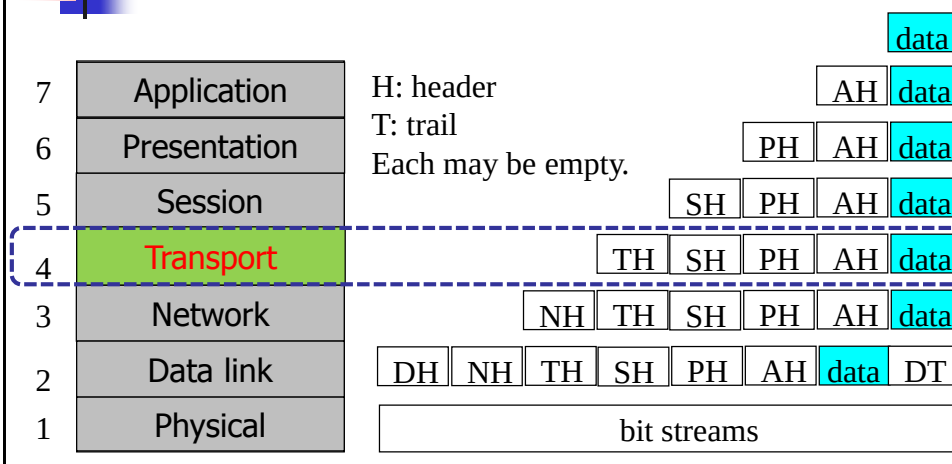
Chapter 6
The Transport Layer

陳瑞奇(Rikki)
亞洲大學資訊工程學系

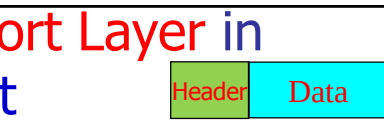
Adapted from Computer Networks,
Andrew S. Tanenbaum, Vrije University, Netherlands
& Computer Networking: A Top Down Approach,
Jim Kurose, Keith Ross

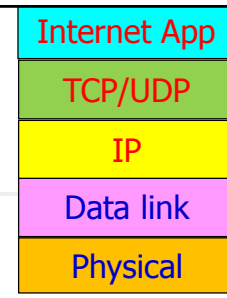
The Transport Layer

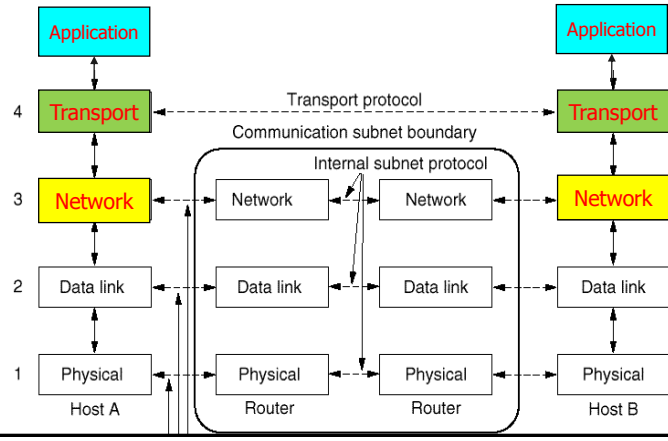


OSI Reference Model

The Transport Layer in the Internet

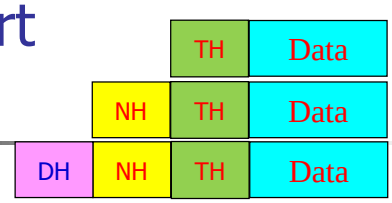


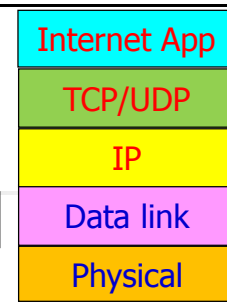


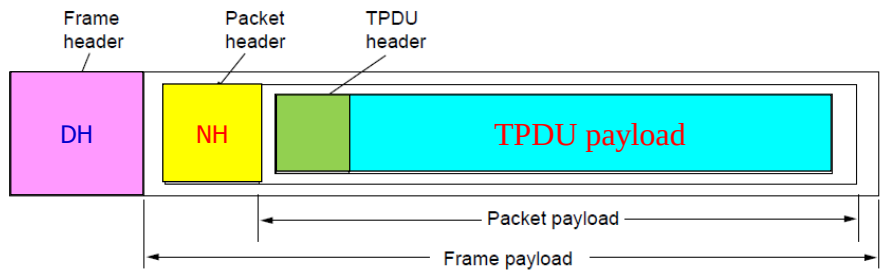


3

Transport Service





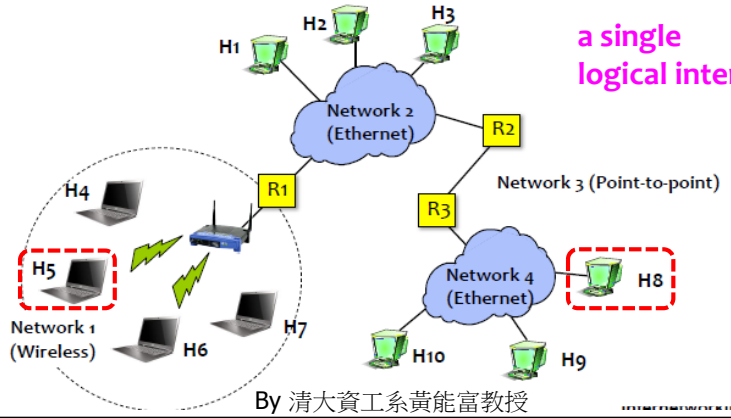


Nesting of TPDUs, packets, and frames.

4

Internetworking

- An arbitrary collection of networks interconnected to provide some sort of **host-to-host** packet delivery service **(host-to-host transmission)**



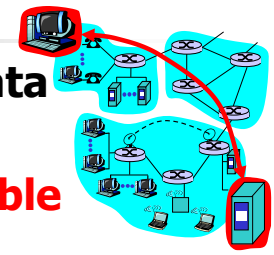
a single logical internetwork

By 清大資工系黃能富教授

5

IP Packet Delivery Model

- **Connectionless model for data delivery**
- **Best-effort delivery (unreliable service)**
  - packets are lost
  - packets are delivered out of order
  - duplicate copies of a packet are delivered
  - packets can be delayed for a long time



Computer Networking, Jim Kurose, Keith Ross

6

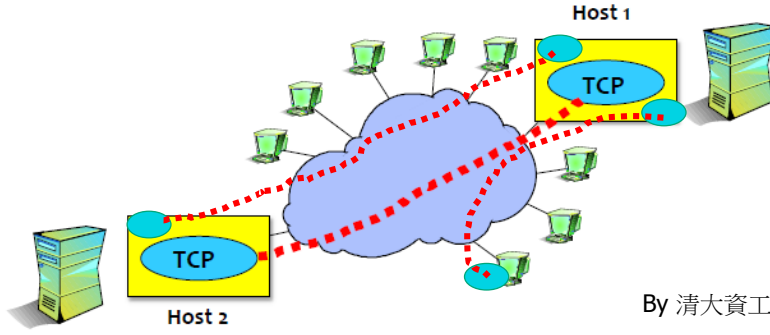
Process-to-process Protocols

- Challenge for Transport Protocols
- **Develop algorithms that turn the unreliable service of the underlying network into the service required by application programs**
  - ***Unreliable service*** ➔ ***Unreliable* service (UDP)**
  - ***Unreliable service*** ➔ ***Reliable* service (TCP)**

7

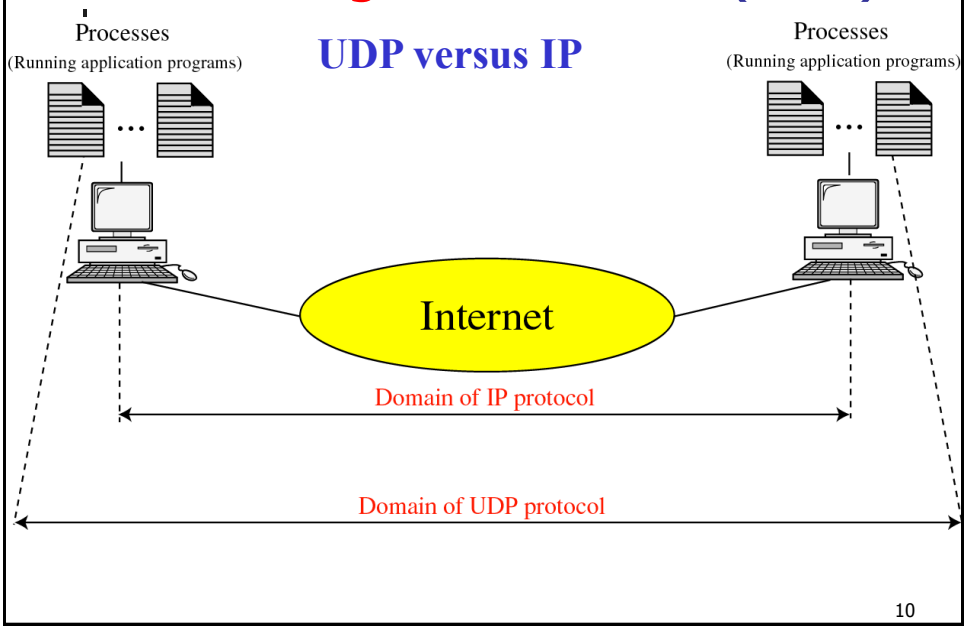
Description of Layers

- Transport Layer (提供不同主機 processes 之間的資料傳送)
  - Implements a process-to-process channel
  - Unit of data exchanges in this layer is called a *message* **segment**
  - *UDP (User Datagram Protocol) – Unreliable service*
  - *TCP (Transmission Control Protocol) – Reliable service*

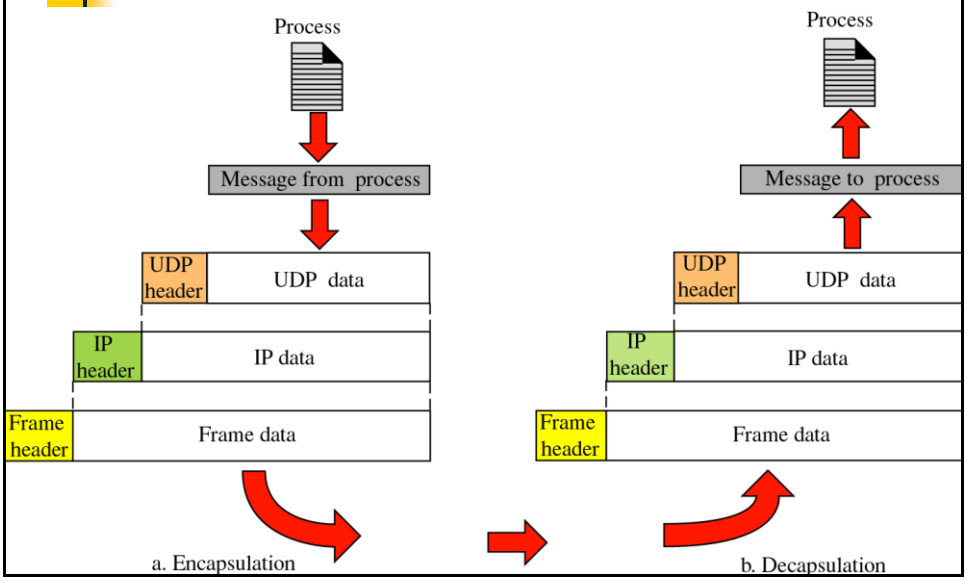


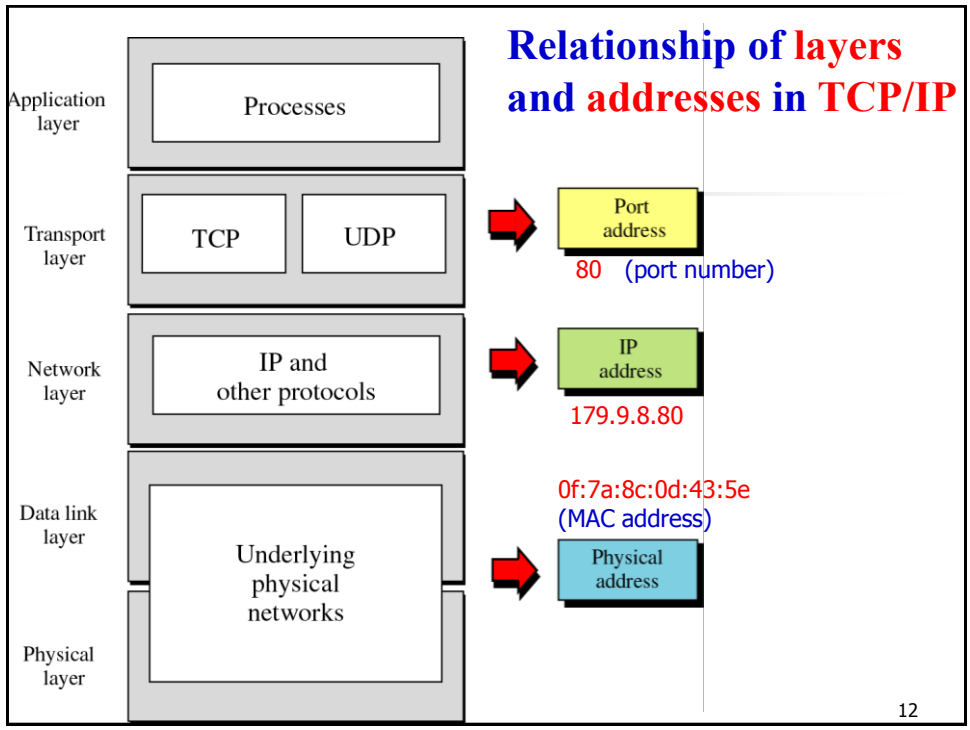
By 清大資工系黃能富教授

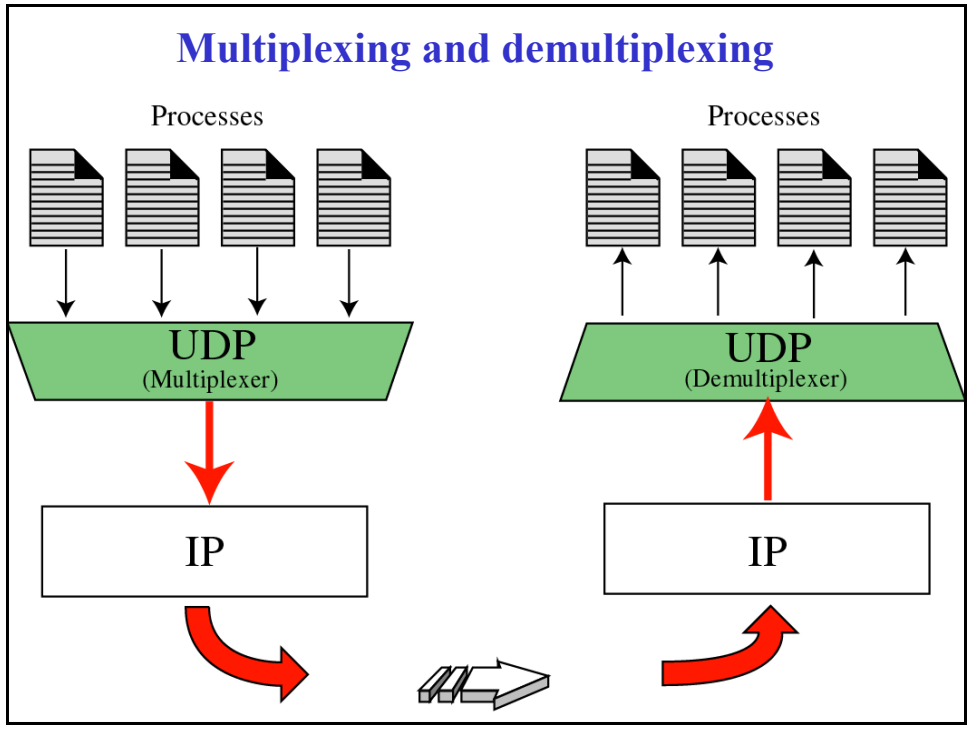
User Datagram Protocol (UDP)



Encapsulation and decapsulation

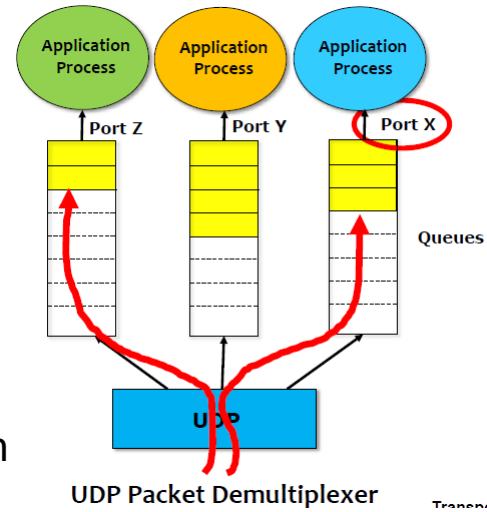



Relationship of layers and addresses in TCP/IP
Application layer
Processes
Transport layer
TCP
UDP
Network layer
IP and other protocols
Data link layer
Underlying physical networks
Physical layer
Port address
80 (port number)
IP address
179.9.8.80
0f:7a:8c:0d:43:5e
(MAC address)
Physical address
12


Multiplexing and demultiplexing
Processes
Processes
UDP
(Multiplexer)
UDP
(Demultiplexer)
IP
IP

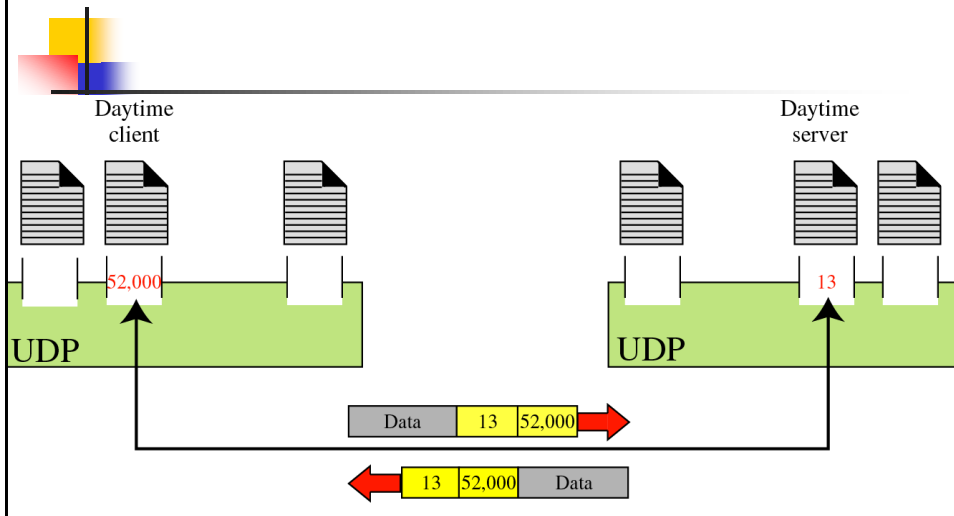
User Datagram Protocol (UDP)

- Simple Demultiplexer
- Extends host-to-host delivery service of the underlying network into a process-to-process communication service
- Adds a level of demultiplexing which allows multiple application processes on each host to share the network



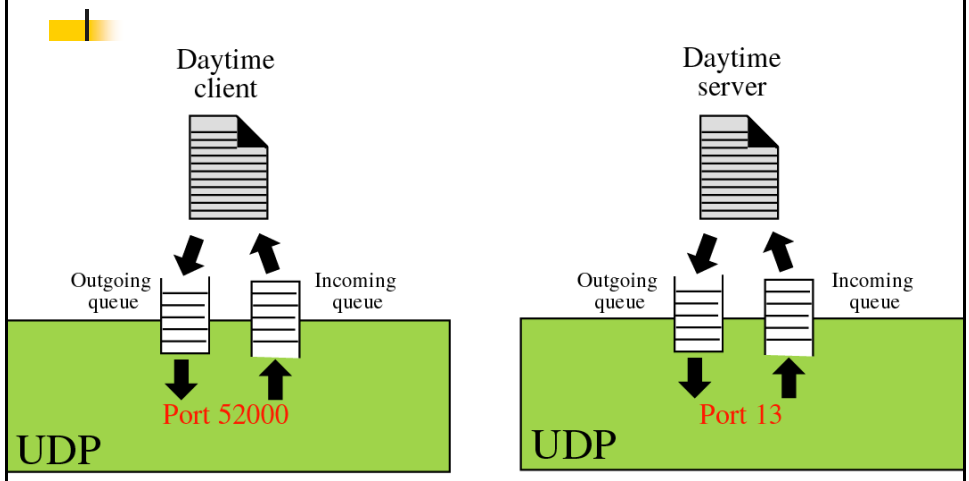
By 清大資工系黃能富教授 14

UDP Port numbers



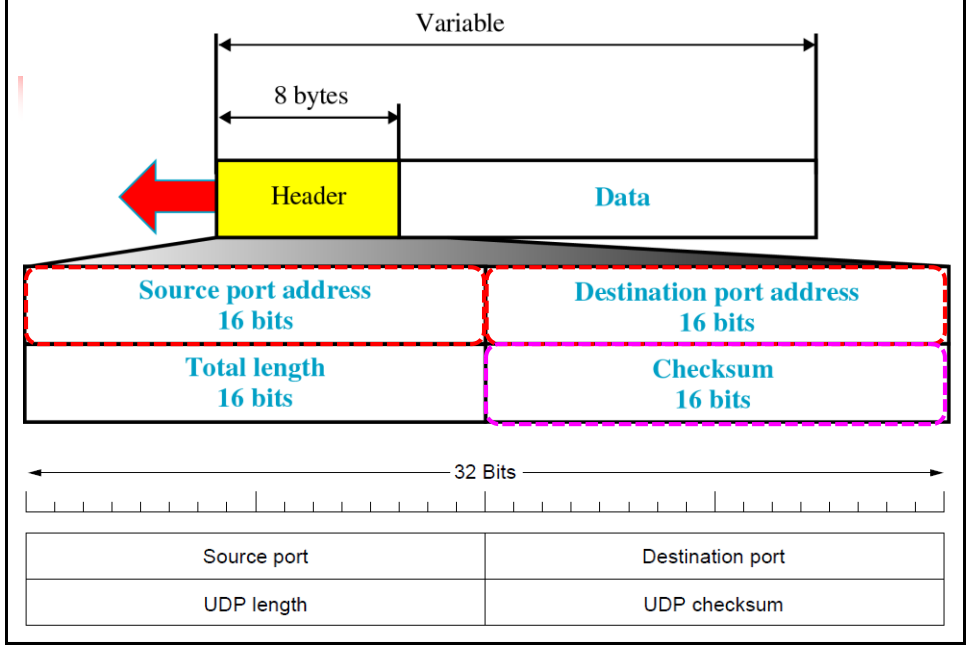
15

Queues in UDP

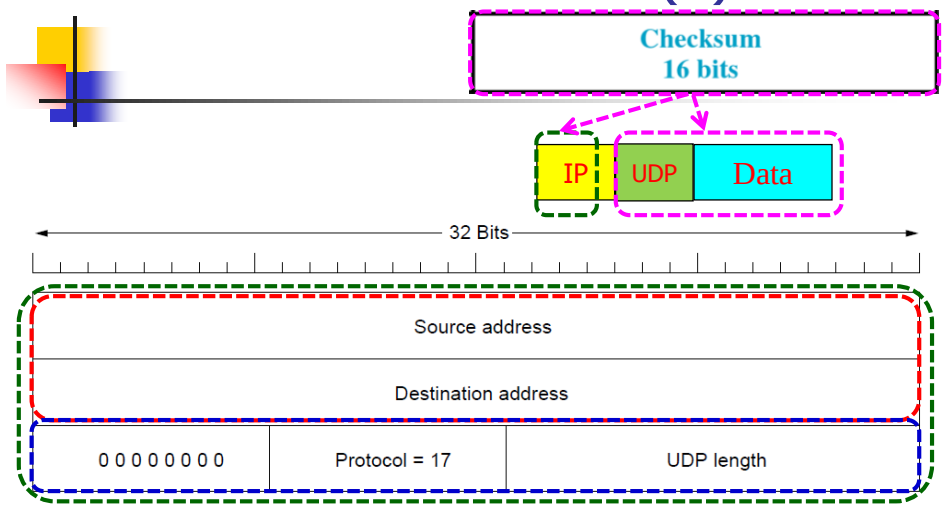


16

UDP Datagram Format



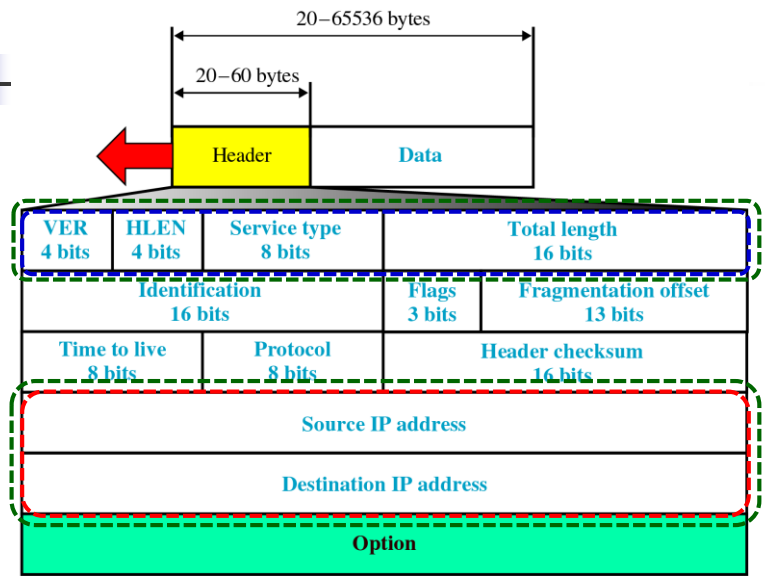
Introduction to UDP (2)



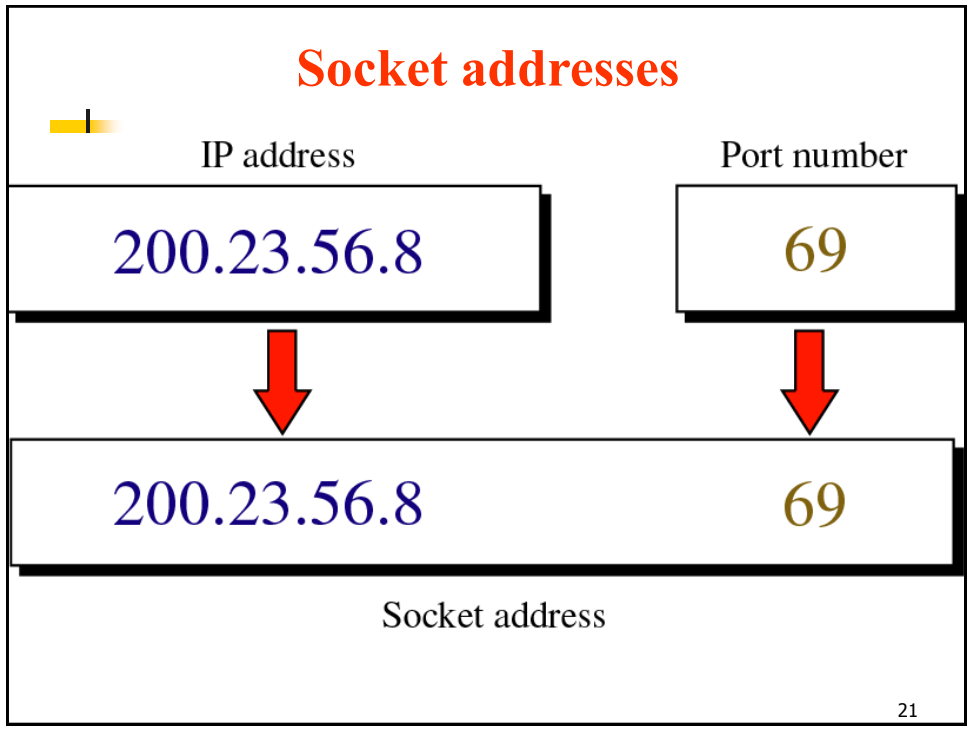
The IPv4 pseudoheader included in the UDP checksum.

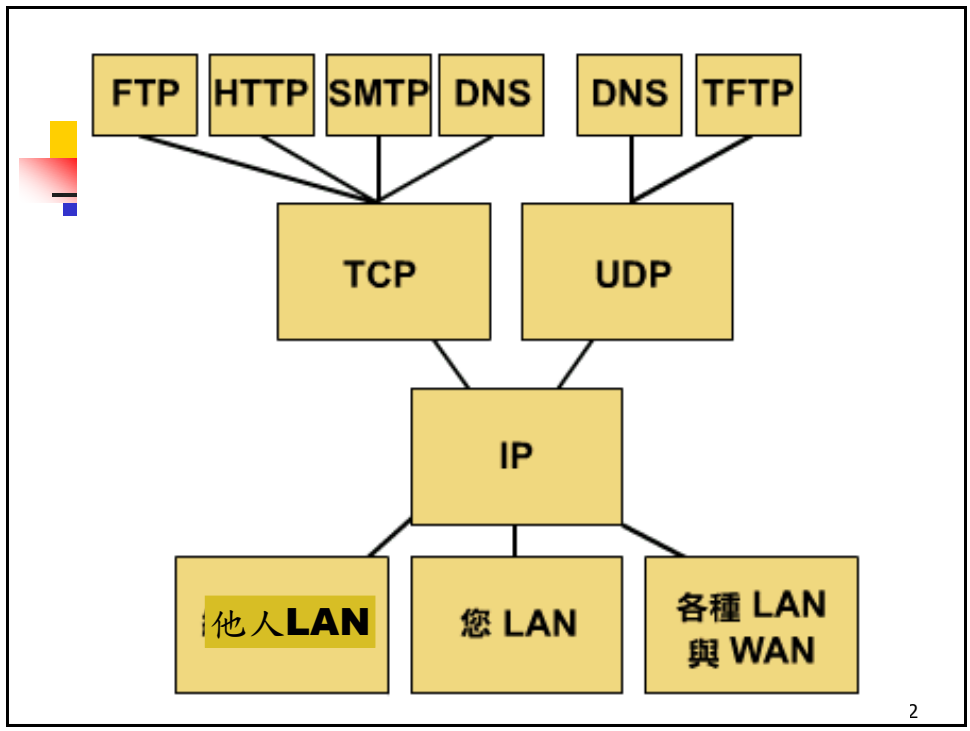
19

IP Datagram

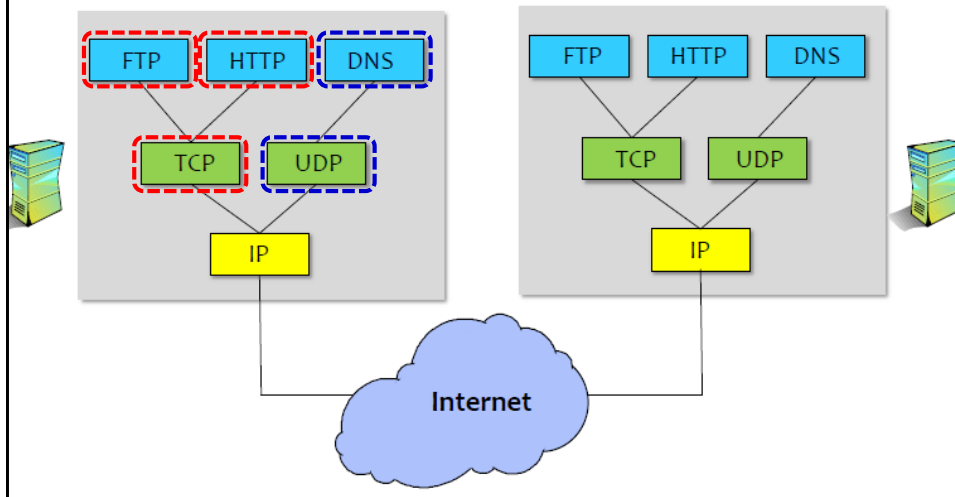


20


Socket addresses
IP address
Port number
200.23.56.8
69
200.23.56.8
69
Socket address
21


FTP
HTTP
SMTP
DNS
DNS
TFTP
TCP
UDP
IP
他人LAN
您 LAN
各種 LAN
與 WAN
2

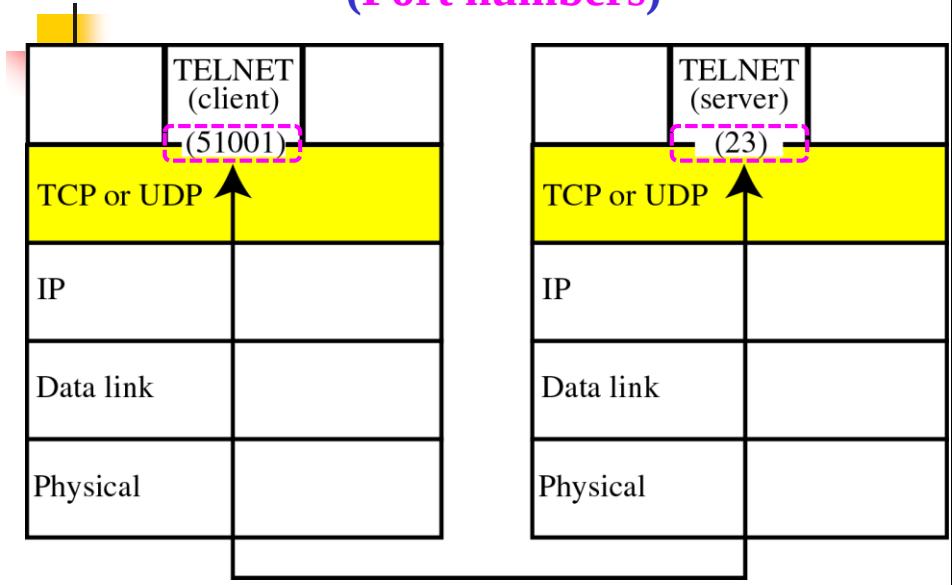
Protocol Architecture

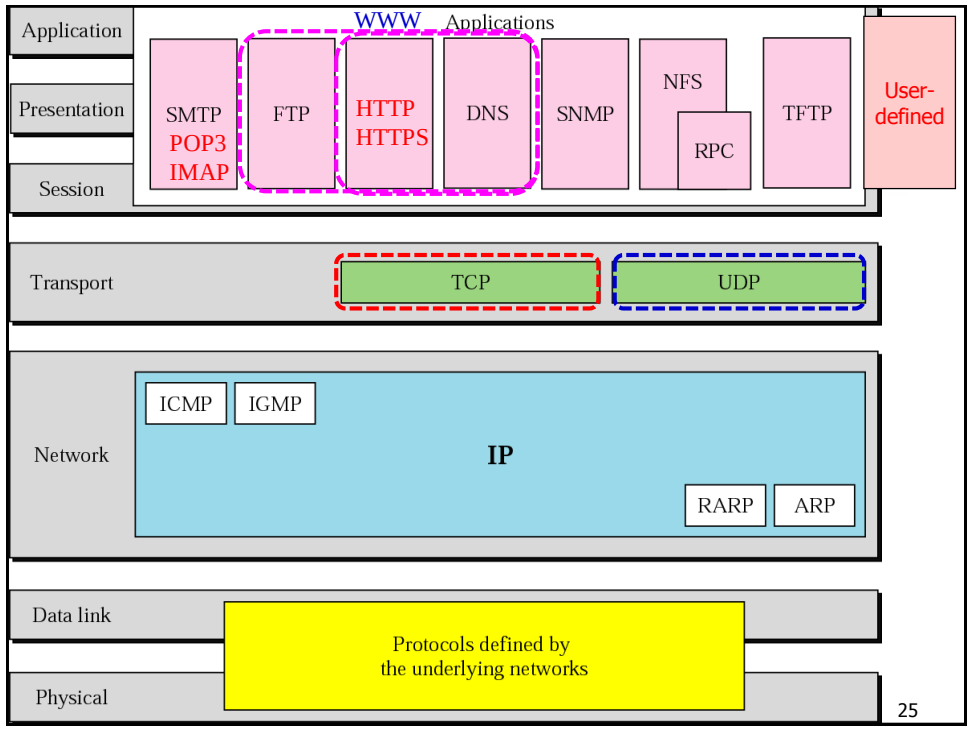


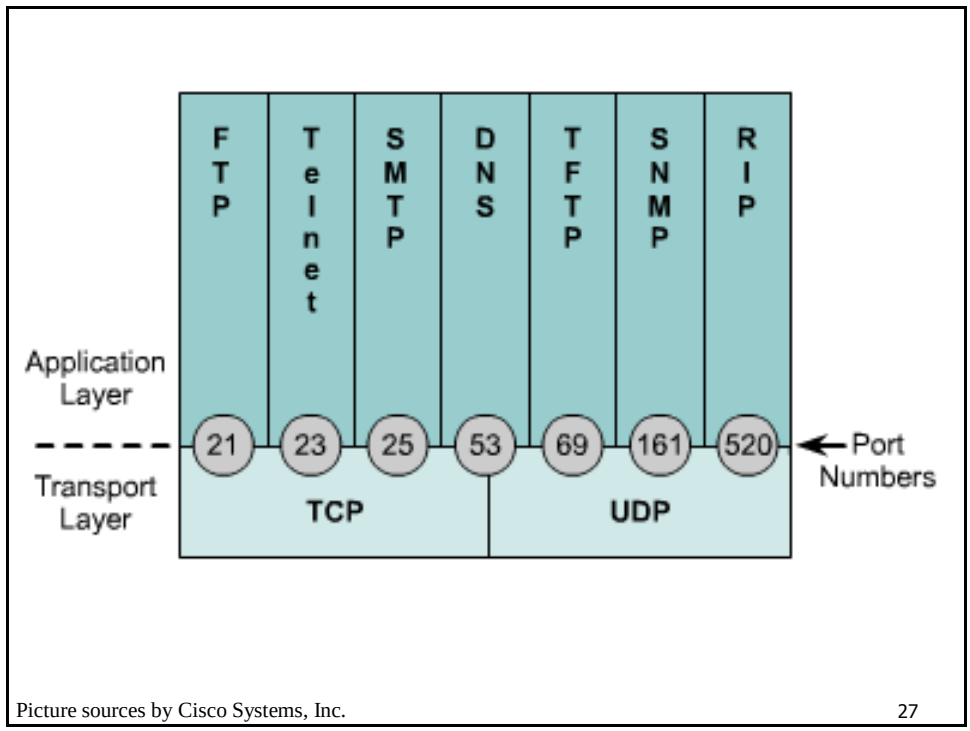
Example of a protocol architecture

nodes are the protocols and links the "depends-on" relation

Port Addresses
(Port numbers)




Application
Presentation
Session
WWW Applications
SMTP
POP3
IMAP
FTP
HTTP
HTTPS
DNS
SNMP
NFS
RPC
TFTP
User-defined
Transport
TCP
UDP
Network
ICMP
IGMP
IP
RARP
ARP
Data link
Protocols defined by
the underlying networks
Physical
25


FTP
Telnet
SMTP
DNS
TFTP
SNMP
RIP
Application Layer
21
23
25
53
69
161
520
Port Numbers
Transport Layer
TCP
UDP
Picture sources by Cisco Systems, Inc.
27

The TCP Service Model (1)

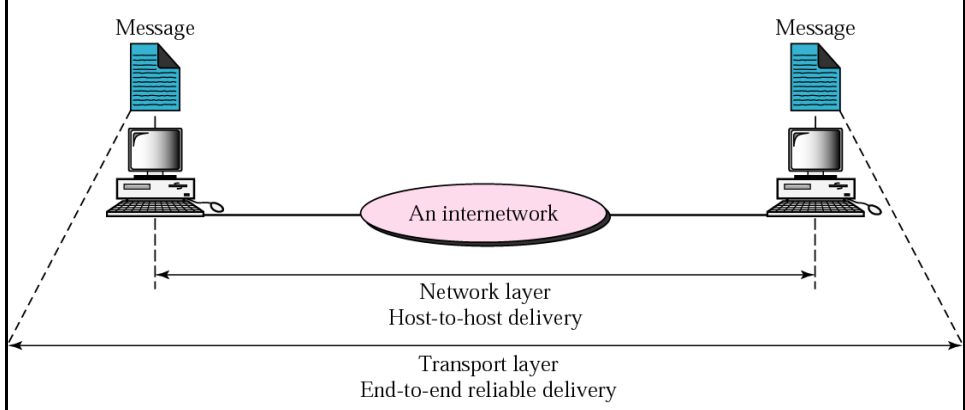
Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

Some assigned ports

28

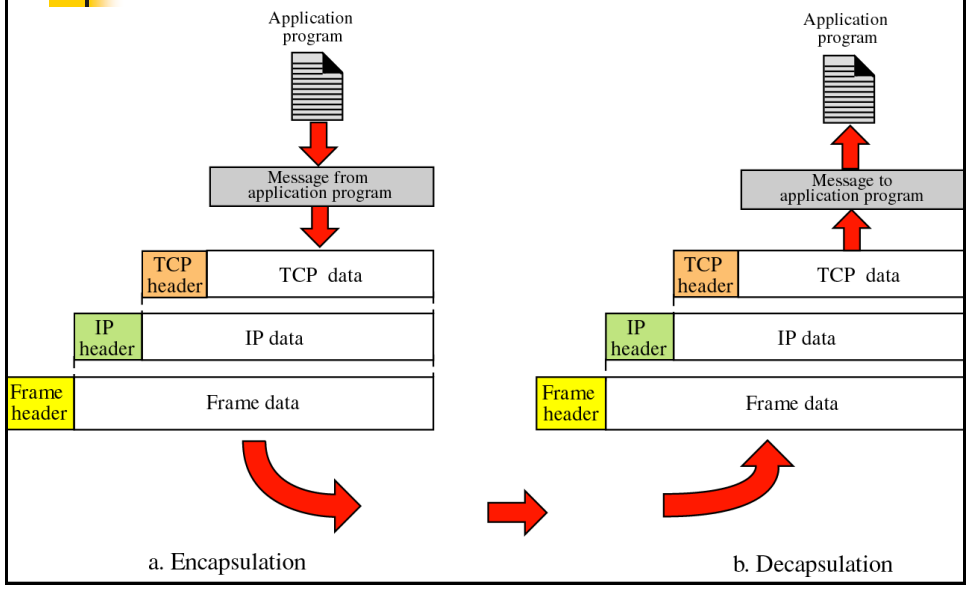
Reliable end-to-end delivery of a message

TCP (Transmission Control Protocol)

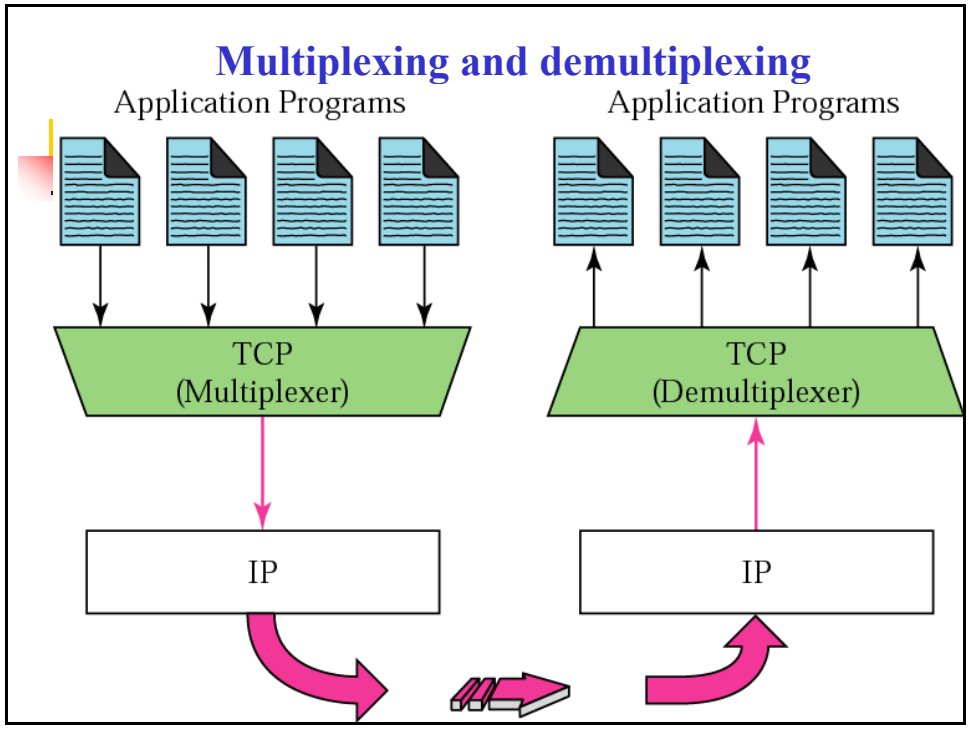


29

Encapsulation and decapsulation

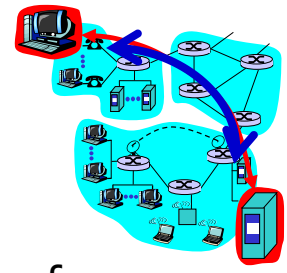


Multiplexing and demultiplexing



Reliable Byte Stream protocol (TCP)

- TCP offers the following services
  - Reliable
  - Connection oriented
  - Byte-stream service

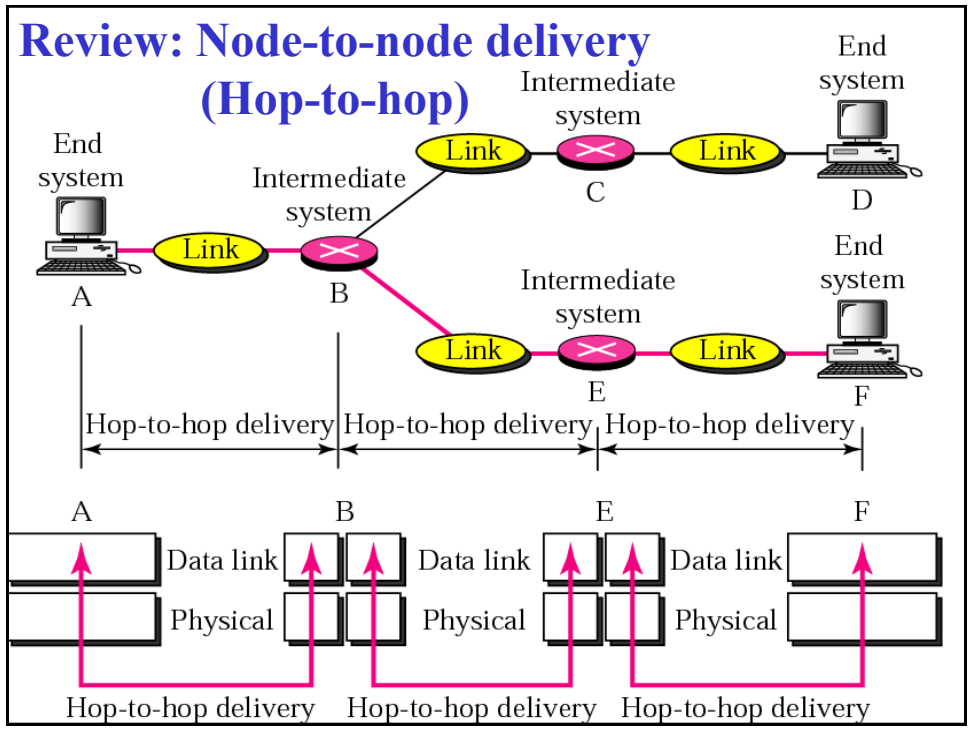


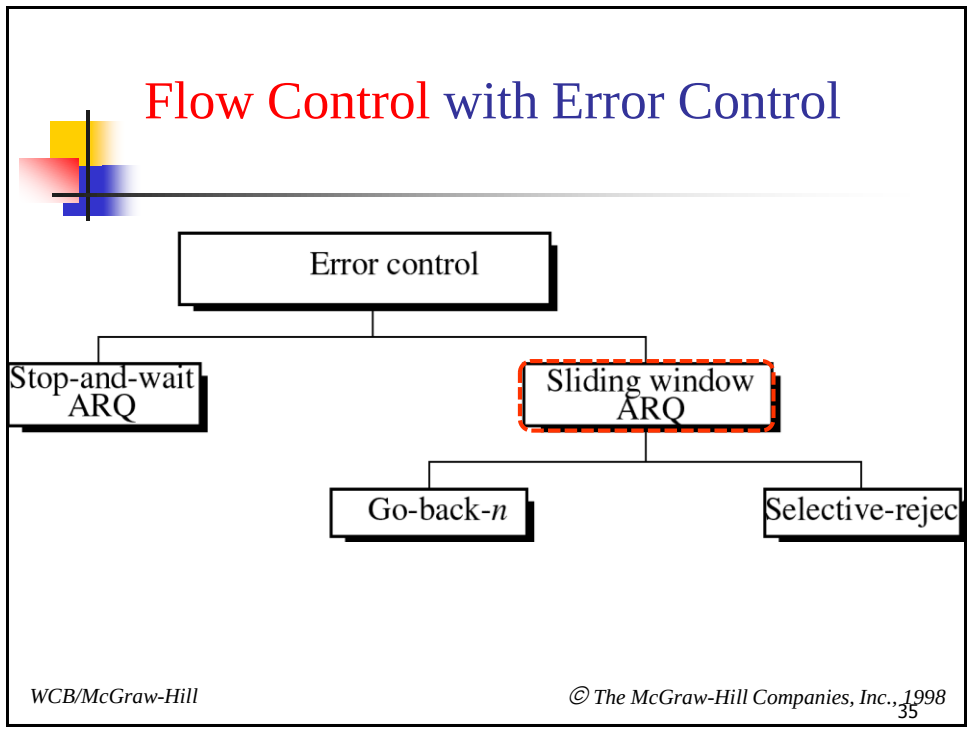
- Flow control: preventing senders from overrunning the capacity of the receivers
- Congestion control: preventing too much data from being injected into the network, thereby causing the Internet to become overloaded

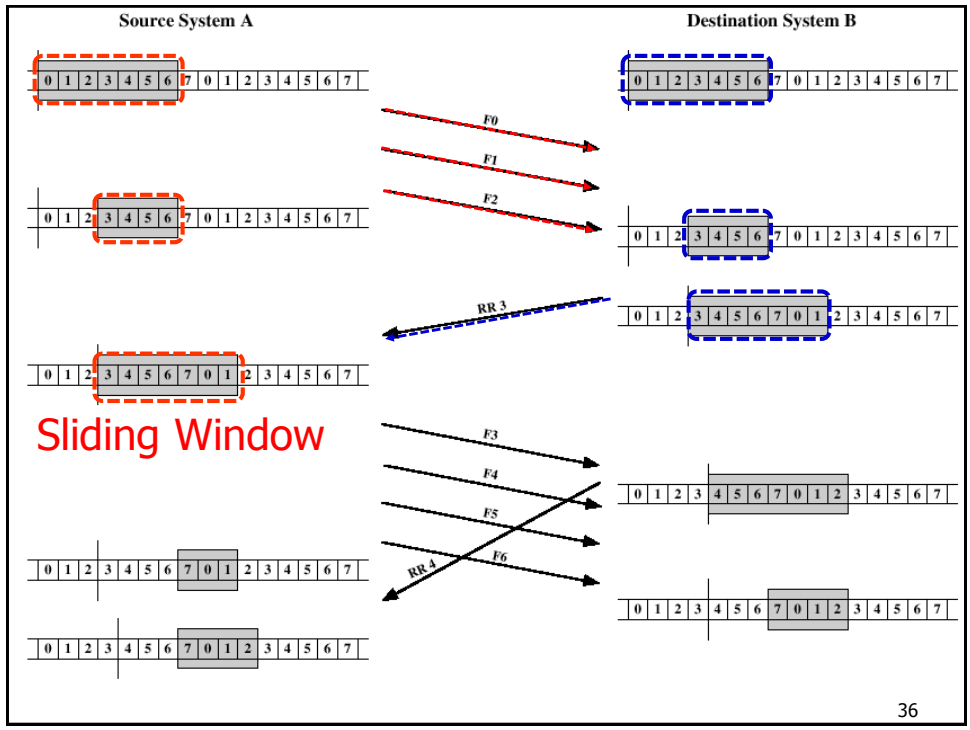
32

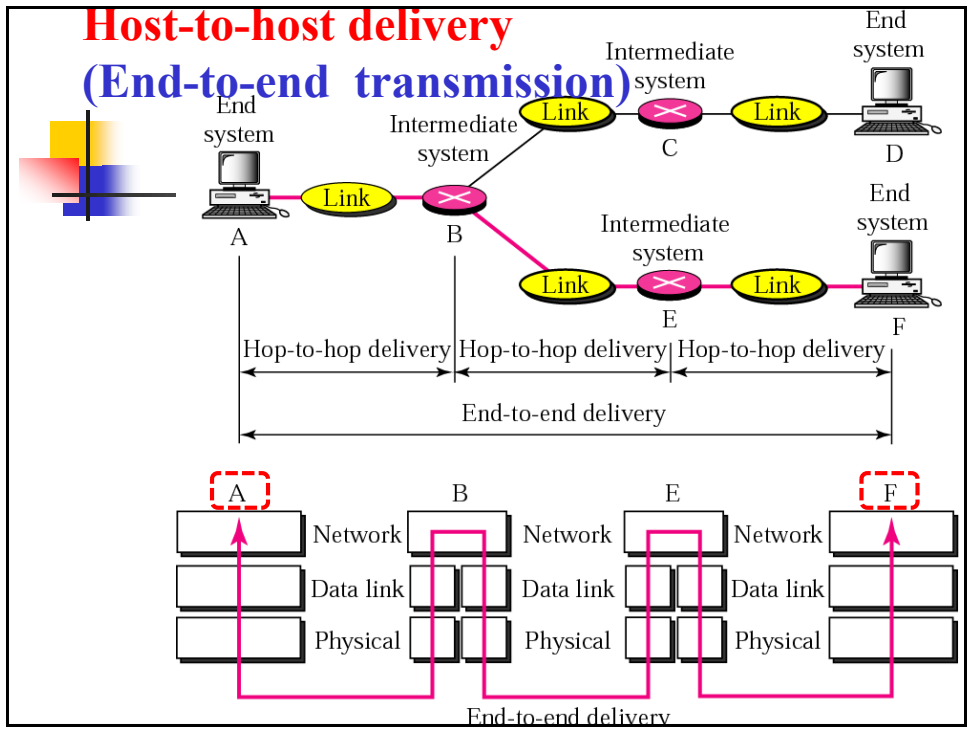
End-to-end Issues

- TCP runs over the Internet rather than a point-to-point link
- The TCP sliding window algorithm need to consider:
  - TCP supports logical connections between processes that are running on two different computers in the Internet
  - TCP connections are likely to have widely different RTT times
  - Packets may get reordered in the Internet

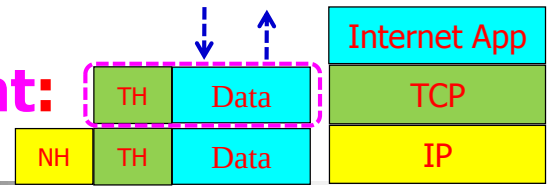

Review: Node-to-node delivery
(Hop-to-hop)
End system
Intermediate system
End system
Link
Link
Link
C
D
A
B
Link
Intermediate system
End system
Link
Link
E
F
Hop-to-hop delivery
Hop-to-hop delivery
Hop-to-hop delivery
A
B
E
F
Data link
Data link
Data link
Physical
Physical
Physical
Hop-to-hop delivery
Hop-to-hop delivery
Hop-to-hop delivery


Flow Control with Error Control
Error control
Stop-and-wait ARQ
Sliding window ARQ
Go-back-n
Selective-rejec
WCB/McGraw-Hill
© The McGraw-Hill Companies, Inc., 1998
35


Source System A
Destination System B
F0
F1
F2
RR 3
F3
F4
F5
RR 4
F6
Sliding Window
36


Host-to-host delivery
(End-to-end transmission)
End system
Intermediate system
Link
A
B
C
D
E
F
Hop-to-hop delivery
End-to-end delivery
Network
Data link
Physical

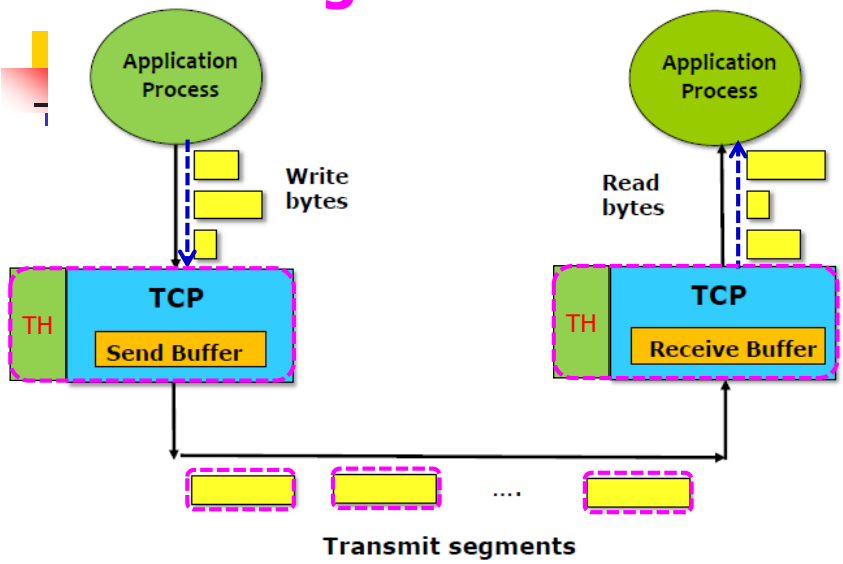
TCP Segment:



- TCP is a byte-oriented protocol
- The sender writes bytes into a TCP connection and the receiver reads bytes out of the TCP connection.
- The source TCP buffers enough bytes from the sending process to fill a reasonably sized packet and then sends this packet to its peer on the destination host.
- The destination TCP then puts the contents of the packet into a receive buffer, and the receiving process reads from this buffer.

By 清大資工系黃能富教授 38

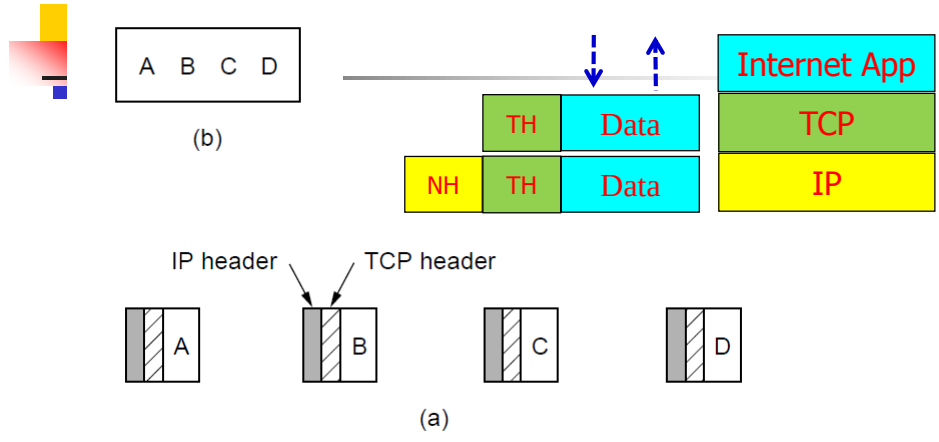
TCP Segment



By 清大資工系黃能富教授

How TCP manages a byte stream. 39

The TCP Service Model (2)

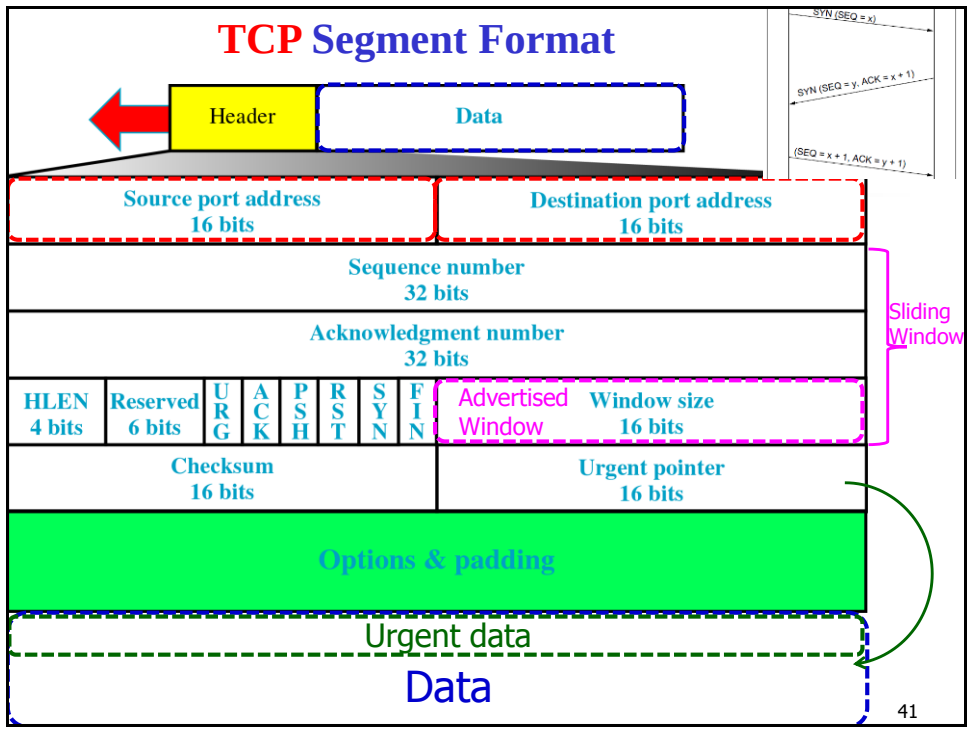


(a) Four 512-byte segments sent as separate IP diagrams

(b) The 2048 bytes of data delivered to the application in a single READ call

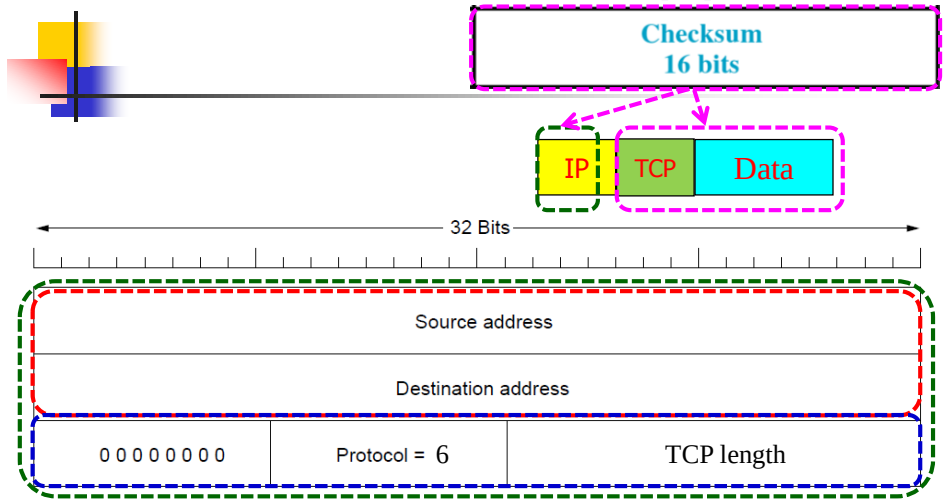
40

TCP Segment Format



41

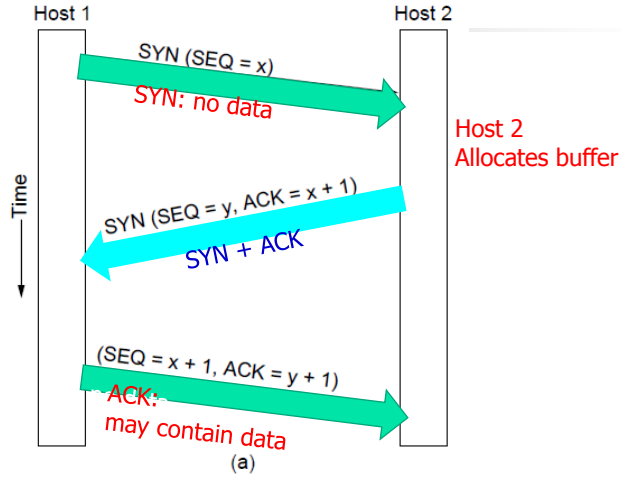
Introduction to TCP (2)



The IPv4 pseudoheader included in the TCP checksum.

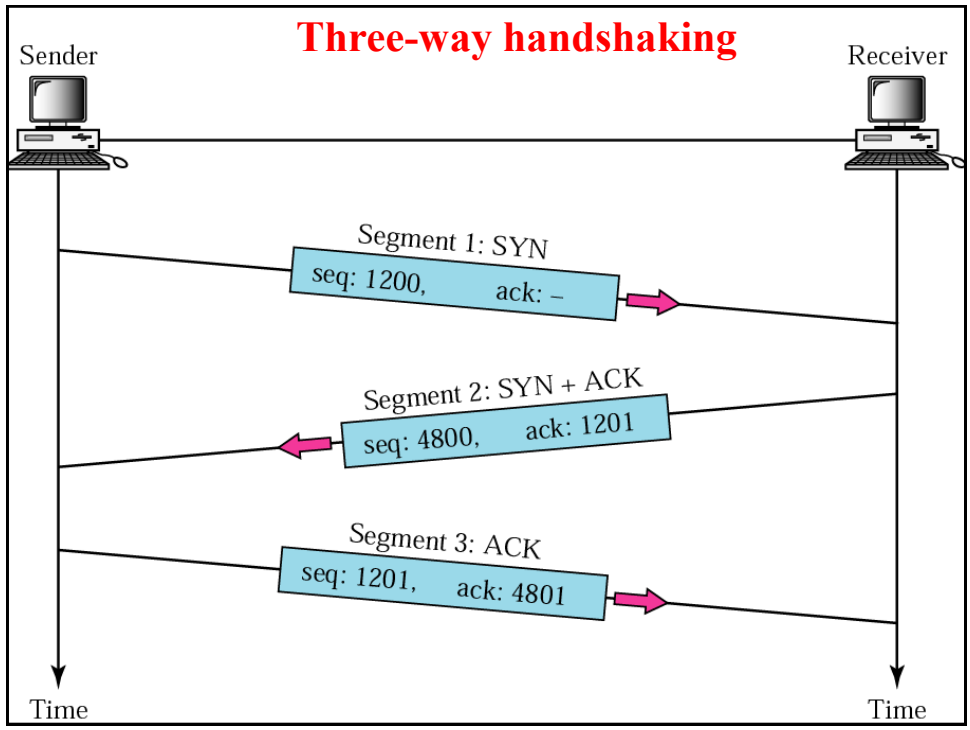
42

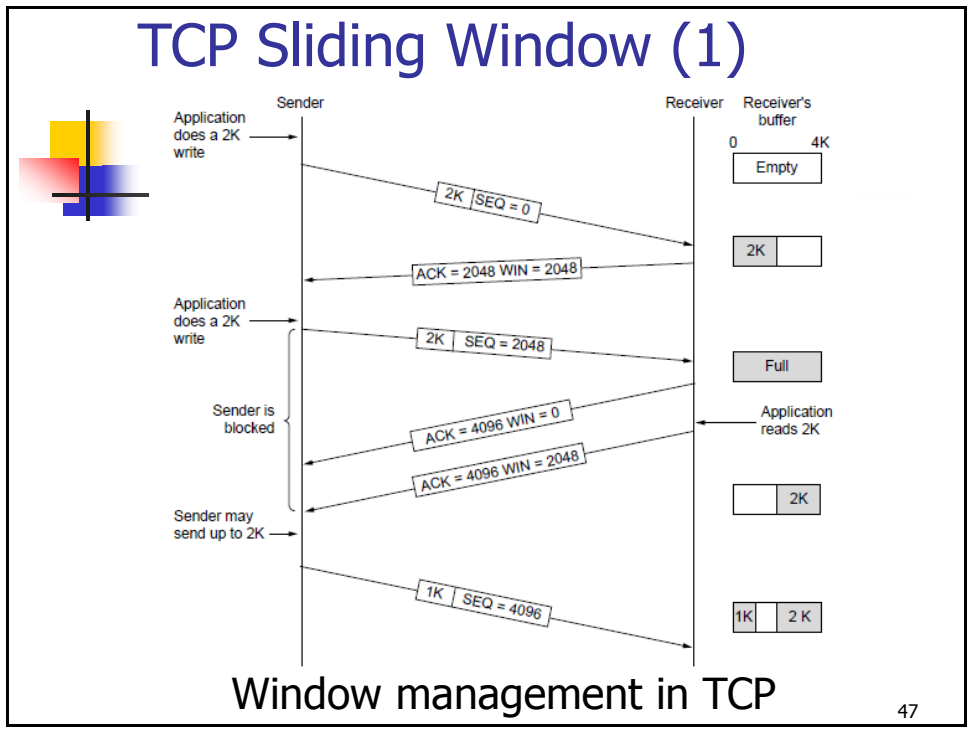
TCP Connection Establishment
Three-way handshake



(a) TCP connection establishment in the normal case.

43


Three-way handshaking
Sender
Receiver
Segment 1: SYN
seq: 1200, ack: –
Segment 2: SYN + ACK
seq: 4800, ack: 1201
Segment 3: ACK
seq: 1201, ack: 4801
Time
Time


TCP Sliding Window (1)
Sender
Receiver
Receiver's buffer
0
4K
Empty
Application does a 2K write
2K SEQ = 0
ACK = 2048 WIN = 2048
2K
Application does a 2K write
2K SEQ = 2048
Full
Sender is blocked
ACK = 4096 WIN = 0
Application reads 2K
ACK = 4096 WIN = 2048
2K
Sender may send up to 2K
1K SEQ = 4096
1K
2 K
Window management in TCP
47

TCP Connection Management (cont.)

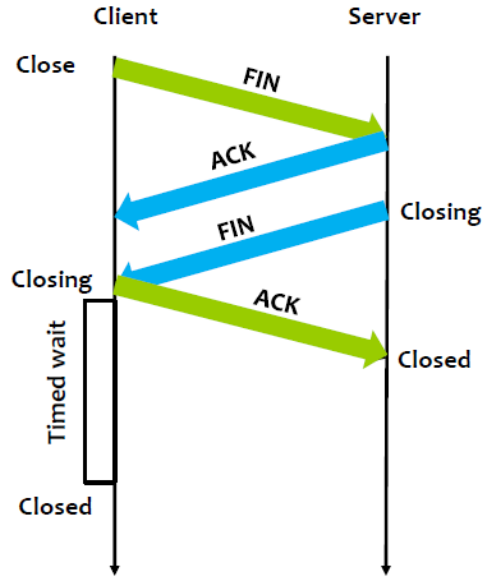
Closing a connection:

client closes socket:

```
clientSocket.close()
;
```

Step 1: Client sends TCP FIN control segment to server

Step 2: Server receives FIN, replies with ACK. Closes connection, sends FIN.



By 清大資工系黃能富教授

48

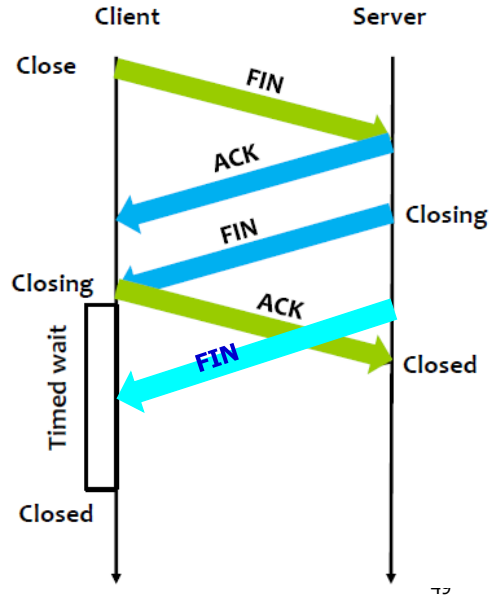
TCP Connection Management (cont.)

Step 3: Client receives FIN, replies with ACK.

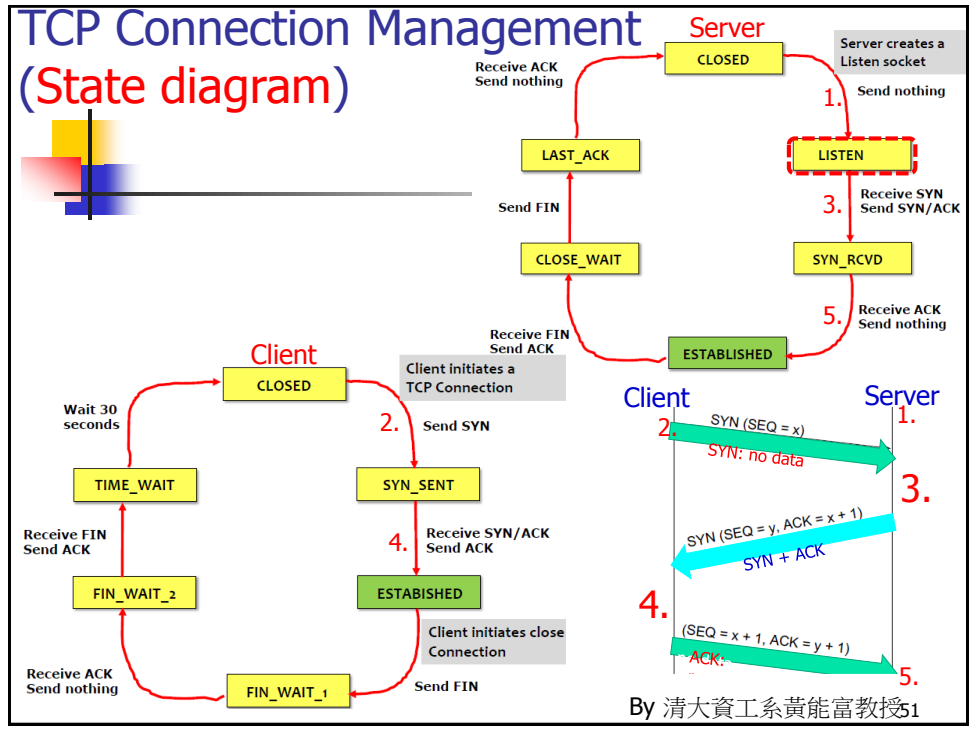
- Enters "timed wait" - will respond with ACK to received FINs

Step 4: Server receives ACK. Connection closed.

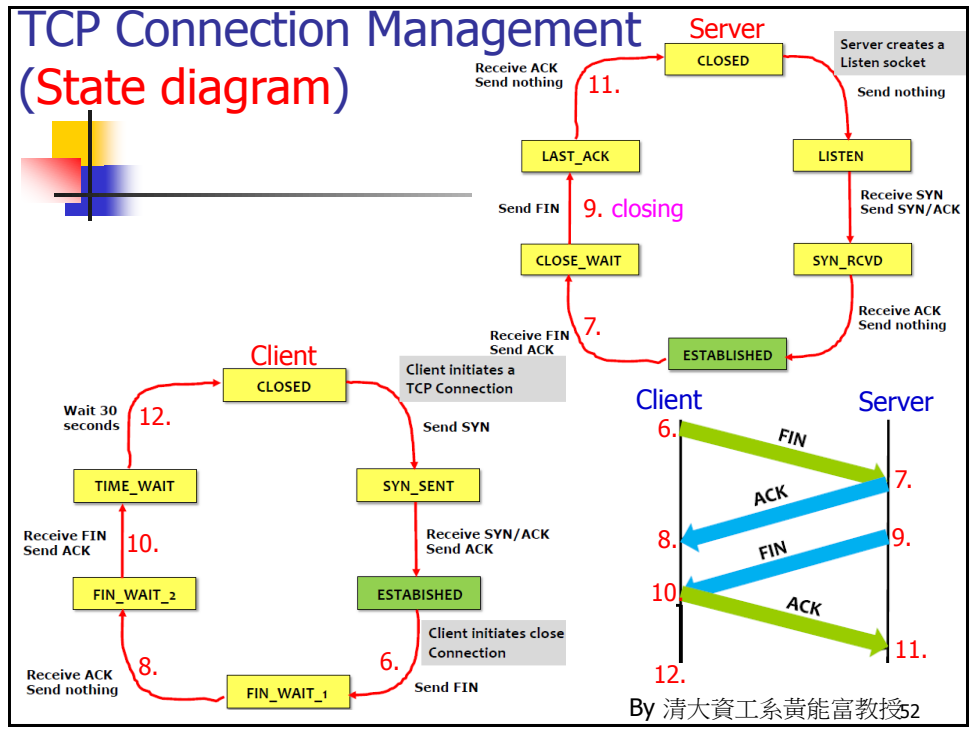
Note: with small modification, can handle simultaneous FINs.



By 清大資工系黃能富教授



By 清大資工系黃能富教授51



By 清大資工系黃能富教授52

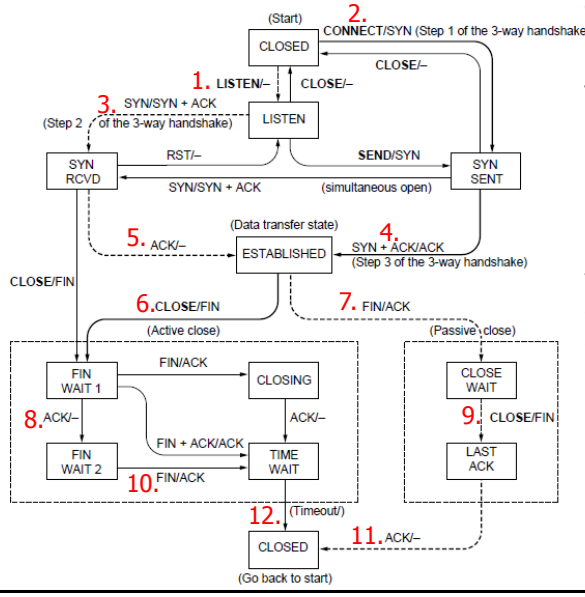
TCP Connection Management Modeling (1)

The states used in the TCP connection management finite state machine.

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 application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIME WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

53

TCP Connection Management Modeling (2)



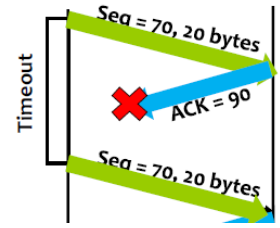
TCP connection management finite state machine.

The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. Each transition is labeled by the event causing it and the action resulting from it, separated by a slash.

54

Timeout value for Retransmission

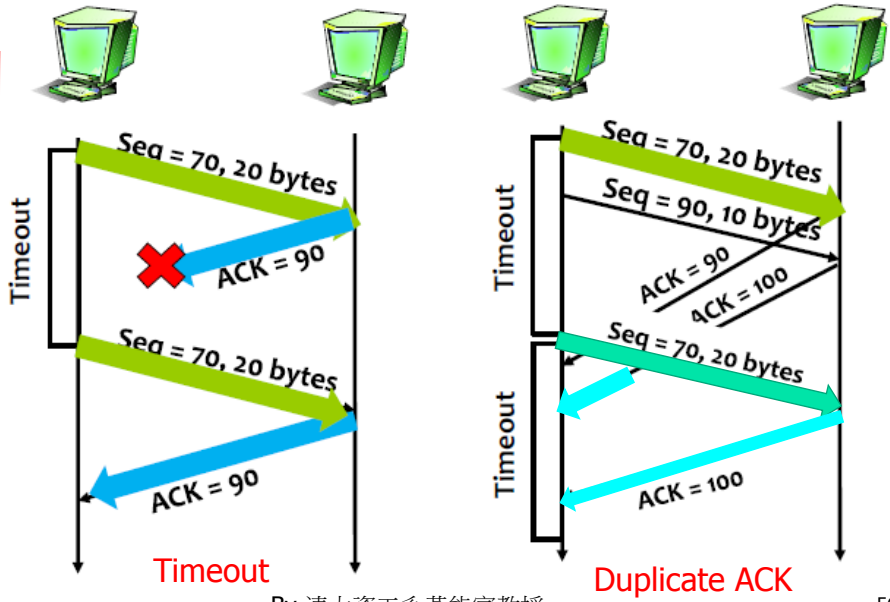
- Measure SampleRTT for each segment/ ACK pair
- Compute weighted average of RTT
  - **EstRTT**= **a** x **EstRTT**+ (1 -**a**)x SampleRTT
  - **a** between 0.8 and 0.9
- Set timeout based on EstRTT
  - TimeOut = 2 x EstRTT



By 清大資工系黃能富教授

55

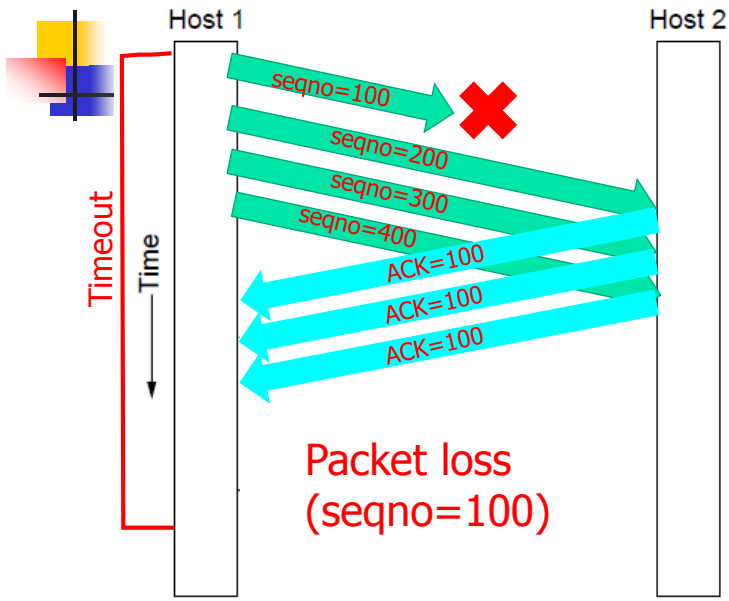
TCP retransmission scenarios



By 清大資工系黃能富教授

56

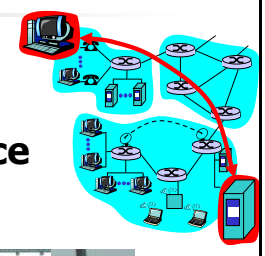
3-duplicate ACK



57

TCP Congestion Control

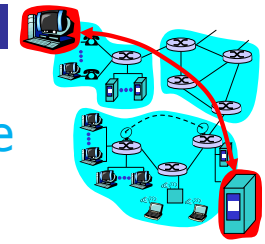
- **Flow control**
  - **Advertised Window**
- **Congestion control**
  - **Congestion window*: self-clocking* by using ACKs to pace the transmission of packets**



Source: https://cc0.wfublog.com/ or http://photopin.com/ or Google

8

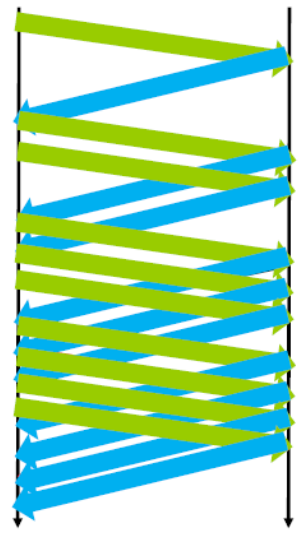
TCP Congestion Control



- Additive Increase Multiplicative Decrease (AIMD)
  - TCP's **effective window** is revised as follows:
  - MaxWindow=MIN (CongestionWindow, AdvertisedWindow)
  - **EffectiveWindow**= MaxWindow– (LastByteSent– LastByteAcked).
- A TCP source is allowed to send no faster than the slowest component can accommodate
  - the network or the destination host

Additive Increase Multiplicative Decrease (AIMD)

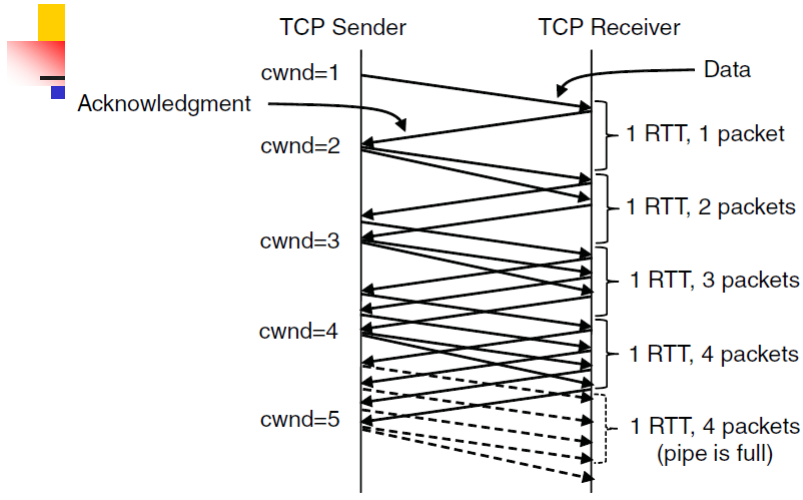
A packet = maximum segment size (MSS) bytes



Additive increase

One more packet is added for each RTT

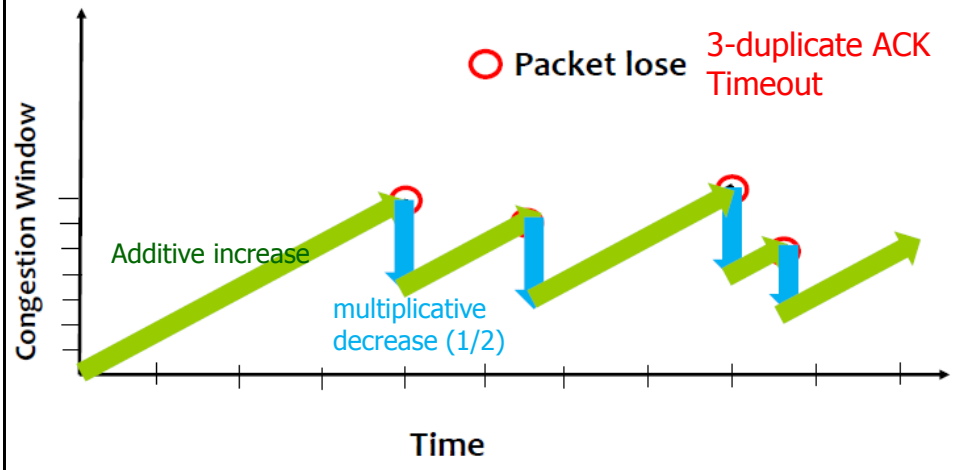
TCP Congestion Control (1)



Additive increase from an initial congestion window of 1 segment.

61

- Additive Increase Multiplicative Decrease (AIMD)
- Trace: Sawtooth behavior



By 清大資工系黃能富教授

62

TCP Congestion Control

- **Slow start**: to increase the congestion window rapidly from a cold start.
- Slow start effectively increases the congestion window exponentially, rather than linearly.

http://auto.ferrari.com/zh_CN/wp-content/uploads/sites/11/2015/02/news-ferrari-580x362.jpg
http://news.xinhuanet.com/auto/2009-04/15/xinsrc_362040615093710923227 28.jpg

63

Slow Start

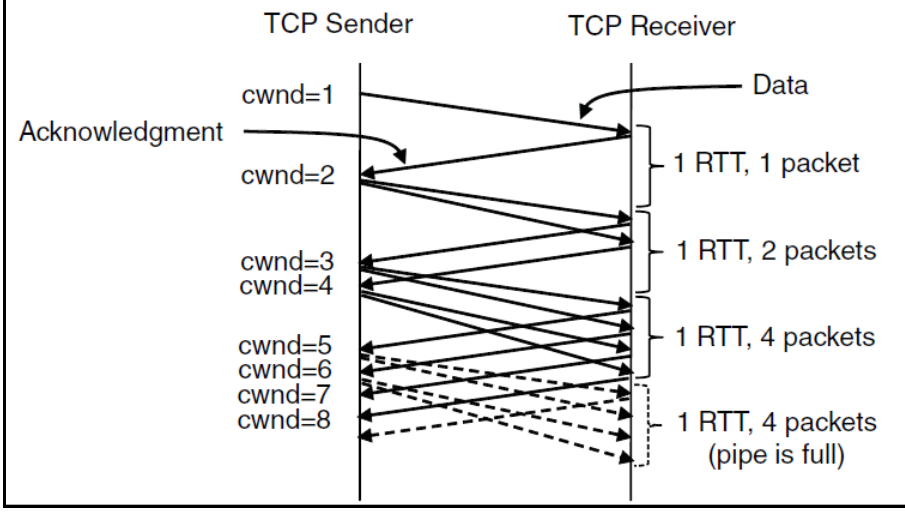
- Initially, the CongestionWindow= 1 packet.
- Example: MSS = 500 bytes, RTT = 200 msec, initial rate = 20 kbps
- 每收到一個ACK就加1 packet (MSS)
- Upon receiving the corresponding two ACKs, TCP increments CongestionWindow by 2—one for each ACK—and next sends four packets.
- TCP effectively doubles the number of packets it has in transit every RTT.

By 清大資工系黃能富教授

64

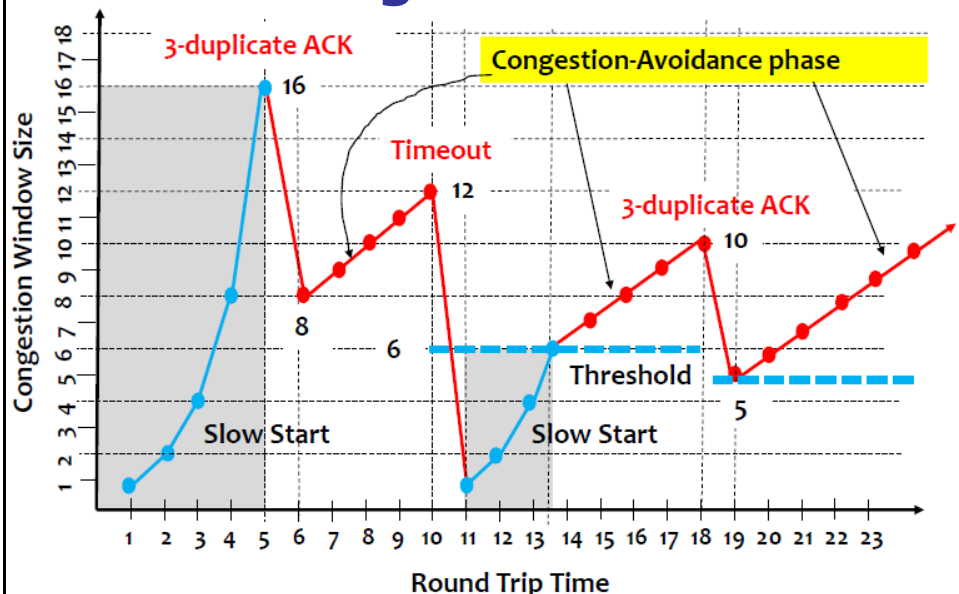
TCP Congestion Control (2)

Slow start from an initial congestion window of 1 segment



5

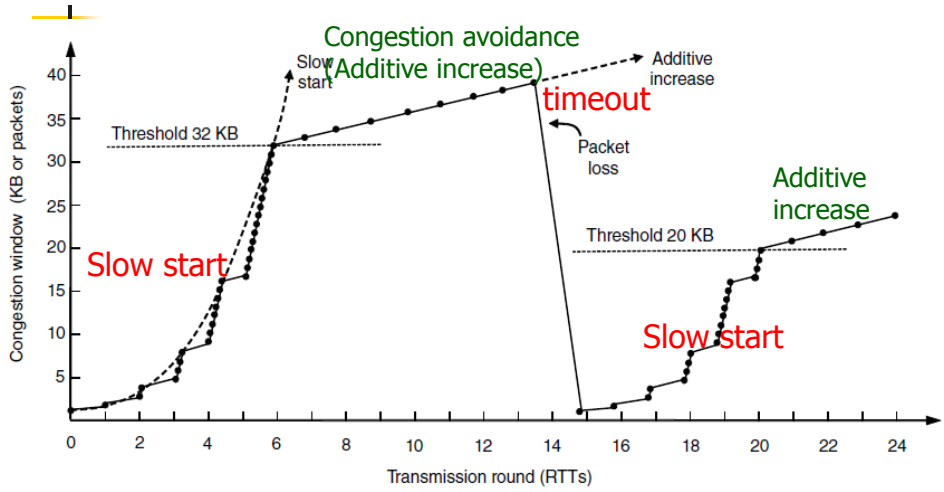
TCP Congestion Control



By 清大資工系黃能富教授

66

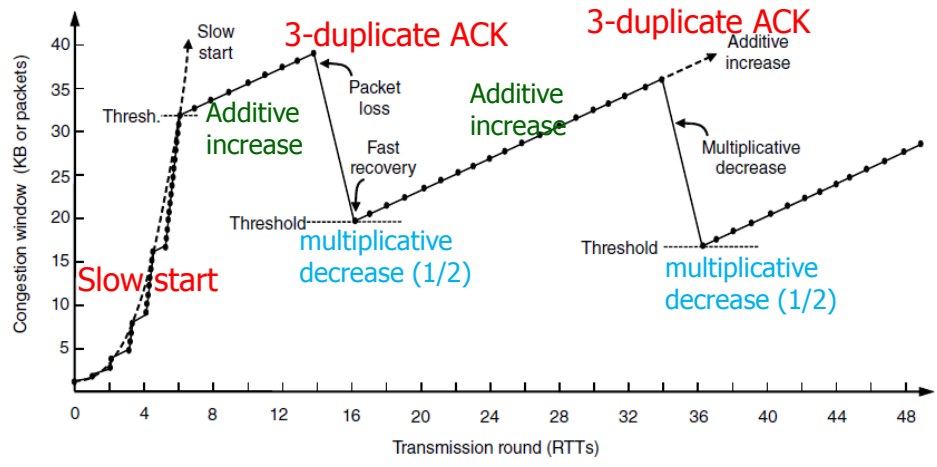
TCP Congestion Control (3)



Slow start followed by additive increase in TCP Tahoe.

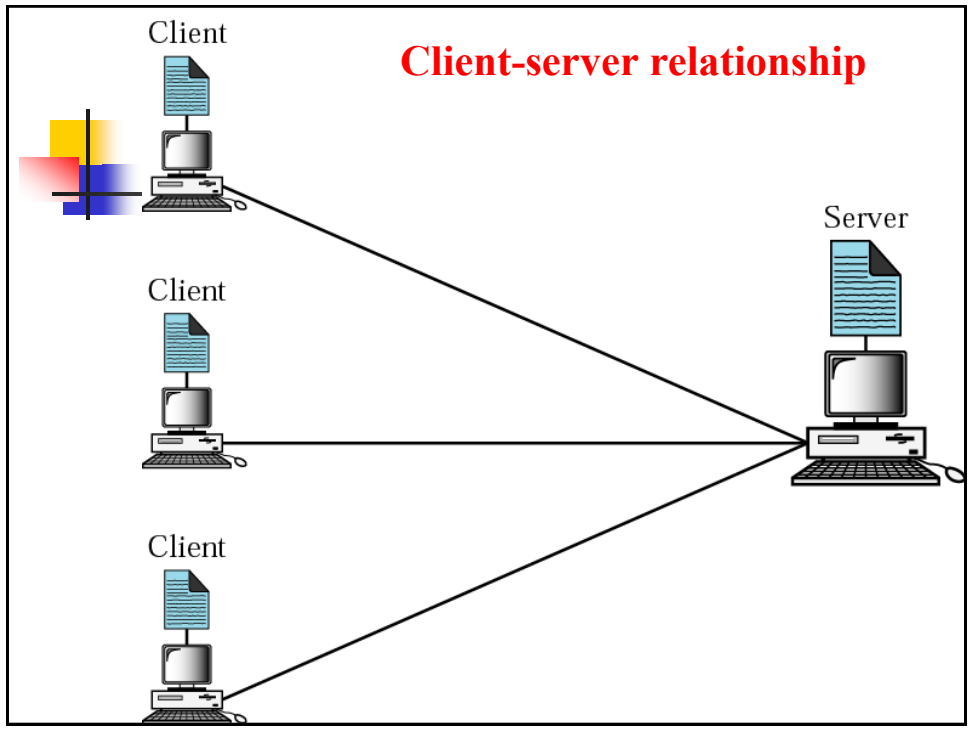
67

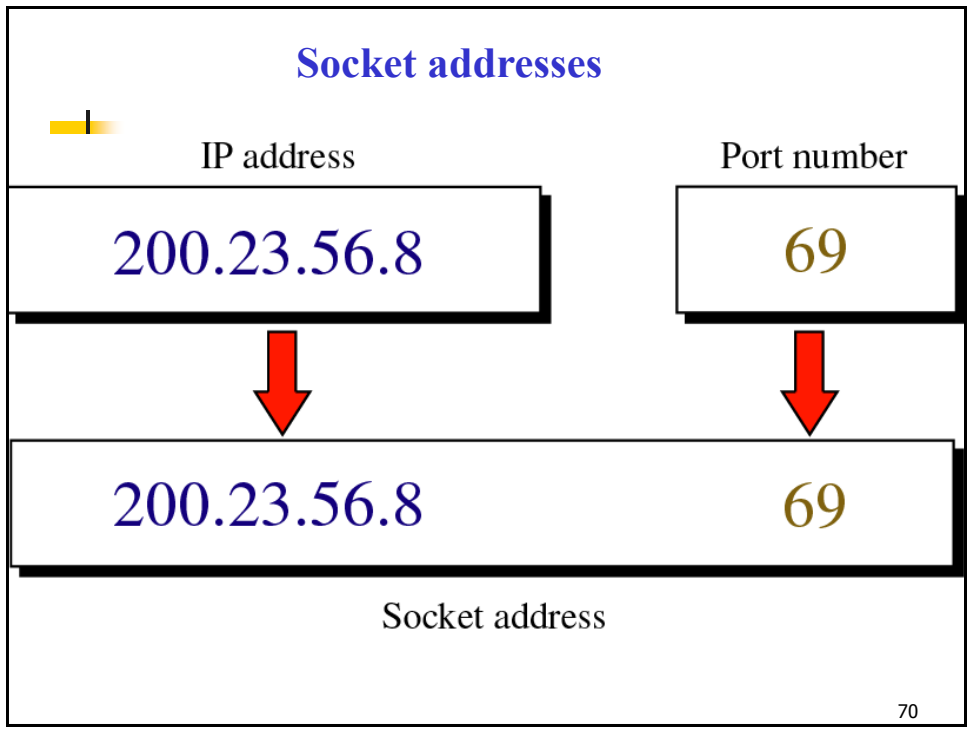
TCP Congestion Control (4)

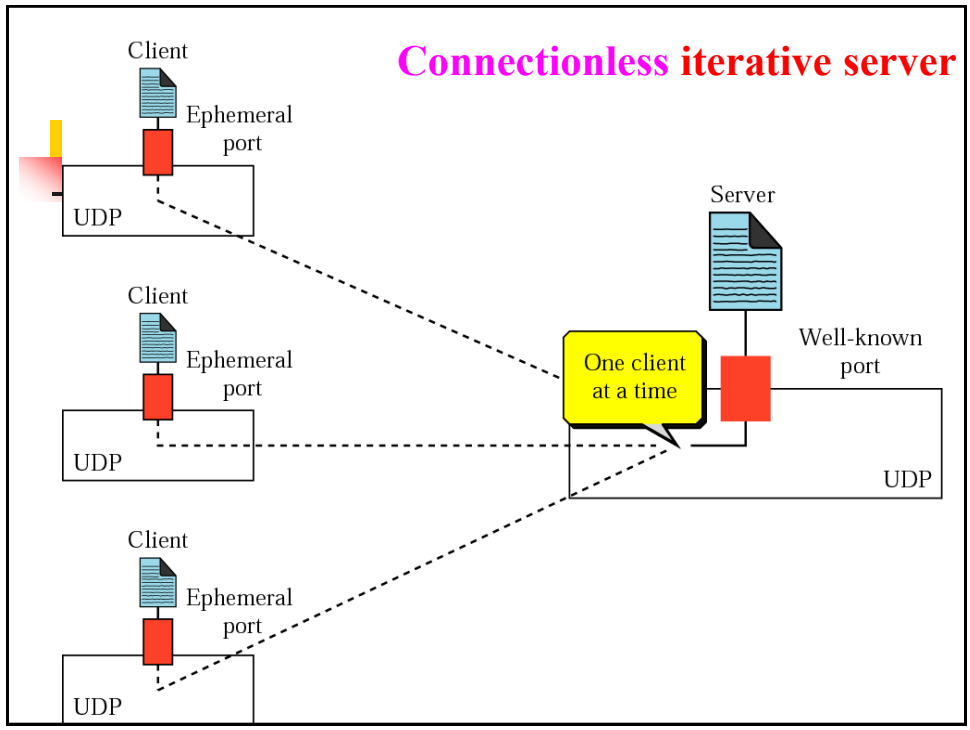


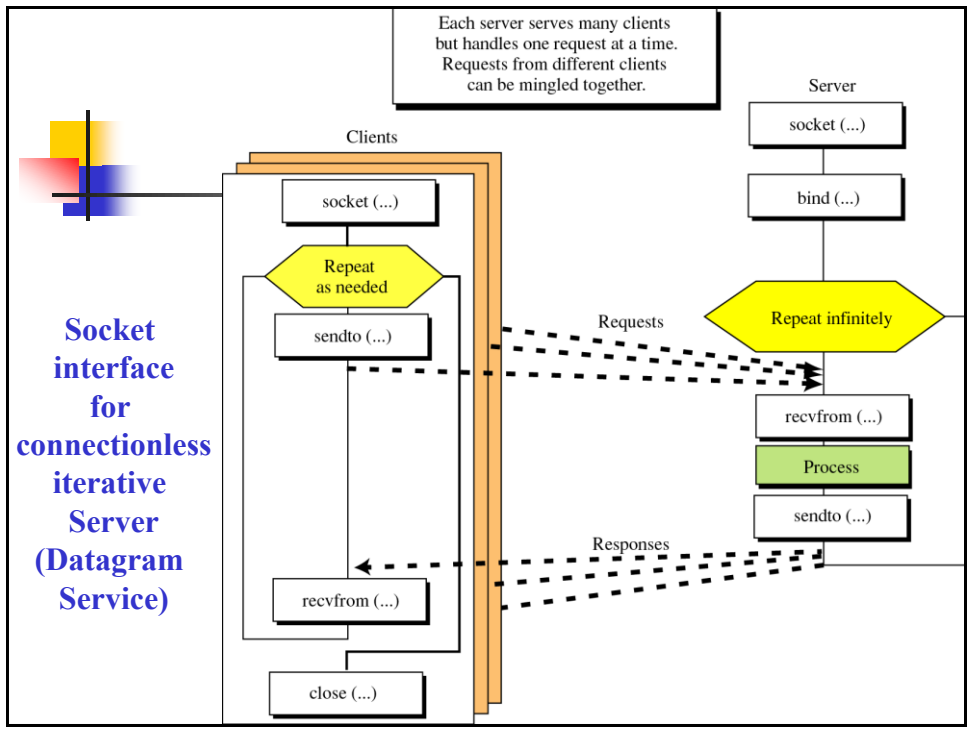
Fast recovery and the sawtooth pattern of TCP Reno.

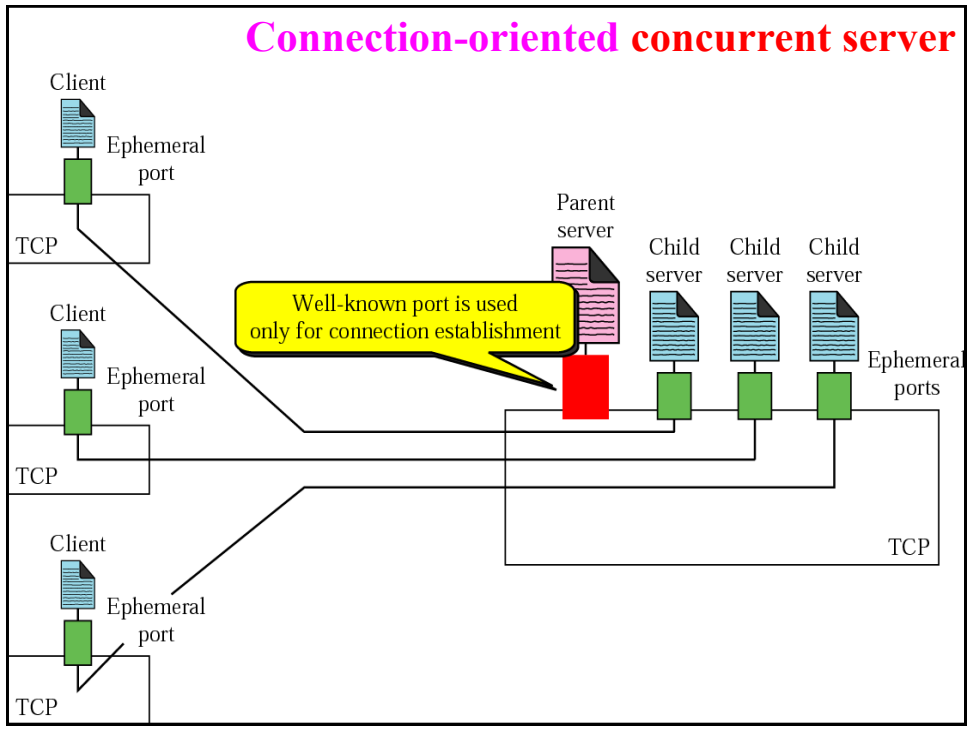
68

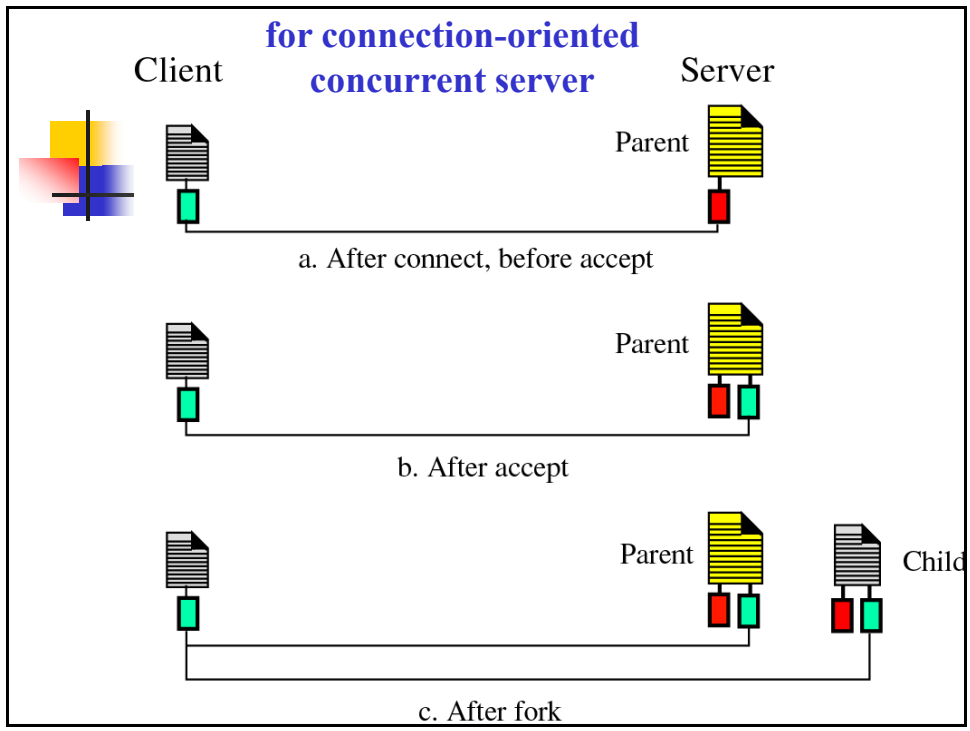

Client-server relationship
Client
Client
Client
Server


Socket addresses
IP address
Port number
200.23.56.8
69
200.23.56.8
69
Socket address
70

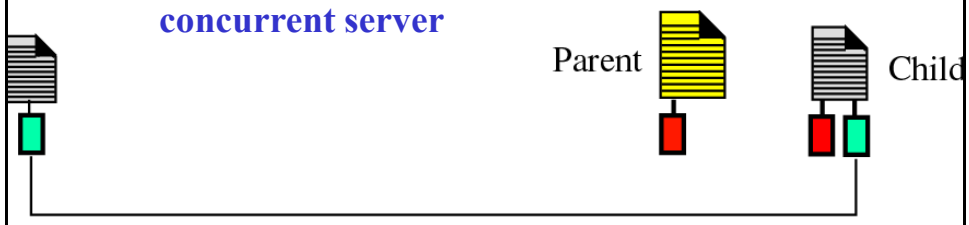

Connectionless iterative server
Client
Ephemeral port
UDP
Client
Ephemeral port
UDP
Client
Ephemeral port
UDP
Server
One client at a time
Well-known port
UDP


Each server serves many clients but handles one request at a time. Requests from different clients can be mingled together.
Socket interface for connectionless iterative Server (Datagram Service)
Clients
socket (...)
Repeat as needed
sendto (...)
recvfrom (...)
close (...)
Server
socket (...)
bind (...)
Repeat infinitely
Requests
recvfrom (...)
Process
sendto (...)
Responses

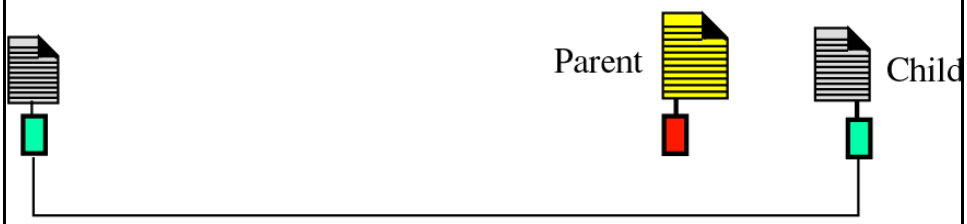

Connection-oriented concurrent server
Client
Ephemeral port
TCP
Client
Ephemeral port
TCP
Client
Ephemeral port
TCP
Parent server
Child server
Child server
Child server
Well-known port is used only for connection establishment
Ephemeral ports
TCP


for connection-oriented concurrent server
Client
Server
Parent
a. After connect, before accept
Parent
b. After accept
Parent
Child
c. After fork

for connection-oriented concurrent server



d. After parent closes ephemeral port

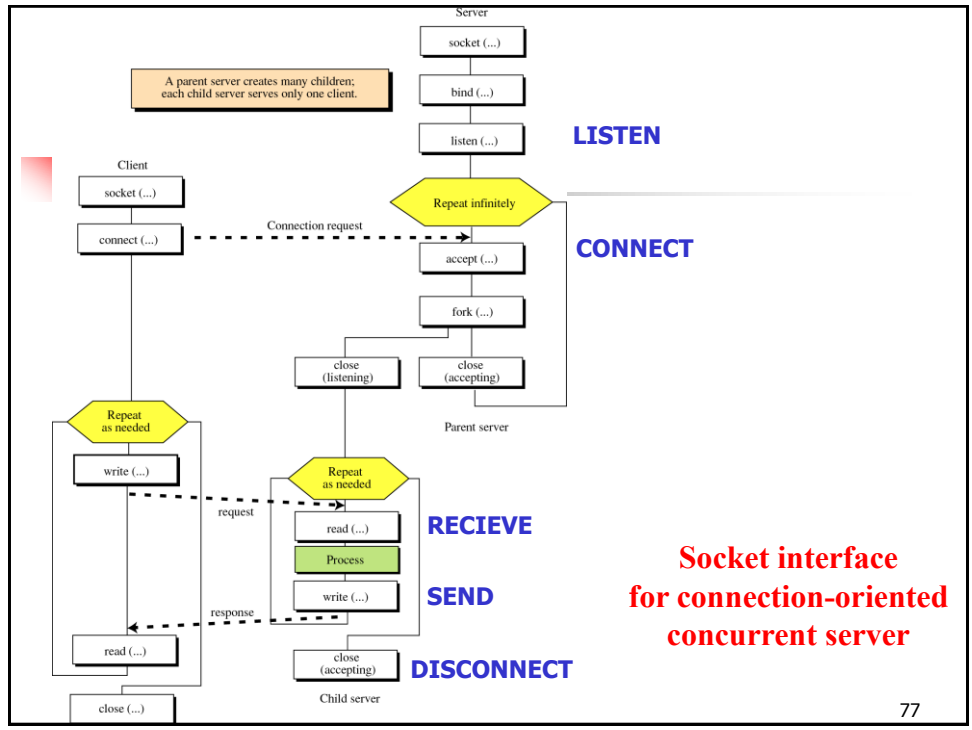


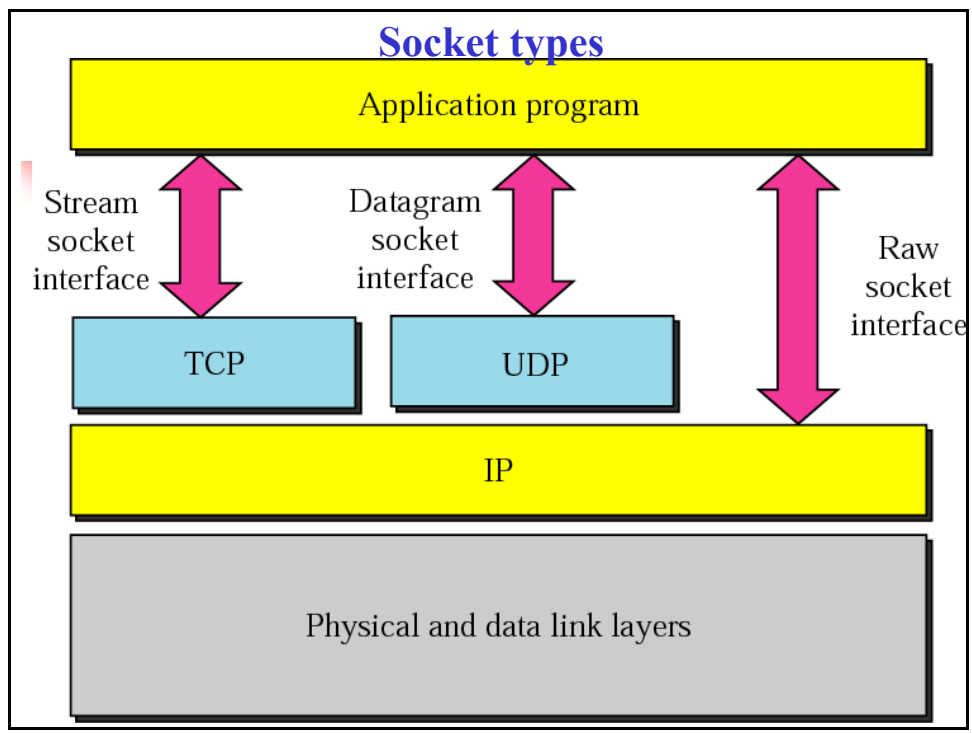
e. After child closes well-known port

75

Example of a server program with parent and child processes

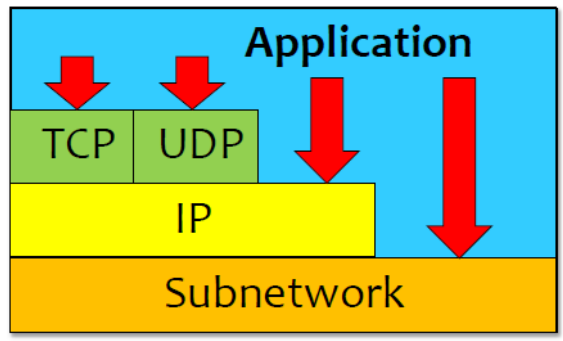
```
.....

void main (void)
{
  ..... ;
  .....;
  pid_t  pid ;
  for ( ; ; )
      {
            Connection from client
            pid  = fork () ;
            if (pid ! = 0 )
            {
              Code  for parent
            }
            else
            {
              Code  for child
            }
      }
  }
```


Server
socket (...)
A parent server creates many children;
each child server serves only one client.
bind (...)
listen (...)
LISTEN
Client
socket (...)
Repeat infinitely
Connection request
connect (...)
CONNECT
accept (...)
fork (...)
close
(listening)
close
(accepting)
Parent server
Repeat
as needed
write (...)
Repeat
as needed
request
read (...)
RECIEVE
Process
write (...)
SEND
response
read (...)
close
(accepting)
DISCONNECT
close (...)
Child server
Socket interface
for connection-oriented
concurrent server
77


Socket types
Application program
Stream
socket
interface
Datagram
socket
interface
Raw
socket
interface
TCP
UDP
IP
Physical and data link layers

Internet Architecture

- Defined by IETF
- Three main features
  - Does not imply strict layering. The application is free to bypass the defined transport layers and to directly use IP or other underlying networks



By 清大資工系黃能富教授 79

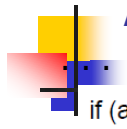
Example of Socket Programming: An Internet File Server (1)

```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345          /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096              /* block transfer size */

int main(int argc, char *argv)
{
  int c, s, bytes;
  char buf[BUF_SIZE];              /* buffer for incoming file */
  struct hostent *h;               /* info about server */
  struct sockaddr_in channel;      /* holds IP address */

  . . .
```

Client code using sockets

80

Example of Socket Programming: An Internet File Server (2)



```
if (argc != 3) fatal("Usage: client server-name file-name");
h = gethostbyname(argv[1]);                    /* look up host's IP address */
if (!h) fatal("gethostbyname failed");

s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s <0) fatal("socket");
memset(&channel, 0, sizeof(channel));
channel.sin_family= AF_INET;
memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
channel.sin_port= htons(SERVER_PORT);

c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
if (c < 0) fatal("connect failed");

. . .
```

Client code using sockets

81

Example of Socket Programming: An Internet File Server (3)

```
. . .
c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
if (c < 0) fatal("connect failed");

/* Connection is now established. Send file name including 0 byte at end. */
write(s, argv[2], strlen(argv[2])+1);

/* Go get the file and write it to standard output. */
while (1) {
    bytes = read(s, buf, BUF_SIZE);        /* read from socket */
    if (bytes <= 0) exit(0);               /* check for end of file */
    write(1, buf, bytes);                  /* write to standard output */
  }
}

fatal(char *string)
{
  printf("%s\n", string);
  exit(1);
}
```

Client code using sockets

82

Example of Socket Programming: An Internet File Server (4)

```
#include <sys/types.h>                  /* This is the server code */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345               /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                   /* block transfer size */
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
  int s, b, l, fd, sa, bytes, on = 1;
  char buf[BUF_SIZE];                   /* buffer for outgoing file */
  struct sockaddr_in channel;           /* holds IP address */

      . . .
```

Server code

83

Example of Socket Programming: An Internet File Server (5)

```
. . .
  /* Build address structure to bind to socket. */
  memset(&channel, 0, sizeof(channel));       /* zero channel */
  channel.sin_family = AF_INET;
  channel.sin_addr.s_addr = htonl(INADDR_ANY);
  channel.sin_port = htons(SERVER_PORT);

  /* Passive open. Wait for connection. */
  s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);    /* create socket */
  if (s < 0) fatal("socket failed");
  setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

  b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
  if (b < 0) fatal("bind failed");

  l = listen(s, QUEUE_SIZE);                  /* specify queue size */
  if (l < 0) fatal("listen failed");

. . .
```

Server code

84

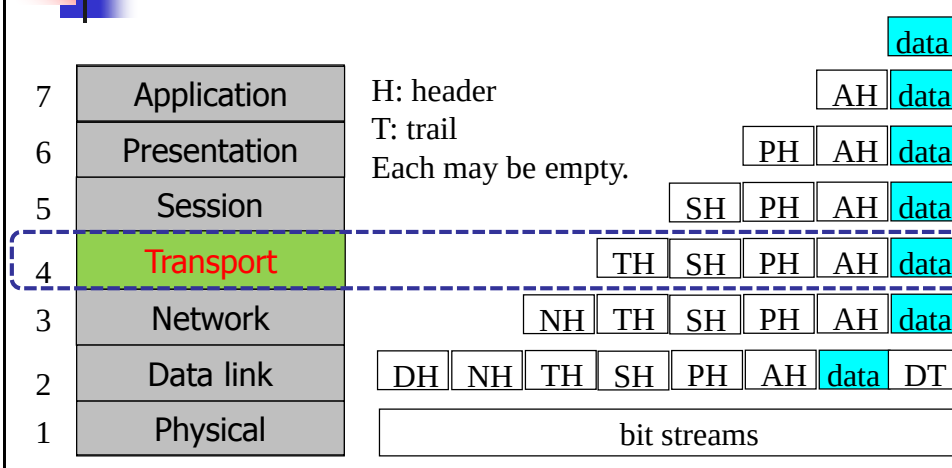
Example of Socket Programming: An Internet File Server (6)

```
. . .
/* Socket is now set up and bound. Wait for connection and process it. */
while (1) {
    sa = accept(s, 0, 0);                    /* block for connection request */
    if (sa < 0) fatal("accept failed");
    read(sa, buf, BUF_SIZE);                 /* read file name from socket */
    /* Get and return the file. */
    fd = open(buf, O_RDONLY);                /* open the file to be sent back */
    if (fd < 0) fatal("open failed");
    while (1) {
        bytes = read(fd, buf, BUF_SIZE); /* read from file */
        if (bytes <= 0) break;               /* check for end of file */
        write(sa, buf, bytes);               /* write bytes to socket */
    }
    close(fd);                               /* close file */
    close(sa);                               /* close connection */
  }
}
```

Server code

85

The Transport Layer



OSI Reference Model

86

End of Chapter 6

Questions?

Thank you!