CSC465 – Computer Networks

Dr. J. Harrison

These slides were produced almost entirely from material by Behrouz Forouzan for the text "TCP/IP Protocol Suite (2nd Edition)", McGraw Hill Publisher

---

Chapter 12

Transmission Control Protocol (TCP)

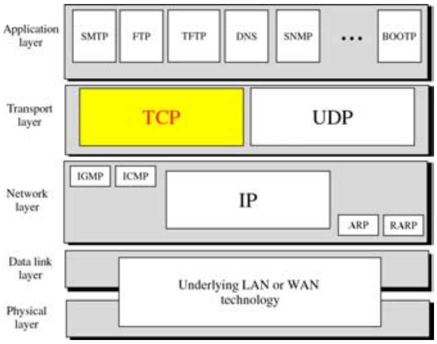
CONTENTS

- **PROCESS-TO-PROCESS COMMUNICATION**
- **TCP SERVICES**
- **NUMBERING BYTES**
- **FLOW CONTROL**
- **SILLY WINDOW SYNDROME**
- **ERROR CONTROL**
- **TCP TIMERS**

CONTENTS (continued)

- **CONGESTION CONTROL**
- **SEGMENT**
- **OPTIONS**
- **CHECKSUM**
- **CONNECTION**
- **STATE TRANSITION DIAGRAM**
- **TCP OERATION**
- **TCP PACKAGE**

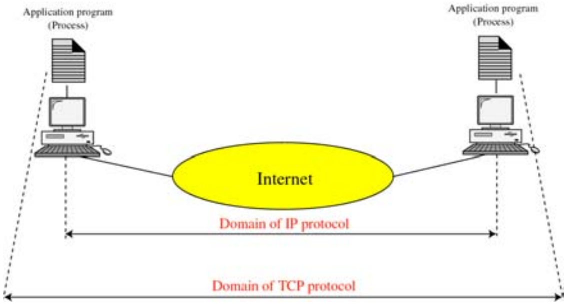
Position of TCP in TCP/IP protocol suite



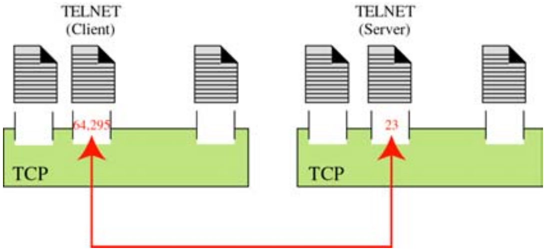
12.1

PROCESS TO PROCESS COMMUNICATION

TCP versus IP



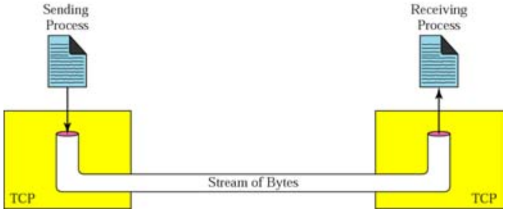
Port numbers



12.2

TCP SERVICES

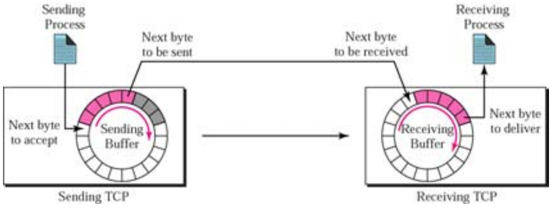
Stream delivery



TCP is a byte stream oriented protocol

TCP unlike UDP, which sends decoupled, apparently unrelated datagrams

Sending and receiving buffers

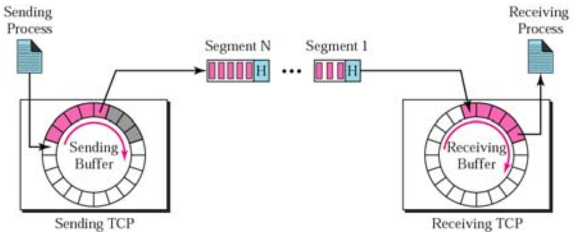


Handles disparity between speed of the producing and consuming processes

TCP Segments

- The IP layer, as a service provider for TCP, needs to send data in packets, not as stream of bytes
- TCP groups a number of bytes together into a packet called a segment
- Segment encapsulated into an IP datagram
- Each segment can be a different size
- Process transparent to sending/receiving processes

TCP segments



Other TCP Properties

- Full-Duplex Service
  - Segments can flow in both directions
  - Each TCP has sending & receiving buffers
- Connection-Oriented Service
  1. A's TCP informs B's TCP & gets approval from B
  2. A & B's TCP exchange data in both directions
  3. When both complete A & B destroy their buffers
- Reliable Service

12.3

NUMBERING BYTES

Numbering Bytes

- TCP keeps track of segments but no segment #
- Instead, *byte numbers* are retained
  - The bytes being transferred in each connection are numbered
  - Numbering starts with a random number
- After bytes numbered, sequence # assigned to each segment
- Sequence # for each segment is the number of the first byte carried in that segment

Example 1

Imagine a TCP connection is transferring a file of 6000 bytes. The first byte is numbered 10010. What are the sequence numbers for each segment if data is sent in five segments with the first four segments carrying 1,000 bytes and the last segment carrying 2,000 bytes?

Solution

The following shows the sequence number for each segment:

Segment 1 ➔ 10,010 (10,010 to 11,009)

Segment 2 ➔ 11,010 (11,010 to 12,009)

Segment 3 ➔ 12,010 (12,010 to 13,009)

Segment 4 ➔ 13,010 (13,010 to 14,009)

Segment 5 ➔ 14,010 (14,010 to 16,009)

Acknowledgement Number

- Communication in TCP is full duplex
- Each party numbers bytes, usually with a different starting byte number
- The sequence # in each direction shows the number of the first byte carried by the segment
- Each party also uses an ack # to confirm bytes it received
- Ack# indicates # of next byte that the party expects to receive

Acknowledgement Number

- Ack# is cumulative
  1. Party takes # of last byte received
  2. Increments by 1
  3. Announces this sum as ack#

Note

The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receives. The acknowledgment number is cumulative.

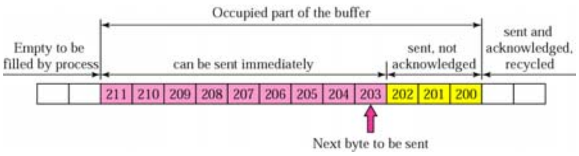
12.4

FLOW CONTROL

Flow Control

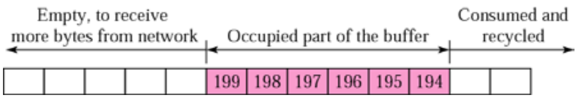
- Defines amount of data a source can send before receiving an acknowledgement from the destination
- TCP uses a ***sliding window*** to make transmission more efficient as well as to control the flow of data so that the destination does not become overwhelmed with data.
- Windows marks what data is to be sent, has been sent but not ack'ed, and what data has been sent and ack'ed

Sender buffer



- "circular" buffer shown "flat"
- Sender could sent everything up to and including byte 211 but this could overflow the receiver and force retransmission
- The sender must adjust itself to the number of locations available at the receiver

Receiver window



- Next byte to be consuming by destination process is 194
- Receiver expects byte 200 (was sent but not received)
- How many more bytes can receiver store? 7 (13 – 6)
- Receiver Window is 7

Sender buffer and sender window

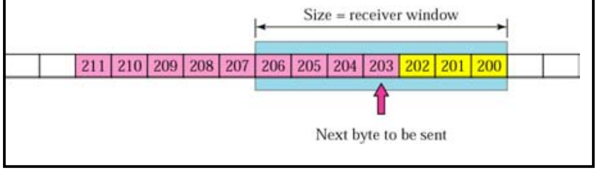
Flow control is achieved if the sender creates a window with a size less than or equal to the receiver window

Sender windows contains bytes sent and not ack'ed and those that can be sent

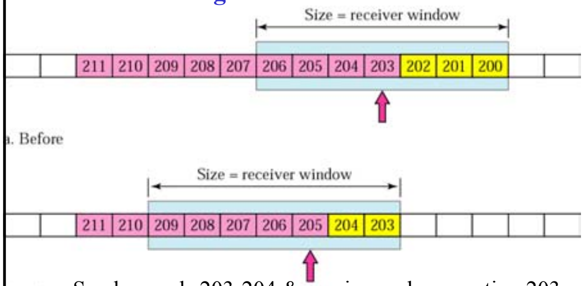
Ex: Sender Windows size = Receiver Window size = 7

Only 4 more bytes can be sent (203-206) because 3 sent already

207-211 cannot be considered for sending until news from recvr.



Sliding the sender window



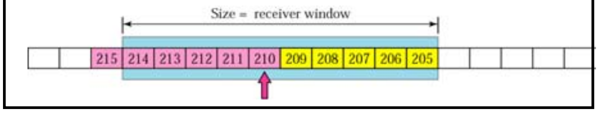
Sender sends 203-204 & receives ack requesting 203

Sender Window can "slide";

Buffer positions 200-202 recycle

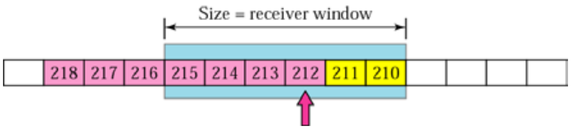
Expanding the sender window

- If the receiving process consumes faster than it receives, size of receiving window expands
- Situation can be relayed to sender
- Sender can expand its window size
- Here receiver has ack'ed 203-204 and set win size = 10
- Sender has sent 205-209 and added 212-215 and set window size to 10 like receiver



Shrinking the sender window

- If receiving process consumes data slower than it receives, receiving windows shrinks
- Received informs sender to reduce Sender Window Size
- If receiver had buffer size = 10, received 5 (205-209) and consumed 1 (205), Receiver Window now 6
- Note sending process produced 3 (216-218)



Closing Sender Window

- Occurs when receiver buffer totally full
- Receiver Window size is 0
- Relayed to sender who closes window
  - Left and right window boundaries overlap
- Sender cannot send until receiver announces a nonzero window value

In TCP, the sender window size is totally controlled by the receiver window value. However, the actual window size can be smaller if there is congestion in the network.

Some Points about TCP's Sliding Windows:

1. ***The source does not have to send a full window's worth of data.***
2. ***The size of the window can be increased or decreased by the destination.***
3. ***The destination can send an acknowledgment at any time.***

12.5

SILLY WINDOW SYNDROME

Silly Window Syndrome

- Problems can occur when either the sending process sends slowly or the receiving process consumes slowly
- Each situation results in small segment sizes, which can reduce efficiency
- If data only 1 byte, IP header (20) + TCP header (20) so segment 41 bytes.
- This inefficient network usage is called: ***Silly Window Syndrome***

Nagle's Algorithm

1. Sending TCP sends first piece of data receiving from sending process even if only 1 byte
2. Sending TCP then accumulates data in output buffer until either:
    1. Receiving TCP sends acknowledgement OR
    2. Enough data has accumulated to fill a max size segment
3. Step 2 repeats. Segment 3 must be sent when *ack* is received by sender for Segment 2

Syndrome Created by Receiver

- When receiving process consumes too slowly
- Clark's Solution: Receiver sends ack when data arrives but announces a window size of 0 until:
    - enough space to accommodate segment of max size
    - OR half of buffer is empty
- Another solution: *Delayed Acknowledgement*
    - Segment not ack'ed immediately
    - Delay stops sender from sliding its window
    - After data in window is sent, sender stops
    - Dis:May force sender to retransmit if delay is too long

---

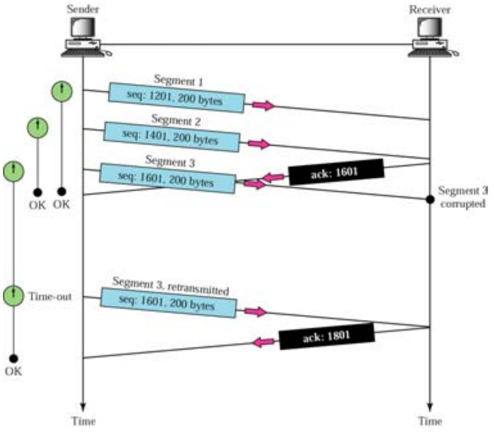
12.6

ERROR CONTROL

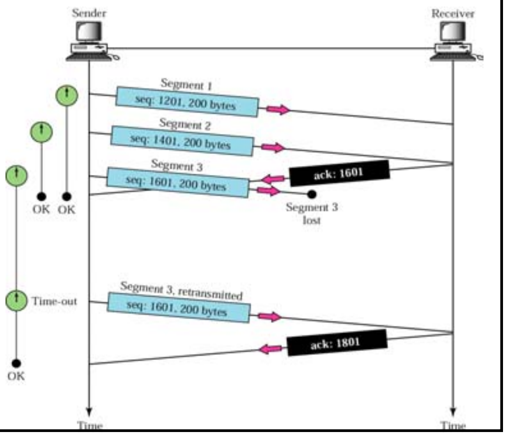
Error Control

- TCP must address transmission errors
- Errors must be corrected after detection
- TCP uses three simple tools:
  1. Checksum (checksum field)
  2. Acknowledgement (no *negative ack*)
  3. Time-out (no ack by timeout implies corrupt or lost)

Example: Corrupted Segment



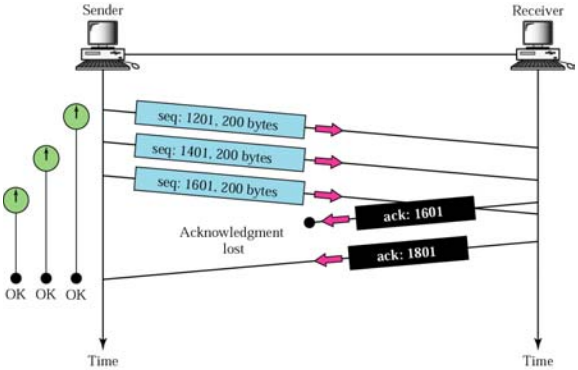
Lost Segment (same as corrupt)



More Error Control

- Duplicate Segment
  - Can occur when an ack not received by sender before timeout expires
  - When a packet arrives with same *seq#* as already received segment, destination TCP discards packet
- Out-of-Order Segment
  - Segments can arrive out of order (IP is used)
  - Out-of-order segment not ack'ed until all previous segments are received
  - Can't delay ack too long or retransmission will occur

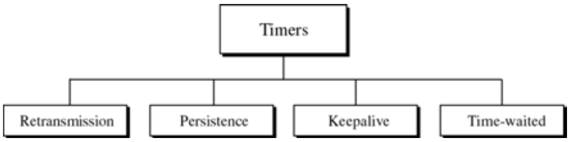
Lost acknowledgment



12.7

TCP TIMERS

TCP timers



Retransmission Timer

- For lost or discarded segment, TCP employs a transmission timer
- Measures waiting time for an ack of a segment
- When TCP sends a segment, time is created for that segment
- If an ack is received for the segment before timer expires, timer is destroyed
- If timer expires before ack arrives, the segment is retransmitted and timer is reset

Retransmission Timer

- Different connections require different retransmission time settings
- If the retransmission time is set too short, acks will not have time to return & segments will be prematurely retransmitted
- If the retransmission time is set too long, sending process will wait unnecessarily for retransmissions to occur
- Retransmission times should not be fixed even for one connection due to changing traffic levels

Persistence Timer

- Addresses zero (0) window size advertisement
- Sender will stop sending until ack received from destination TCP
- If ack gets lost, destination TCP will wait indefinitely for more data from the sender
- This deadlock situation must be avoided
- After persistence timer elapses, sender sends a probe segment (only 1 byte)
- Probe alerts destination TCP that ack was lost and must be resent

Keepalive Timer

- Implemented in some TCP servers
- Prevents a long idle connection between two connected TCP implementations
- Timer is typically set at 2 hours
- After timer elapses, 10 “probe” segments are rapidly sent
- If no response after 10 probes, it is assumed that the client is down so connection is terminated

Time-Waited Timer

- Used during connection termination
- Keeps connection alive long enough for any remaining FIN segments to arrive (which are then discarded)

12.8

CONGESTION CONTROL

TCP assumes that the cause of a lost segment is due to congestion in the network.

If the cause of the lost segment is congestion, retransmission of the segment not only does not remove the cause, it aggravates it.

Multiplicative decrease

Assume maximum window size is 32 segments

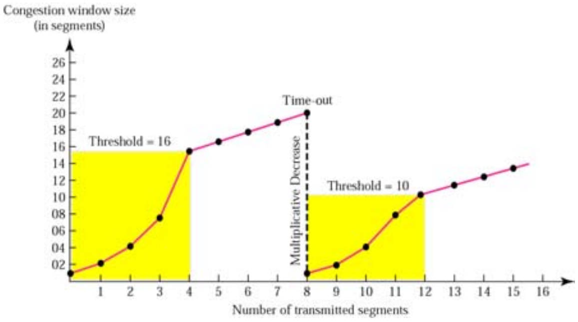
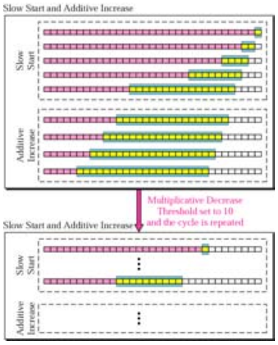


Figure 12-18

Congestion avoidance strategies



12.9

SEGMENT

Figure 12-19

TCP segment format

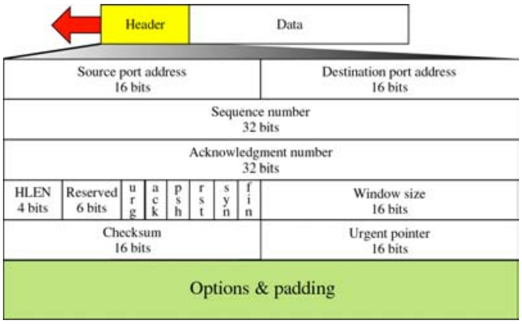
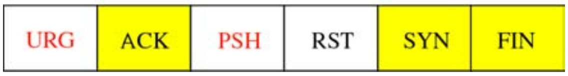


Figure 12-20

Control field

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection



12.10

OPTIONS

Figure 12-21

Options

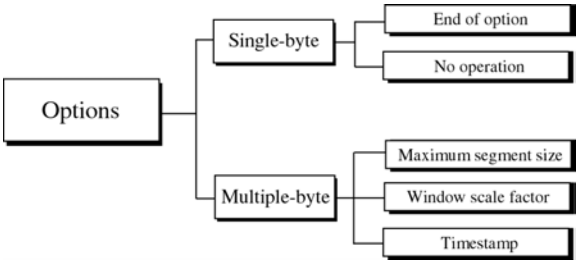
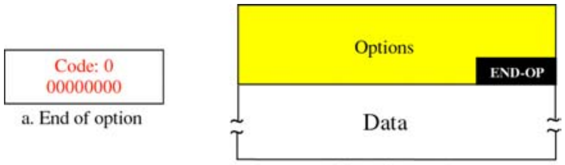


Figure 12-22

End of option option

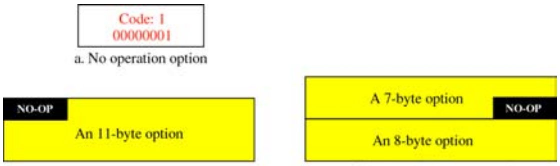


a. End of option

b. Used for padding

Figure 12-23

No operation option



a. No operation option

b. Used to align beginning of an option

c. Used to align the next option

Figure 12-24

Maximum segment size option

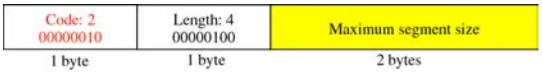


Figure 12-25

Window scale factor option

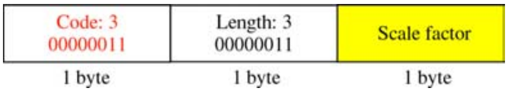
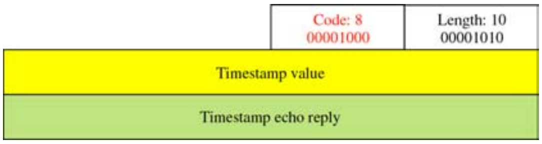


Figure 12-26

Timestamp option

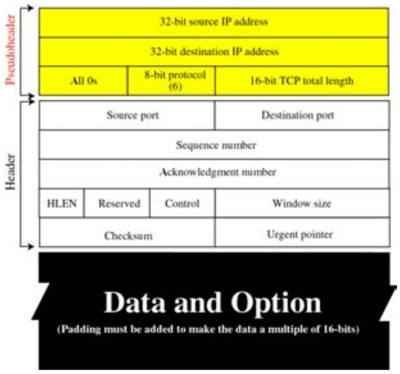


12.11

CHECKSUM

Figure 12-12

Pseudoheader added to the TCP datagram



12.12

CONNECTION

Figure 12-28

Three-way handshaking

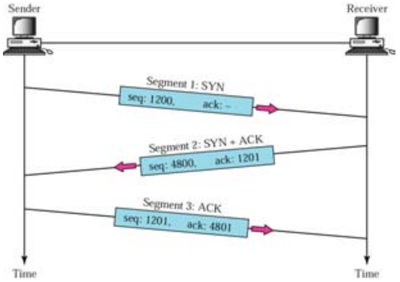
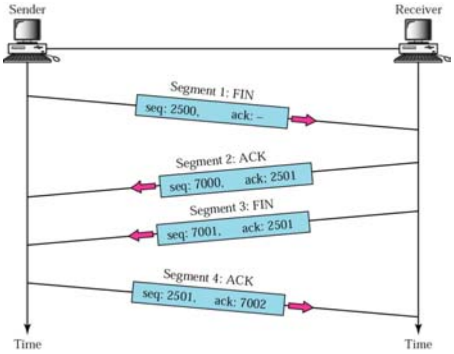


Figure 12-29

Four-way handshaking



12.13

STATE TRANSITION DIAGRAM

Figure 12-30

State transition diagram

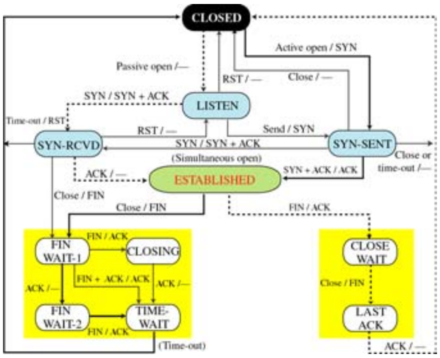


Figure 12-31

Client states

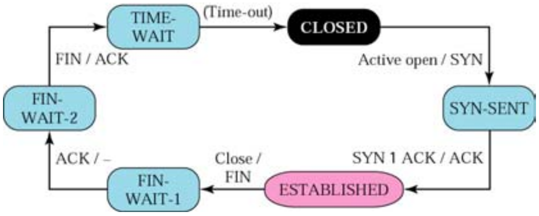
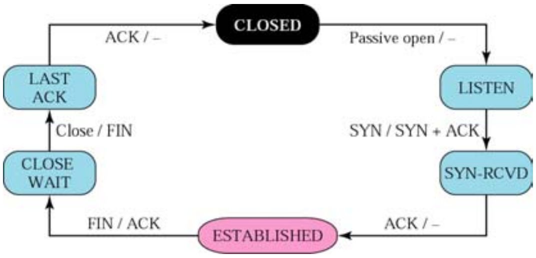


Figure 12-32

Server states



12.14

TCP OPERATION

Figure 12-33

Encapsulation and decapsulation

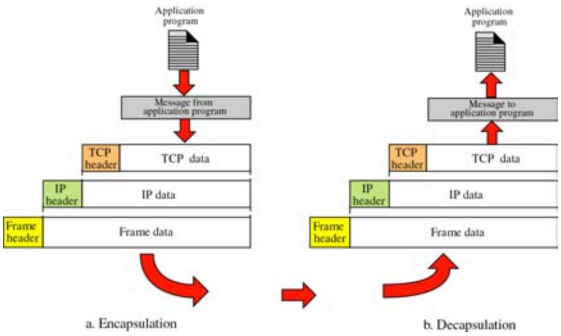
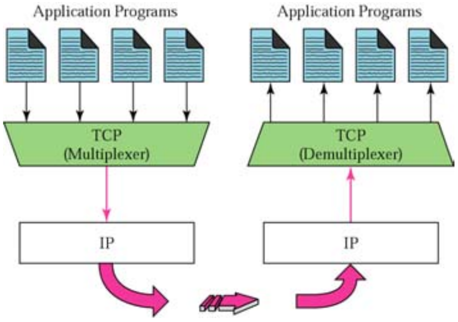


Figure 12-34

Multiplexing and demultiplexing



12.15

TCP PACKAGE

Figure 12-35

TCP package

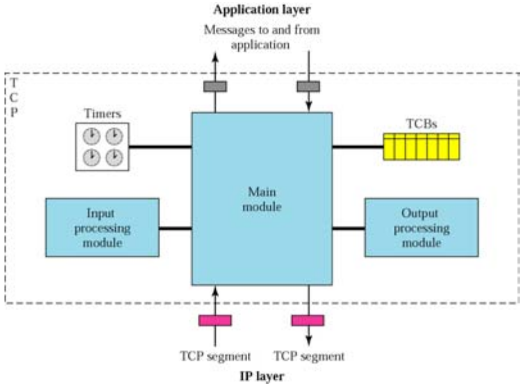


Figure 12-36

TCBs

Transmission control blocks