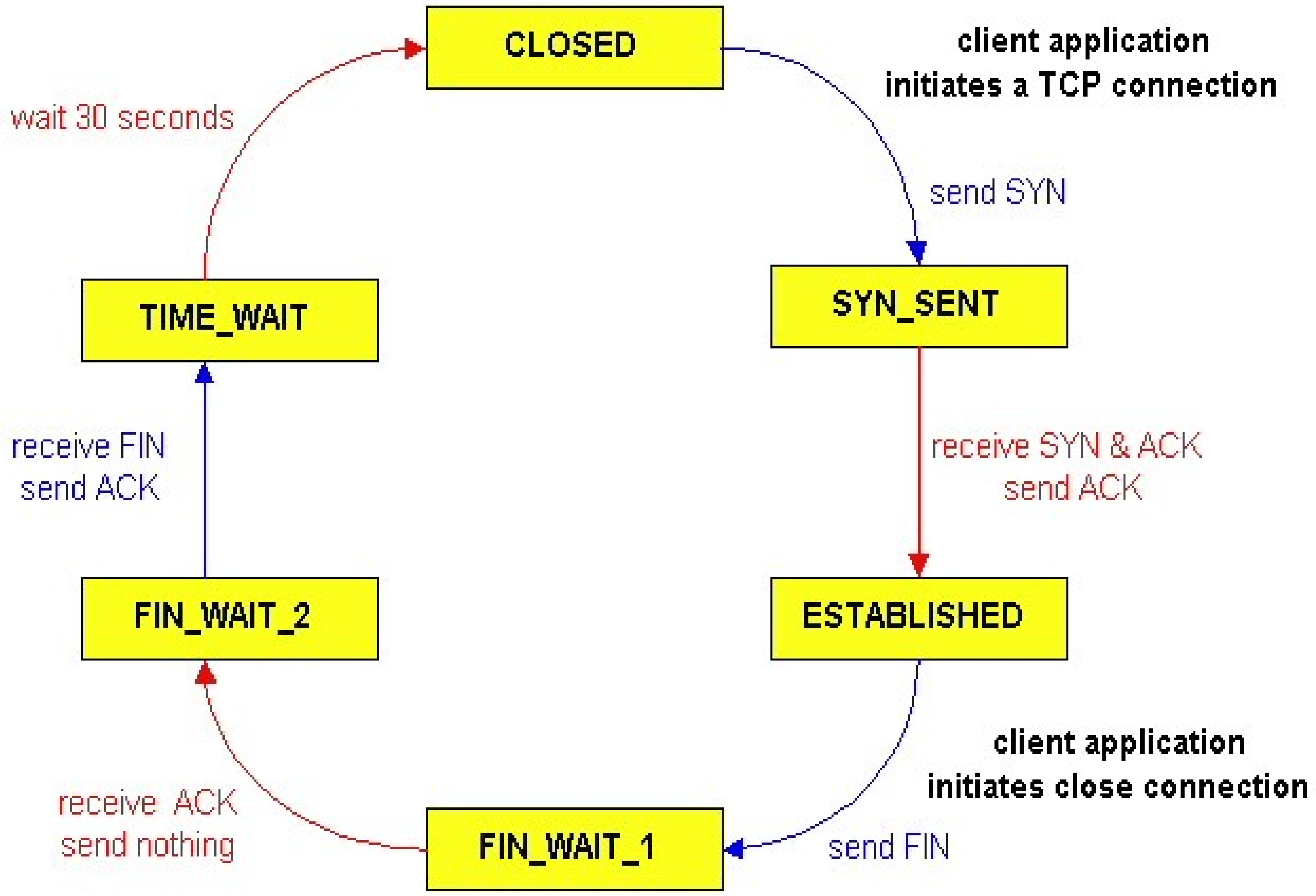
Problem with 2-way handshake is that SYNs themselves are not protected with sequence numbers
3-way handshake protects against delayed SYNs

Wait for 1 MSL (30s to 2 min) upon boot before initiating connection

TCP connection termination



TCP client states



TCP server states