TCP Connection Control

TCP Connection Establishment

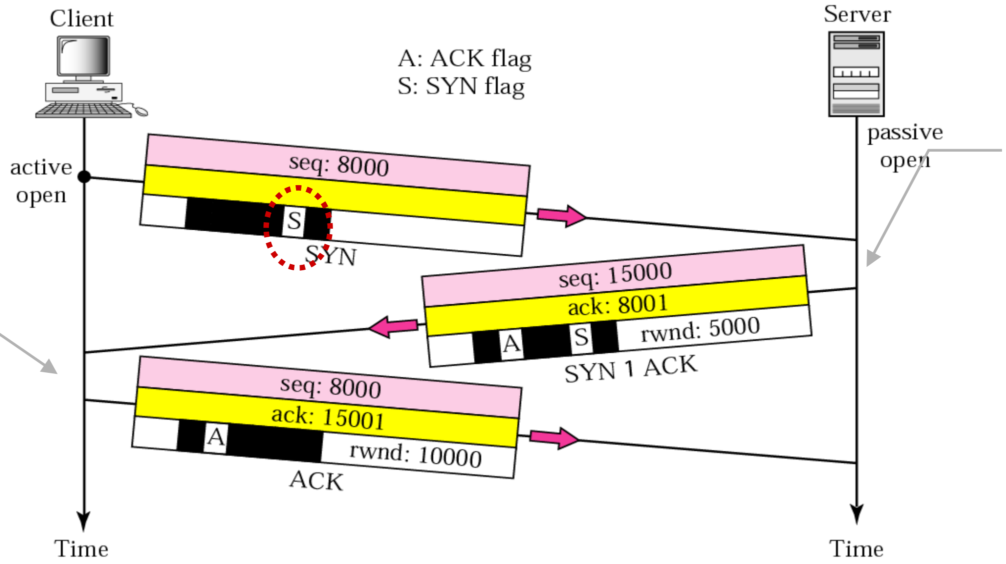
– TCP sender and receiver must establish “connection” before sending any data (this includes initialization of TCP variables: sequence #s, flow control info, ...)

“Three-Way Handshake”:

(1) client sends a SYN segment to server, which includes

- Source and Destination Port
- SYN = 1
- Sequence Number = *client_seq* – randomly chosen to prevent certain security attacks (attacker cannot spoof TCP connection without sniffing TCP packets)
- no application data!!!

allocate send and receive buffers and variables (window size defined)



allocate send and receive buffers and variables (window size defined)

TCP Connection Control (cont.)

“Three-Way Handshake” (cont.):

(2) once server receives SYN segment, it allocates buffers and variables to the connection, and sends a connection-granted segment (SYNACK) back to client

- SYN = 1
- Sequence Number = *server_seq*
- ACK = *client_seq + 1* – only (+1) because no user data have been sent
- server receive (client send) window size defined
- no application data!!!

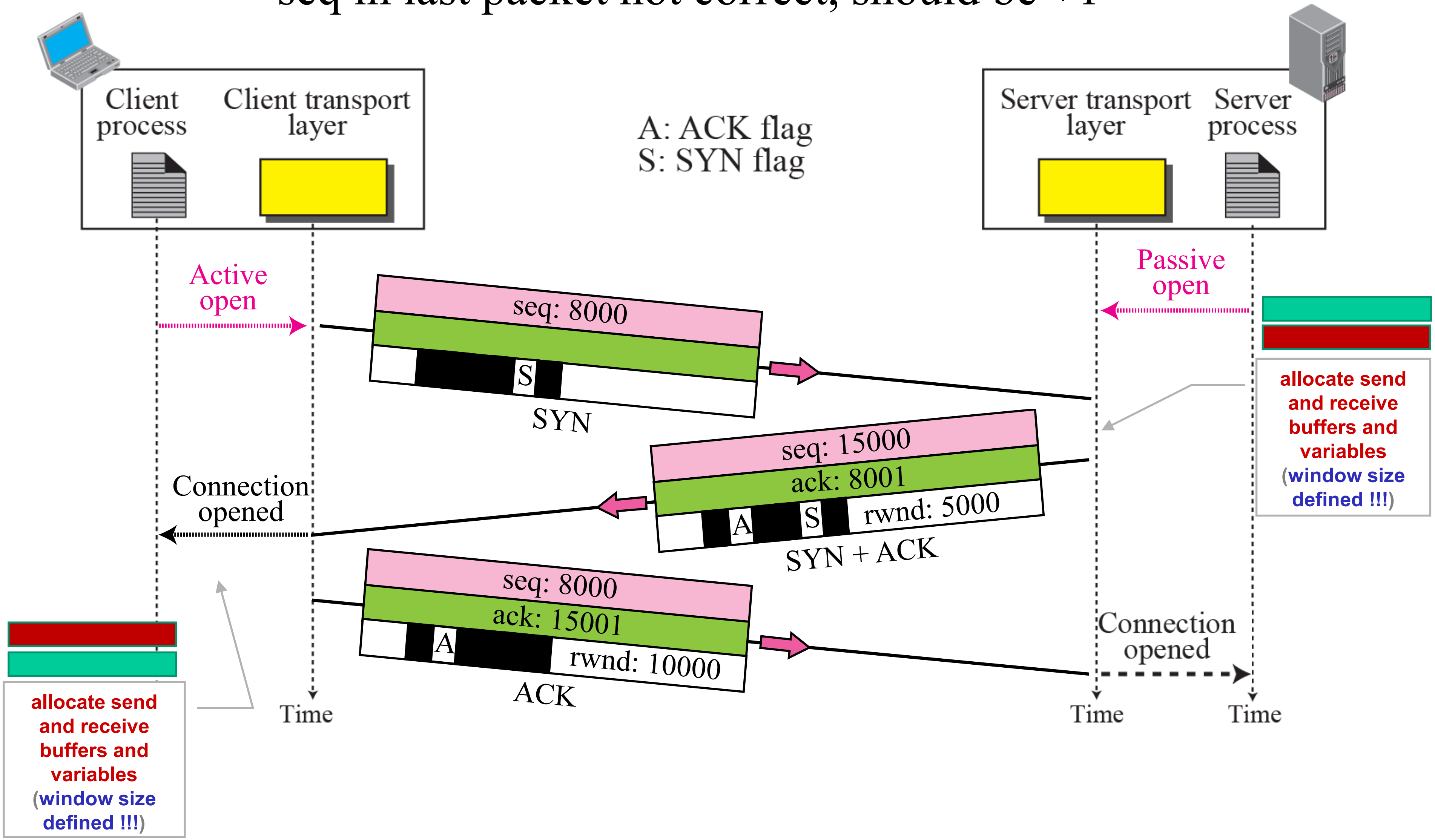
(3) upon receiving SYNACK segment, client also allocates buffers and variables to the connection, and sends the last segment to server (ACK)

- SYN = 0 – connection is established!
- Sequence Number = *client_seq + 1*
- ACK = *server_seq + 1*
- client send (server receive) window size defined
- no application data (although allowed in some implementations)

Why 3-way instead of 2-way handshake?!

TCP Connection Control (cont.)

seq in last packet not correct, should be +1



TCP Connection Control (cont.)

Example **[Two-Way Handshake: deadlock problem with obsolete SYN segment]**

"Two-Way Handshake": only steps (1) and (2) of Three-Way Handshake
- resources allocated already at the first SYN request - NOT used as it would lead to resource-starvation problems …

Assume an old SYN segment has survived the termination of a TCP connection between A and B.

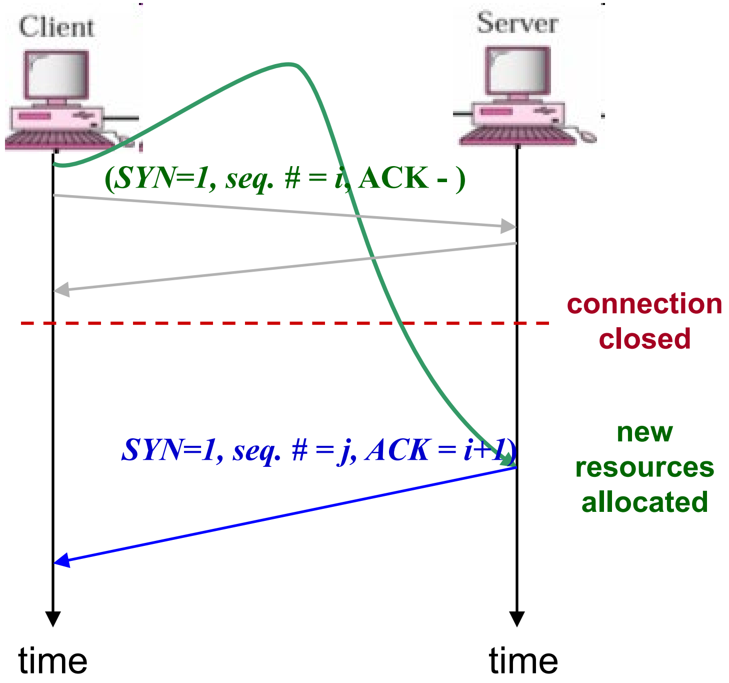
(1) Old (*SYN=1, seq. # = i*, ACK -) arrives at B.

(2) B assumes that this is a fresh request and responds with (*SYN=1, seq. # = j, ACK = i+1*).

(3) A rejects the segment from B as an old duplicate.

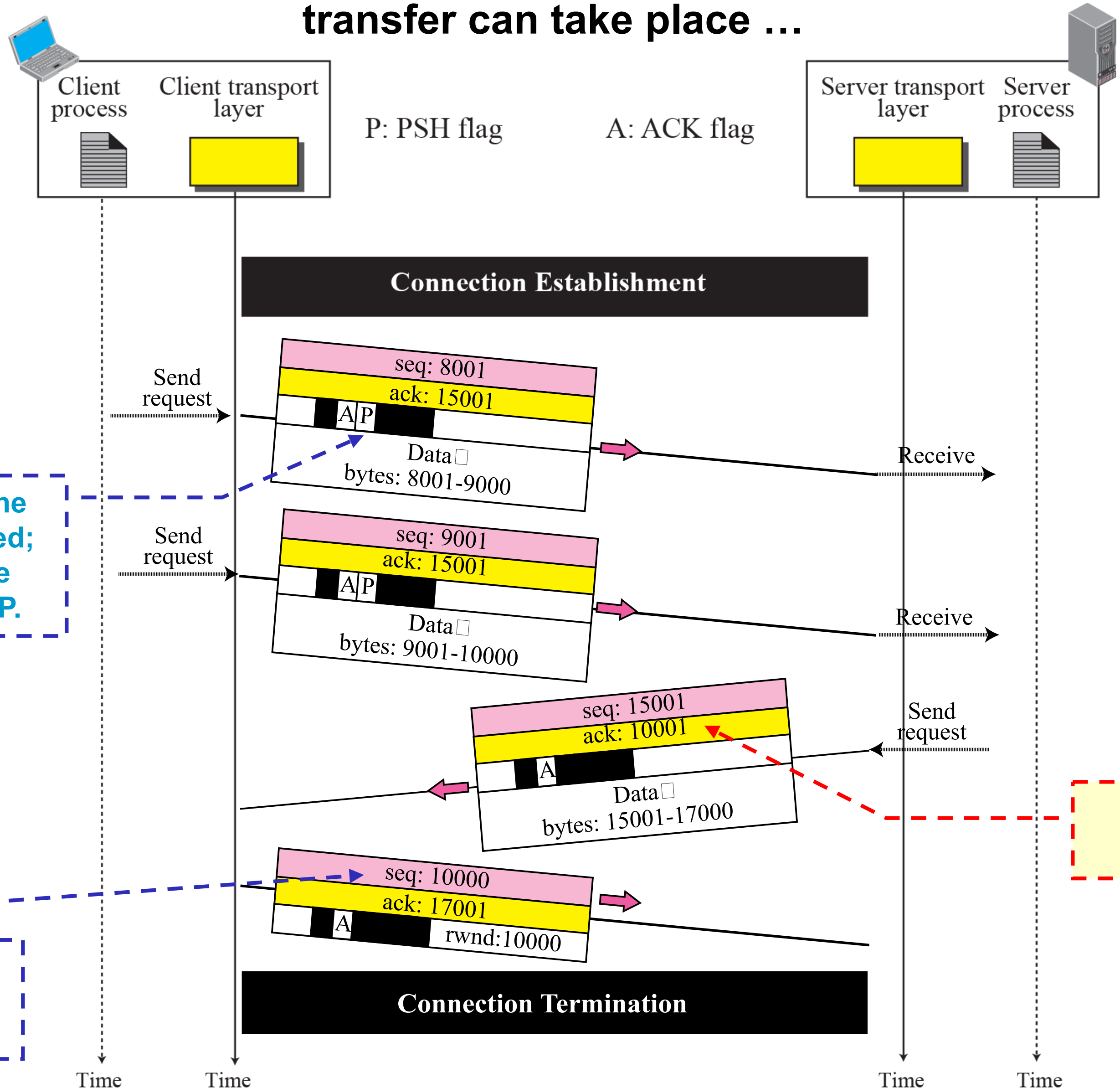
Now B is open, assuming the connection is established, but currently A does not have any data to send.
A knows nothing about what happened.

In 3-way handshake procedure A is require to send the last packet within a specified relatively short interval of time! The actual data can come (much) later.



TCP Connection Control (cont.)

TCP Data Transfer – **after connection is established, bidirectional data transfer can take place …**



Do not wait for the window to be filled; send data to the application ASAP.

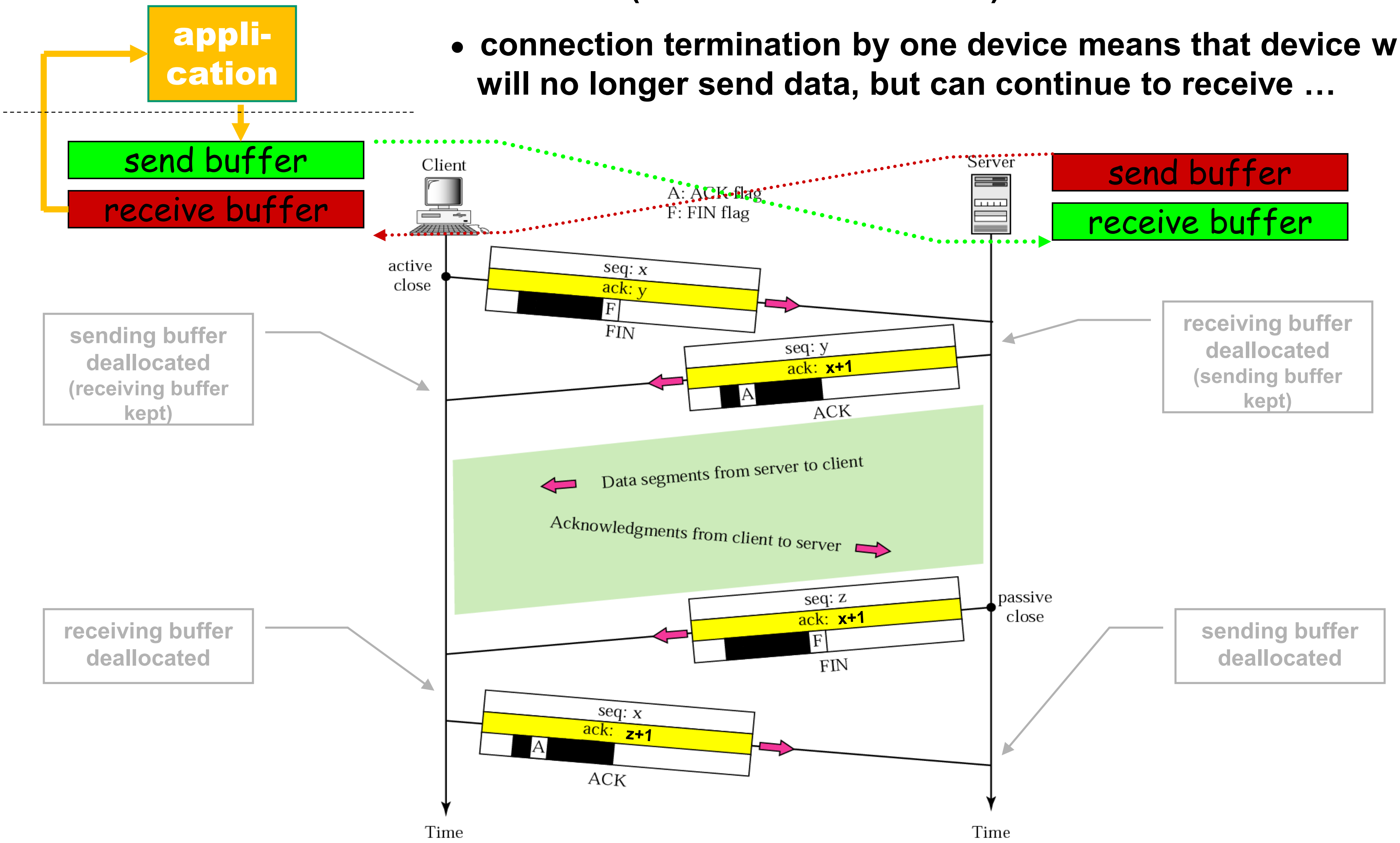
Cumulative ACK!

What is the purpose of this packet.

TCP Connection Control (cont.)

TCP Connection Termination – **TCP provides graceful close, i.e., <u>independent termination</u> of each direction of connection – when connection ends, resources (buffers and variables) are deallocated**

- **connection termination by one device means that device will will no longer send data, but can continue to receive …**



sending buffer deallocated (receiving buffer kept)

receiving buffer deallocated (sending buffer kept)

receiving buffer deallocated

sending buffer deallocated

TCP Connection Control (cont.)

“Four-Way Handshake” – Pair of Two-Way Handshakes :

(1) client sends a FIN segment to server, which includes

- **FIN = 1**
- **no application data**

(2) when server receives FIN segment, it immediately acknowledges the segment and notifies destination application about termination request

- **ACK = *client_seq + 1* – only (+1) because no user data have been sent**
- **possibly some application data**

(3) server can continue sending data to client – when it does not have any more data to send, it sends its own FIN segment

- **FIN = 1**
- **no application data**

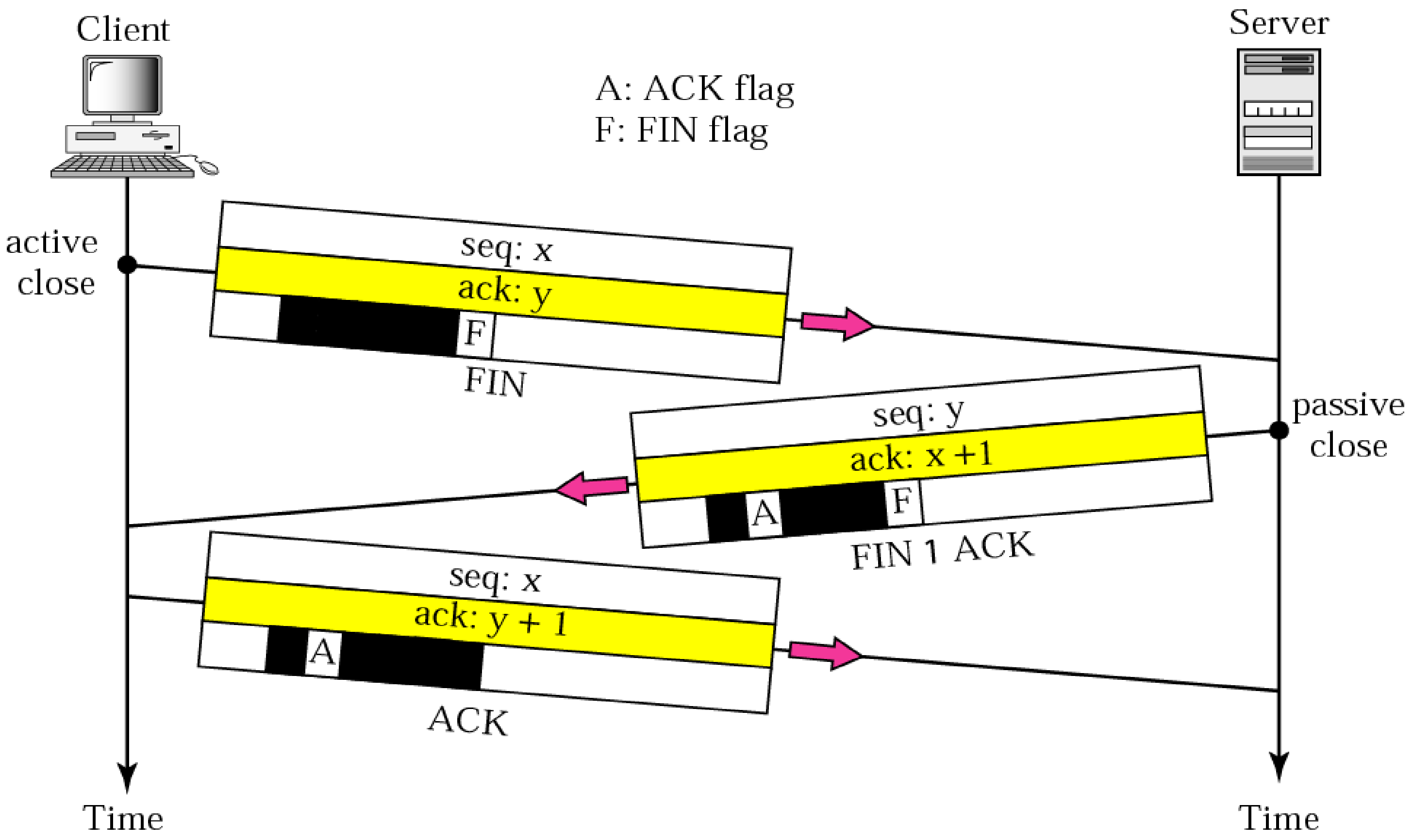
(4) client sends fourth segment to confirm receipt of FIN segment from server

- **ACK = *server_seq + 1* – only (+1) because no user data have been sent**
- **no application data**

TCP connection is full-duplex ⇒
must be explicitly closed in both directions!

TCP Connection Control (cont.)

Example **[connection termination using three-way handshake]**



TCP Connection Control (cont.)

TCP Connection Resetting – allows devices to deal with problem situations, such as half-open connection or receipt of unexpected messages

- to use this feature, device detecting the problem sends a TCP segment with RST flag set to 1
- receiving device either returns to LISTEN state (server), or closes connection and returns to CLOSED state (client)

TCP Resetting Examples

(a) Denying a Connection

The client TCP has requested a connection to a nonexistant port. The server TCP sends a segment with its RST bit set, to annul the request.

(b) Terminating an Idle Connection

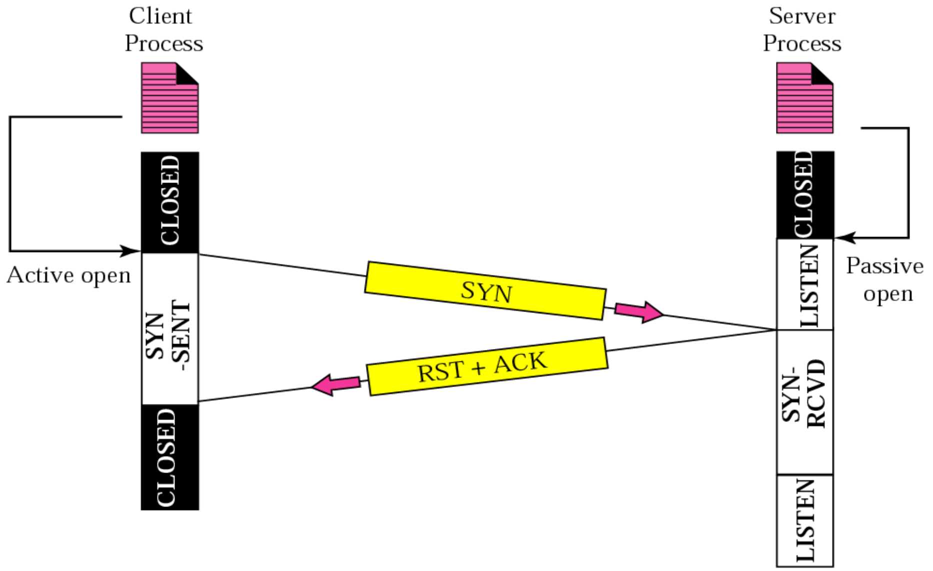
TCP on one side discovers that TCP on the other side has been idle for a long time, so it sends an RST segment to destroy the connection. (see "timers", next lecture)

(c) Aborting a Connection

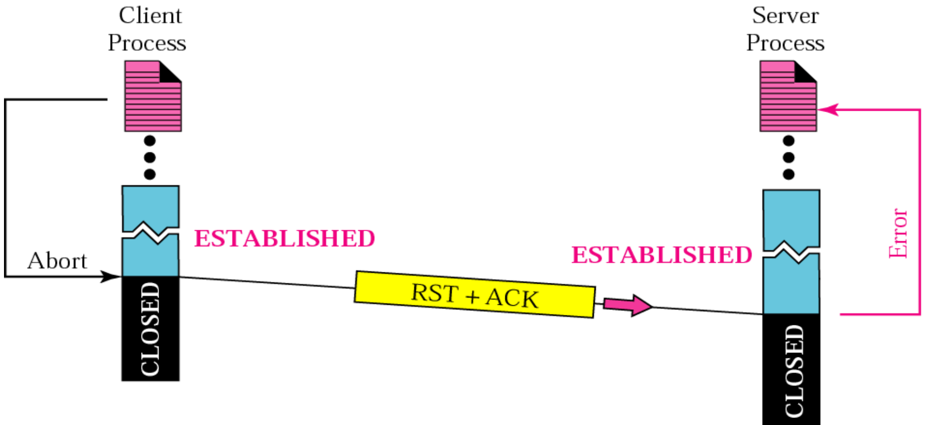
One TCP wants to abort a connection due to an abnormal situation. So, it sends an RST segment to the other TCP to close the connection.

TCP Connection Control (cont.)

Example [special case 1: denying a connection]



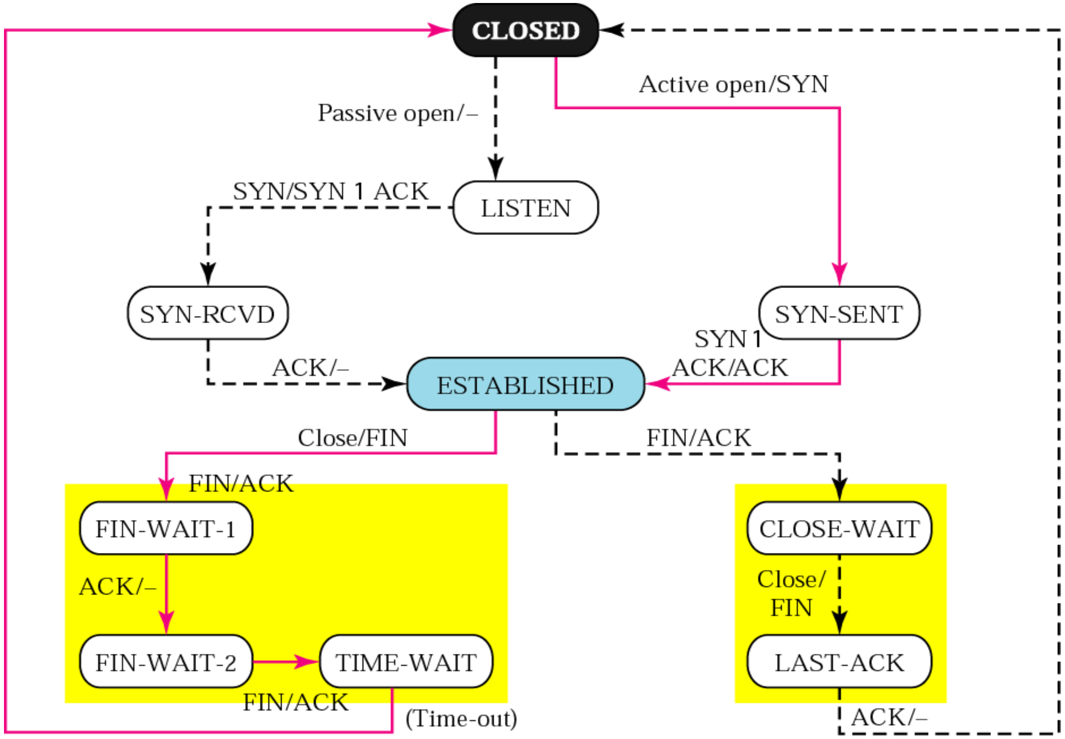
Example [special case 2: aborting a connection]



TCP Connection Control (cont.)

TCP Client/Server State Transition Diagram – states are shown using ovals, transitions are shown using directed lines

- each line has two strings separated by a slash: 1st string = input that TCP receives, 2nd string = output that TCP sends
- dotted lines = server, solid lines = client



TCP Connection Control (cont.)

State		Description
CLOSED		There is no connection.
LISTEN	S	The server is waiting for calls from the client.
SYN-SENT	C	A connection request is sent; waiting for acknowledgment.
SYN-RCVD	S	A connection request is received.
ESTABLISHED		Connection is established.
FIN-WAIT-1	C	The application has requested the closing of the connection.
FIN-WAIT-2	C	The other side has accepted the closing of the connection.
TIME-WAIT	C	Waiting for retransmitted segments to die.
CLOSE-WAIT	S	The server is waiting for the application to close.
LAST-ACK	S	The server is waiting for the last acknowledgment.

TCP Connection Control (cont.)

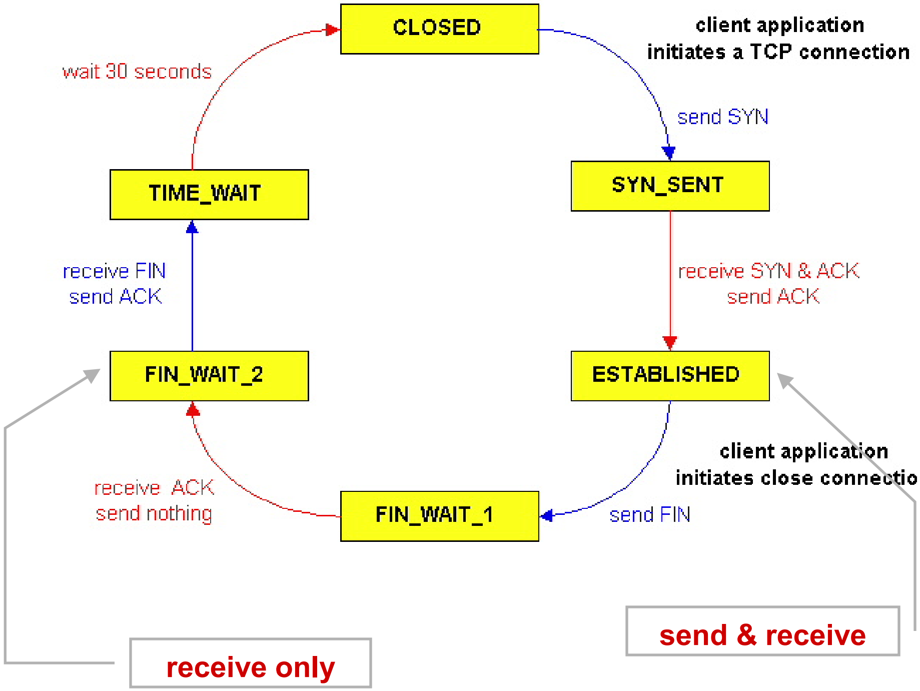
TCP Client Lifecycle

(1) TCP client starts in **CLOSED** state.

(2) While in this state, TCP client can receive an active open request from client application program. It, then, sends a SYN segment to TCP server and goes to the **SYN-SENT** state.

(3) While SYN-SENT state, TCP client can receive a SYN+ACK segment from TCP server. It, then, sends an ACK to TCP server and goes to **ESTABLISHED** (data transfer) state. TCP client remains in this state as long as it sends and receives data.

(4) While in ESTABLISHED state, TCP client can receive a close request from the client application program. It sends a FIN segment to TCP server and goes to **FIN-WAIT-1** state.

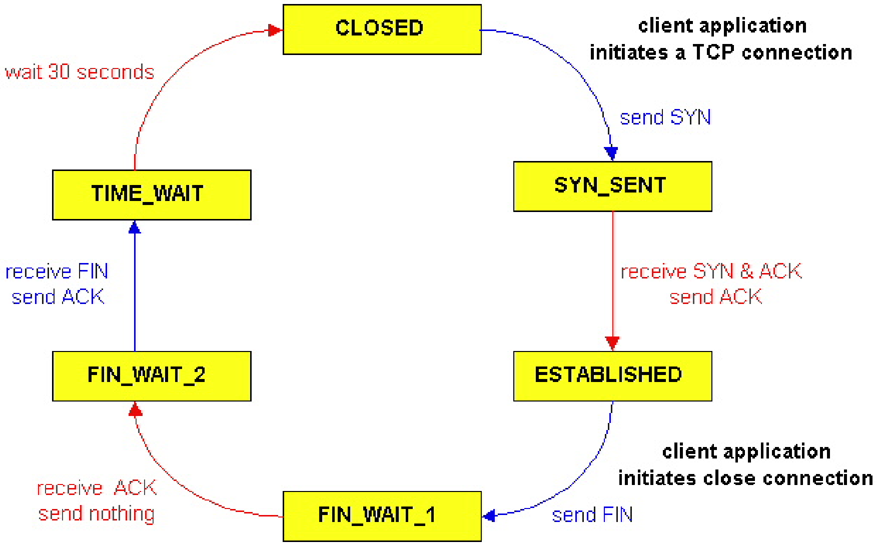


TCP Connection Control (cont.)

TCP Client Lifecycle (cont.)

(5) While in FIN-WAIT-1 state, TCP client waits to receive an ACK from TCP server. When the ACK is received, TCP client goes to FIN-WAIT-2 state. It does not send anything. Now the connection is closed in one direction.

(6) TCP client remains in FIN-WAIT-2 state, waiting for TCP sever to close the connection from its end. Once TCP client receives a FIN segment from TCP server, it sends an ACK segment and goes to the TIME-WAIT state.



(7) When in TIME-WAIT state, TCP client starts a timer and waits until the timer goes off. The TIME-WAIT timer is set twice the maximum segment lifetime (2MSL). The client remains in this state before totally closing to ensure that ACK segment it sent was received. (If another FIN arrives from TCP server, ACK segment is retransmitted and the TIME-WAIT timer is restared at 2MSL.) Also, 2MSL ensures that all segments from the old connection are cleared from the network at the end of TIME-WAIT state.

TCP Connection Control (cont.)

TCP Server Lifecycle

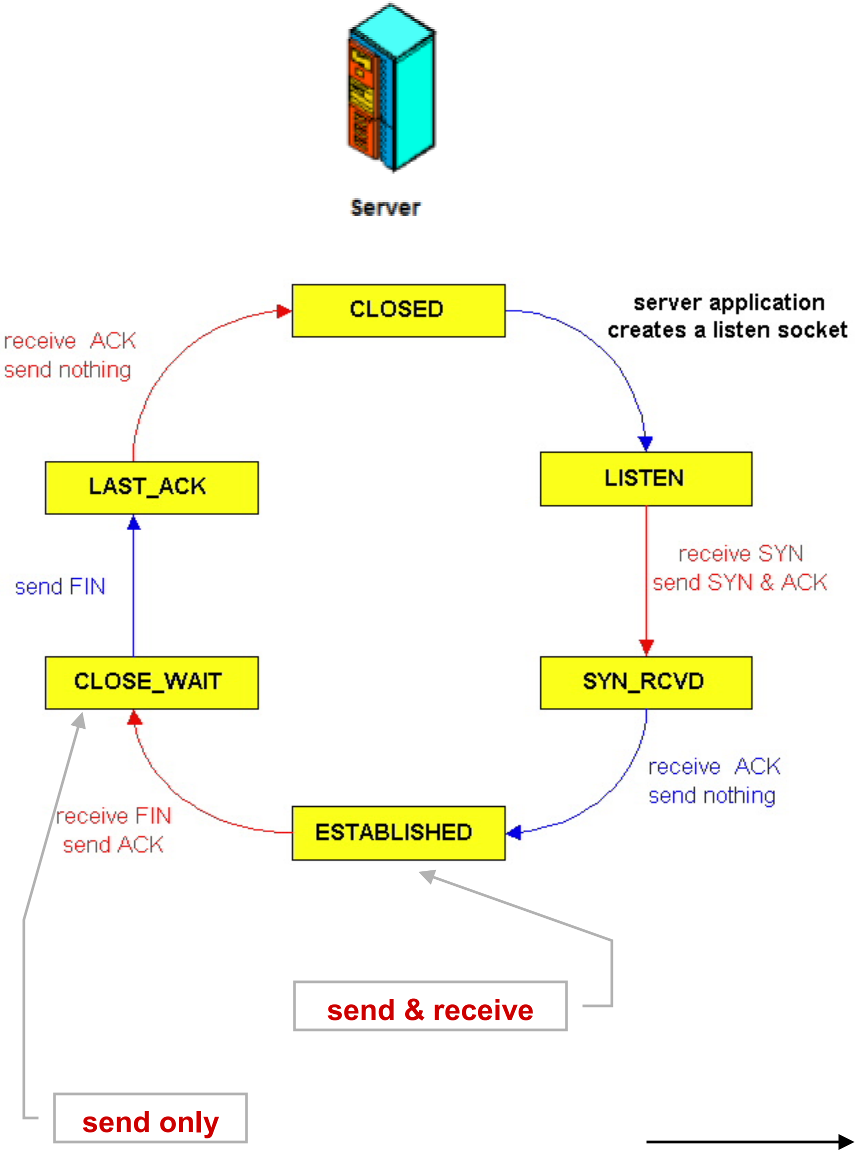
Theoretically, TCP server can by in any of the 11 states. However, it normally operates in one of the following states:

(1) TCP server starts in CLOSED state

(2) While in this state, TCP server can receive a passive open request from server application program. It, then, goes to LISTEN state.

(3) While in LISTEN state, TCP server can receive a SYN segment from TCP client. IT sends a SYN + ACK segment to TCP client and then goes to SYN-RCVD state.

(4) While in SYN-RCVD state, TCP server can receive an ACK segment from client TCP. It, then, goes to ESTABLISHED (data transfer) state. TCP client remains in this state as long as it sends and receives data



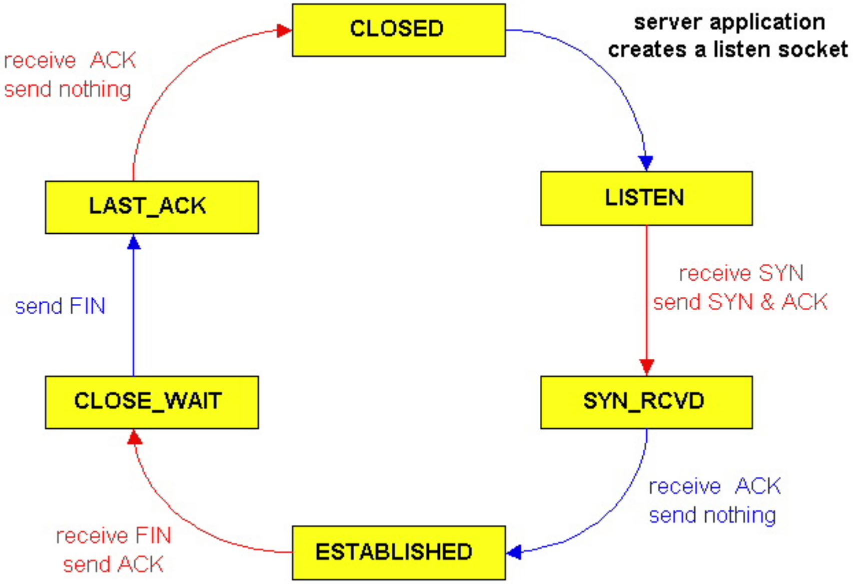
TCP Connection Control (cont.)

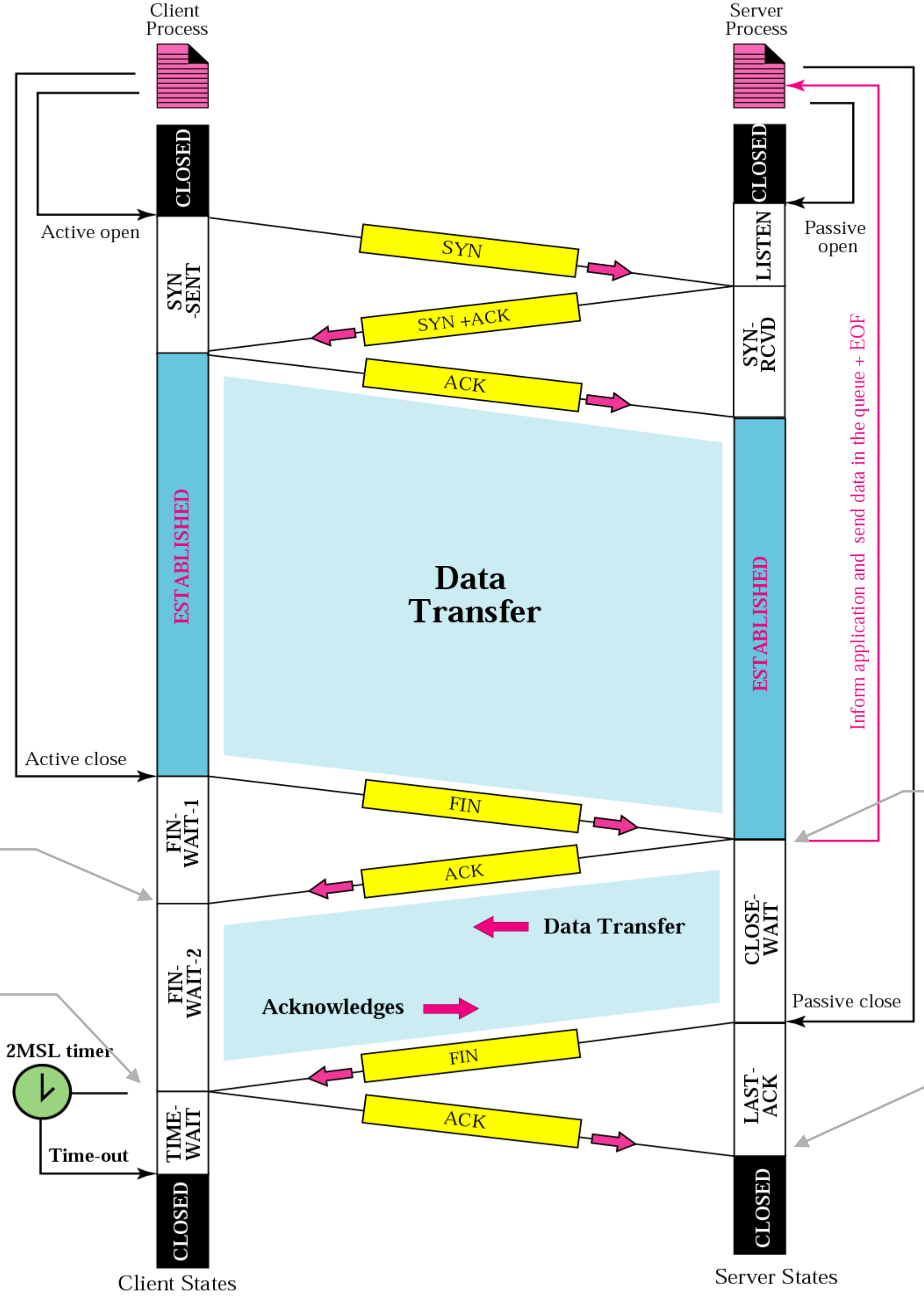
TCP Server Lifecycle (cont.)

(5) While in ESTABLISHED state, TCP server can receive a FIN segment from TCP client, which means that client wants to close the connection. TCP server then sends an ACK segment to TCP client and goes to CLOSE-WAIT state.

(6) While in CLOSE-WAIT state, TCP server waits until it receives a close request from its own server program/application. It then sends a FIN segment to TCP client and goes to LAST-ACK state.

(7) When in LAST-ACK state, TCP server waits for the last ACK segment. It then goes to CLOSED state.





receiving buffer deallocated

sending buffer deallocated

receiving buffer deallocated

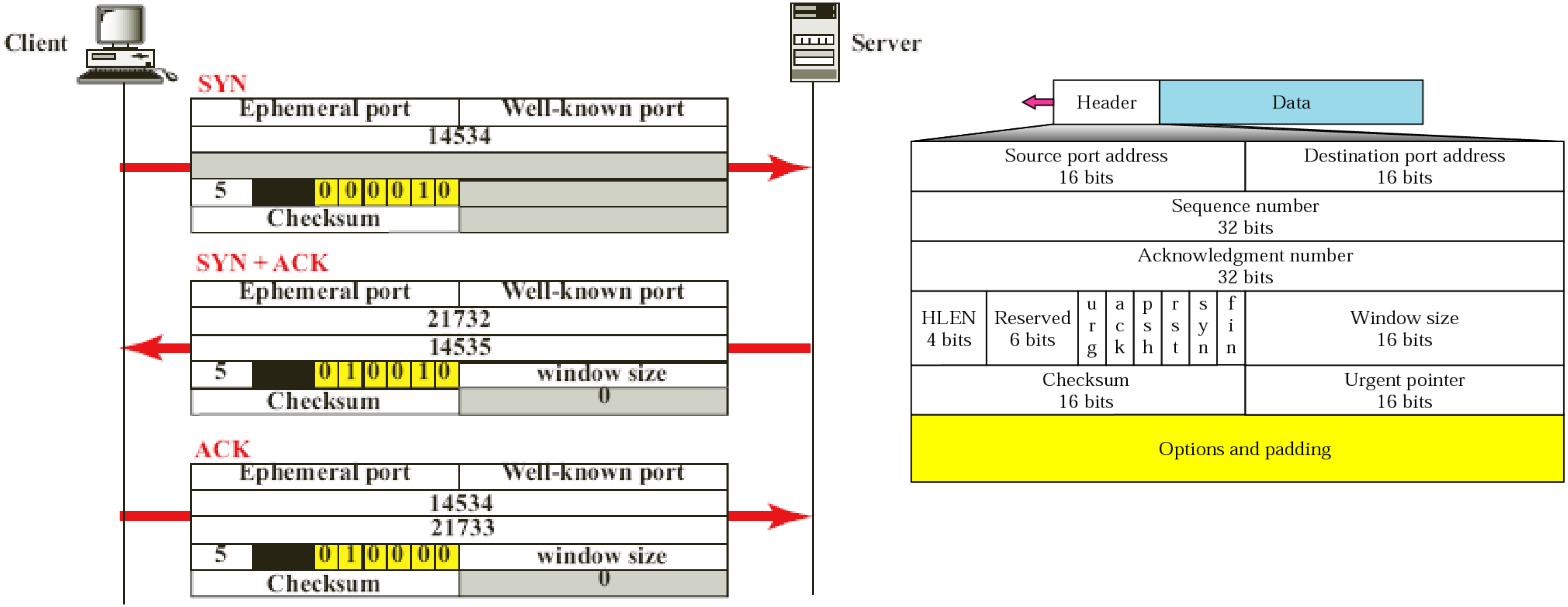
sending buffer deallocated

TIMEWAIT a long enough interval to ensure that all ACKs are received and any packets from this connection disappear from the network …

TCP Connection Control (cont.)

Example [TCP connection establishment]

TCP opens a connection using an initial sequence umber ISN of 14,534. The other party opens the connection with an ISN of 21,732. Show the Three TCP segments during the connection establishment.



Client

Server

SYN

Ephemeral port | Well-known port
14534
5 | 0 0 0 0 1 0
Checksum

SYN + ACK

Ephemeral port | Well-known port
21732
14535
5 | 0 1 0 0 1 0 | window size
Checksum | 0

ACK

Ephemeral port | Well-known port
14534
21733
5 | 0 1 0 0 0 0 | window size
Checksum | 0

Header | Data

Source port address 16 bits | Destination port address 16 bits
Sequence number 32 bits
Acknowledgment number 32 bits
HLEN 4 bits | Reserved 6 bits | u r g | a c k | p s h | r s t | s y n | f i n | Window size 16 bits
Checksum 16 bits | Urgent pointer 16 bits
Options and padding

Exercise

1. Show the entries for the header of a UDP user datagram that carries a messages from an Echo Server to an Echo Client. Fill the checksum with 0s. Choose an appropriate ephemeral port number and the correct well-known port number. The length of the data is 10 bytes.

2. TCP opens a connection using an initial sequence number (ISN) of 14534. The other party opens the connection with an ISN of 21732.
   Show the three segments during the connection establishment phase.