

TCP/IP

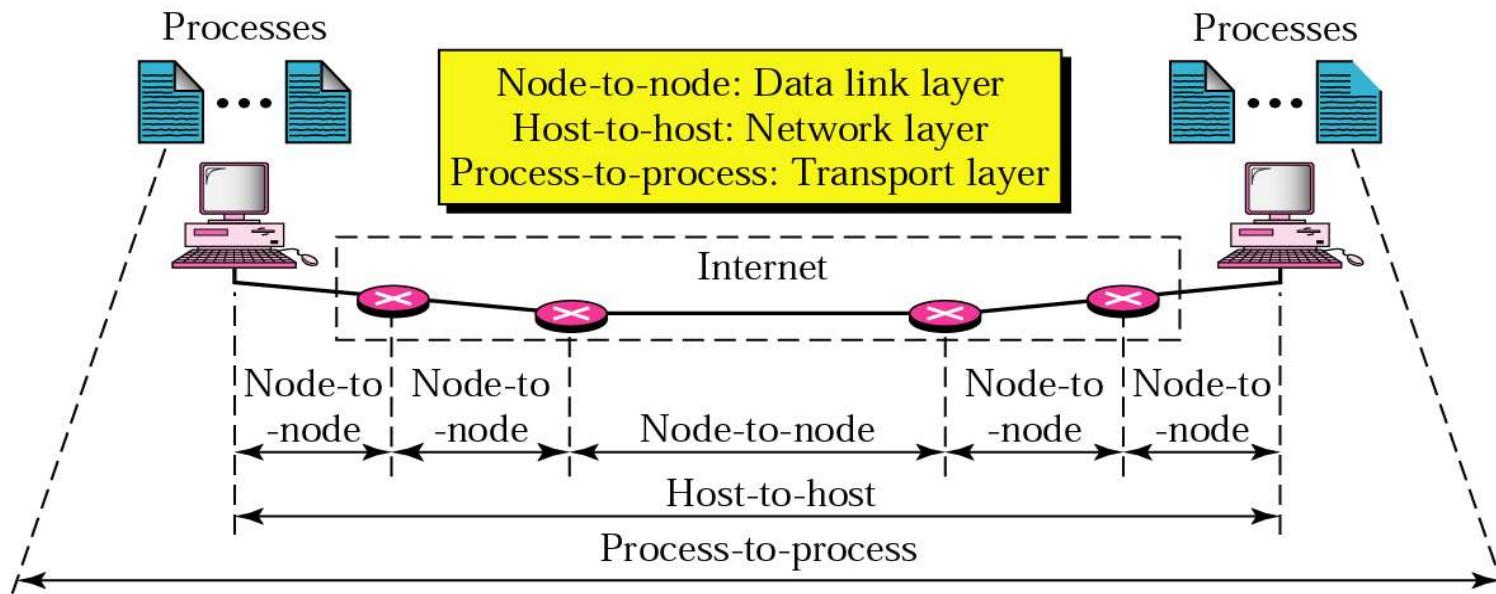
–Transmission Control Protocol

Transmission Control Protocol



프로세스간 전달 (1/4)

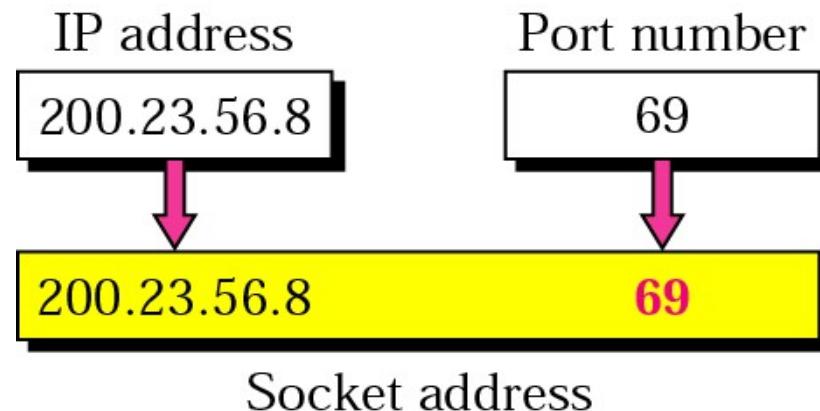
❖ 프로세스간 전달 형태



프로세스간 전달 (2/4)

◆ 소켓 주소

- ▶ IP 주소와 포트 번호의 조합
- ▶ 전송층 프로토콜은 한 쌍의 클라이언트 소켓 주소와 서버 소켓 주소의 소켓 주소들을 필요





포트의 개요

◆ 개요

- ▶ TCP나 UDP를 사용할 때의 상호간의 이동통로
- ▶ 상 하위 프로토콜은 같은 포트번호를 사용한다
- ▶ 클라이언트는 서버와 접속할 때 포트번호를 할당 받아 사용한다.
- ▶ 한 컴퓨터에서 여러 서비스를 동시에 사용할 수 있는 것은 포트 번호가 있기 때문 0~65534



포트

- ▶ well-known (reserved) port : 0~1023
IANA 가 잘 알려진 인터넷 서비스(예, ftp, smtp, http)에게 할당한 port 번호
- ▶ 유닉스 /etc/services 파일에 기술되어 있음
- ▶ registered port : 1024~49151
- ▶ ephemeral(dynamic) port : 49152~65534
- ▶ server
 - ✖ bind()를 통해 자신의 port 번호를 명시함
port 번호를 “0”으로 지정한 경우 시스템에서 임의의 포트 번호를 할당 해준다.
- ▶ client
 - ✖ 자신의 port 번호를 알고 있을 필요가 없으며, connect() 시에 임의의 포트번호를 할당 받는다.



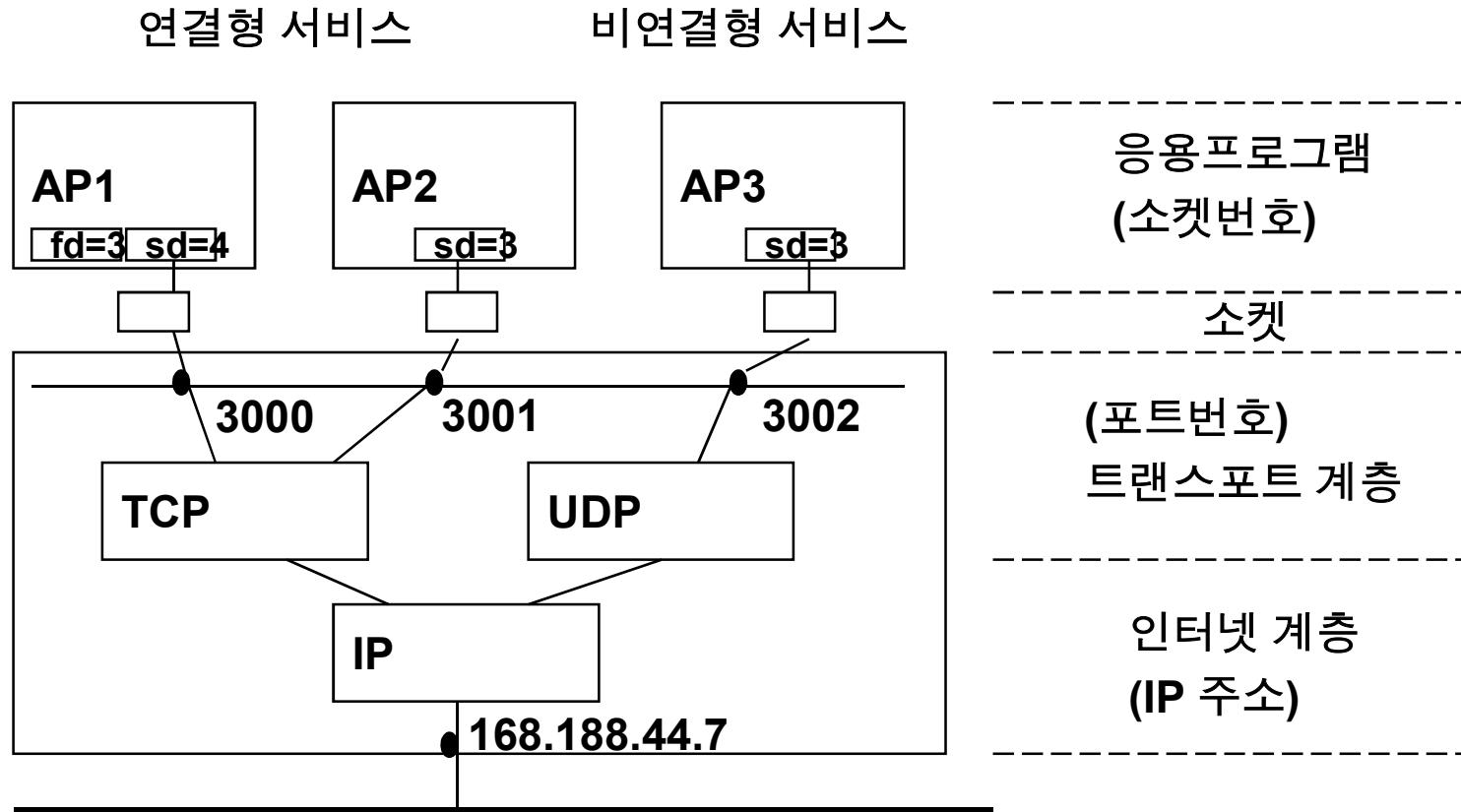
서비스 기본 포트 번호

❖ 기본 서비스들

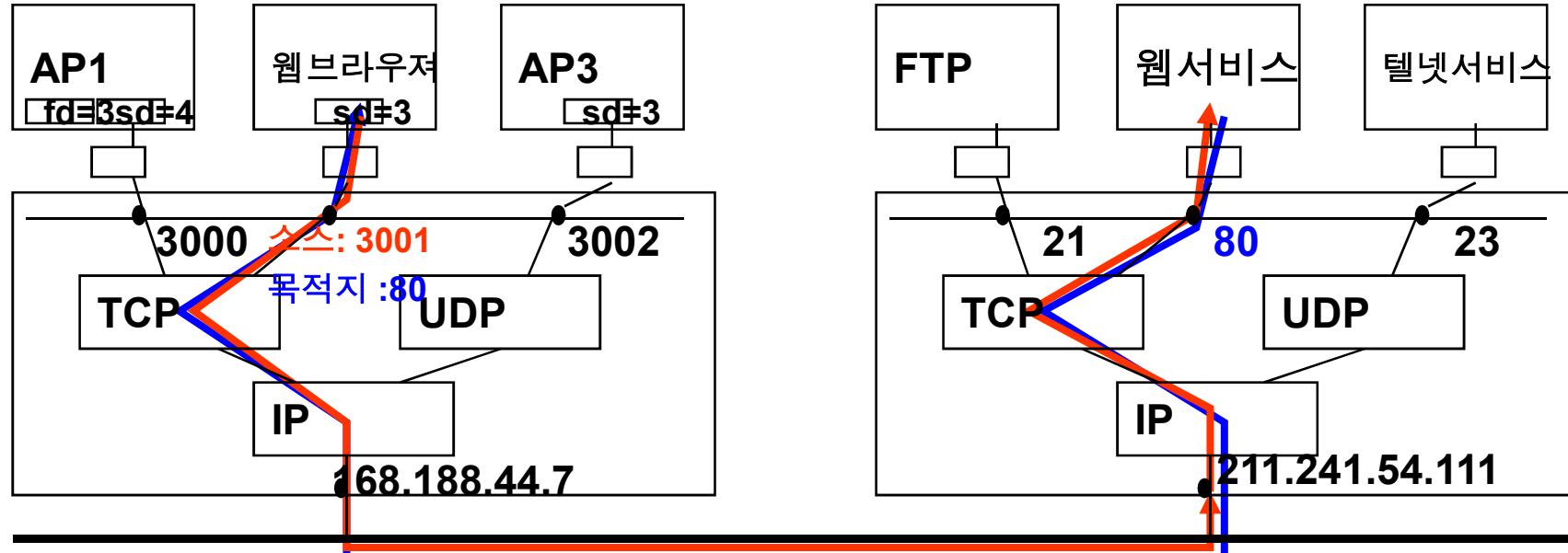
- ▶ Echo service : 단순히 클라이언트의 문자를 반복으로 보내주는 서비스
- ▶ Daytime service : 서버의 날짜와 시간을 알려주는 서비스
- ▶ Telnet : 23번을 기본포트로 원격로그인 한다.
- ▶ FTP : 21번을 기본 포트로 로그인 한다.
- ▶ HTTP : 웹 브라우저는 80번 포트를 기본으로 사용하고 있다.

Service	Port
Echo Service	7
Daytime	9
Telnet(remote login)	23
FTP	21
HTTP(web)	80

파일 번호와 포트 번호 IP주소의 관계



소스 포트와 목적지 포트



- ❖ 1) 서버의 IP주소 211.241.54.111로 접근
- ❖ 2) 목적지포트 80으로 서비스 요청
- ❖ 3) 요청한 클라이언트 3001 포트로 응답

프로세스간 전달 (3/4)

❖ 다중화 및 역 다중화

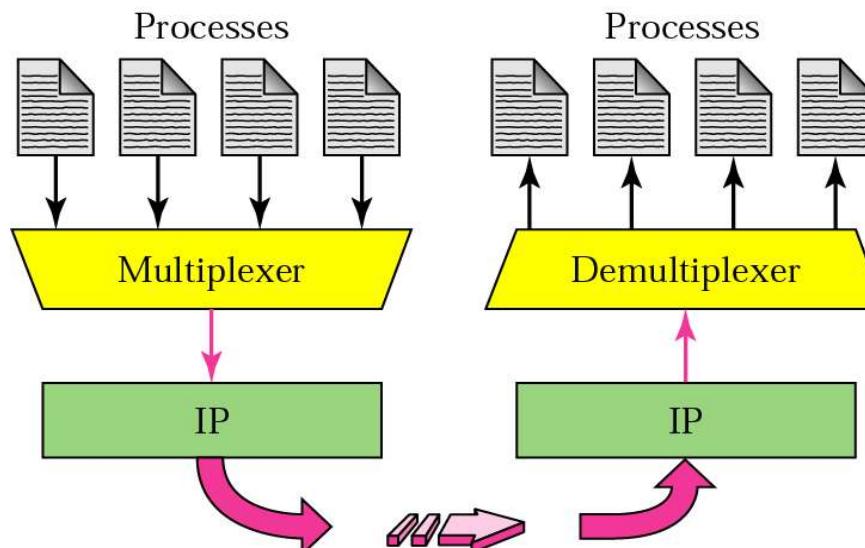
▶ 다중화

↳ 다수의 프로세스와 하나의 전송 층 프로토콜(UDP 또는 TCP)

▶ 역 다중화

↳ 수신 측에서의 일 대 다의 관계

↳ 오류 검사와 헤더를 제거 후 포트 번호에 근거하여 적절한 프로세스에 전달





프로세스간 전달 (4/4)

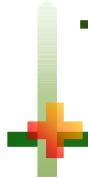
❖ 신뢰성 대비 신뢰성

▶ 신뢰성

- ↳ 전송 층에서는 흐름 제어와 오류 제어를 구현하여 신뢰성 있는 전송층 프로토콜을 사용
- ↳ 느리고 복잡한 서비스 제공

▶ 비 신뢰성

- ↳ 응용 프로그램이 자신의 흐름 제어와 오류 제어 기법을 가진 경우
- ↳ 빠른 서비스가 필요하거나 서비스 특성상 흐름 제어 및 오류 제어가 불필요할 경우(실시간 응용) 사용



TCP 개요

◆ 개요

- ▶ 연결 위주 전송 방식 (**Connection-Oriented**)
- ▶ 전송 데이터의 신뢰성과 무결성을 보장하기 위한 것
- ▶ 신뢰성 있는 전송
- ▶ TCP 연결경로를 통하여 데이터를 전송하고 이에 대한 응답(ack)을 받음으로써 그 데이터가 올바르게 전송되었음을 보장
- ▶ 데이터 흐름제어나 순서제어를 하기 때문에 전송속도가 늦다.
- ▶ 양방향 연결 (**가상회선 연결방식**)
- ▶ 통신 버퍼 이용 전달 및 특별한 구조가 없는 바이트 스트림 형태로 수송



TCP에서 사용하는 포트

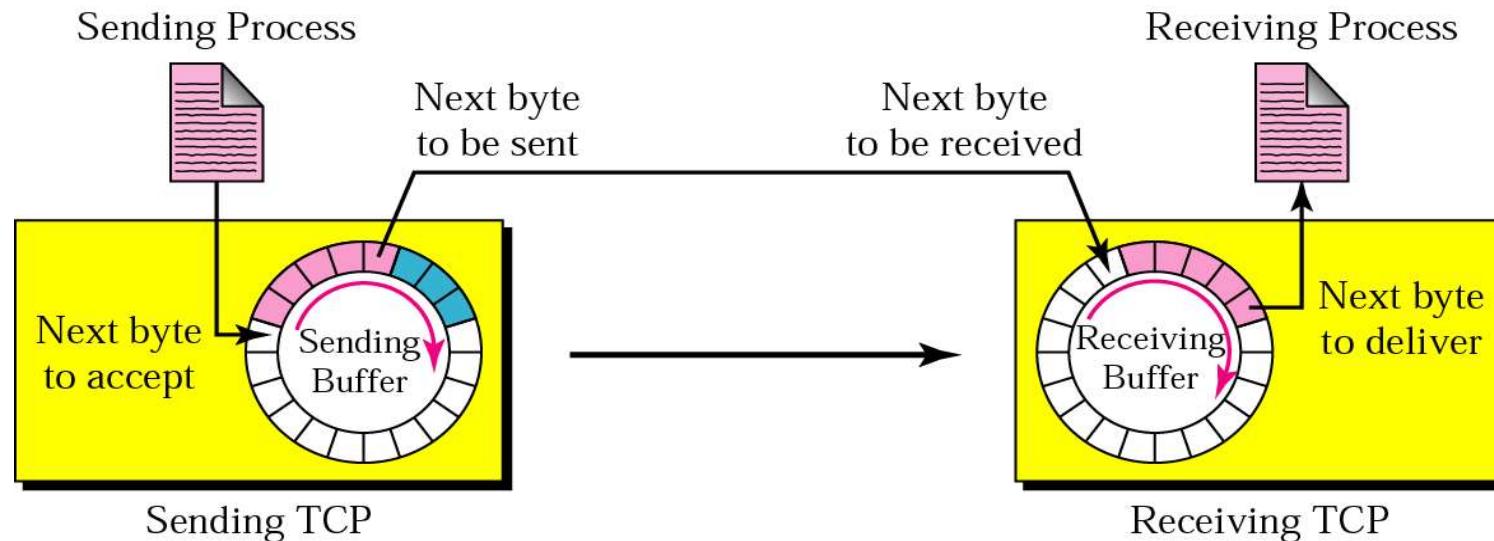
Port	protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (data connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

<Table. TCP에서 사용하는 알려진 포트들>

TCP 송수신 버퍼

❖ 송신과 수신 버퍼

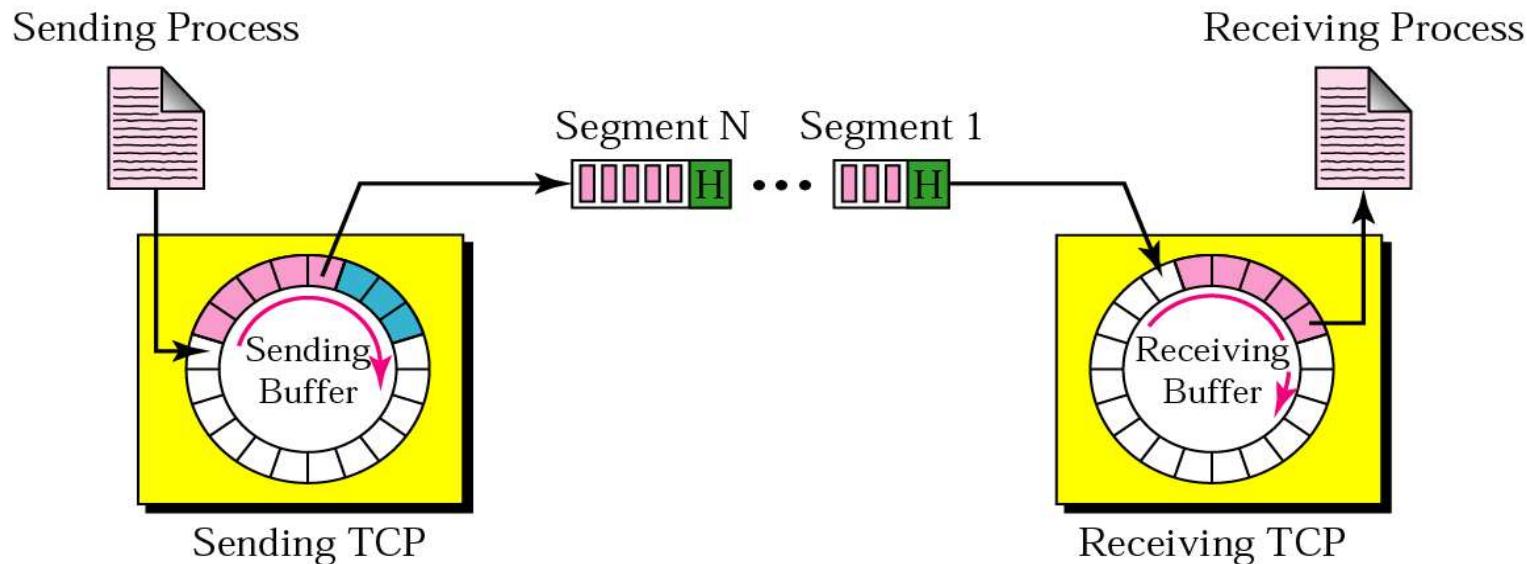
- ▶ 송/수신 프로세스가 똑같은 속도로 데이터를 만들고 처리할 수 없기 때문에 버퍼가 필요
- ▶ 송신 및 수신 버퍼를 사용



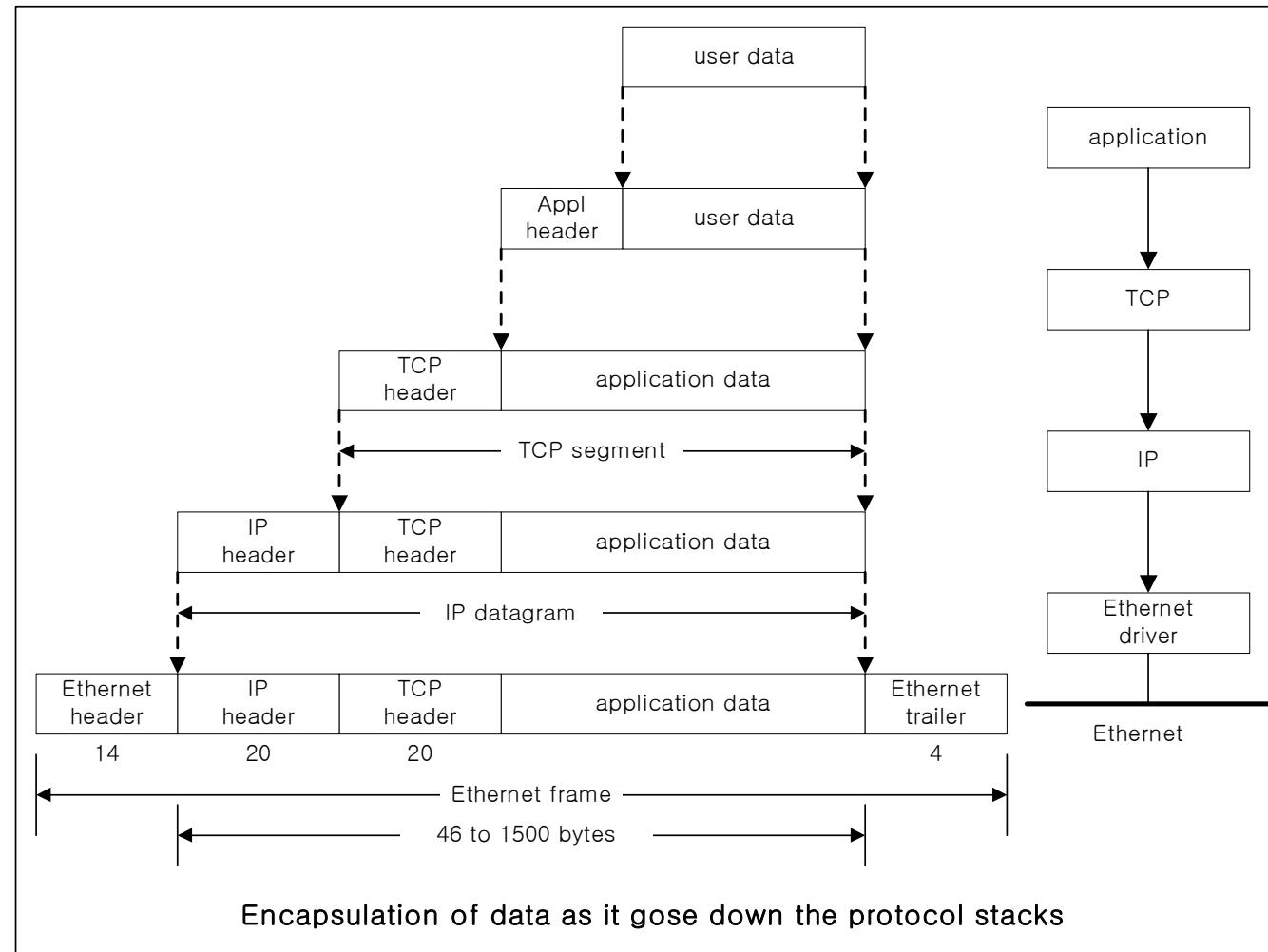
바이트와 세그먼트

❖ 바이트와 세그먼트

- ▶ 세그먼트라고 하는 패킷으로 **다수의 바이트를 묶어서 그룹화**
- ▶ 각 세그먼트에 헤더를 붙이고 전송을 위해 **IP 계층으로 전달**
- ▶ 세그먼트들은 **IP 데이터그램으로 캡슐화되고 전송**



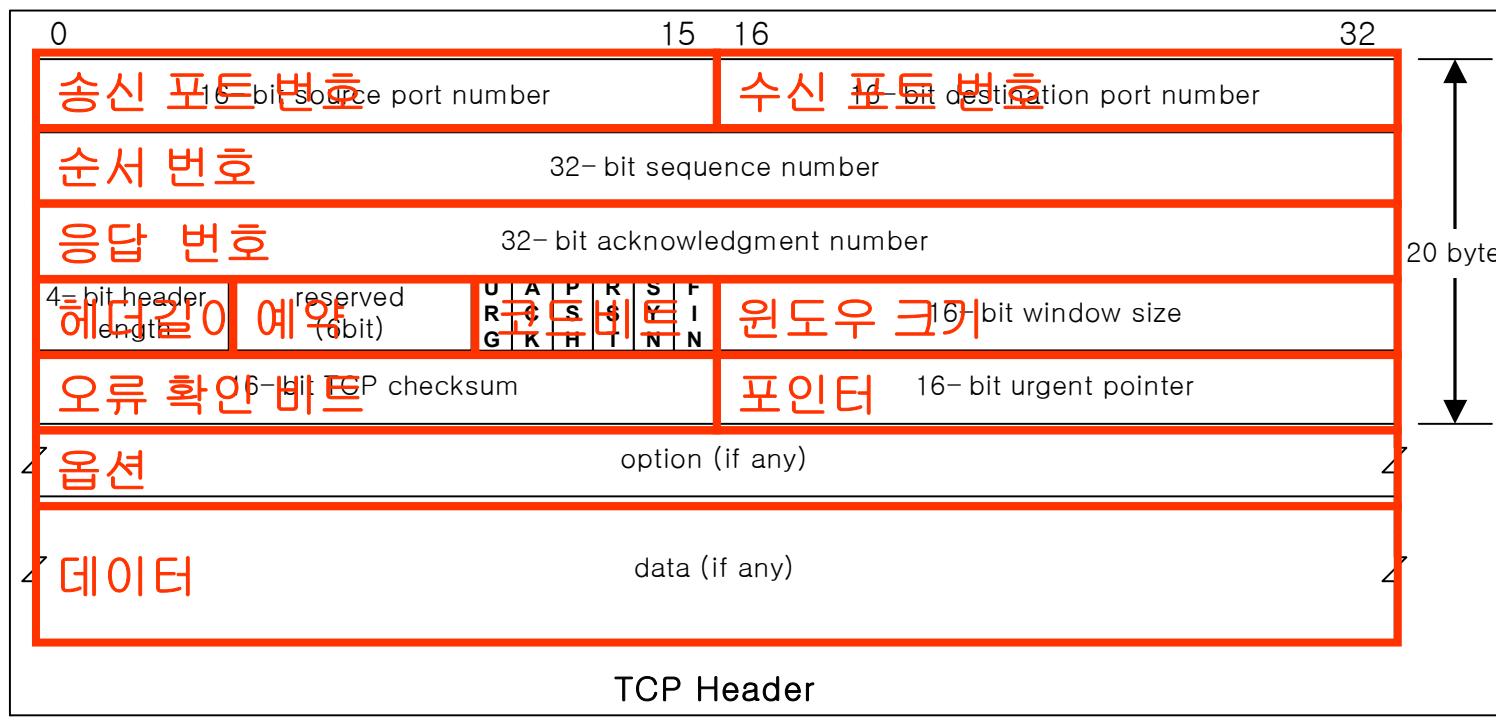
TCP 캡슐화 위치



TCP Header (1/2)

❖ TCP 헤더

- ▶ 송신 포트와 수신 포트 : 요청할 포트와 받을 포트
 - ↳ Ex) 80, 3001
- ▶ 순서번호 : 송신 시 차례로 부여되는 번호 (순서대로 전송)
- ▶ 응답번호 : 수신할 때 순서가 부여되는 번호



TCP Header (2/2)

▶ 제어

- ↳ 여섯 개의 다른 제어 비트 또는 플래그를 정의
- ↳ 흐름 제어, 연결 설정과 종료, 데이터 전송 모드를 지원

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push

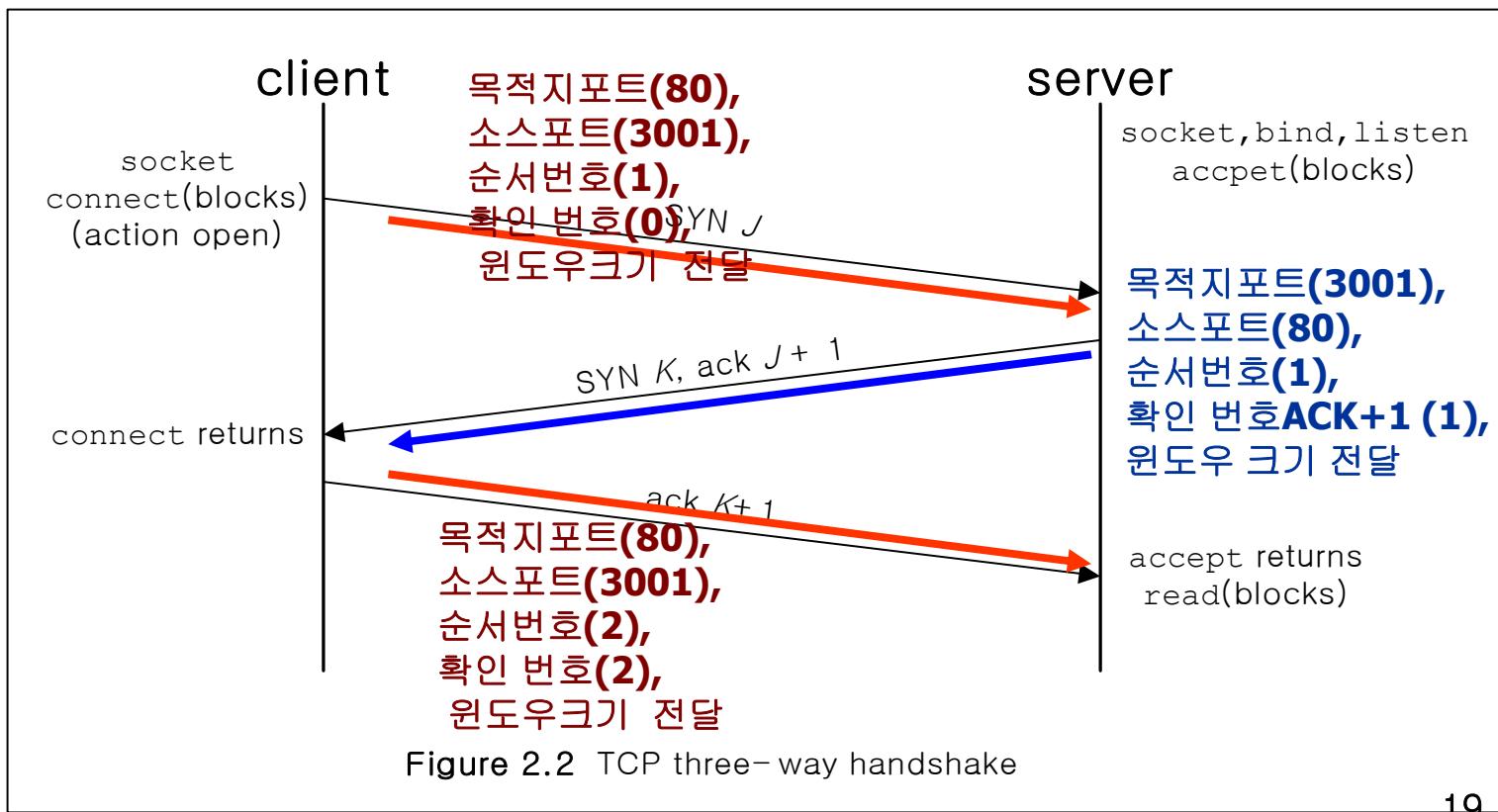
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

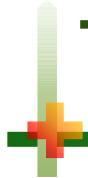
URG	ACK	PSH	RST	SYN	FIN
-----	-----	-----	-----	-----	-----

3-way handshake

❖ TCP의 연결 확정 과정이라 불린다.

- ▶ 1) 클라이언트가 서버로 SYN(Synchronize)신호 보냄
- ▶ 2) 서버가 응답으로 SYN, ACK 신호를 보낸다.
- ▶ 3) SYN, ACK를 정상 수신하면 응답으로 다시 ACK를 보낸다





TCP 플래그

TCP 플래그	Dump 약자	Description
SYN	"S"	TCP 연결의 첫 번째 부분인 세션 연결 요청
ACK	"ack"	송신자로부터 데이터 영수증으로 받음을 알림. 다른 플래그와 함께 결합하여 사용. 부가적 기능
FIN	"F"	수신호스트로의 연결을 정상적으로 마칠때 사용
RESET	"R"	수신호스트와의 연결을 즉시 종료하기 위해 사용
PUSH	"p"	응용 소프트웨어 데이터를 송신호스트로 "전송" 데이터 수신시에 버퍼가 채워지기를 기다리지 않고, 어플리케이션으로 전달(telnet) - 대역폭의 효율도바는 응답성에 초점
URGENT	"urg"	"긴급한" 데이터가 다른 데이터에 우선함
Placeholder	"."	연결 과정이 SYN, FIN, RESET이나 PUSH 플래그를 가지고 있지 않다면, 마침표(.)가 목적지 포트후에 표시



3-way handshake

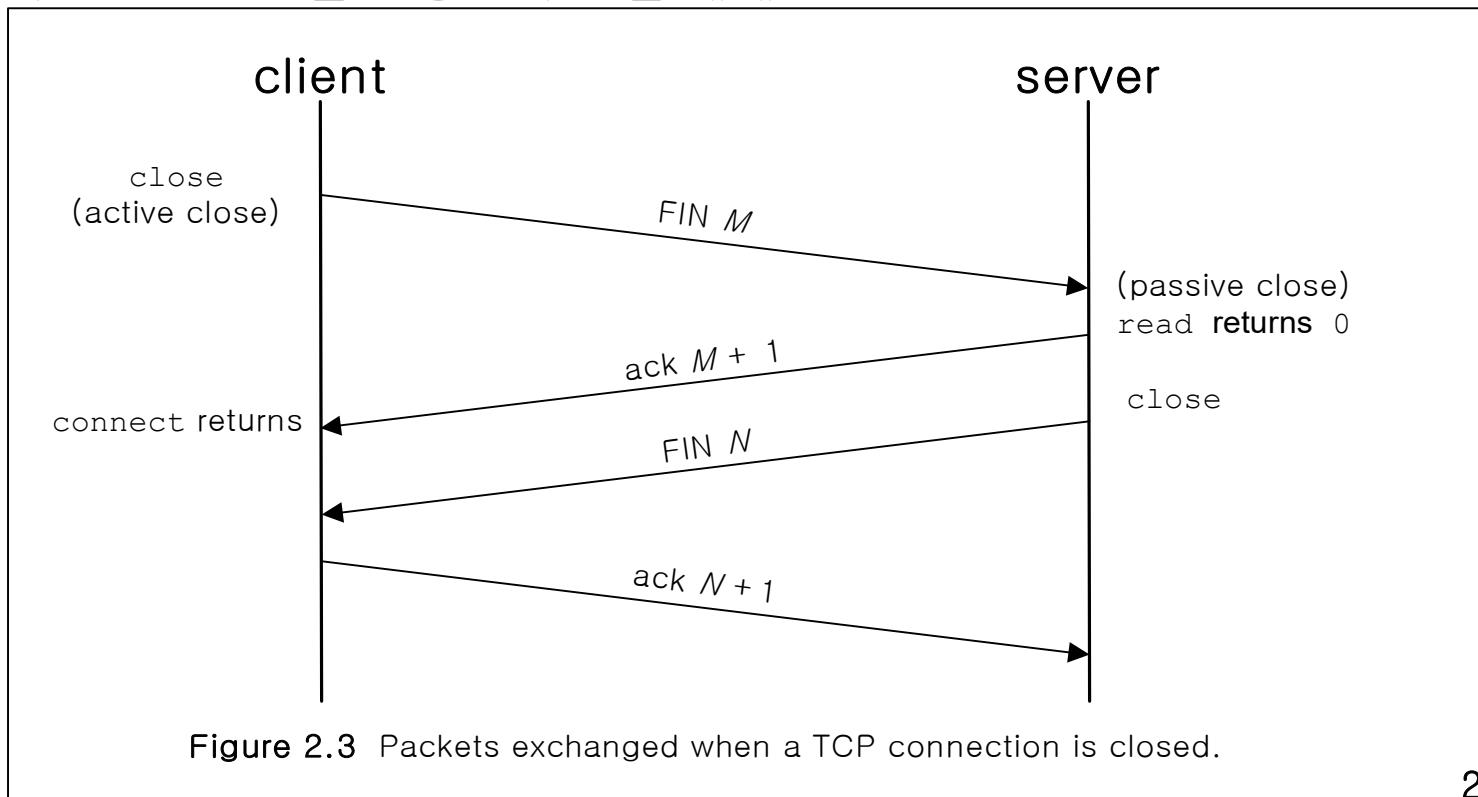
❖ 연결 재설정

- ▶ 현재 연결이 훼손된 경우 요청
- ▶ 훼손 원인
 - ↳ 한쪽 TCP가 존재하지 않는 포트에 대해 연결 요청을 받은 경우
 - ↳ 한쪽 TCP가 비정상적인 상황으로 인해 연결 중지를 원할 경우
 - ↳ 한쪽 TCP가 다른 쪽 TCP가 오랫동안 유휴 상태인 것을 발견한 경우 연결을 없애기 위해 RST 세그먼트 전송

연결 종료과정

❖ 연결 종료 과정

- ▶ 1) 클라이언트에서 순서번호를 실어 FIN 전송
- ▶ 2) 클라이언트 순서번호에 1을 더하고 확인번호 전송
- ▶ 3) 서버의 순서번호를 실어 FIN 전송
- ▶ 4) 확인번호 $N+1$ 을 전송하여 연결 해제

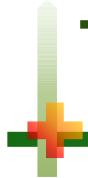


실습: ethereal을 이용한 패킷 살펴보기

❖ 목 표

- ▶ target에서 telnet 실행 후 호스트에서 연결수립 및 종료과정을 살펴보자.

No. .	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.123.110	192.168.123.102	TCP	1032 > telnet [SYN] Seq=2743608351 Ack=0 Win=5840 Len=0
2	0.000076	192.168.123.102	192.168.123.110	TCP	telnet > 1032 [SYN, ACK] Seq=2969044057 Ack=2743608352 Win=5840 Len=0
3	0.002270	192.168.123.110	192.168.123.102	TCP	1032 > telnet [ACK] Seq=2743608352 Ack=2969044058 Win=5840 Len=0



TCP 상태 천이도 (1/3)

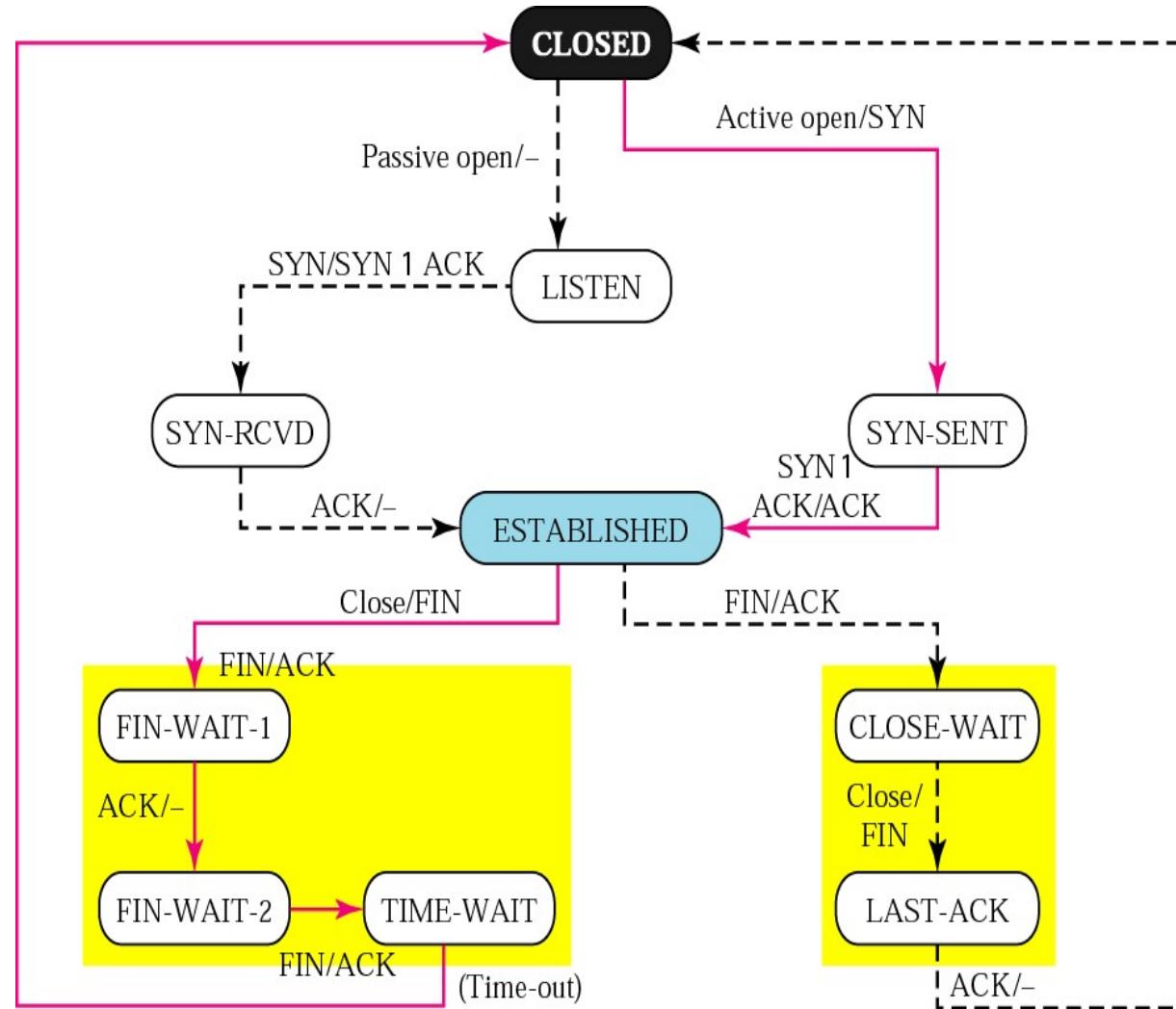
❖ 개요

- ▶ TCP 소프트웨어는 유한 상태 기계(finite state machine, FSM)로 구현됨
 - ▶ 유한 상태 기계는 제한된 상태의 수를 가지고 동작
 - ▶ 사건에 따라 상태는 변경됨
-
- ▶ netstat 명령을 통해 TCP 상태를 확인할 수 있다.
 - ↳ \$ netstat -nt

```
[root@localhost ~]# netstat -nt
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp      1      0 192.168.1.56:55820      152.3.220.166:80      CLOSE_WAIT
tcp      1      0 192.168.1.56:44454      208.209.50.16:80      CLOSE_WAIT
tcp      0    132 ::ffff:192.168.1.56:22  ::ffff:192.168.1.55:1435  ESTABLISHED
```

TCP 상태 천이도 (2/3)

Shape	Purpose
타원	상태
화살표 있는 직선	상태 천이
점선	서버
실선	클라이언트





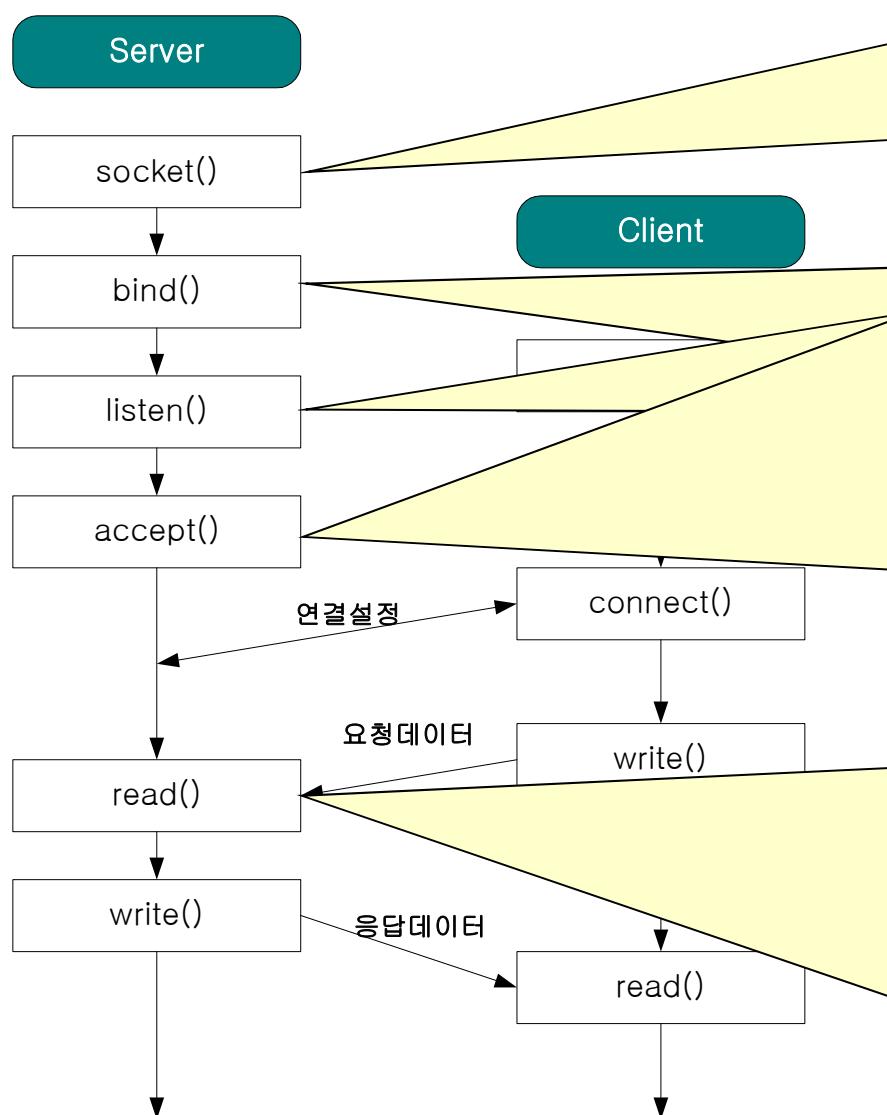
TCP 상태 천이도 (3/3)

❖ TCP 상태

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

<Table. TCP의 상태>

유닉스 프로그램 관점의 연결 시나리오



메시저 예제서자

Listen() (sockfd backlog)

accept(sockfd *newr *addr)

Read()

서버의 패킷 데이터를 읽어오기

Write()

데이터 보내기

한다.



Socket과 family의 조합

	AF_UNIX	AF_INET	AF_NS
SOCK_STREAM	사용가능	TCP	SPP
SOCK_DGRAM	사용가능	UDP	IDP
SOCK_RAW	사용불가	IP	사용가능
SOCK_SEQPACKET	사용불가	사용불가	SPP



실습: simple_tcp_server

❖ 목 표

- ▶ 간단한 TCP 예제를 살펴본다.
- ▶ 소켓을 생성하는 방법을 배운다.
- ▶ ethereal 프로그램으로 3way handshake 과정과 TCP 헤더를 살펴보자
- ▶ 소켓 인터페이스는 향후에 자세히 살펴보도록 한다.

실습: simple_tcp_server

❖ inet.h

```
1 - ****
2 * Filename: inet.h
3 * Title: Simple TCP Server
4 * Desc: Simple TCP Server
5 ****
6 #include <stdio.h>
7 #include <sys/types.h>
8 #include <sys/socket.h>
9 #include <netinet/in.h>
10 #include <arpa/inet.h>
11
12 #define SERV_UDP_PORT 7000 /* UDP Server port */
13 #define SERV_TCP_PORT 7000 /* TCP Server port */
14 #define SERV_HOST_ADDR "192.168.0.2" /* Server IP address */
15
16 char *pname;
```

실습: simple_tcp_server

❖ server.c

```
1 - ****
2 * Filename: server.c
3 * Title: Simple TCP Server
4 * Desc: Simple TCP Server
5 ****
6 #include "inet.h"
7
8 int main ( int argc, char* argv[] ) {
9     int sockfd, newsockfd, clilen;
10    struct sockaddr_in cli_addr;
11    struct sockaddr_in serv_addr;
12    char buff[30];
13    pname = argv[0];
14
15    if ( (sockfd = TODO: 스트림 소켓 지정하기 ) < 0 ) {
16        puts( "Server: 스트림 소켓을 열 수 없습니다." );
17        exit(1);
18    }
19
20    /* 서버의 주소를 등록하여 클라이언트가 접속 가능하게 한다. */
21    bzero((char *) &serv_addr, sizeof(serv_addr));
22    serv_addr.sin_family = AF_INET;
23    serv_addr.sin_addr.s_addr = htonl( INADDR_ANY );
24    serv_addr.sin_port = htons( SERV_TCP_PORT );
25
26    if ( TODO: 어드레스 바인딩 하기 ) < 0 ) {
27        puts( "Server: 지역어드레스로 바인딩 할 수 없습니다." );
28        exit(1);
29    }
30
```

실습: simple_tcp_server

❖ server.c (cont.)

```
31     TODO: 동시 접속 큐 설정하기 - 5로 설정
32     clilen = sizeof( cli_addr );
33     newsockfd = TODO: 클라이언트 요청을 받아들이기 위한 Accept ;
34
35     if ( newsockfd < 0 ) {
36         puts("Server: accept error!");
37         exit(1);
38     }
39
40     if ( read(newsockfd, buff, 20) <= 0 ) {
41         puts( "Server: readn error!" );
42         exit(1);
43     }
44     printf("Server: Received String = %s \n", buff);
45     close( sockfd );
46     close( newsockfd );
47
48     return 0;
49 }
```

실습: simple_tcp_server

❖ client.c

```
1 - ****
2 * Filename: client.c
3 * Title: Simple TCP Server
4 * Desc: Simple TCP Server
5 ****
6 #include "inet.h"
7
8 int main ( int argc, char *argv[] ) {
9     int sockfd;
10    struct sockaddr_in serv_addr;
11    pname = argv[0];
12
13    /* 연결하고자 하는 서버의 주소 및 포트번호를 구조체 serv_addr에 채운다 */
14    bzero((char *) &serv_addr, sizeof(serv_addr));
15    serv_addr.sin_family = AF_INET;
16    serv_addr.sin_addr.s_addr = inet_addr( SERV_HOST_ADDR );
17    serv_addr.sin_port = htons( SERV_TCP_PORT );
18
19    /* TCP Socket Open */
20    if ((TODO: 스트림 소켓 열기) < 0) {
21        puts("Client: 소켓을 열수 없습니다.");
22        exit(1);
23    }
24
25    /* 서버에 접속 */
26    if (( TODO: 서버에 접속하기 ) < 0) {
27        puts("Client: 서버에 연결할 수 없습니다.");
28        exit(1);
29    }
30 }
```

실습: simple_tcp_server

❖ client.c (Cont.)

```
31 -     if ( write( sockfd, "Hello world!~", 20) < 20 ) {
32         puts( "Client: writen error" );
33         exit(1);
34     }
35
36     close( sockfd );
37 }
```



흐름제어(Flow Control)

❖ 흐름 제어(flow control)

- ▶ 목적지로부터 확인 응답을 받기 전에 발신자가 송신할 수 있는 데이터의 양을 정의
- ▶ TCP는 응용 프로그램으로부터 전달된 데이터 버퍼에 부여되어서 송신할 준비가 되어 있는 윈도우를 정의
- ▶ TCP는 슬라이딩 윈도우 프로토콜에 정의된 최대한 많은 양의 데이터를 송신

❖ 슬라이딩 윈도우 (sliding window protocol)

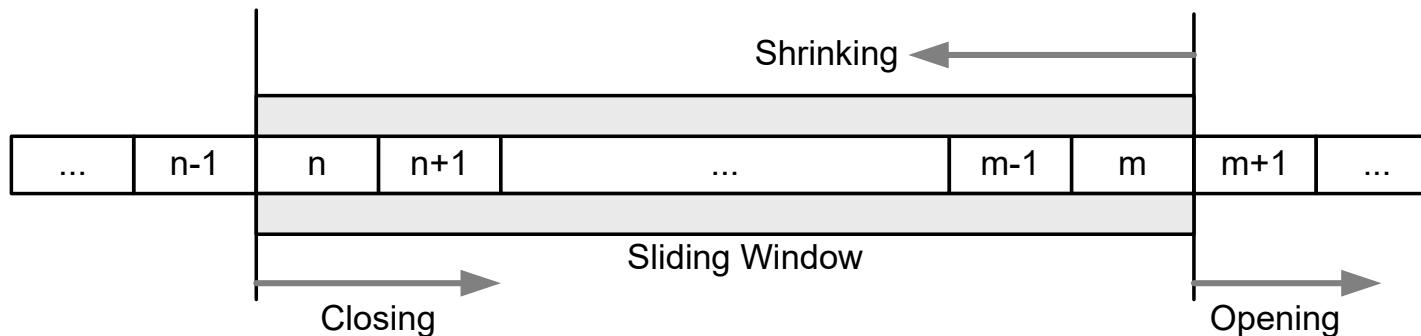
- ▶ 슬라이딩 윈도우는 목적지가 데이터로 넘쳐나지 않도록 하기 위하여 데이터의 흐름을 제어할 뿐만 아니라 전송을 좀 더 효과적으로 만들기 위하여 사용된다. TCP의 슬라이딩 윈도우는 바이트 단위로 제어된다.

Sliding window (1/3)

- ❖ 슬라이딩 윈도우 프로토콜

- ▶ 윈도우 크기는 수신 윈도우(receiver window: rwnd)와 혼잡 윈도우(congestion window: cwnd)중에서 작은 값으로 정해진다.
- ▶ 윈도우는 수신측에 의해서 열릴 수 있으며 또한 닫힐 수 있지만, 축소될 수는 없다.
- ▶ 목적지는 윈도우 축소가 일어나지 않는 한 언제든지 확인응답을 전송할 수 있다.

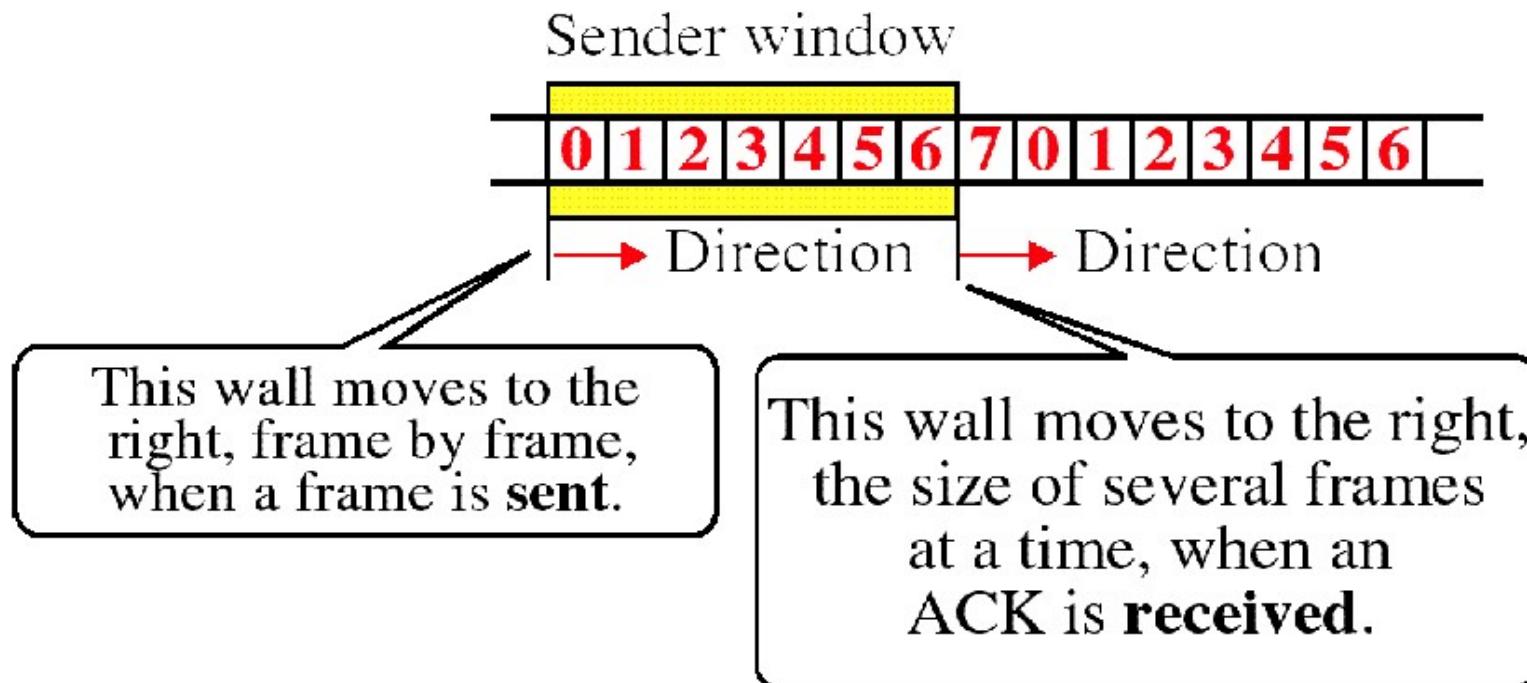
Window Size = $\min(rwnd, cwnd)$



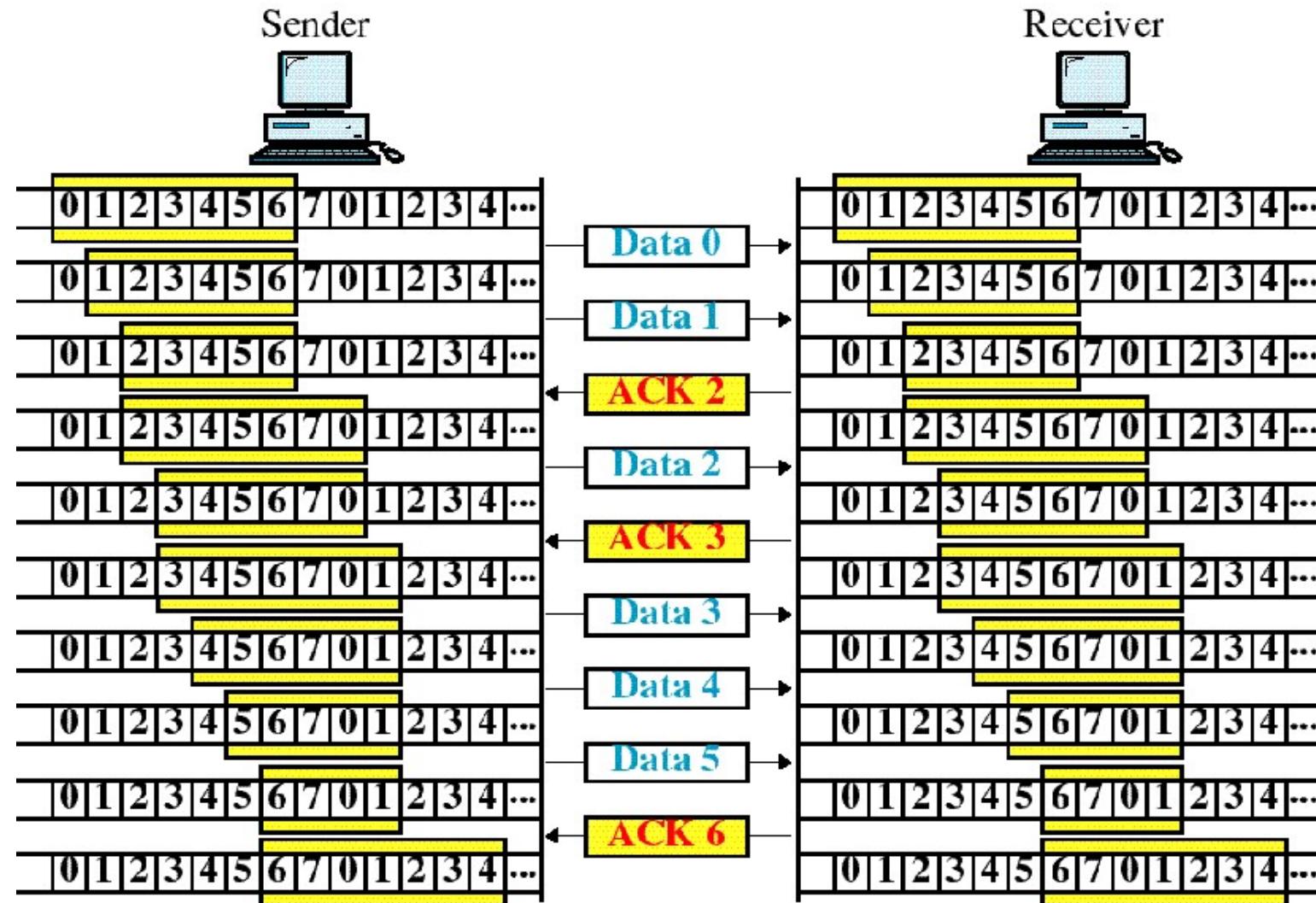
<Fig. 슬라이딩 윈도우>

Sliding window (2/3)

- ▶ 개념적으로 송신기의 슬라이딩 윈도우는 데이터프레임이 보내지면 왼쪽으로부터 줄어든다.
- ▶ 송신기의 슬라이딩 윈도우는 확인응답 ACK을 받으면 오른쪽으로 확장한다.



Sliding window (3/3)



<Fig. 슬라이딩 윈도우 예제>



Nagle 알고리즘

❖ 개요

- ▶ 너무 작은 패킷들이 네트워크에서 마구 훌러 다니면 실제 전달한 내용 보다 더 커지는 비효율이 발생한다.
- ▶ 한번에 한 바이트씩 데이터를 천천히 발생하는 경우 어리석은 윈도우 신드롬이 발생 한다.
- ▶ 이러한 단점을 해결하기 위해 **TCP**는 어느 정도 시간을 기다릴 것인가를 고려 해야 한다. → 네이글(Nagle) 알고리즘으로 해결(RFC 896)

❖ 알고리즘의 동작

- ▶ 1. 송신 **TCP**는 단지 한 **byte**라 하더라도 송신 응용 프로그램으로부터 수신하는 첫 데이터를 세그먼트로 만든 후 전송한다.
- ▶ 2. 첫 번째 세그먼트를 전송한 후에, 송신 **TCP**는 수신 **TCP**로부터 **ack**를 받거나 또는 충분한 데이터가 출력 버퍼에 저장되기 전까지는 데이터를 출력 버퍼에 저장한다. 위 둘 중 한 경우가 발생하면 세그먼트를 전송한다.
- ▶ 3. 나머지 전송 기간 동안 2번째 단계를 반복한다.



수신 측에서의 신드롬

◆ 수신 측 문제

- ▶ 수신측에서 짧은 시간 동안에 도착한 데이터 패킷들에 대해서 일일이 즉시 ACK를 보내면 헤더만 딸린 패킷을 보내게 되기 때문에 앞선 문제점과 비슷한 상황이 발생할 수 있다.
- ▶ 이때는 이미 도착한 부분만 ACK를 보내는 경우 약간 보류하여 전송하는 ACK 기법을 지연 ACK(Delayed ACK)라고 한다.
- ▶ 대부분의 OS에서는 200msec 이내의 시간을 사용하고 있다.



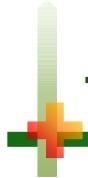
TCP autotuning

❖ 개념

- ▶ 몇몇 OS에서는 TCP 소켓 버퍼의 크기를 전송량에 비례하여 자동으로 조정하는 기능이 제공된다.
- ▶ 초기 지정된 크기는 이후 전송량에 비례해서 줄거나 늘거나 할 수 있다.

❖ TCP autotuning 리눅스 커널 설정

- ▶ net.ipv4.tcp_moderate_rcvbuf 수신버퍼에 TCP autotuning 적용
 - ▶ net.ipv4.tcp_rmem 송신 버퍼 크기 값
 - ▶ net.ipv4.tcp_wmem 수신 버퍼 크기 값
 - ▶ net.core.rmem_max 지정 가능한 송신 버퍼 제한 값
 - ▶ net.core.wmem_max 지정 가능한 수신 버퍼 제한 값
-
- ▶ # sysctl net.ipv4.rmp_rmem
net.ipv4.tcp_rmem = 4096 87380 2097152
 - ▶ 소켓 옵션 중 SO_RCVBUF, SO_SNDBUF 옵션으로 지정 가능하다.



오류제어 (1/2)

❖ 오류제어

- ▶ TCP는 신뢰성 있는 전송 층 프로토콜
- ▶ 손상된 세그먼트, 유실된 세그먼트, 순서가 어긋난 세그먼트, 중복된 세그먼트들을 찾아내기 위한 기법 포함
- ▶ 사용 도구
 - ↳ 검사합 (Checksum)
 - ↳ 확인 응답
 - ↳ 시간 종료(time-out)

❖ 유실 또는 손상 세그먼트

- ▶ 손상된 세그먼트는 최종 목적지에 의해 폐기
- ▶ 유실된 세그먼트는 어떤 중간 노드에 의해 폐기



오류제어 (2/2)

❖ 중복 세그먼트

- ▶ 확인 응답이 시간 종료 전에 도착하지 않을 때 발신지 TCP에 의해 생성
- ▶ 목적지 TCP는 동일 순서번호를 가진 또 다른 세그먼트 수신 시 그 세그먼트를 폐기

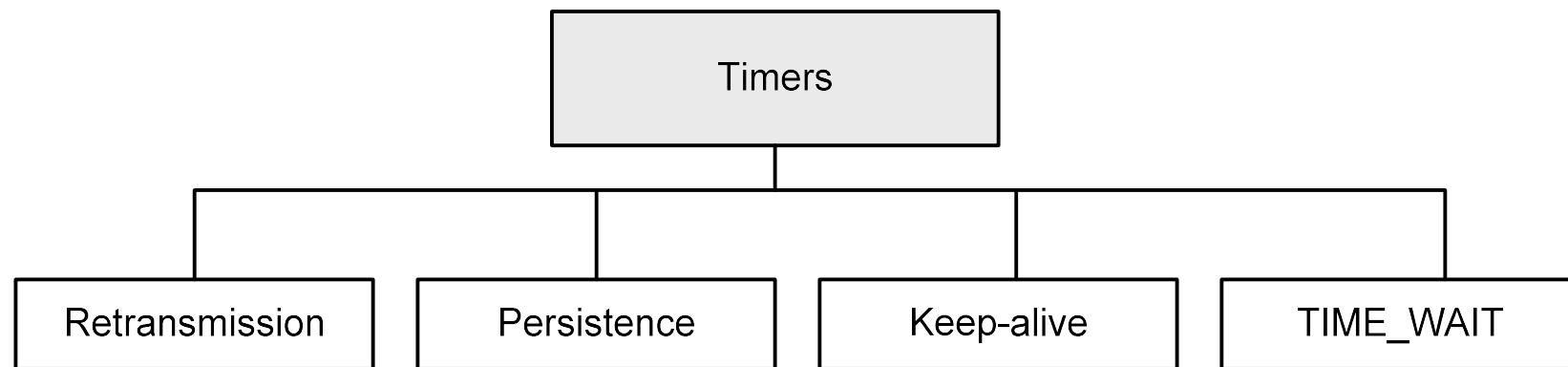
❖ 순서 없는 세그먼트(out-of-order segment)

- ▶ 목적지 TCP는 순서에 어긋나게 도착한 세그먼트 앞의 세그먼트 도착 전 까지 확인 응답 지연
- ▶ 세그먼트의 타이머 만기 시 재전송

TCP 타이머

❖ TCP 타이머의 종류

▶ TCP의 동작을 순조롭게 수행하기 위해 다음과 같은 타이머가 필요하다.



<**Fig. TCP 타이머의 종류**>

TCP에서의 타이머

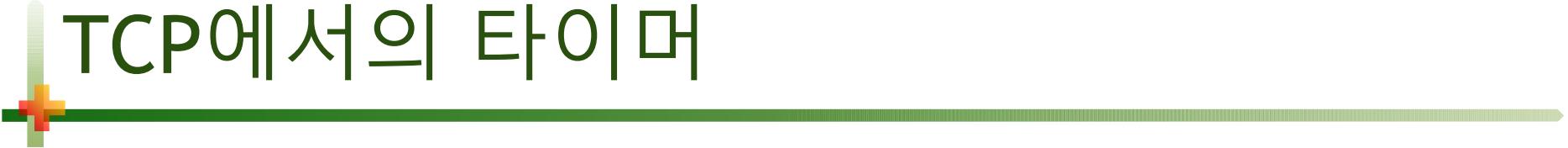
❖ 재전송 타이머(Retransmission timer)

- ▶ 세그먼트의 확인 응답을 기다리는 시간을 처리
- ▶ 타이머 종료 전 확인 응답을 수신 시 타이머는 제거
- ▶ 타이머 종료까지 확인 응답 미 수신 시 세그먼트는 재전송, 세그먼트는 다시 초기화
- ▶ RTO (Retransmission time-out) 값으로 설정
- ▶ 재전송 시간 계산
 - ↳ 고정된 재전송 시간 사용 불가
 - ↳ 동적인 재전송 시간 사용
 - ↳ 왕복시간(round-trip time, RTT)을 기반으로 동적으로 생성
 - ↳ 지수 백오프(Exponential Backoff) 기법을 사용하여 재전송 시간을 2배씩 늘려 가는 방법을 사용할 수 있다.

TCP에서의 타이머

❖ 영속 타이머(Persistence Timer)

- ▶ 문제: 확인 응답이 유실될 경우 송수신 TCP가 서로 상대의 송신을 기다리는 교착 상태 발생
- ▶ 해결책
 - ↳ 영속 타이머 사용
 - ↳ 크기 0을 갖는 윈도우를 수신 시 영속 타이머 작동
 - ↳ 영속 타이머 종료 시 프로브(probe)라는 특별 세그먼트 전송
 - ↳ 프로브는 수신 TCP에게 확인 응답의 손실과 재송신 요구를 경고
- ▶ 영속 타이머 값
 - ↳ 재전송 시간의 값으로 설정된다. 확인응답을 받지 못하면 값은 두배가 되고 초기화 된다.
 - ↳ 그 이후에는 윈도우가 재 개시될 때까지 매 60초마다 하나의 프로브 세그먼트를 전송한다.



TCP에서의 타이머

❖ 연결 유지 타이머(keep-alive timer)

- ▶ 두 개의 TCP 사이 오랫동안 유류 연결을 방지하기 위해 구현
- ▶ 연결 유지 타이머를 통해 클라이언트와의 연결 종료
 - ↳ 시간 종료 : 2시간
 - ↳ 프로브 세그먼트 송신 간격 : 75초 간격 10번
 - ↳ 10번을 전송하였음에도 불구하고 수신하지 못하였을 경우 상대방이 다운되었다고 간주하고 연결을 종료한다.

❖ 시간 대기 타이머(time-waited timer)

- ▶ 2(MSL) 타이머
- ▶ 연결 종료하는 동안 사용



그밖에 TCP 요소

- ❖ 데이터 밀어 넣기(**Pushing Data**)

- ▶ 원도우가 채워지는 것을 기다리지 않고 즉시 세그먼트를 전송 및 수신 요청

- ❖ 긴급 데이터(**Urgent Data**)

- ▶ 처리 순서에 관계없이 긴급히 처리 요청하기 위해 사용
 - ▶ URG 비트를 설정한 세그먼트 전송



UDP(User Datagram Protocol)

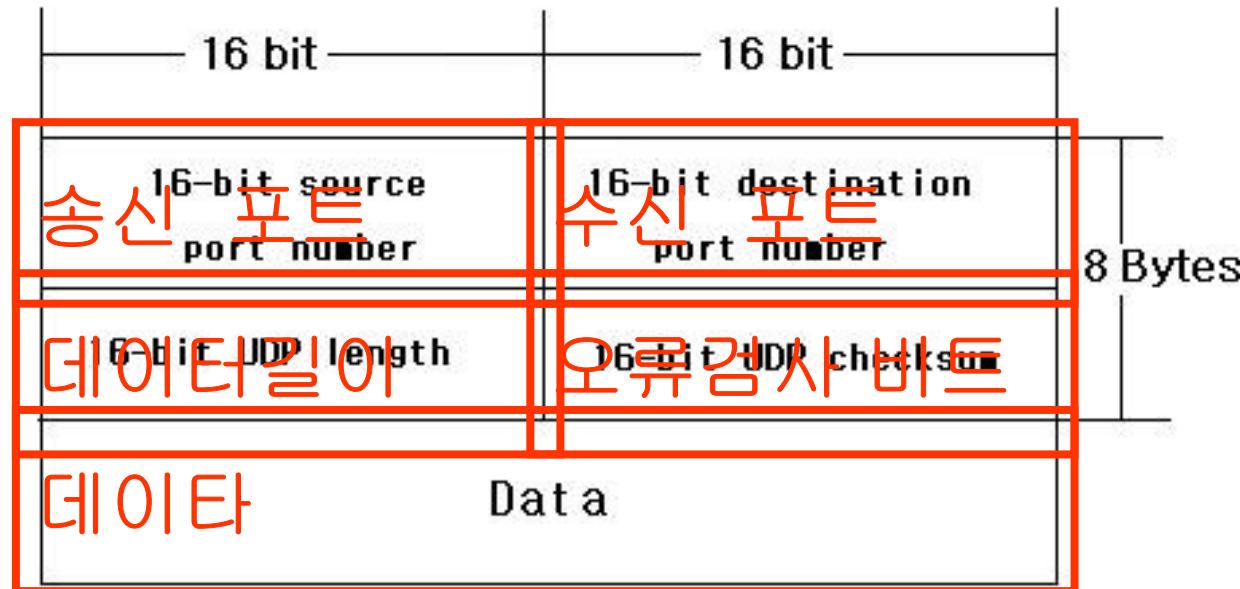
❖ 개요

- ▶ UDP는 datagram 기반의 transport layer protocol이다.
- ▶ UDP는 비 연결형 데이터 전송 프로토콜이다.
- ▶ UDP는 신뢰할 수 없다.
(Packet이 목적 호스트에 꼭 도달한다는 보장은 없다.)
- ▶ 최대 UDP datagram 크기 이론적인 IP datagram의 최대 크기는 65535byte이다.
- ▶ RFC 768

- ▶ Ex) grep snmp /etc/services

UDP Header

- ❖ TCP헤더에 비해 적은 정보만을 필요로 한다.
 - ▶ UDP 길이 = IP 길이 - IP 헤더 길이





UDP를 사용하는 어플리케이션

- ❖ UDP에서 사용하는 응용 프로토콜들
 - ▶ DNS 질의 시
 - ↳ DNS의 질의는 빠른 응답요구를 요하기 때문
 - ▶ TFTP (Trivial File Transfer Protocol)
 - ↳ 덜 신뢰하는 파일전송
 - ▶ NFS (Network File System)
 - ▶ RIP (Routing Information Protocol)
 - ↳ 반복적인 라우팅 테이블의 갱신을 위해
 - ▶ ...



UDP의 장, 단점

◆ 장 점

- ▶ 연결설정 과정이 필요 없으므로 오버헤드를 줄인다.
- ▶ 오류발생을 허용하는 신속한 전송을 처리하기에 적합하다.
- ▶ 멀티캐스팅과 브로드캐스팅을 위한 전송 프로토콜로 적절하다.
- ▶ RIP와 같은 경로 간선 프로토콜에서 사용된다.

◆ 단 점

- ▶ 데이터 그램의 상실
- ▶ 데이터 그램의 순서 뒤바뀜
- ▶ 중복



UDP의 사용 용도

❖ UDP의 사용 용도

- ▶ 흐름 및 오류 제어를 하지 않는 간단한 요청-응답 통신을 요구하는 프로세서
- ▶ 내부 흐름 및 오류 제어 기법을 가진 프로세스
- ▶ 멀티캐스팅을 위한 적당한 전송 프로토콜
- ▶ 라우팅 정보 프로토콜과 같은 경로 갱신 프로토콜을 위하여 사용
- ▶ 실시간 전송 프로토콜(RTP)과 함께 사용

UDP의 오버플로우

❖ UDP 오버플로우

- ▶ 데이터를 전달함에 있어 흐름제어를 하지 않으므로 수신 측 해당 포트가 열려있지 않아도 전송 되므로 데이터그램이 버려지게 된다.
- ▶ UDP는 소켓 버퍼의 영향을 받는데 즉, 응용 프로그램이 보내고자 하는 최대 크기의 유저 데이터그램이 버퍼 크기보다 작아야 한다.
- ▶ 이런 소켓 버퍼에 의존적인 경향 때문에 UDP는 소켓의 수신 버퍼가 오버플로우가 되면 버려지게 된다.
- ▶ 이 버려진 패킷은 netstat -s 명령으로 살펴볼 수 있다.

```
$ netstat -s  
...  
Udp:  
8567 packets received  
32 packets to unknown port received  
67 packet receive errors  
9567 packets sent
```



실습: UDP를 사용하는 예제

◆ 목표

- ▶ Datagram 서버를 만들어보자.
- ▶ 서버는 포트번호를 인자로 받고 , 클라이언트는 서버 IP와 포트번호를 명령 행 인자로 받는다.
- ▶ Ethereal을 이용하여 패킷을 분석해 보자.

실습: UDP example - server_udp.c

```
1  /* **** */
2  * Filename: server_udp.c
3  * Title: Simple UDP Server
4  * Desc: datagram server를 만든다.
5  *       포트번호는 실행시 인자로 넘겨준다.
6  *****/
7  #include <sys/types.h>
8  #include <sys/socket.h>
9  #include <netinet/in.h>
10 #include <netdb.h>
11 #include <stdio.h>
12
13 void error(char *msg)
14 {
15     perror(msg);
16     exit(0);
17 }
18
19 int main(int argc, char *argv[])
20 {
21     int sock, length, fromlen, n;
22     struct sockaddr_in server;
23     struct sockaddr_in from;
24
25     char buf[1024];
26
27     if (argc < 2) {
28         fprintf(stderr, "ERROR, no port provided\n");
29         exit(0);
30     }
31 }
```

실습: UDP example - server_udp.c

```
32     sock = TODO: 데이터그램 소켓 열기      ;
33     if (sock < 0) error("Opening socket");
34
35     length = sizeof(server);
36
37     bzero(&server,length);
38     server.sin_family      = AF_INET;
39     server.sin_addr.s_addr = INADDR_ANY;
40     server.sin_port        = htons(atoi(argv[1]));
41
42     if ( bind( sock, (struct sockaddr *)&server, length ) < 0 )
43         error("binding");
44     fromlen = sizeof(struct sockaddr_in);
45
46     while (1) {
47
48         n = TODO: recvfrom 사용 하여 buf 메시지 받기
49
50
51
52         if (n < 0) error( "recvfrom" );
53         write( 1, "Received a datagram: ", 21 );
54         write( 1, buf, n );
55
56         n = TODO: sendto를 이용하여 “Got your message\n”를 전송하기
57
58
59
60
61
62
63
64         if (n < 0) error("sendto");
65
66     }
```

실습: UDP example - client_udp.c

```
1  /* ****
2  * Filename: client_udp.c
3  * Title: Simple UDP Client
4  * Desc: datagram Client를 만든다.
5  * 명령행 인자는 서버IP와 포트번호
6  *****/
7 #include <sys/types.h>
8 #include <sys/socket.h>
9 #include <netinet/in.h>
10 #include <arpa/inet.h>
11 #include <netdb.h>
12 #include <stdio.h>
13
14 void error(char *);
15
16 int main(int argc, char *argv[])
17 {
18     int sock, length, n;
19     struct sockaddr_in server, from;
20     struct hostent *hp;
21
22     char buffer[256];
23
24     if (argc != 3) { printf("Usage: server port\n");
25                     exit(1);
26     }
27
28     sock = TODO: 데이터그램 소켓 열기
29     if (sock < 0) error("socket");
30 }
```

실습: UDP example - client_udp.c

```
31 server.sin_family = AF_INET;
32 hp = gethostbyname(argv[1]);
33 if (hp==0) error("Unknown host");
34
35 bcopy((char *)hp->h_addr,
36       (char *)&server.sin_addr,
37       hp->h_length);
38 server.sin_port = htons(atoi(argv[2]));
39
40 length=sizeof(struct sockaddr_in);
41 printf("Please enter the message: ");
42
43 bzero(buffer,256);
44 fgets(buffer,255,stdin);
45
46 n= TODO: sendto를 사용하여 buffer에 담겨진 내용을 보낸다.
47
48 if (n < 0) error("Sendto");
49
50 n = TODO: recvfrom을 사용하여 buffer에 보내온 메시지 담아놓기.
51 if (n < 0) error("recvfrom");
52
53 write(1,"Got an ack: ",12);
54 write(1,buffer,n);
55 }
56 //-----
57 void error(char *msg)
58 {
59 perror(msg);
60 exit(0);
61 }
```