AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

19CSE301
COMPUTER NETWORKS
3-0-0 3

TRANSPORT LAYER

Chapter 3: Transport Layer

- Flow Control
- Connection management

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

TCP flow control

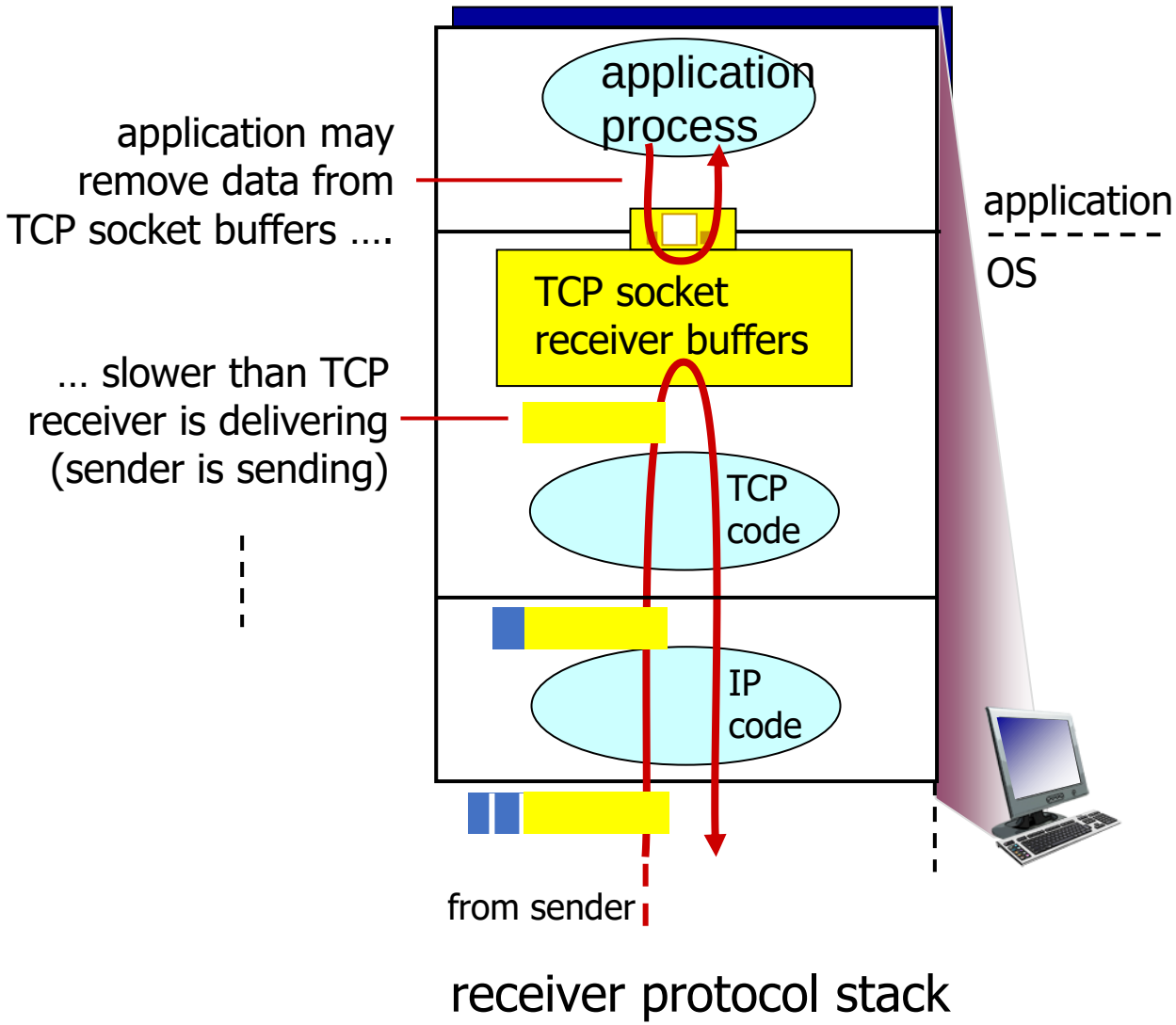
The hosts on each side of a TCP connection set aside a receive buffer for the connection. When the TCP connection receives bytes that are correct and in sequence, it places the data in the receive buffer. The associated application process will read data from this buffer, but not necessarily at the instant the data arrives. Indeed, the receiving application may be busy with some other task and may not even attempt to read the data until long after it has arrived. If the application is relatively slow at reading the data, the sender can very easily overflow the connection's receive buffer by sending too much data too quickly.

TCP provides a flow-control service to its applications to eliminate the possibility of the sender overflowing the receiver's buffer. Flow control is thus a speed-matching service—matching the rate at which the sender is sending against the rate at which the receiving application is reading

TCP flow control

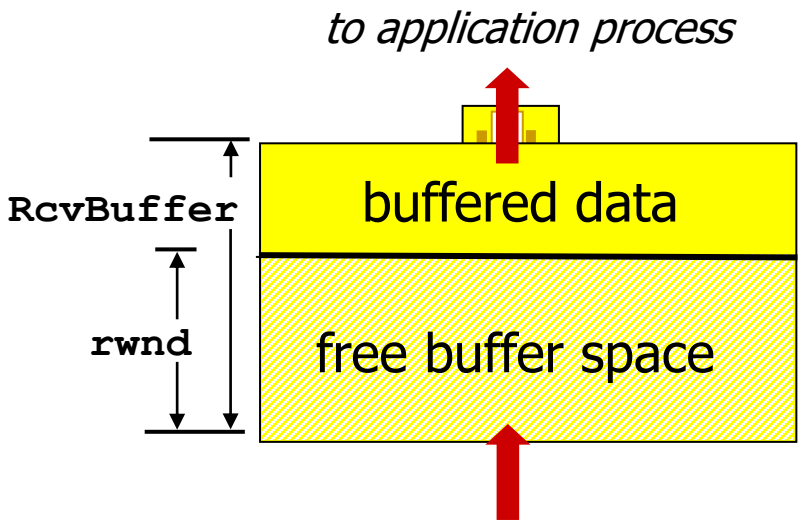
flow control

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast



TCP flow control

- receiver "advertises" free buffer space by including **`rwnd`** value in TCP header of receiver-to-sender segments
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unacked ("in-flight") data to receiver's **`rwnd`** value
- guarantees receive buffer will not overflow



receiver-side buffering

TCP flow control

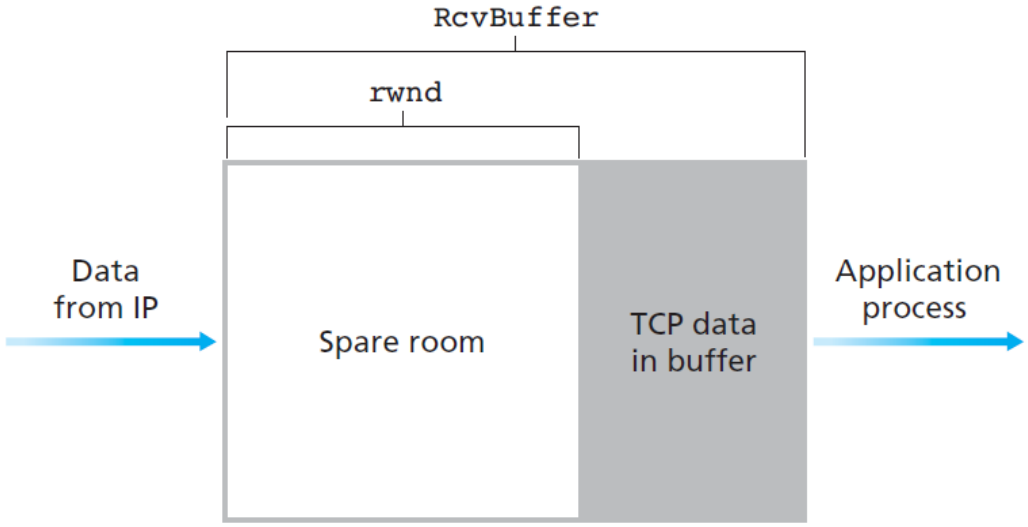
TCP provides flow control by having the *sender* maintain a variable called the **receive window.**
Suppose that Host A is sending a large file to Host B over a TCP connection. Host B allocates a **receive buffer** to this connection; denote its size by **RcvBuffer**. From time to time, the application process in Host B reads from the buffer.

- `LastByteRead`: the number of the last byte in the data stream read from the buffer by the application process in B
- `LastByteRcvd`: the number of the last byte in the data stream that has arrived from the network and has been placed in the receive buffer at B

$$\texttt{LastByteRcvd} - \texttt{LastByteRead} \leq \texttt{RcvBuffer}$$

The receive window, denoted `rwnd` is set to the amount of spare room in the buffer

$$\texttt{rwnd} = \texttt{RcvBuffer} - [\texttt{LastByteRcvd} - \texttt{LastByteRead}]$$



3.38 ♦ The receive window (`rwnd`) and the receive buffer (`RcvBuffer`)

spare room changes with time, `rwnd` is dynamic

How does the connection use the variable `rwnd` to provide the flow-control service?

TCP flow control

How does the connection use the variable `rwnd` to provide the flow-control service?

- Host B tells Host A how much spare room it has in the connection buffer by placing its current value of rwnd in the receive window field of every segment it sends to A.

Initially, Host B sets `rwnd = RcvBuffer`

- Host A in turn keeps track of two variables, `LastByteSent` and `Last-ByteAcked`, which have obvious meanings.
- Note that the difference between these two variables, `LastByteSent – LastByteAcked`, is the amount of unacknowledged data that A has sent into the connection.
- By keeping the amount of unacknowledged data less than the value of `rwnd`, Host A is assured that it is not overflowing the receive buffer at Host B.
- Thus, Host A makes sure throughout the connection's life that

$$\texttt{LastByteSent} - \texttt{LastByteAcked} \leq \texttt{rwnd}$$

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

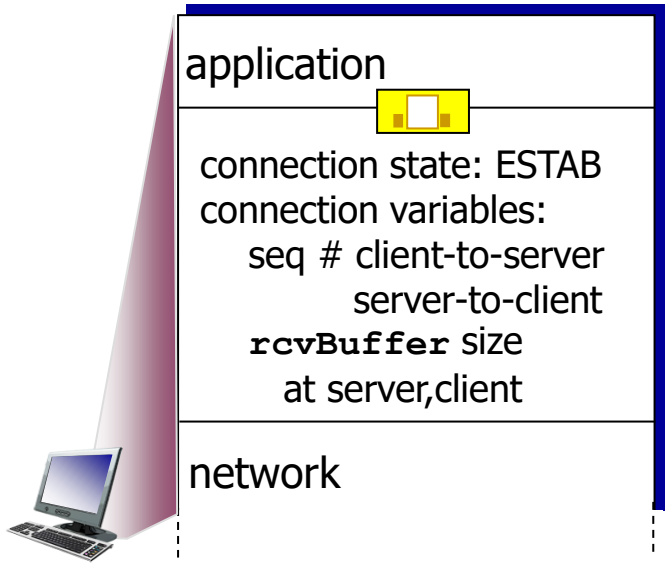
3.6 principles of congestion control

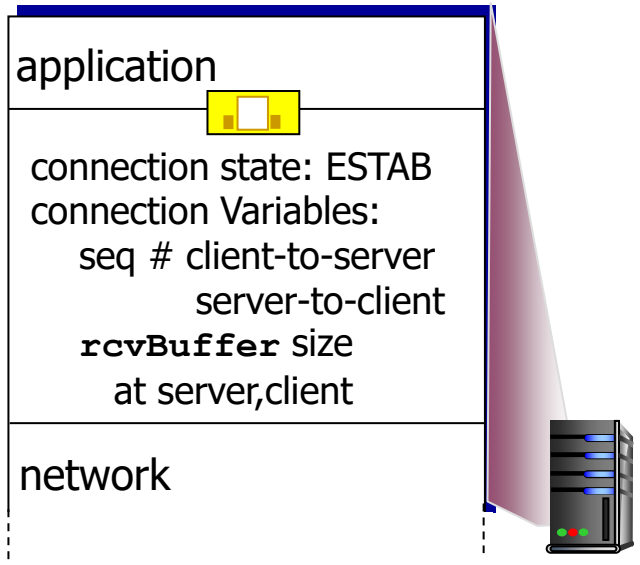
3.7 TCP congestion control

Connection Management

before exchanging data, sender/receiver “handshake”:

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters



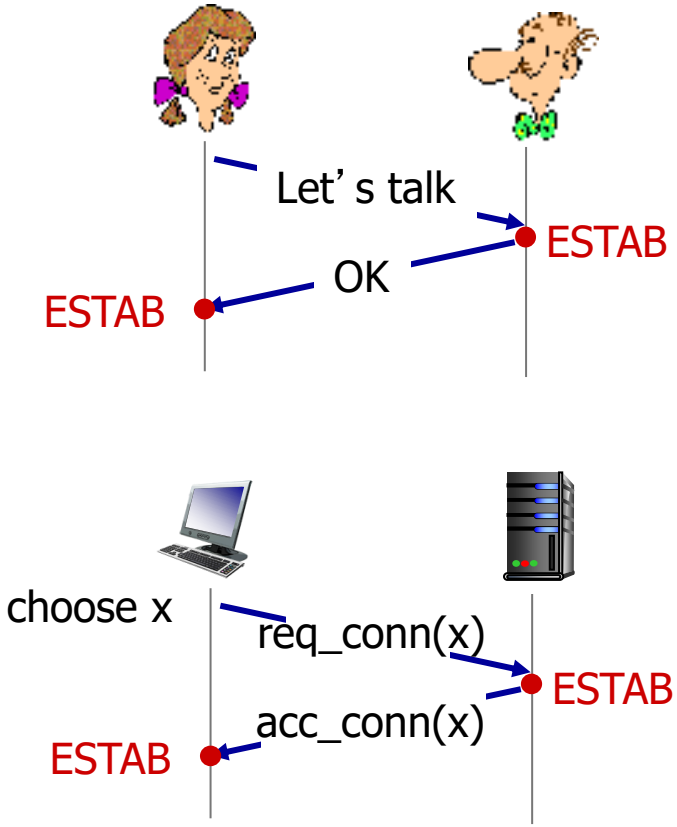



```
Socket clientSocket =
  newSocket("hostname","port
   number");
```

```
Socket connectionSocket =
   welcomeSocket.accept();
```

Agreeing to establish a connection

2-way handshake:



Q: will 2-way handshake always work in network?

- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can’t “see” other side

TCP- Three-way handshake

Step 1.

- The client-side TCP first sends a special TCP segment to the server-side TCP. This special segment contains no application-layer data. special segment is referred to as a `SYN` segment ans `SYN` bit set to 1.
- In addition, the client randomly chooses an initial sequence number (`client_isn`) and puts
- this number in the sequence number field of the initial TCP `SYN` segment. This segment is encapsulated within an IP datagram and sent to the server.

Step 2.

- The server extracts the `TCP SYN` segment. Allocates the TCP buffers and variables to the connection, and sends a connection-granted segment to the client TCP
- First, the SYN bit is set to 1. Second, the acknowledgment field of the TCP segment header is set to `client_isn+1`.
- Finally, the server chooses its own initial sequence number (`server_isn`) and puts this value in the sequence number field of the TCP segment header"I received your `SYN` packet to start a connection with your initial sequence number, client_isn. I agree to establish this connection. My own initial sequence number is `server_isn`." The connectiongranted segment is referred to as a `SYNACK` segment

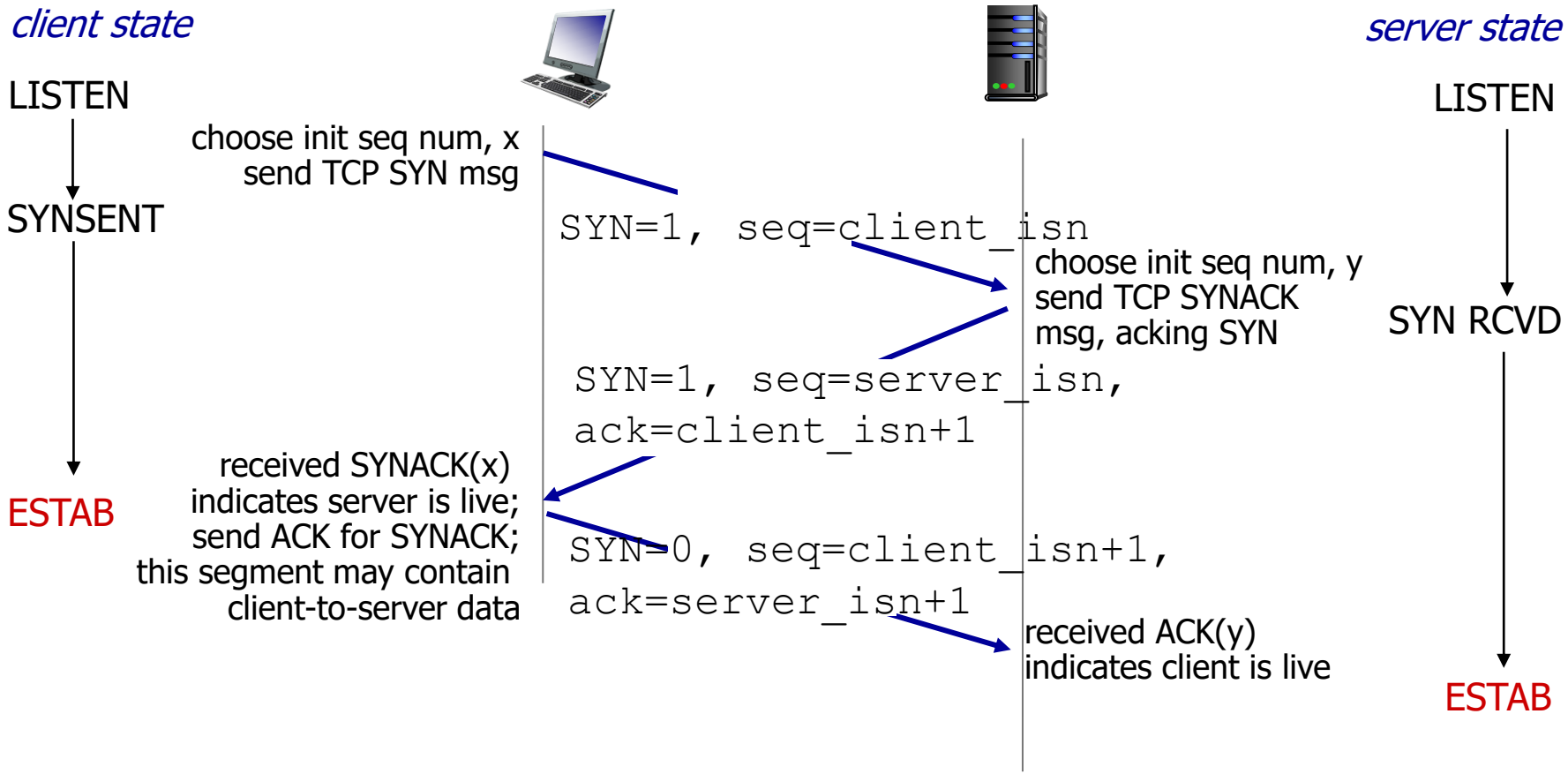
TCP- Three-way handshake

Step 3. Upon receiving the `SYNACK` segment, the client also allocates buffers and variables to the connection. The client host then sends the server yet another segment; this last segment acknowledges the server's connection-granted segment (the client does so by putting the value `server_isn+1` in the acknowledgment field of the TCP segment header). The `SYN` bit is set to zero, since the connection is established. This third stage of the three-way handshake may carry client-to-server data in the segment payload.

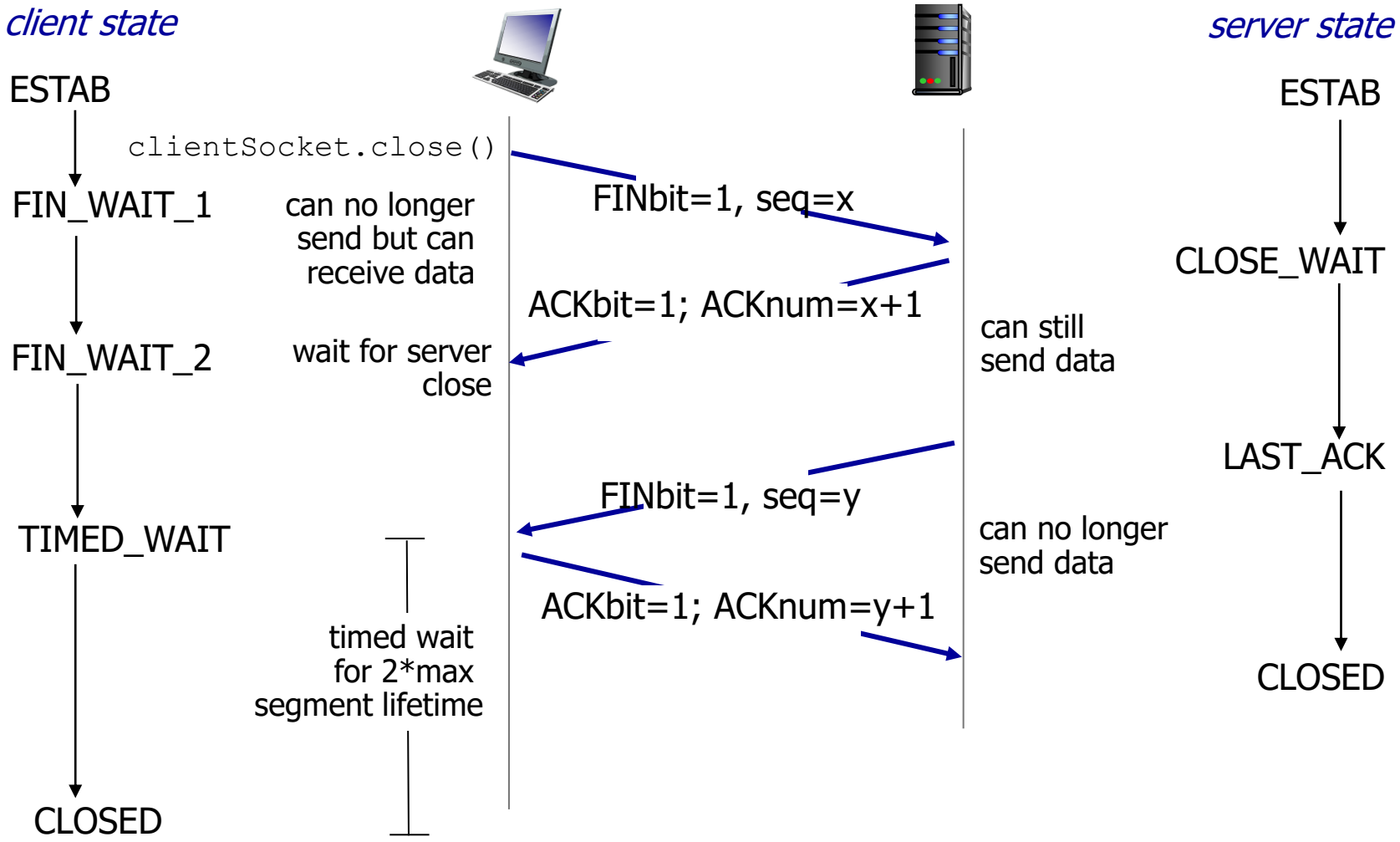
Once these three steps have been completed, the client and server hosts can send segments containing data to each other

End the connection :Either of the two processes participating in a TCP connection can end the connection. When a connection ends, the "resources" (that is, the buffers and variables) in the hosts are deallocated

TCP 3-way handshake



TCP: closing a connection



Sequence of TCP states- Client and Server

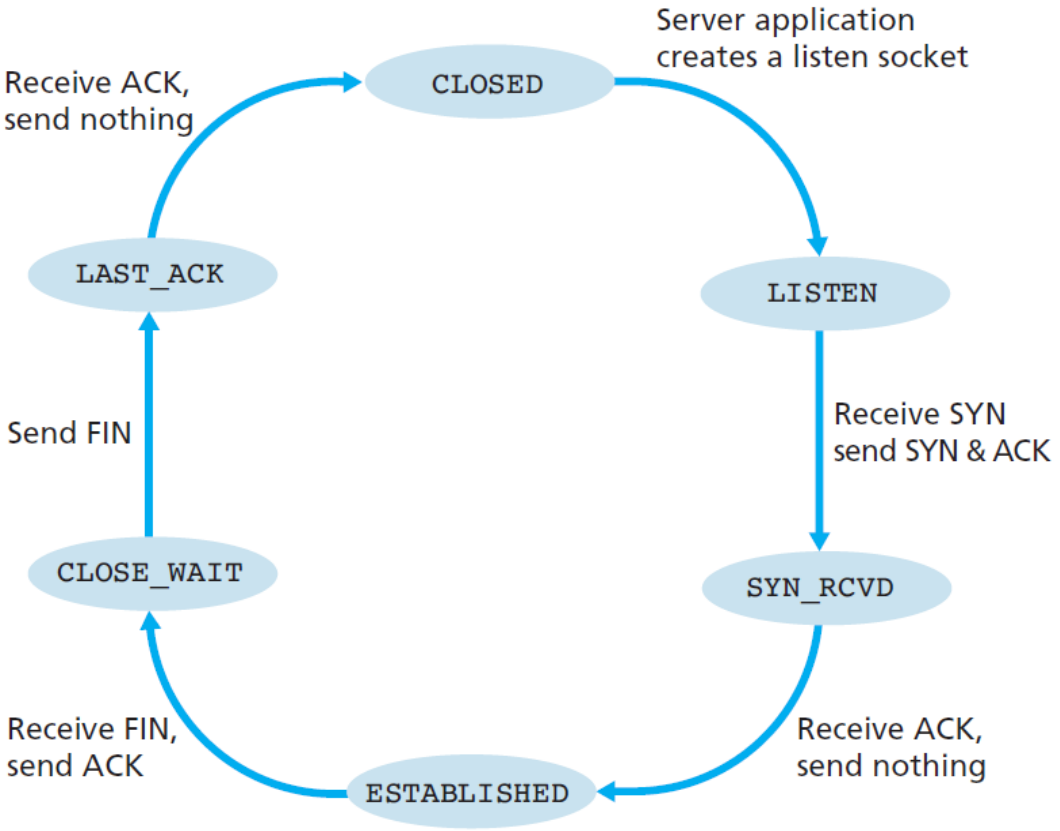


Figure 3.42 ♦ A typical sequence of TCP states visited by a server-side TCP

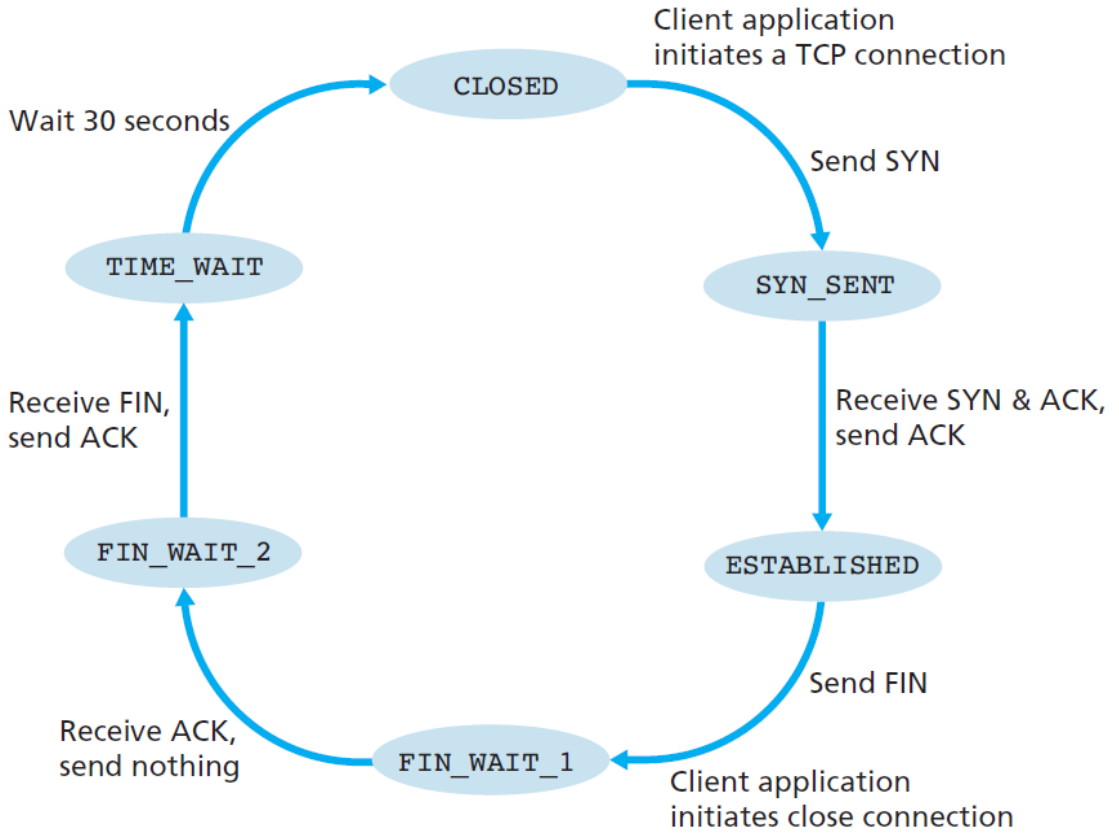
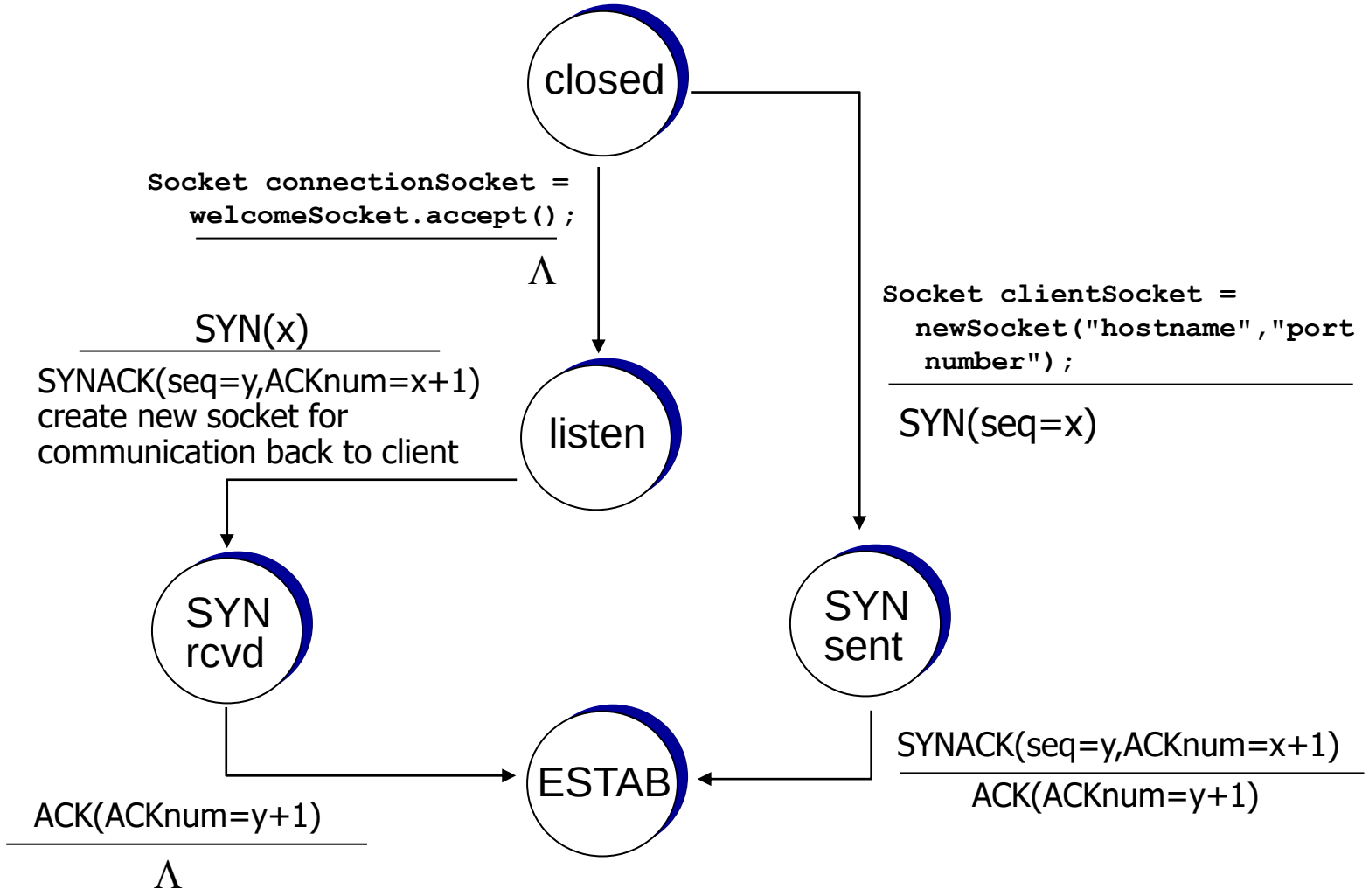


Figure 3.41 ♦ A typical sequence of TCP states visited by a client TCP

TCP 3-way handshake: FSM



TCP: closing a connection

- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

Namah Shivaya