NETWORK PROGRAMMING
BE (SE/IT)

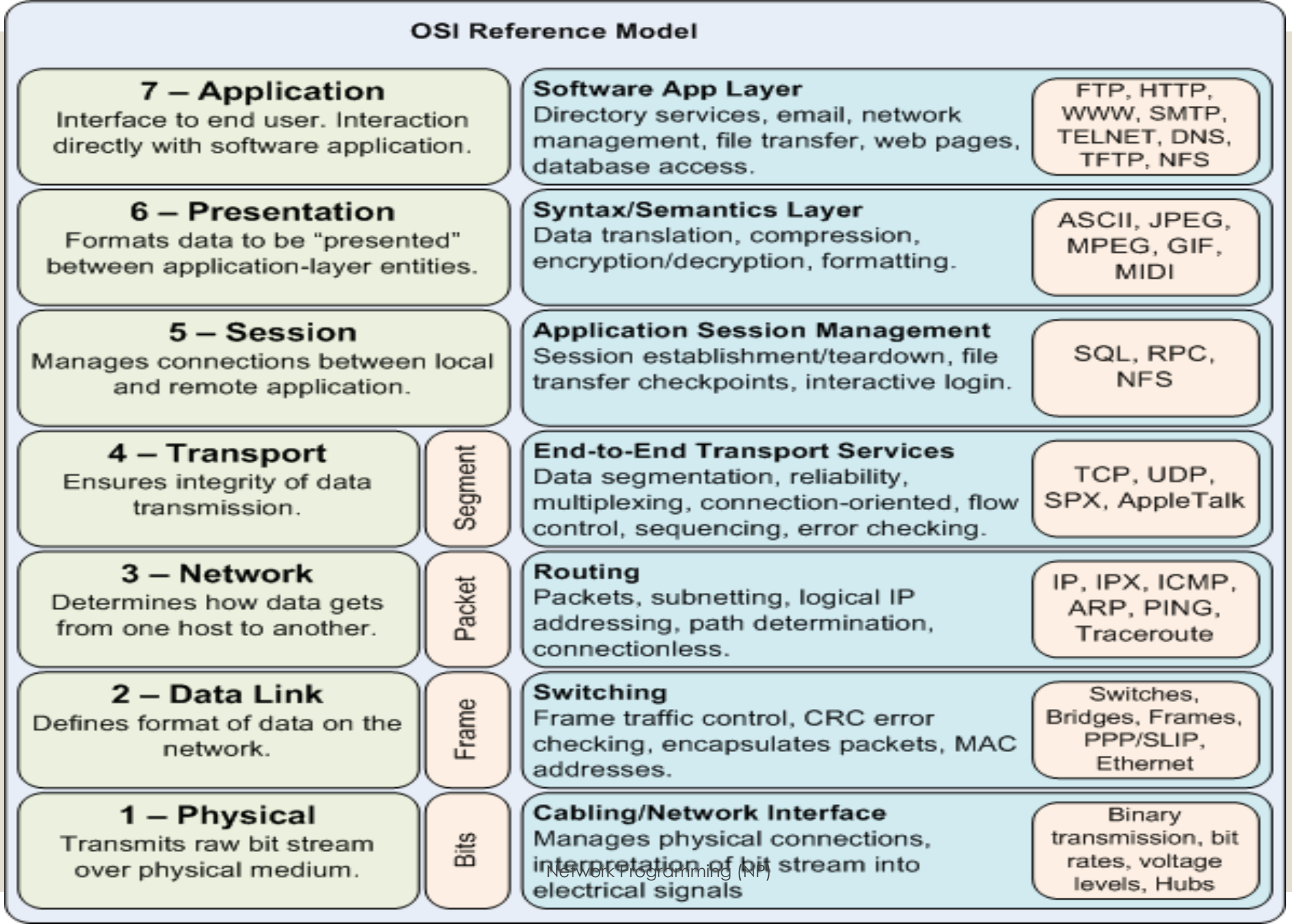
COMPILED BY:

ASSOC. PROF. MADAN KADARIYA
NCIT

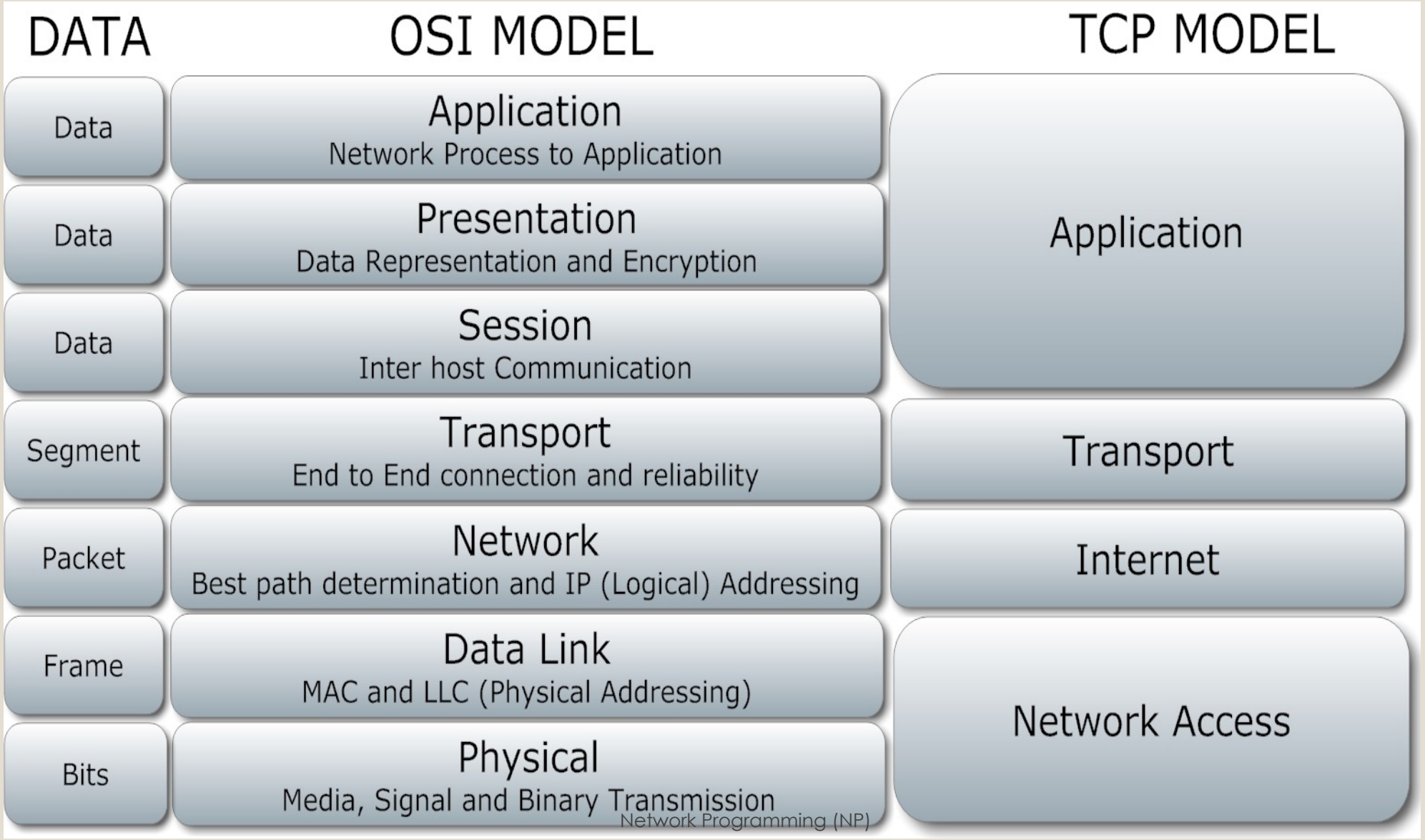
CHAPTER 1: NETWORK PROGRAMMING FUNDAMENTALS

Outline

1. Introduction to Networking and network programming
2. Client/Server mode
3. Communication Protocol (TCP, IP, UDP, SCTP)
4. TCP state transition Diagram
5. Protocol comparison


OSI Reference Model
7 – Application
Interface to end user. Interaction directly with software application.
Software App Layer
Directory services, email, network management, file transfer, web pages, database access.
FTP, HTTP, WWW, SMTP, TELNET, DNS, TFTP, NFS
6 – Presentation
Formats data to be "presented" between application-layer entities.
Syntax/Semantics Layer
Data translation, compression, encryption/decryption, formatting.
ASCII, JPEG, MPEG, GIF, MIDI
5 – Session
Manages connections between local and remote application.
Application Session Management
Session establishment/teardown, file transfer checkpoints, interactive login.
SQL, RPC, NFS
4 – Transport
Ensures integrity of data transmission.
Segment
End-to-End Transport Services
Data segmentation, reliability, multiplexing, connection-oriented, flow control, sequencing, error checking.
TCP, UDP, SPX, AppleTalk
3 – Network
Determines how data gets from one host to another.
Packet
Routing
Packets, subnetting, logical IP addressing, path determination, connectionless.
IP, IPX, ICMP, ARP, PING, Traceroute
2 – Data Link
Defines format of data on the network.
Frame
Switching
Frame traffic control, CRC error checking, encapsulates packets, MAC addresses.
Switches, Bridges, Frames, PPP/SLIP, Ethernet
1 – Physical
Transmits raw bit stream over physical medium.
Bits
Cabling/Network Interface
Manages physical connections, interpretation of bit stream into electrical signals
Binary transmission, bit rates, voltage levels, Hubs

OSI VS TCP/IP


DATA
OSI MODEL
TCP MODEL
Data
Application
Network Process to Application
Data
Presentation
Data Representation and Encryption
Data
Session
Inter host Communication
Application
Segment
Transport
End to End connection and reliability
Transport
Packet
Network
Best path determination and IP (Logical) Addressing
Internet
Frame
Data Link
MAC and LLC (Physical Addressing)
Bits
Physical
Media, Signal and Binary Transmission
Network Access

Similarities b/w TCP/IP and OSI

1. Both the reference models are based upon layered architecture.
2. The physical layer and the data link layer of the OSI model correspond to the link layer of the TCP/IP model. The network layers and the transport layers are the same in both the models. The session layer, the presentation layer and the application layer of the OSI model together form the application layer of the TCP/IP model.
3. In both the models, protocols are defined in a layer-wise manner.
4. In both models, data is divided into packets and each packet may take the individual route from the source to the destination.

Differences b/w TCP/IP and OSI

OSI

- OSI model is just a framework but not the implementation model.
- It consists of 7-layers.
- It clearly distinguishes between services, interfaces & protocols.
- The protocols are first defined & then the layers are set or fixed. Thus, the protocols in the OSI model can be easily replaced by the technology change.
- The network layer is connectionless or connection oriented.
- The transport layer is connection oriented.

TCP/IP

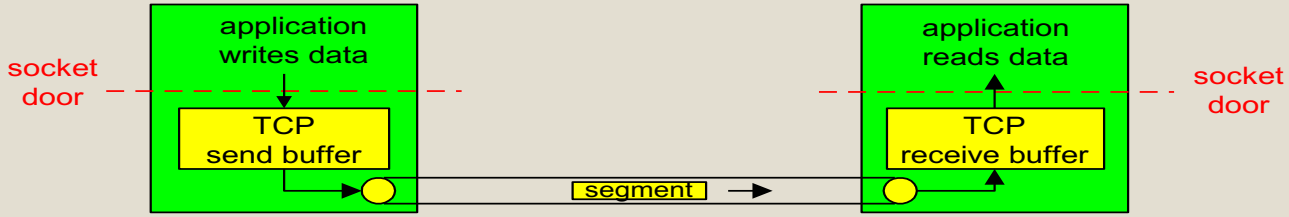
- It is an implementation model.
- It has just 4-layers.
- TCP/IP model fails to distinguish between services, interfaces & protocols.
- The layers are first set & then the protocols are defined. Thus, there may difficulties to replace it by the technology change.
- The network layer is connectionless.
- The transport layer is connectionless or connection oriented.

User Datagram Protocol (UDP)

- **UDP** is a simple **transport-layer protocol**. (RFC 768)
- The application writes a message to a **UDP socket**, which is then encapsulated in a **UDP datagram**, which is then further encapsulated as an IP datagram, which is then sent to its destination.
- There is no guarantee that a **UDP datagram** will ever reach its final destination, that order will be preserved across the network, or that datagrams arrive only once.
- With network programming using UDP is **its lack of reliability**. If a datagram reaches its final destination but the checksum detects an error, or if the datagram is dropped in the network, it is not delivered to the UDP socket and is **not automatically retransmitted**.
- Each UDP datagram has a length. The length of a datagram is passed to the receiving application along with the data.
- UDP provides a **connectionless service**, as there need not be any long-term relationship between a UDP client and server.

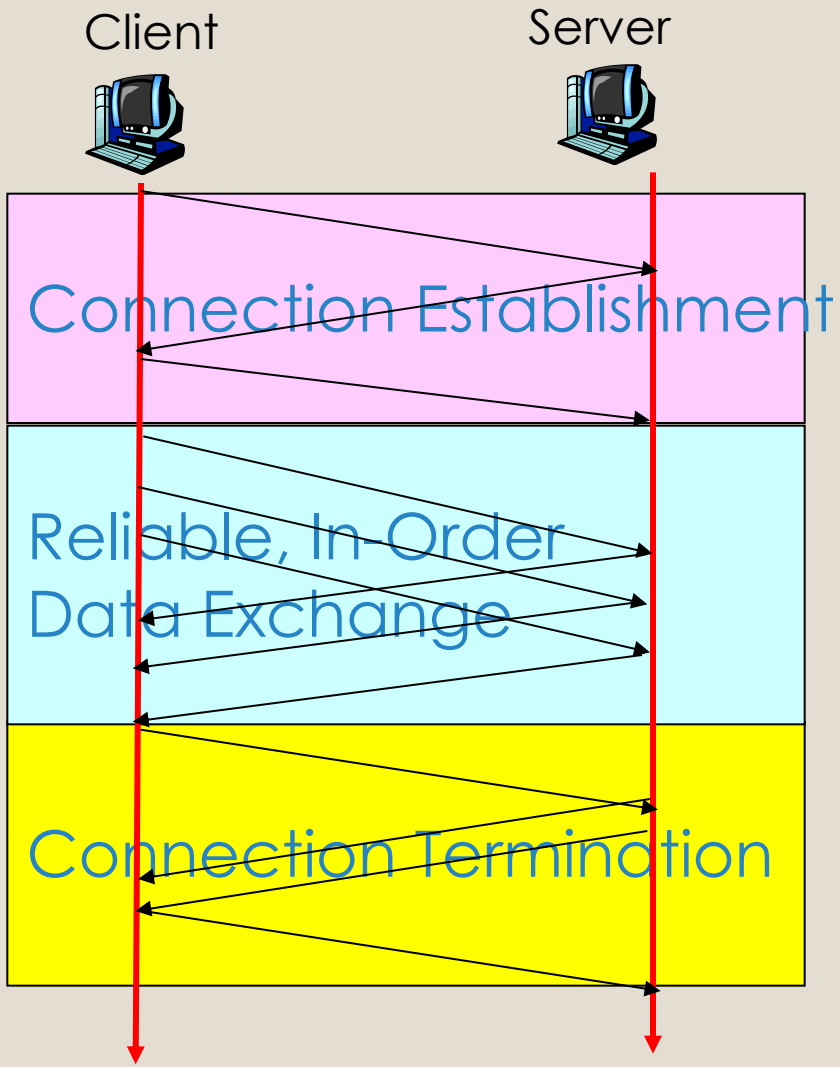
TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581



- point-to-point (unicast):
  - one sender, one receiver
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
  - State resides only at the END systems – Not a virtual circuit!
- full duplex data:
  - bi-directional data flow in same connection (A->B & B->A in the same connection)
  - MSS: maximum segment size
- reliable, in-order *byte steam:*
  - no "message boundaries"
- send & receive buffers
  - buffer incoming & outgoing data
- flow controlled:
  - sender will not overwhelm receiver
- congestion controlled:
  - sender will not overwhelm network

Typical TCP Transaction



- A TCP Transaction consists of 3 Phases
  1. **Connection Establishment**
     - Handshaking between client and server
  2. **Reliable, In-Order Data Exchange**
     - Recover any lost data through retransmissions and ACKs
  3. **Connection Termination**
     - Closing the connection

Few Protocols and Description

Protocol	Description
IPv4	Internet Protocol version 4. IPv4 uses 32-bit addresses and provides packet delivery service for TCP, UDP, SCTP, ICMP, and IGMP.
IPv6	Internet Protocol version 6. IPv6 uses 128-bit addresses.
TCP	Transmission Control Protocol. TCP is a **connection-oriented** protocol that provides a **reliable**, **full-duplex** byte stream to its users
UDP	User Datagram Protocol. UDP is a **connectionless** protocol, and UDP sockets are an example of datagram sockets.
SCTP	Stream Control Transmission Protocol. SCTP is a **connection-oriented** protocol that provides a reliable full-duplex association
ICMP	Internet Control Message Protocol. ICMP handles error and control information between routers and hosts.
IGMP	Internet Group Management Protocol. IGMP is used with multicasting.
ARP	Address Resolution Protocol. ARP maps an IPv4 address into a hardware address (such as an Ethernet address). ARP is normally used on broadcast networks such as Ethernet, token ring, and FDDI, and is not needed on point-to-point networks.
RARP	Reverse Address Resolution Protocol. RARP maps a hardware address into an IPv4 address.

Protocol Comparison

Services/Features	SCTP	TCP	UDP
Connection-oriented	yes	yes	no
Full duplex	yes	yes	yes
Reliable data transfer	yes	yes	no
Partial-reliable data transfer	optional	no	no
Ordered data delivery	yes	yes	no
Unordered data delivery	yes	no	yes
Flow control	yes	yes	no
Congestion control	yes	yes	no
ECN capable	yes	yes	no
Selective ACKs	yes	optional	no
Preservation of message boundaries	yes	no	yes
Path MTU discovery	yes	yes	no
Application PDU fragmentation	yes	yes	no
Application PDU bundling	yes	yes	no
Multistreaming	yes	no	no
Multihoming	yes	no	no
Protection against SYN flooding attacks	yes	no	n/a
Allows half-closed connections	no	yes	n/a
Reachability check	yes	yes	no
Psuedo-header for checksum	no (uses vtags)	yes	yes
Time wait state	for vtags	for 4-tuple	n/a

Transmission Control Protocol (TCP)

1. **Connection**: TCP provides connections between clients and servers. A TCP client establishes a connection with a server, exchanges data across the connection, and then terminates the connection.
2. **Reliability**: TCP requires acknowledgment when sending data. If an acknowledgment is not received, TCP automatically retransmits the data and waits a longer amount of time.
3. **Round-trip time** (RTT): TCP estimates RTT between a client and server dynamically so that it knows how long to wait for an acknowledgment.
4. **Sequencing**: TCP associates a sequence number with every byte (**segment**, unit of data that TCP passes to IP.) it sends. TCP reorders out-of-order segments and discards duplicate segments.
5. **Flow control:** is a set of procedures to restrict the amount of data that sender can send. Stop and Wait Protocol is a **flow control** protocol where sender sends one data packet to the receiver and then stops and waits for its acknowledgement from the receiver.
6. **Full-duplex**: an application can send and receive data in both directions on a given connection at any time.

Stream Control Transmission Protocol (SCTP)

1. Like **TCP**, SCTP provides **reliability**, **sequencing**, **flow control**, and **full-duplex data transfer**.
2. Unlike TCP, SCTP provides:
   I. **Association** instead of "connection": An association refers to a communication between two systems, which may involve more than two addresses due to **multihoming**.
   II. **Message-oriented**: provides sequenced delivery of individual records. Like UDP, the length of a record written by the sender is passed to the receiving application.
   III. **Multihoming**: allows a single SCTP endpoint to support multiple IP addresses. This feature can provide increased robustness against network failure.

TCP Connection Establishment

The following scenario occurs when a TCP connection is established.

1. The server must be prepared to accept an incoming connection. This is normally done by calling **socket, bind**, and **listen** and is called a passive open.
2. The client issues an active open by calling **connect**. This causes the client TCP to send a "**synchronize**" (**SYN**) segment, which tells the server the client's initial sequence number for the data that the client will send on the connection. Normally, there is no data sent with the **SYN**; it just contains an IP header, a TCP header, and possible TCP options.
3. The server must acknowledge (**ACK**) the client's **SYN** and the server must also send its own **SYN** containing the initial sequence number for the data that the server will send on the connection. The server sends its **SYN** and the **ACK** of the client's **SYN** is a single segment.
4. The client must acknowledge the server's SYN.

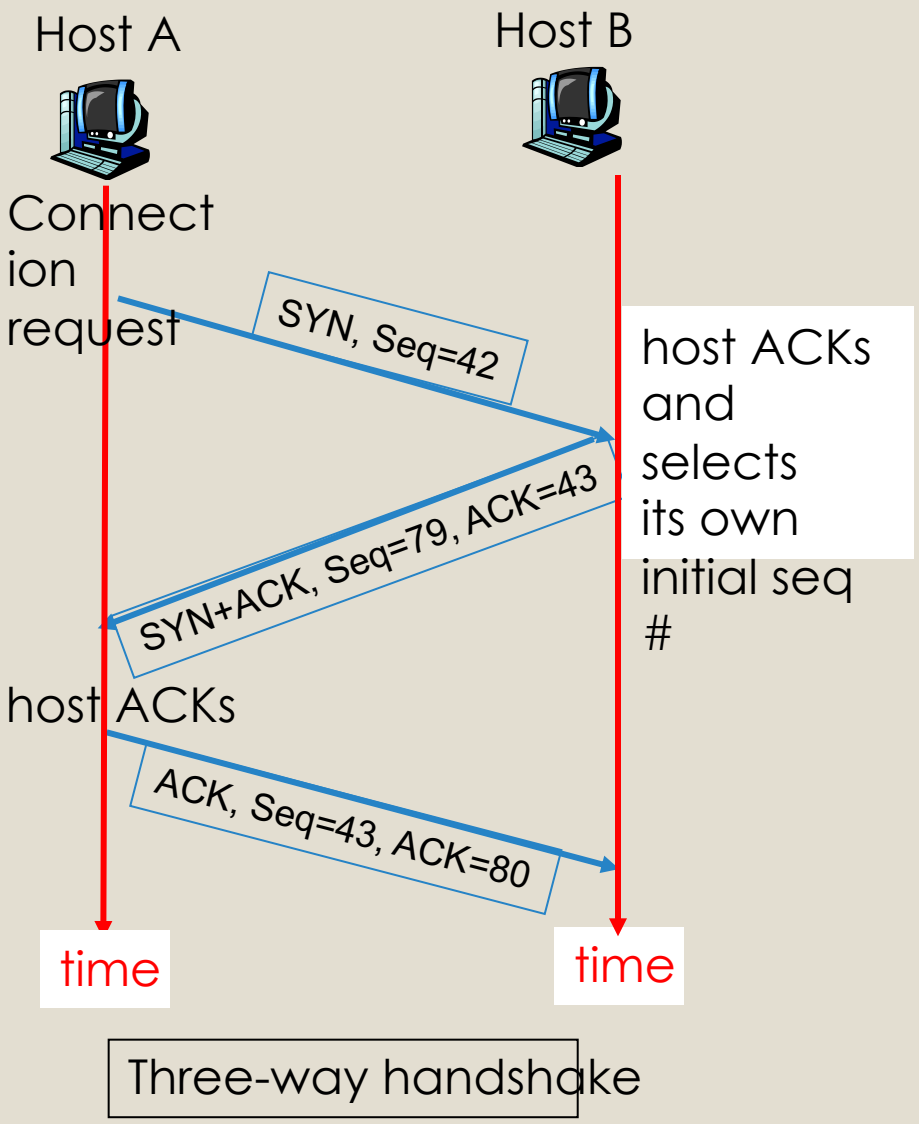
Connection Establishment (cont)

Three way handshake:

Step 1: client host sends TCP SYN segment to server
- specifies a random initial seq #
- no data

Step 2: server host receives SYN, replies with SYNACK segment
- server allocates buffers
- specifies server initial seq. #

Step 3: client receives SYNACK, replies with ACK segment, which may contain data

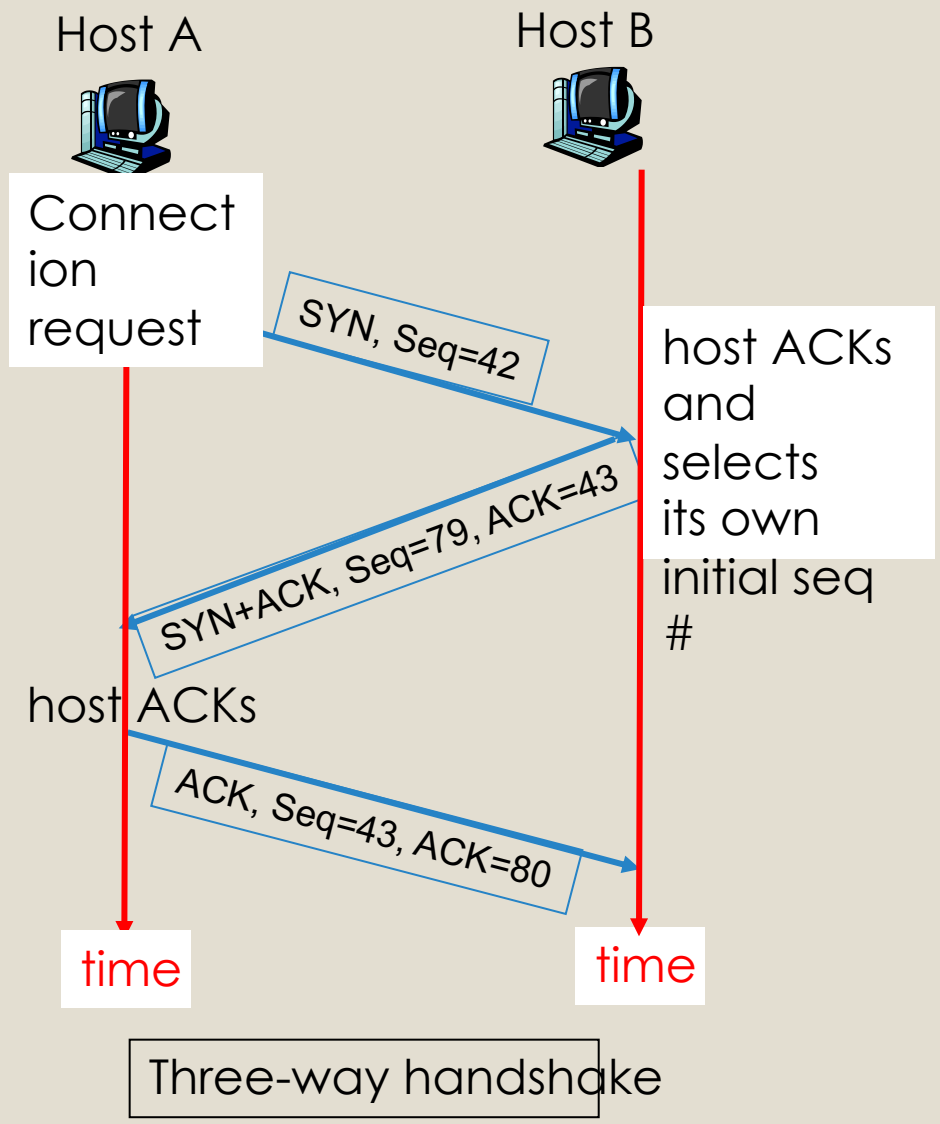


Three-way handshake

Connection Establishment (cont)

Seq. #'s:
- byte stream "number" of first byte in segment's data

ACKs:
- seq # of next byte expected from other side
- cumulative ACK



Three-way handshake

TCP Starting Sequence Number Selection

- Why a random starting sequence #? Why not simply choose 0?
  - To protect against two incarnations of the same connection reusing the same sequence numbers too soon
  - That is, while there is still a chance that a segment from an earlier incarnation of a connection will interfere with a later incarnation of the connection
- How?
  - Client machine seq #0, initiates connection to server with seq #0.
  - Client sends one byte and client machine crashes
  - Client reboots and initiates connection again
  - Server thinks new incarnation is the same as old connection

TCP Connection Termination

1. One application calls **close** first, and we say that this end performs the **active close**. This end's TCP sends a **FIN** segment, which means it is **finished sending data**.
2. The other end that receives the **FIN** performs the **passive close**. The received FIN is acknowledged by **TCP**. The receipt of the **FIN** is also passed to the application as an end-of-file, since the receipt of the **FIN** means the application will not receive any additional data on the connection.
3. Sometime later, the application that received the end-of-file will close its socket. This causes its **TCP** to send a **FIN**.
4. The TCP on the system that receives this final **FIN** (the end that did the active close) acknowledges the **FIN**.

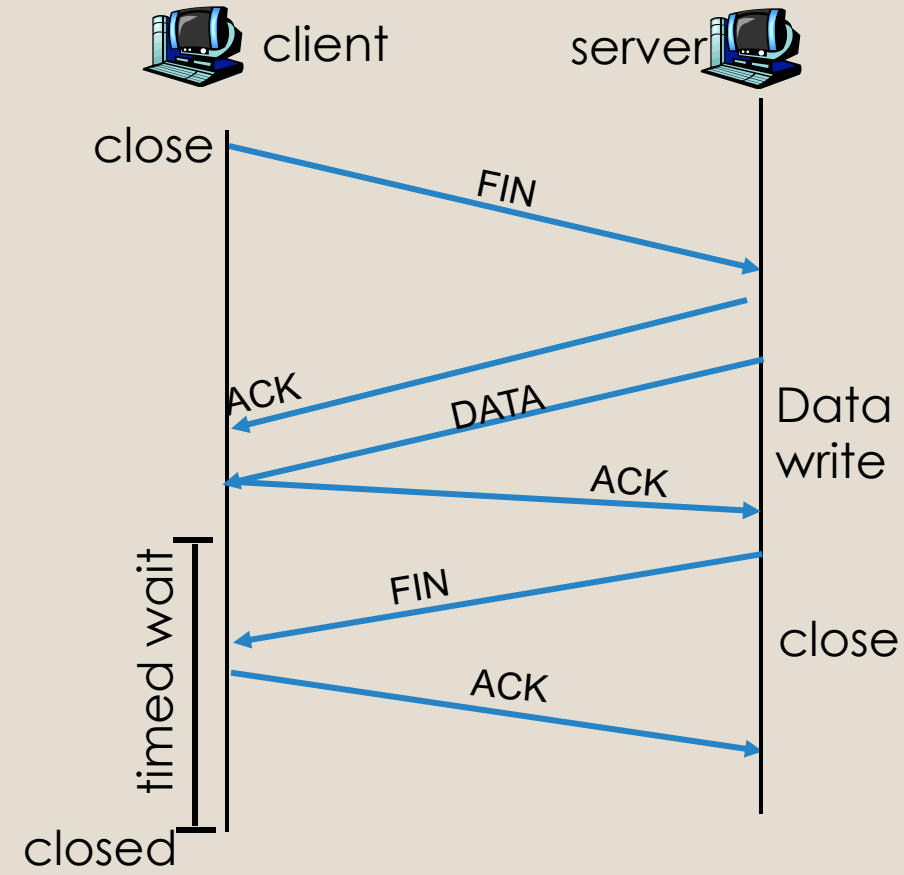
TCP Connection Termination

Closing a connection:

client closes socket:
`clientSocket.close();`

Step 1: client end system sends TCP FIN control segment to server

Step 2: server receives FIN, replies with ACK. Server might send some buffered but not sent data before closing the connection. Server then sends FIN and moves to Closing state.



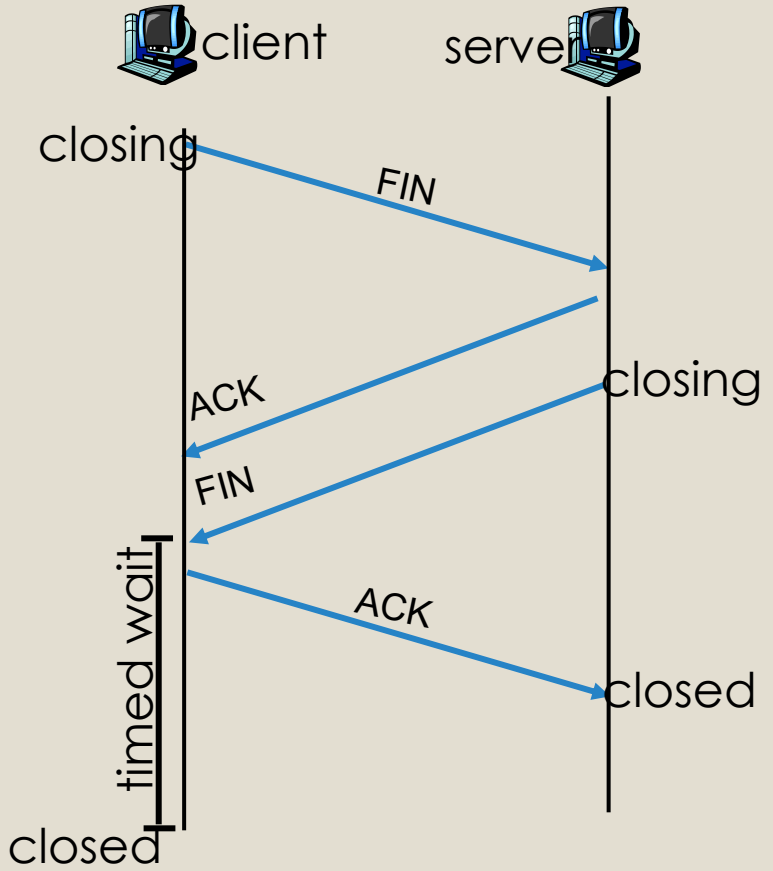
TCP Connection Termination

Step 3: client receives FIN, replies with ACK.

- Enters "timed wait" - will respond with ACK to received FINs

Step 4: server, receives ACK. Connection closed.

- Why wait before closing the connection?
  - If the connection were allowed to move to CLOSED state, then another pair of application processes might come along and open the same connection (use the same port #s) and a delayed FIN from an earlier incarnation would terminate the connection.

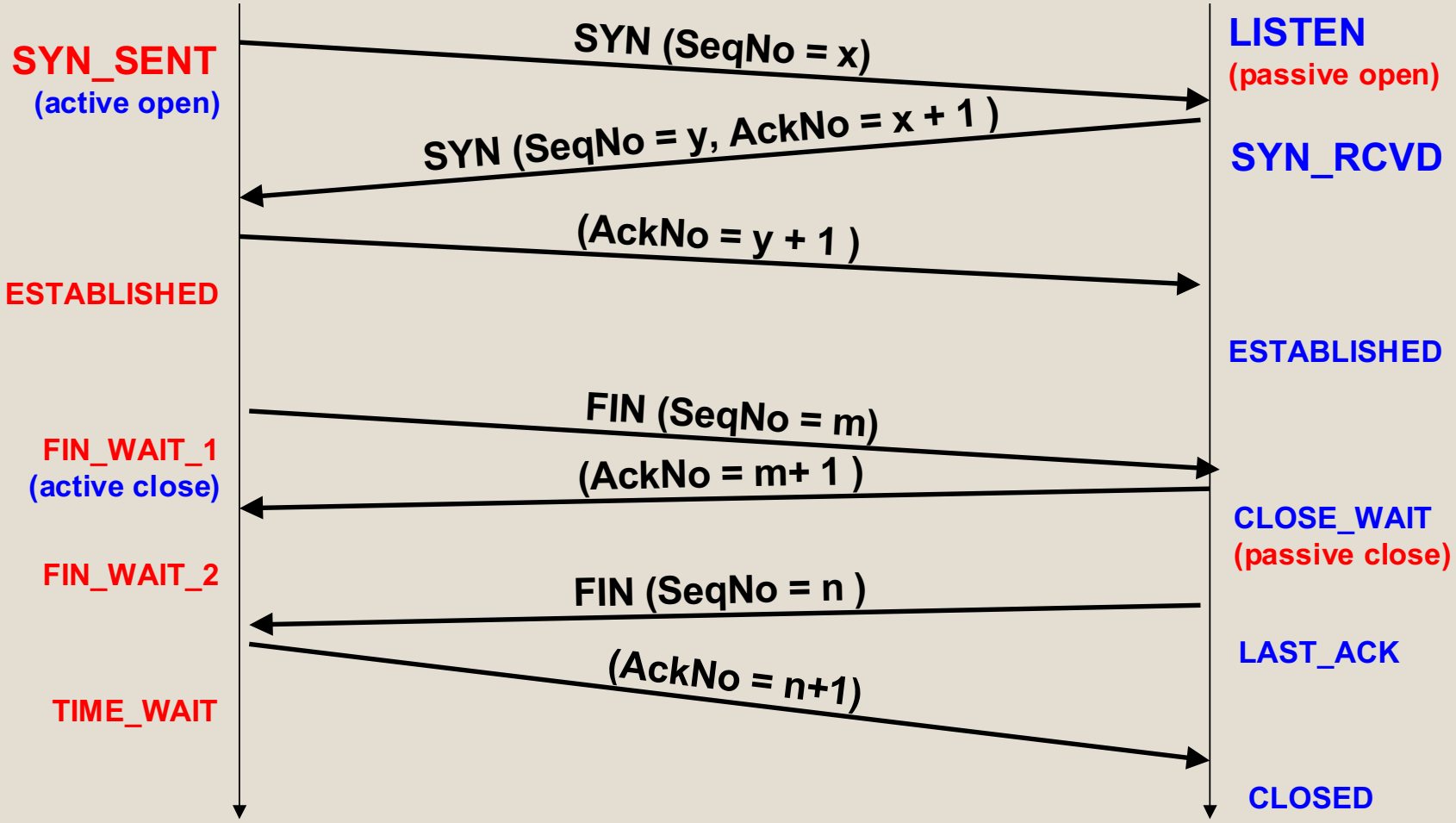


TCP/IP STATE TRANSITION DIAGRAM

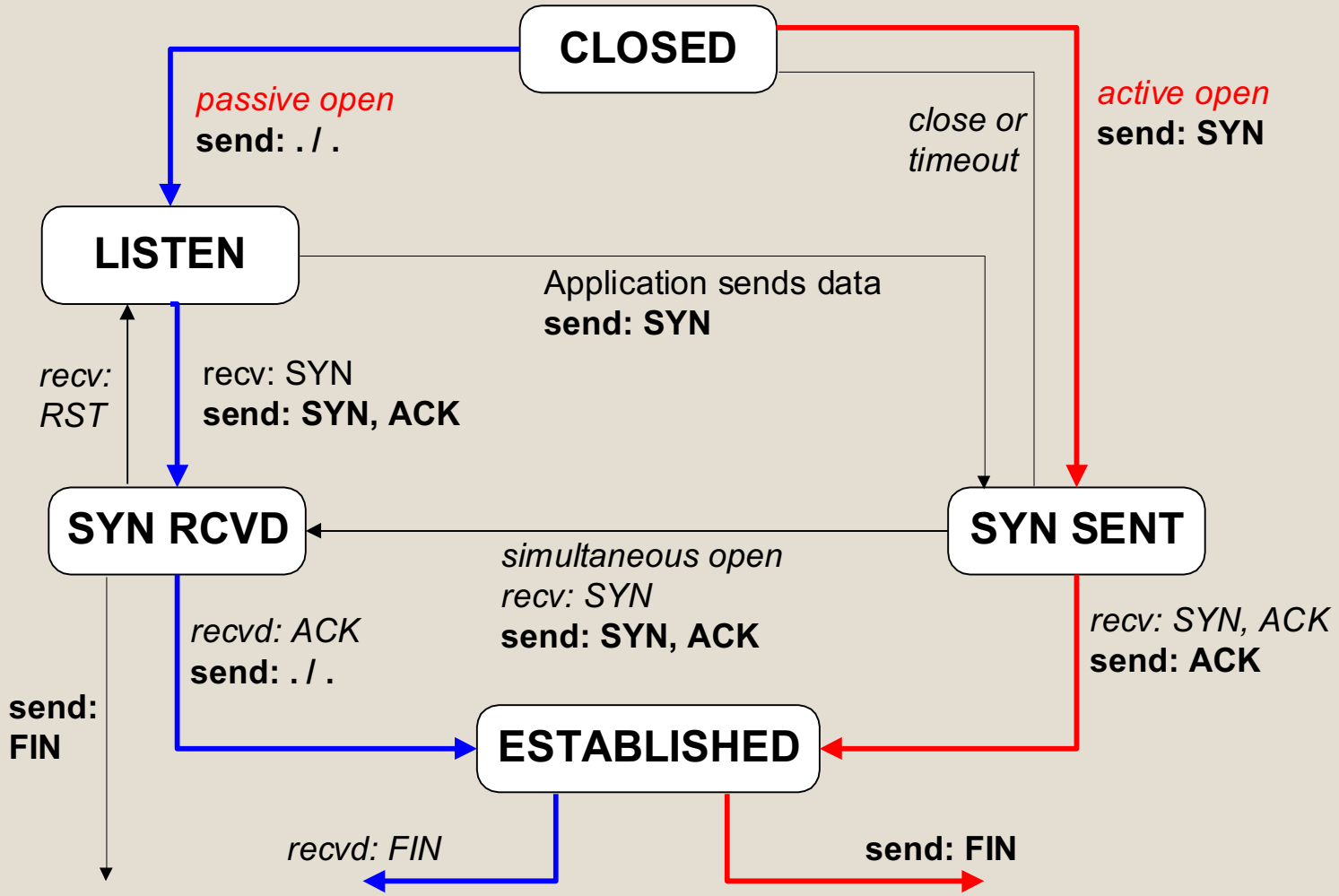
TCP States

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for Ack
SYN SENT	The client has started to open a connection
ESTABLISHED	Normal data transfer state
FIN WAIT 1	Client has said it is finished
FIN WAIT 2	Server has agreed to release
TIMED WAIT	Wait for pending packets (“2MSL wait state”)
CLOSING	Both Sides have tried to close simultanesously
CLOSE WAIT	Server has initiated a release
LAST ACK	Wait for pending packets

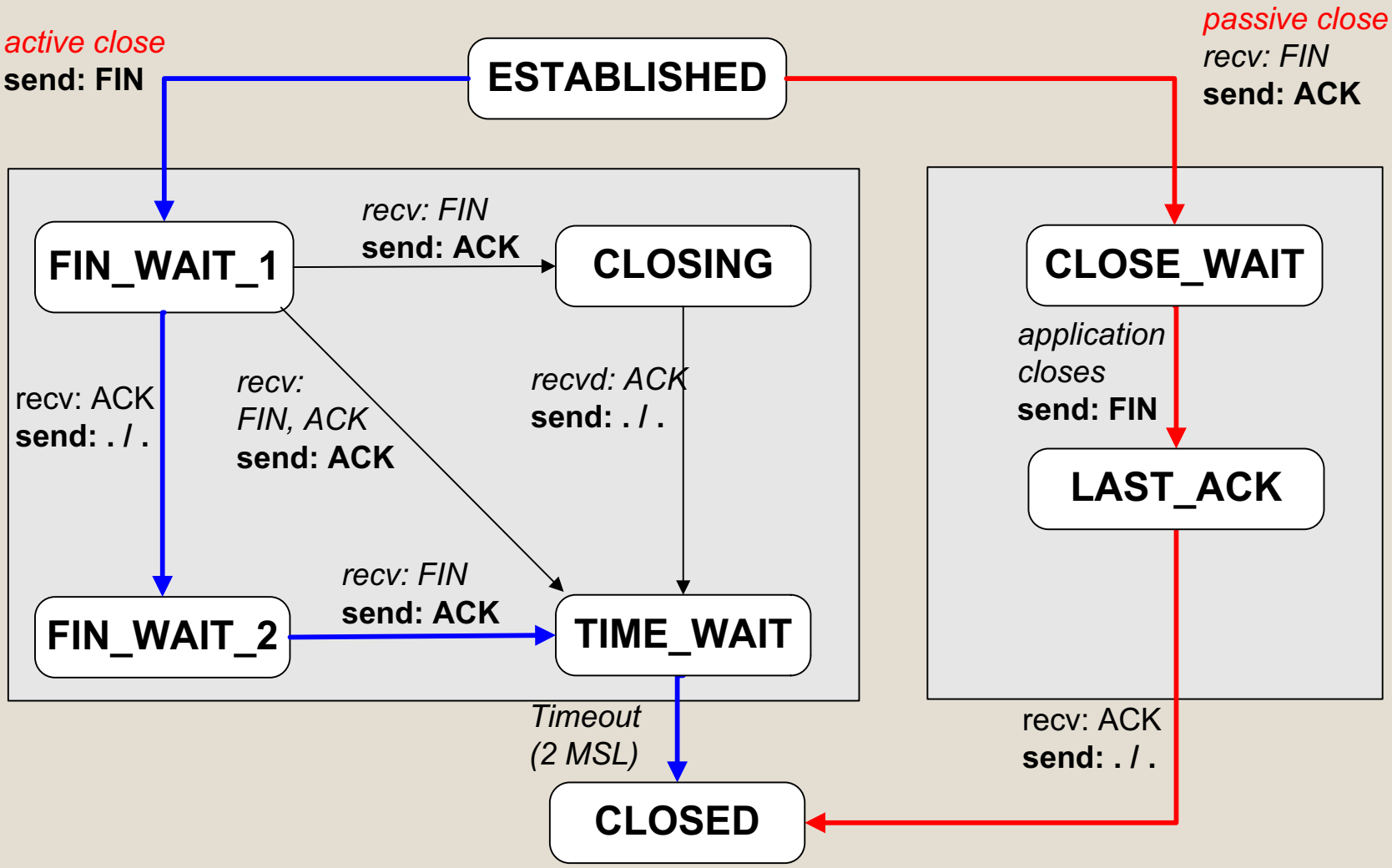
TCP States in "Normal" Connection Lifetime



TCP State Transition Diagram Opening A Connection



TCP State Transition Diagram Closing A Connection



2MSL Wait State

2MSL Wait State = TIME_WAIT

- When TCP does an active close, and sends the final ACK, the connection **must stay in in the TIME_WAIT state for twice the maximum segment lifetime**.

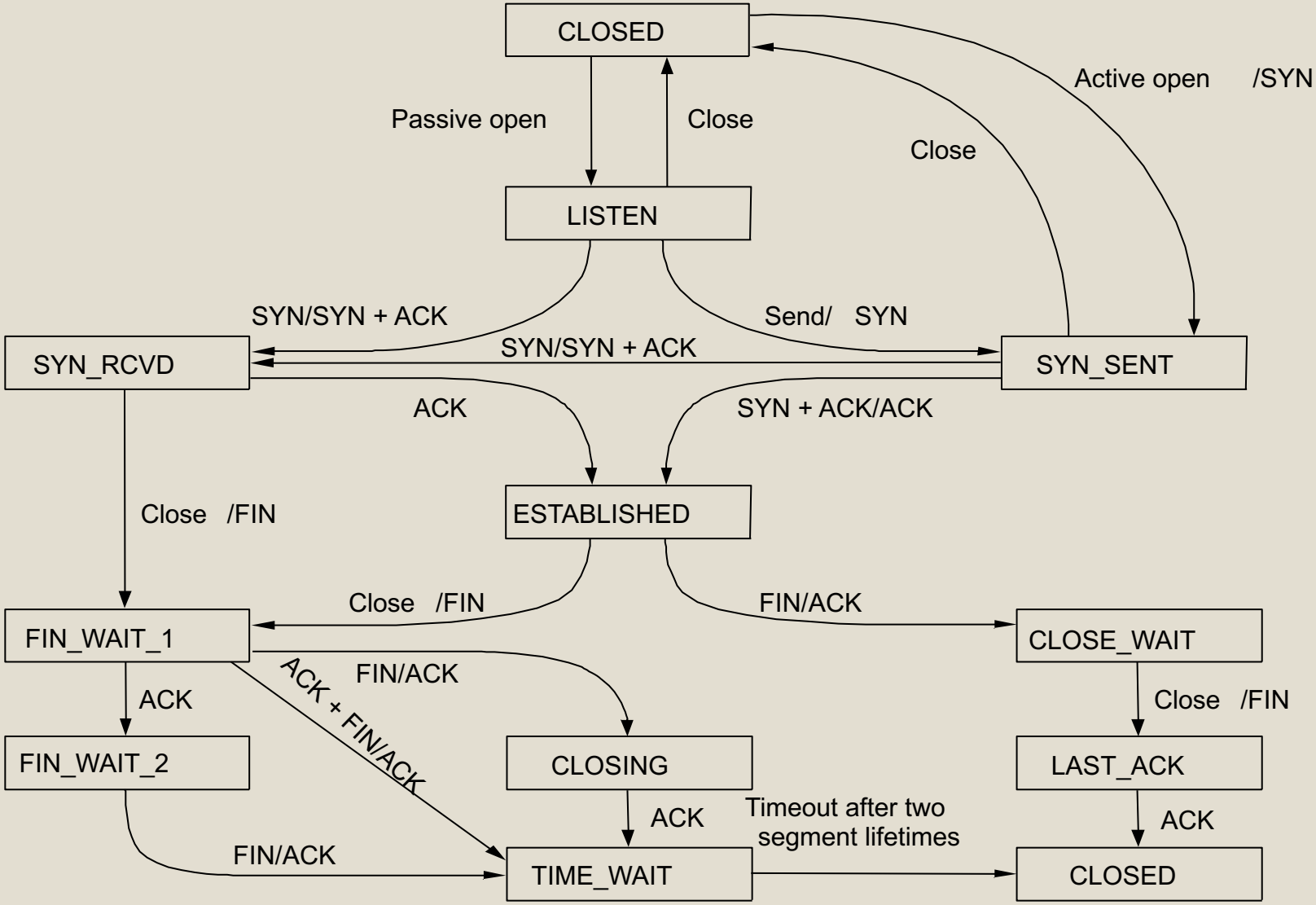
2MSL= 2 * Maximum Segment Lifetime

- Why?
  TCP is given a chance to resent the final ACK. (Server will timeout after sending the FIN segment and resend the FIN)
- The MSL is set to 2 minutes or 1 minute or 30 seconds.

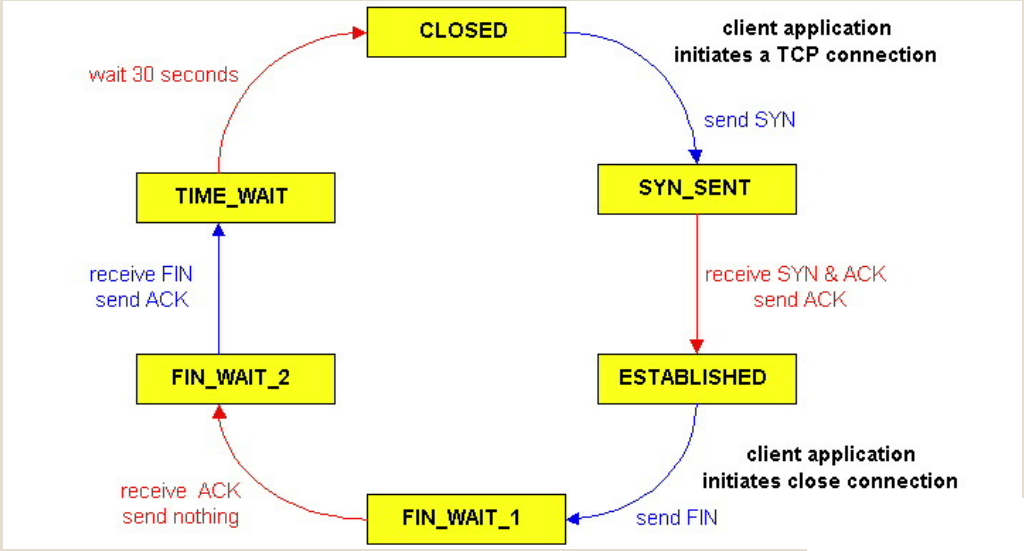
Resetting Connections

- Resetting connections is done by setting the RST flag
- **When is the RST flag set?**
  - Connection request arrives and no server process is waiting on the destination port
  - Abort (Terminate) a connection
    Causes the receiver to throw away buffered data. Receiver does not acknowledge the RST segment

TCP State-Transition Diagram

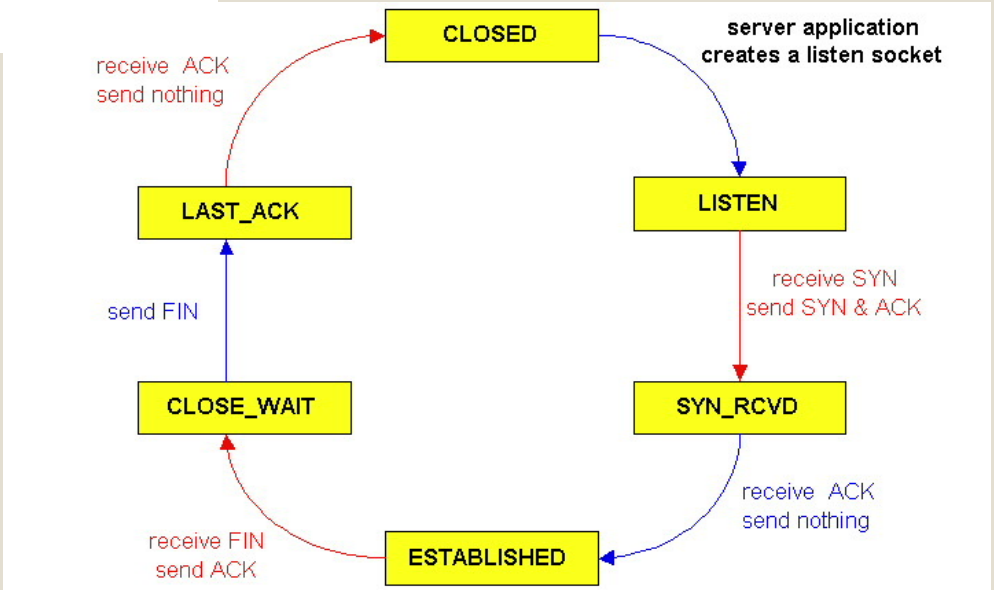


Typical TCP Client/Server Transitions



TCP client lifecycle

TCP server lifecycle



END OF CHAPTER 1