

Global KPU!

글로벌 경쟁력을 갖춘 산업기술 명문대학  
세계를 향해 더 큰 미래를 펼쳐갑니다

## 7장. UDP 서버-클라이언트

네트워크 게임 프로그래밍

- ❖ UDP 서버-클라이언트의 기본 구조와 동작 원리를 이해한다.
- ❖ UDP 응용 프로그램 작성에 필요한 핵심 소켓 함수를 익힌다
- ❖ IPv4와 IPv6 기반 UDP 서버-클라이언트를 작성할 수 있다.
- ❖ 브로드캐스팅(Broadcasting)의 개념을 이해하고 UDP를 이용해 구현할 수 있다.



## ❖ TCP와 UDP의 공통점

- 전송 계층 프로토콜
- 포트 번호를 이용해 주소를 지정
  - 포트 번호를 이용하여 종단(응용 프로그램)간 전송
  - 두 응용 프로그램이 TCP나 UDP를 이용해 통신하려면 반드시 포트 번호를 결정해야 함
- 데이터 오류를 체크
  - 데이터 위변조 확인
  - IP와 달리 TCP와 UDP는 헤더는 물론이고 데이터에 대한 오류도 체크함

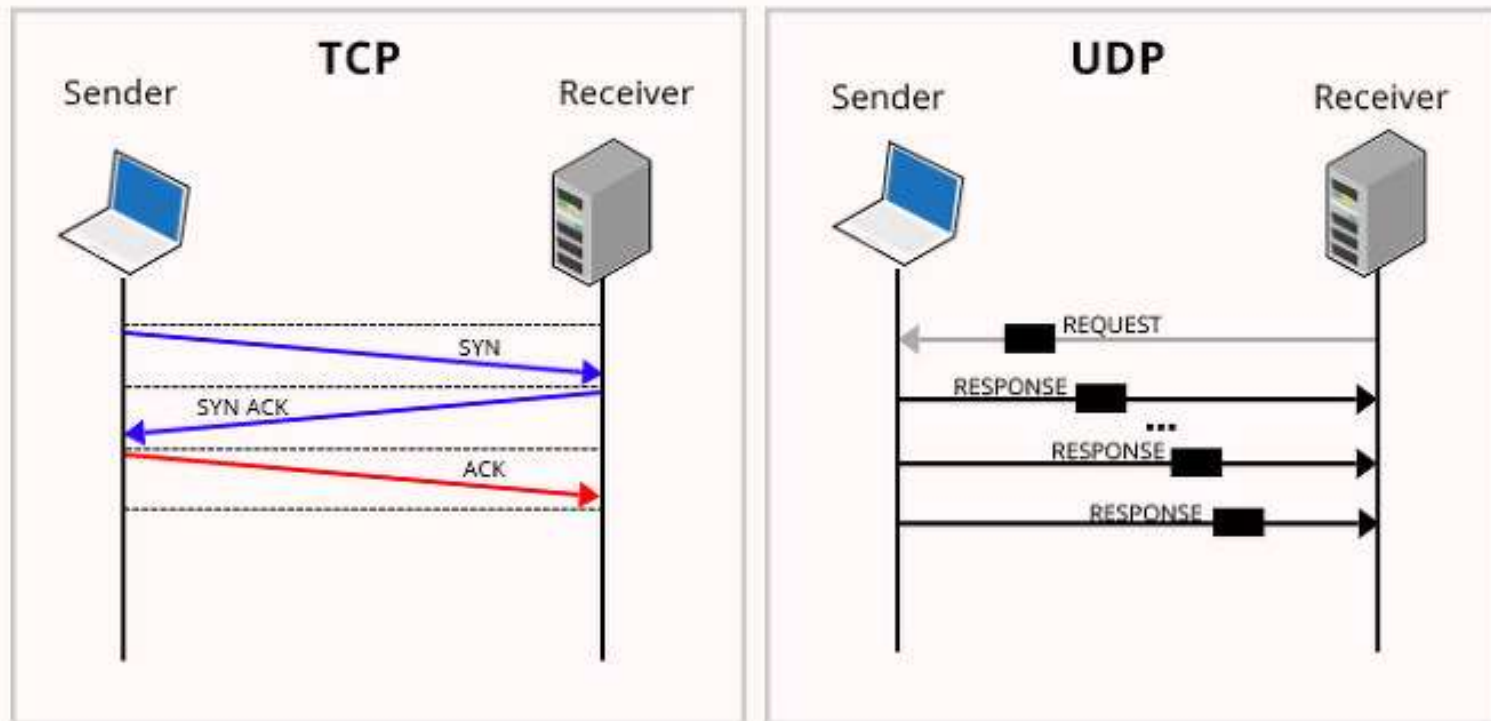


## ❖ TCP와 UDP의 차이점

항목	TCP	UDP
①	연결형 프로토콜(연결지향형) - 연결 설정 후 통신 가능	비연결형 프로토콜 - 연결 설정 없이 통신 가능
②	신뢰성 있는 데이터 전송 - 데이터를 재전송함 - 오류제어 (버퍼 활용 재전송) - 흐름제어 (수신자가 받을수 있을만큼만 보냄, 재조합) - 저성능	신뢰성 없는 데이터 전송 - 데이터를 재전송하지 않음 - 오류제어/흐름제어 기능 미지원 - Timeout등 별도의 프로그래밍 필요 - 보낸사람의 수신자 정보를 확인하여 수신한 메시지 확인 필요 - 서버가 연결이 되지 않은 경우를 대비해 connect() 함수 사용. (NULL 사용)
③	일대일 통신	일대일 통신 일대다 통신
④	데이터 경계 구분 안 함 - 바이트 스트림 서비스 - 경로 설정(connect())하고 데이터 순서 유지	데이터 경계 구분함 - 데이터그램 서비스 - 경로 설정 절차 수행하지 않고 데이터의 순서 유지하지 않음

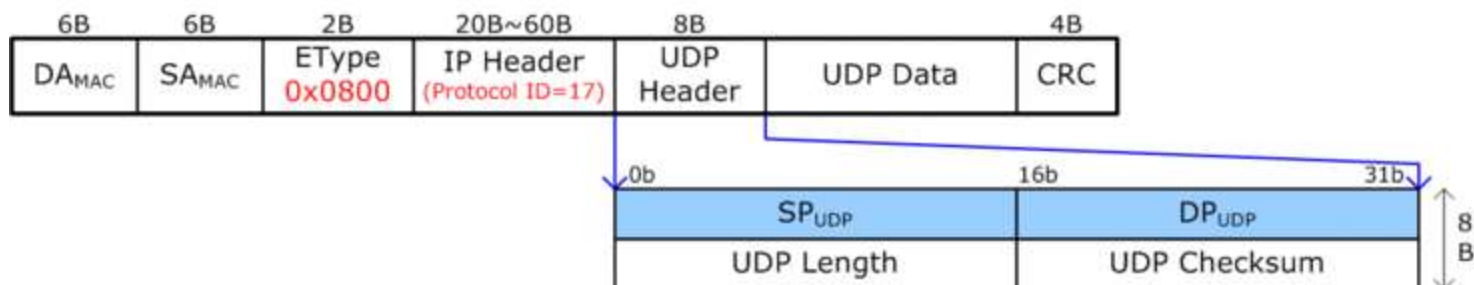
## ❖ TCP와 UDP의 차이점

### TCP Vs UDP Communication

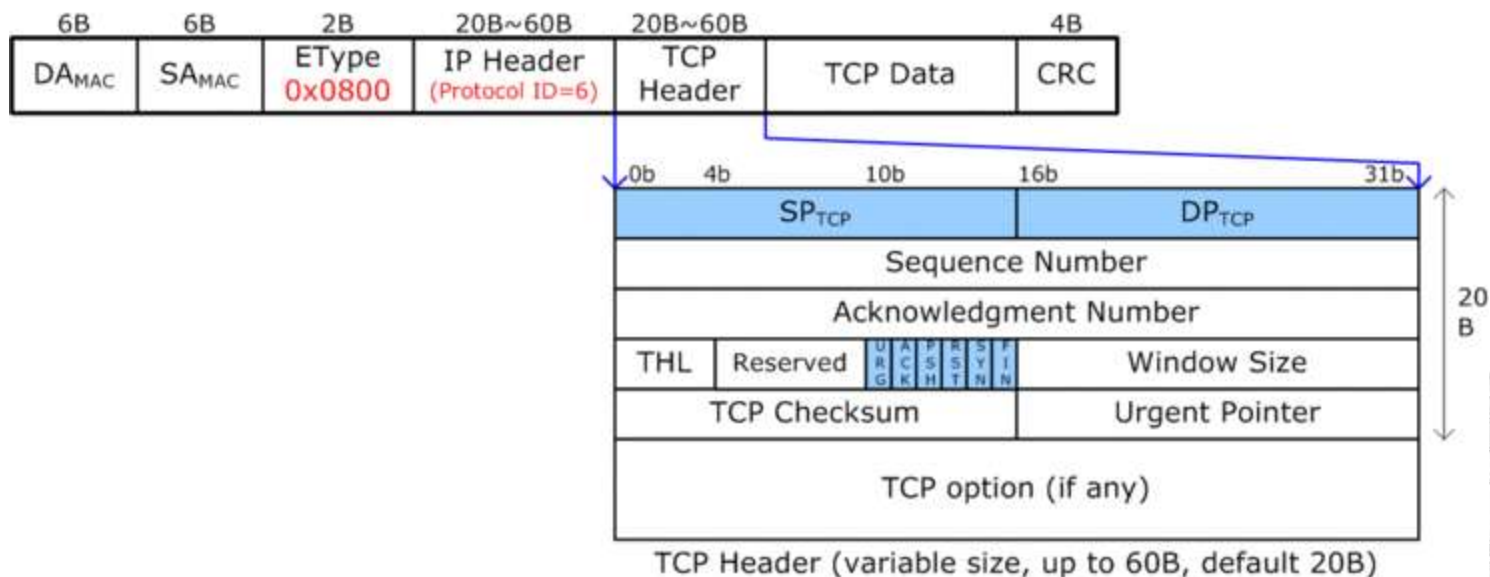


## ❖ UDP 헤더와 TCP 헤더

### UDP Packet



### TCP Packet



## ❖ UDP의 특징

- 연결 설정을 하지 않으므로 connect( ) 함수 불필요
  - UDP는 연결 자체가 없음
- 프로토콜 수준에서 신뢰성 있는 데이터 전송을 보장하지 않으므로, 필요하다면 응용 프로그램 수준에서 신뢰성 있는 데이터 전송 기능을 구현해야 함
- 간단한 소켓 함수 호출 절차만 따르면 다자 간 통신을 쉽게 구현할 수 있음
  - 1:1 혹은 1:다 연결
- TCP와 달리 응용 프로그램이 데이터 경계 구분을 위한 작업을 별도로 할 필요가 없음
- 데이터그램(메시지) 단위 전송이며 하나의 데이터그램은 65535바이트 크기로 제한됨
  - 65535 바이트 이상의 크기는 잘라서 보내야 함
- Not ordered
- 높은 성능
  - 신뢰성을 희생해서 성능 확보



## ❖ UDP를 사용하는 이유

- 성능을 높일 수 있다.
- 실시간 멀티미디어 서비스 프로그램 개발이 용이하다.
  - 연속성이 신뢰성 보다 중요한 서비스
- 개발이 쉽다.
  - 흐름을 관리하기 위한 노력이 필요 없다.
- Agent & Manager 모델에 적합
  - Agent가 Manager로 데이터를 요청
  - 다수의 Manager를 효과적으로 관리
  - 오류에 민감하지 않음 (다시 요청하면 됨)
- 주로 파일, 스트리밍 콘텐츠 전송등에 사용
- TCP는 전화로 비유되고 UDP는 우편으로 비유할 수 있음

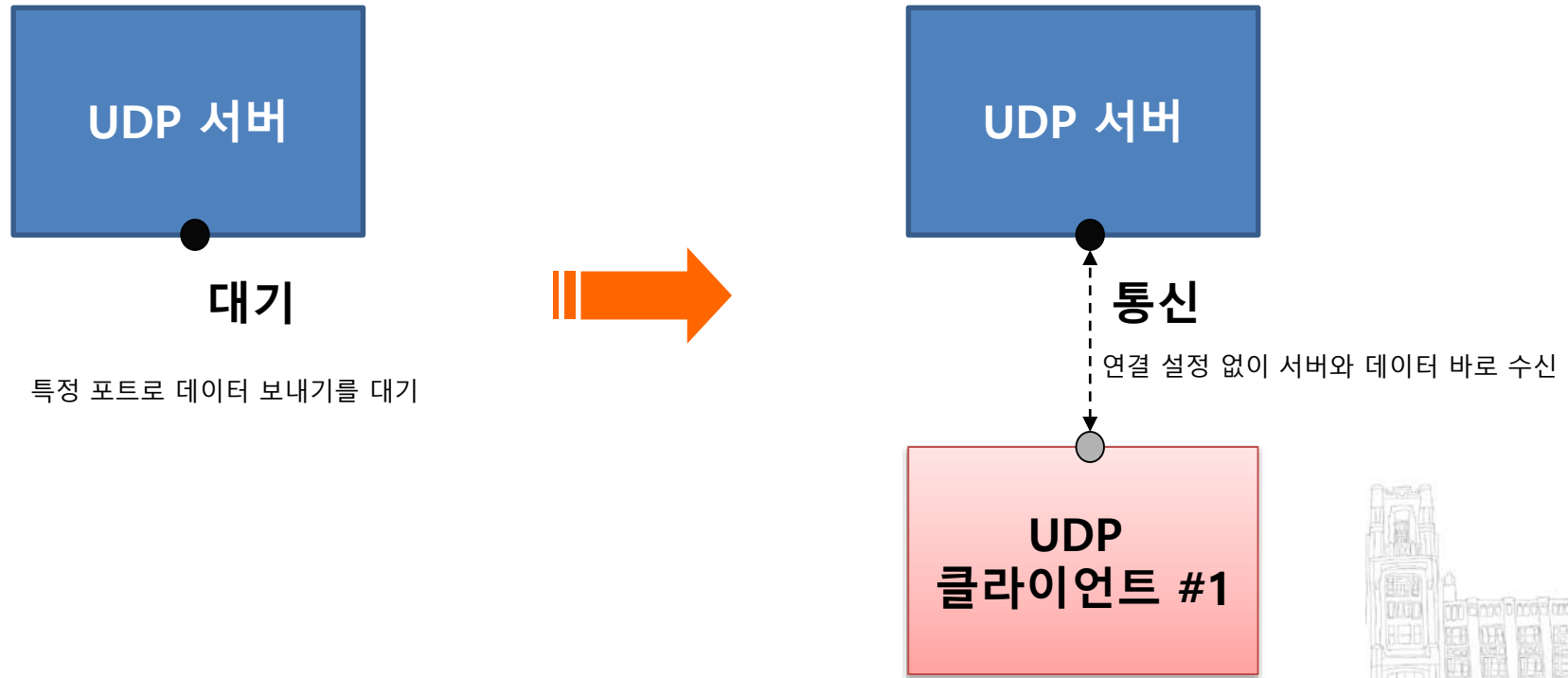




# UDP 서버-클라이언트 동작 원리 (1)

## ❖ UDP 서버-클라이언트 동작 원리

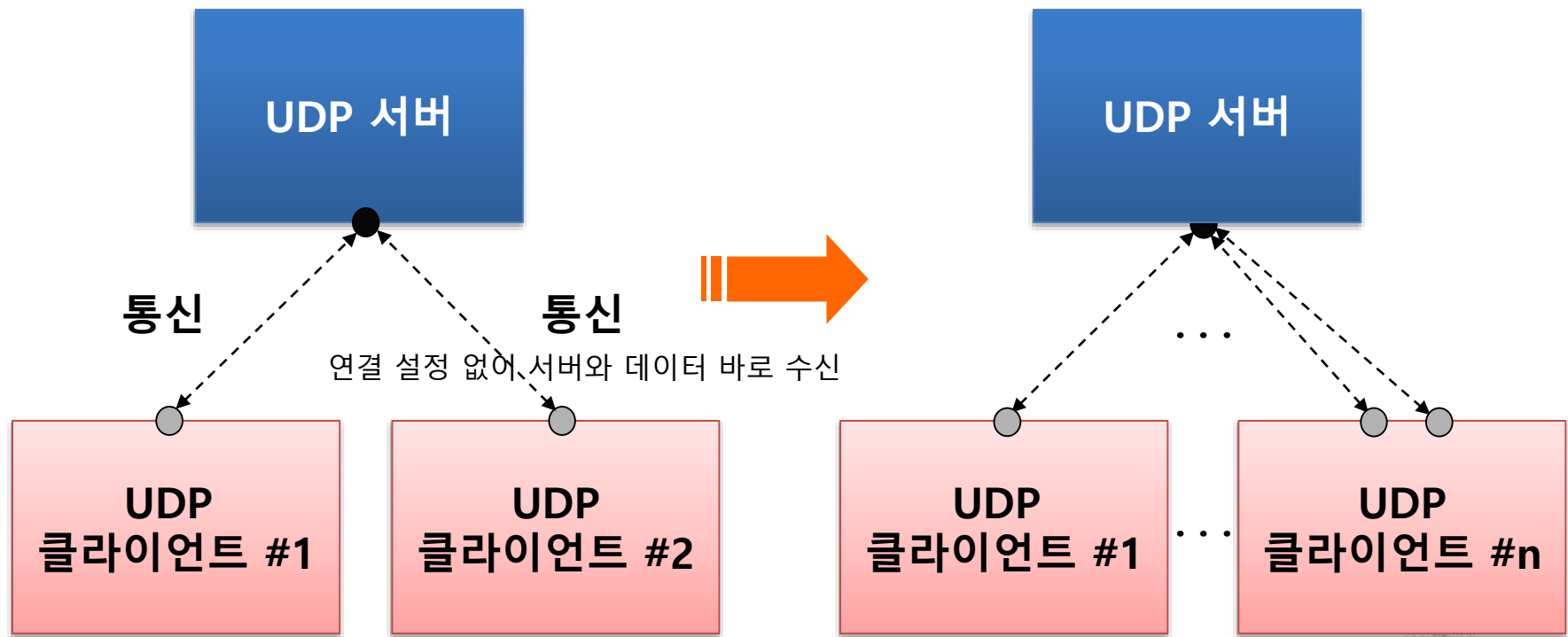
- TCP 서버와 달리 멀티스레드 등의 프로그래밍 기법을 사용하지 않고도 한 소켓으로 여러 클라이언트를 처리할 수 있음 => TCP의 경우 P2P와 같은 형식으로 연결



# UDP 서버-클라이언트 동작 원리 (2)

## ❖ UDP 서버-클라이언트 동작 원리(계속)

- 서버는 순차적으로 데이터를 구분하여 저장/처리



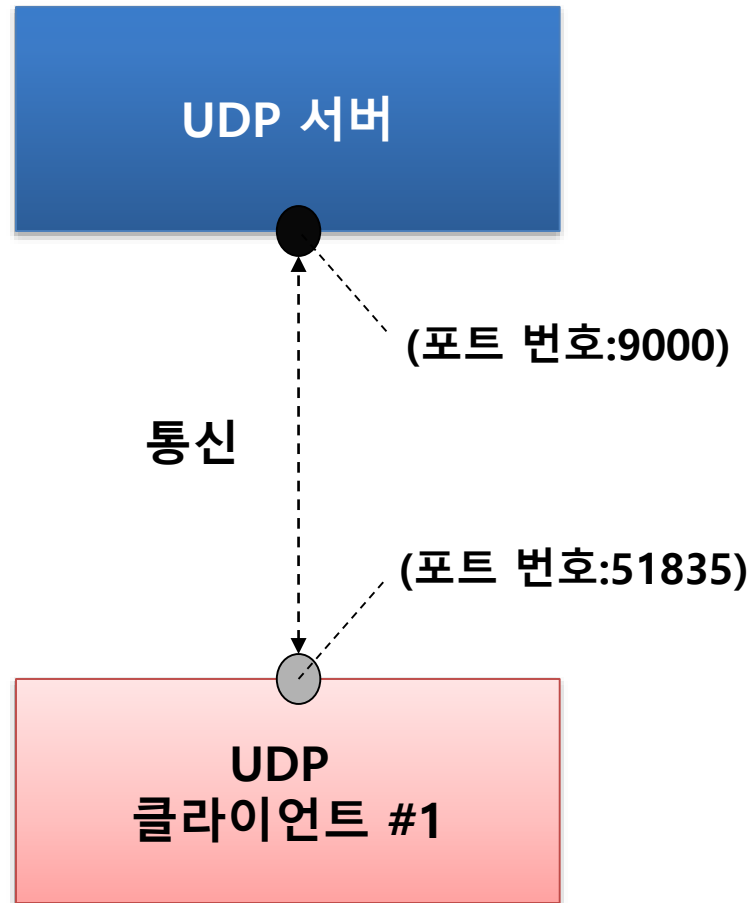
TCP 서버와는 달리 UDP 서버는 소켓을 한 개만 사용



# UDP 서버-클라이언트 예제 (2)

## ❖ UDP 서버-클라이언트 상태 ①

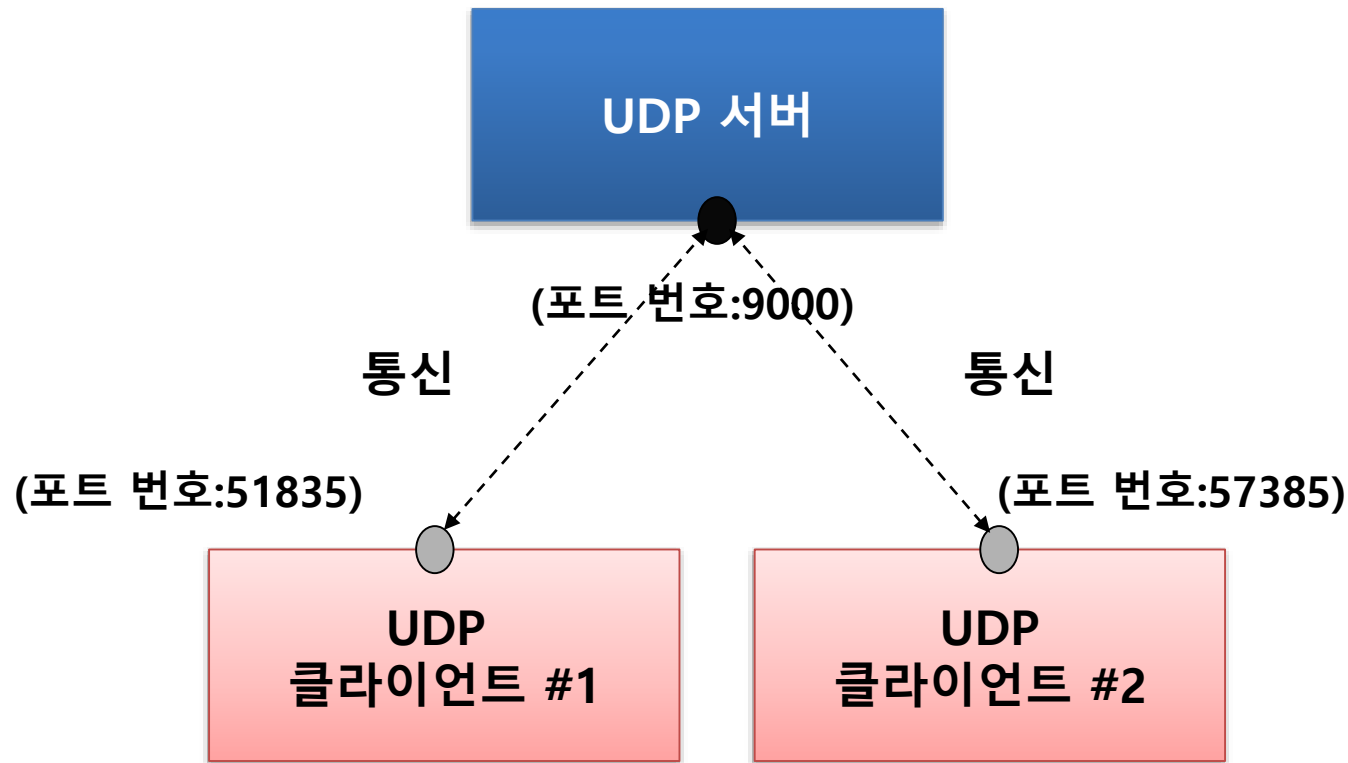
- 한 개의 UDP 서버와 한 개의 UDP 클라이언트



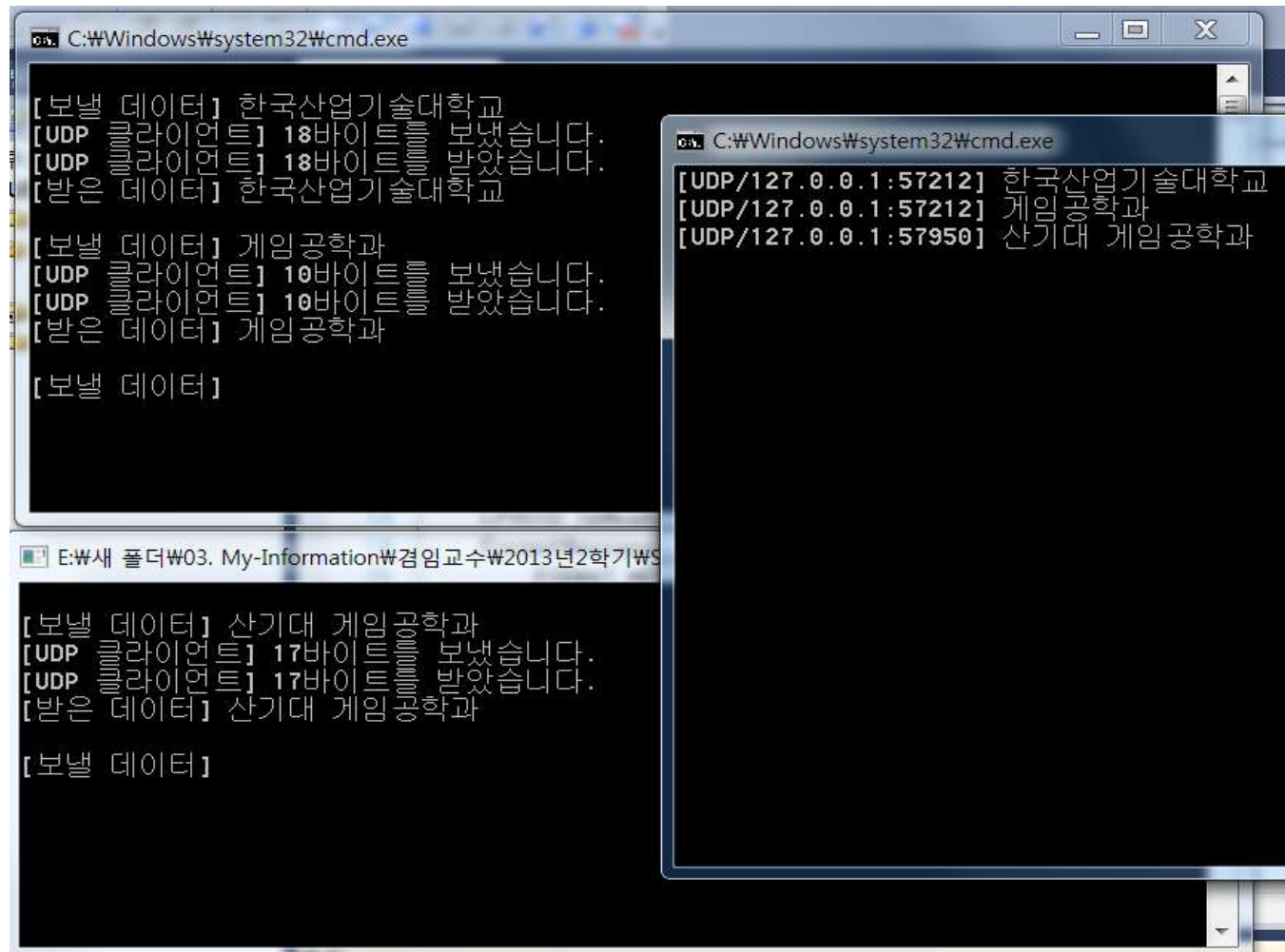
# UDP 서버-클라이언트 예제 (3)

## ❖ UDP 서버-클라이언트 상태 ②

- 한 개의 UDP 서버와 두 개의 UDP 클라이언트



## 실습 7-1 P223~



The image displays three overlapping Windows command prompt windows, each showing the output of a UDP server or client program. The windows are titled "C:\Windows\system32\cmd.exe".

**Top-left window (Server):**

```
[보낼 데이터] 한국산업기술대학교
[UDP 클라이언트] 18바이트를 보냈습니다.
[UDP 클라이언트] 18바이트를 받았습니다.
[받은 데이터] 한국산업기술대학교

[보낼 데이터] 게임공학과
[UDP 클라이언트] 10바이트를 보냈습니다.
[UDP 클라이언트] 10바이트를 받았습니다.
[받은 데이터] 게임공학과

[보낼 데이터]
```

**Top-right window (Client):**

```
[UDP/127.0.0.1:57212] 한국산업기술대학교
[UDP/127.0.0.1:57212] 게임공학과
[UDP/127.0.0.1:57950] 산기대 게임공학과
```

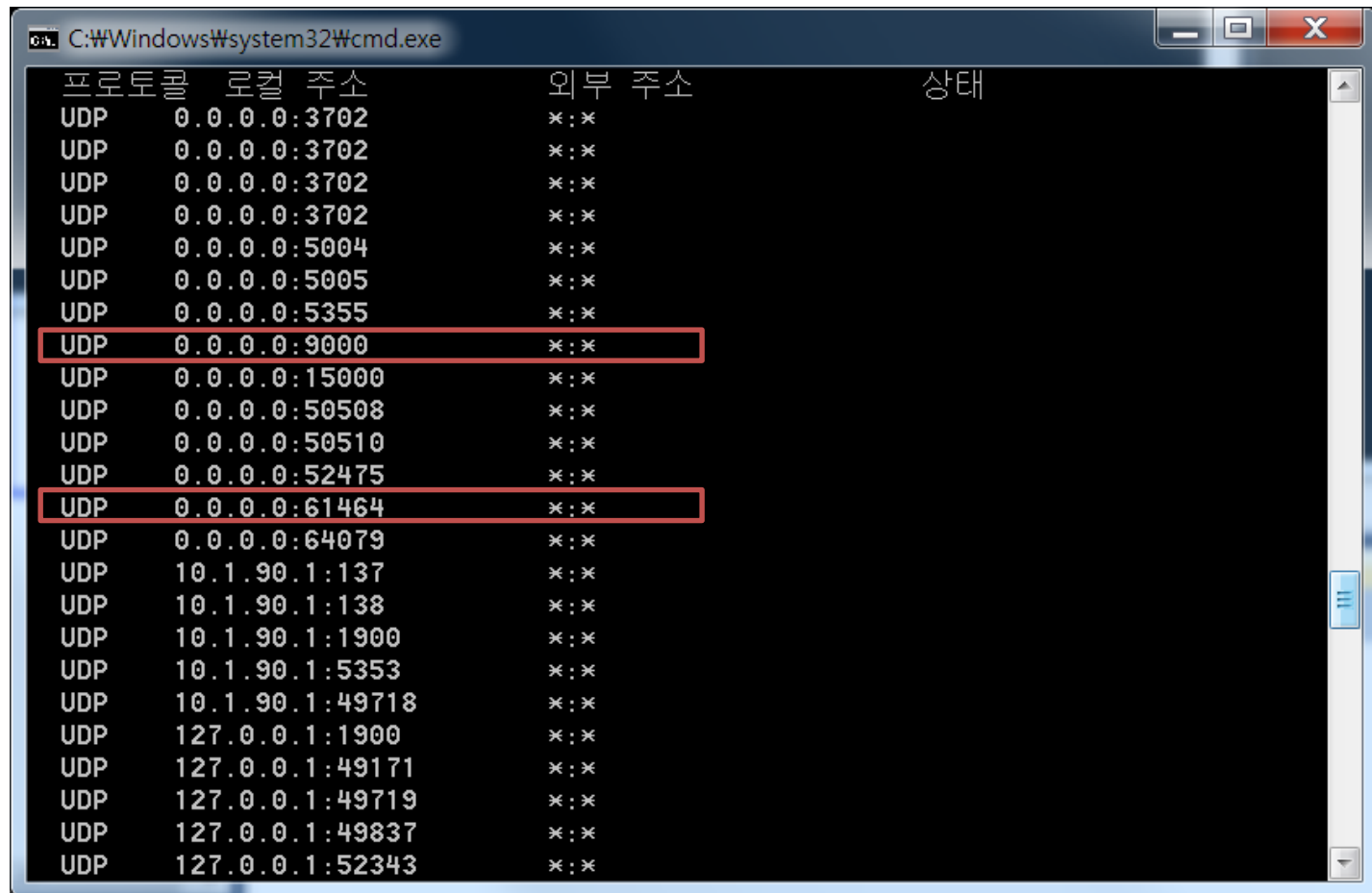
**Bottom window (Client):**

```
[보낼 데이터] 산기대 게임공학과
[UDP 클라이언트] 17바이트를 보냈습니다.
[UDP 클라이언트] 17바이트를 받았습니다.
[받은 데이터] 산기대 게임공학과

[보낼 데이터]
```



## 실습 7-1 P223~



프로토콜	로컬 주소	외부 주소	상태
UDP	0.0.0.0:3702	*:*	
UDP	0.0.0.0:3702	*:*	
UDP	0.0.0.0:3702	*:*	
UDP	0.0.0.0:3702	*:*	
UDP	0.0.0.0:5004	*:*	
UDP	0.0.0.0:5005	*:*	
UDP	0.0.0.0:5355	*:*	
UDP	0.0.0.0:9000	*:*	
UDP	0.0.0.0:15000	*:*	
UDP	0.0.0.0:50508	*:*	
UDP	0.0.0.0:50510	*:*	
UDP	0.0.0.0:52475	*:*	
UDP	0.0.0.0:61464	*:*	
UDP	0.0.0.0:64079	*:*	
UDP	10.1.90.1:137	*:*	
UDP	10.1.90.1:138	*:*	
UDP	10.1.90.1:1900	*:*	
UDP	10.1.90.1:5353	*:*	
UDP	10.1.90.1:49718	*:*	
UDP	127.0.0.1:1900	*:*	
UDP	127.0.0.1:49171	*:*	
UDP	127.0.0.1:49719	*:*	
UDP	127.0.0.1:49837	*:*	
UDP	127.0.0.1:52343	*:*	

## ❖ 소켓 통신을 위해 결정해야 할 요소

### ① 프로토콜

- 통신 규약. 소켓을 생성할 때 결정

### ② 지역 IP 주소와 지역 포트 번호

- 서버 또는 클라이언트 자신의 주소

### ③ 원격 IP 주소와 원격 포트 번호

- 서버 또는 클라이언트가 통신하는 상대의 주소

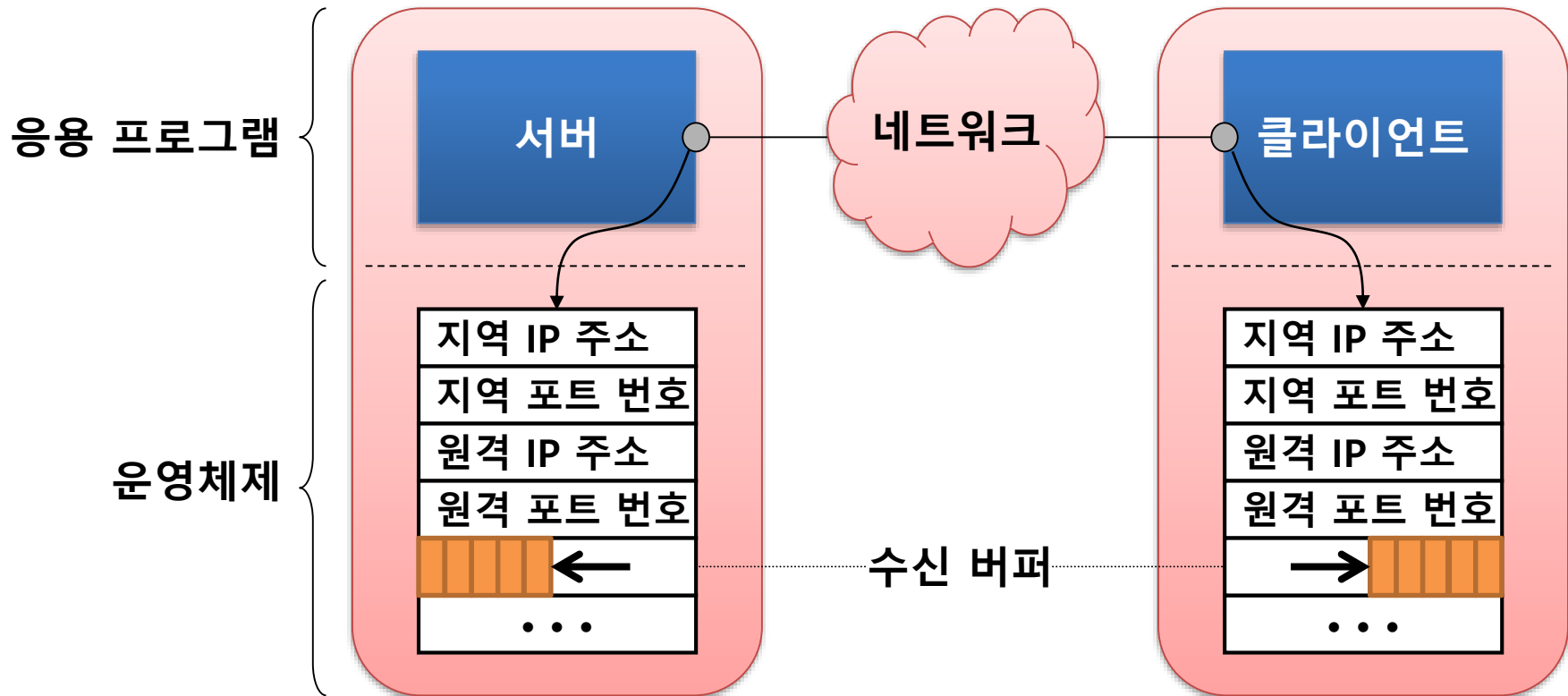




# UDP 서버-클라이언트 분석 (2)

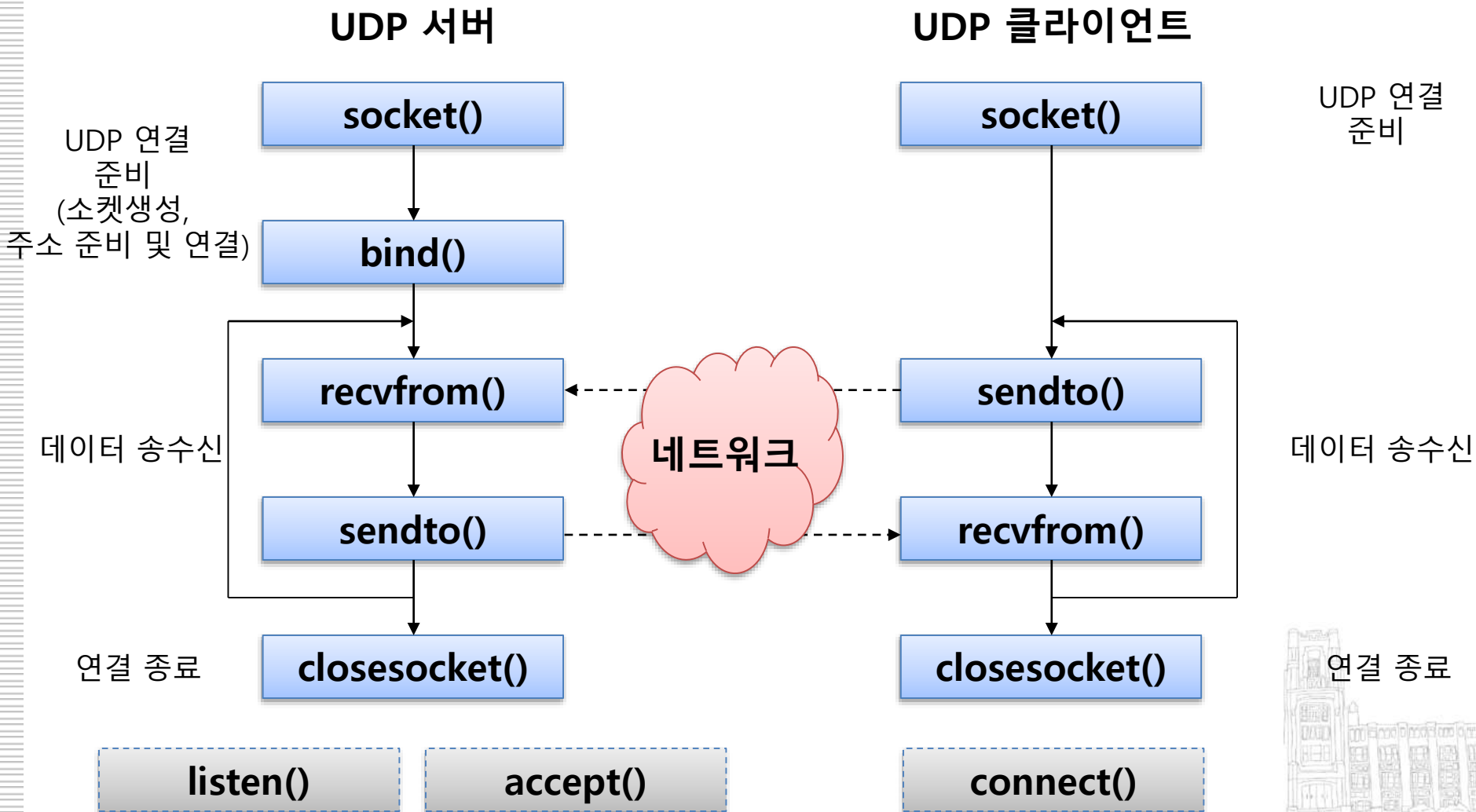
## ❖ 소켓 데이터 구조체

- TCP와는 다르게 송신 버퍼가 필요 없음 - 이유는?



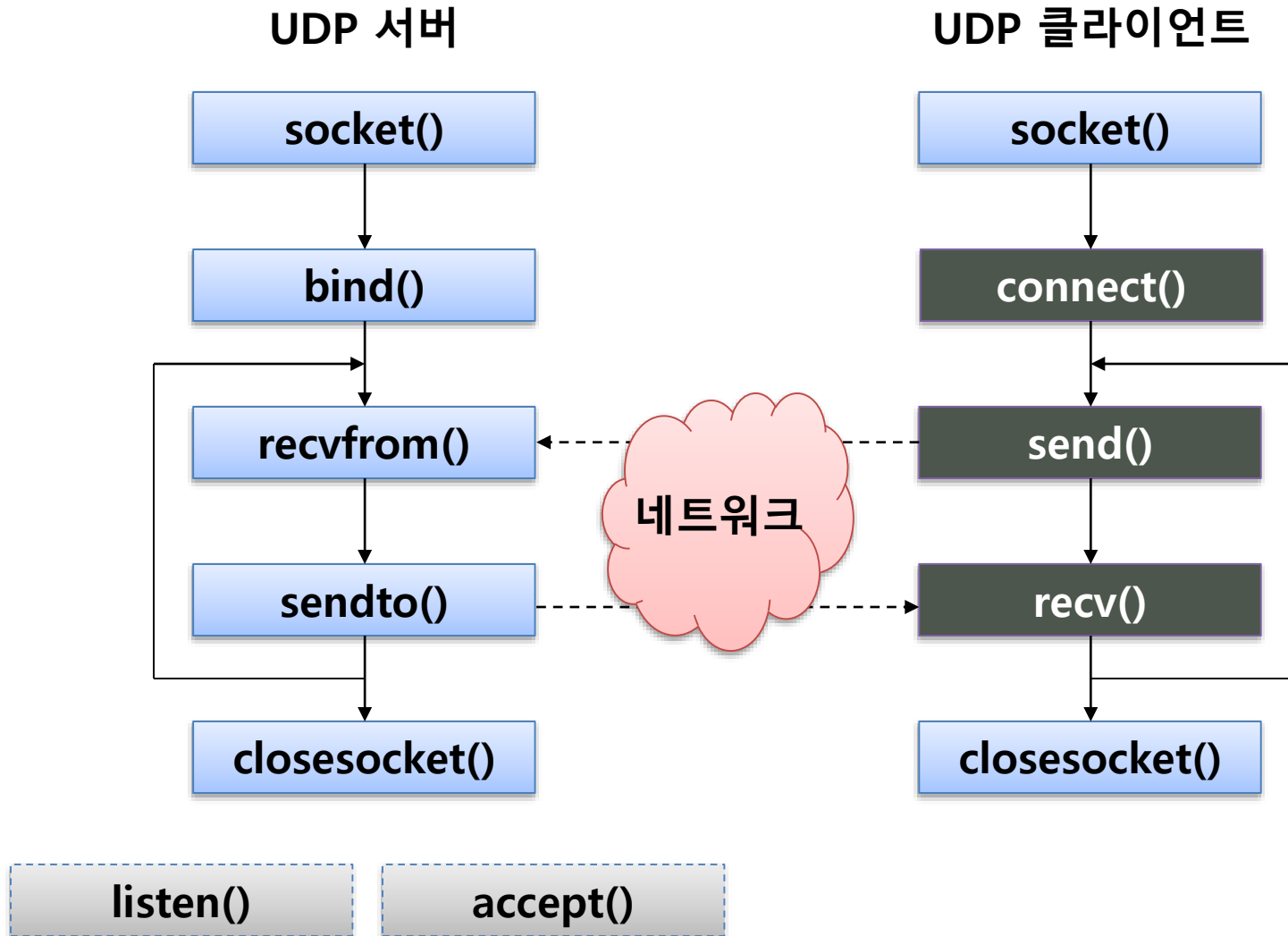
# UDP 서버-클라이언트 분석 (3)

## ❖ UDP 서버-클라이언트 모델 ①



# UDP 서버-클라이언트 분석 (4)

## ❖ UDP 서버-클라이언트 모델 ②



## ❖ 함수원형 (windows type)

- `send(SOCKET s, const char *buf, int len, int flags)`
- `recv(SOCKET s, char *buf, int len, int flags)`
- `sendto(SOCKET s, const char *buf, int len, int flags, const struct sockaddr *to, int tolen)`
- `recvfrom(SOCKET s, char *buf, int len, int flags, struct sockaddr *from, int *fromlen)`



## ❖ 데이터 송수신을 위한 시스템 함수

- `write()`: stream 소켓을 통한 데이터 송신
- `read()`: stream 소켓을 통한 데이터 수신
- `send()`: stream 소켓을 통한 데이터 송신
- `recv()`: stream 소켓을 통한 데이터 송신
- `sendto()`: Datagram 소켓을 통해 데이터 송신
- `recvfrom()`: Datagram 소켓을 통해 데이터 수신



## ❖ sendto() 함수

```
int sendto (  
    SOCKET s,  
    const char *buf, // 보낼 데이터를 담고 있는 응용 프로그램 버퍼의 주소  
    int len,          // 보낼 데이터 크기  
    int flags,  
    const struct sockaddr *to, // 목적지 주소를 담고 있는 소켓 주소 구조체  
    int tolen          // 목적지 주소를 담고 있는 소켓 주소 구조체의 크기  
);
```

성공: 보낸 바이트 수, 실패: SOCKET\_ERROR

- 소켓의 지역 IP 주소와 지역 포트 번호가 아직 결정되지 않은 상태라면 운영체제가 자동으로 결정!
- UDP는 운영체제의 송신 버퍼를 사용하지 않으므로 블로킹 현상이 없음



## ❖ sendto() 함수 사용 예

- connect 함수로, 서버를 명시하기 위해서 사용.
- 실제 connect 과정을 수행하는 건 아님.
- 데이터를 전송 할때, 주소 복사 과정을 생략할 수 있음

// sendto() 함수로 데이터를 보낸다.

```
retval = sendto(sock, buf, strlen(buf), 0,  
                NULL, sizeof(serveraddr));
```

```
if(retval == SOCKET_ERROR) 오류 처리;
```

```
printf("%d바이트를 보냈습니다.\n", retval);
```



## ❖ sendto() 함수 사용 예

```
// 소켓 주소 구조체를 수신자의 IP 주소와 포트 번호로 초기화한다.  
SOCKADDR_IN serveraddr;  
  
...  
  
// 송신용 버퍼를 선언하고 데이터를 넣는다.  
char buf[BUFSIZE];  
  
...  
  
// sendto() 함수로 데이터를 보낸다.  
retval = sendto(sock, buf, strlen(buf), 0,  
                (SOCKADDR *)&serveraddr, sizeof(serveraddr));  
if(retval == SOCKET_ERROR) 오류 처리;  
printf("%d바이트를 보냈습니다.\n", retval);
```





## ❖ recvfrom() 함수

```
int recvfrom (  
    SOCKET s,  
    char *buf, //받은 데이터를 저장할 응용 프로그램 버퍼의 주소  
    int len,   // 응용 프로그램 버퍼의 크기  
    int flags,  
    struct sockaddr *from, // 소켓 주소 구조체를 전달하면 송신자의 IP와 포트로 채워짐  
    int *fromlen //정수형 변수를 from이 가리키는 소켓 주소 구조체의 기로 초기화한 후 전달  
);
```

**성공: 받은 바이트 수, 실패: SOCKET\_ERROR**

- UDP 패킷 데이터를 한 번에 하나만 읽을 수 있음!
- 받은 데이터는 버퍼의 크기만큼 읽고 이상은 버림.
- 받는 순서대로 처리(구분은 송신자의 정보로 구분)



## ❖ recvfrom() 함수 사용 예

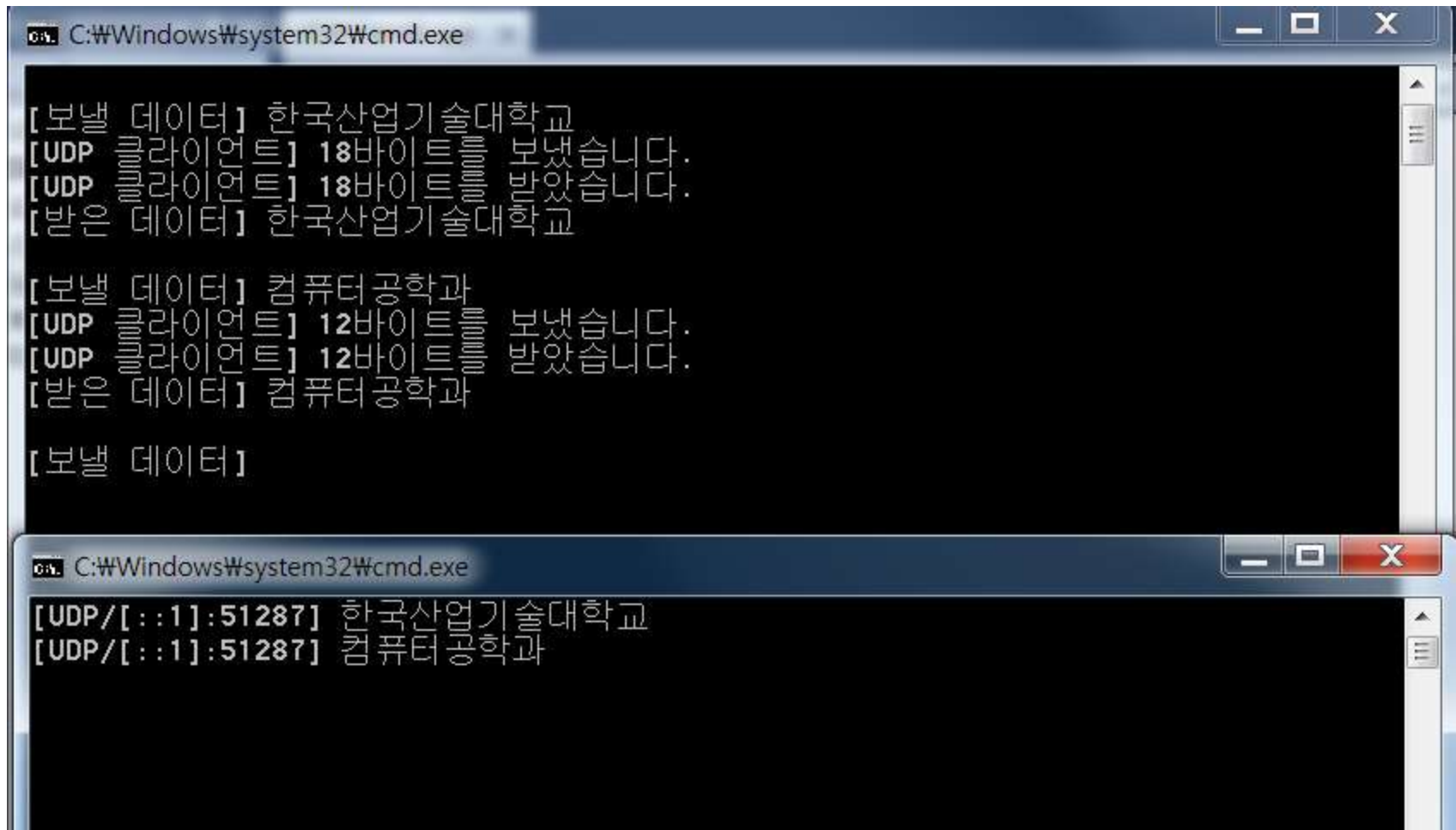
```
// 통신 상대의 주소를 저장하기 위한 변수를 선언한다.  
SOCKADDR_IN peeraddr;  
int addrlen;  
// 수신용 버퍼를 선언한다.  
char buf[BUFSIZE];  
// recvfrom() 함수로 데이터를 받는다.  
addrlen = sizeof(peeraddr);  
retval = recvfrom(sock, buf, BUFSIZE, 0,  
    (SOCKADDR *)&peeraddr, &addrlen);  
if(retval == SOCKET_ERROR) 오류 처리;  
printf("%d바이트를 받았습니다.\n", retval);
```

## ❖ IPv4 코드 ⇒ IPv6 코드

- ws2tcpip.h 헤더 파일을 포함
- 소켓 생성 시 AF\_INET 대신 AF\_INET6를 사용
- 소켓 주소 구조체로 SOCKADDR\_IN 대신 SOCKADDR\_IN6를 사용
  - 구조체를 변경하면 구조체 필드명도 그에 따라 변경
  - 서버에서 주로 사용하는 INADDR\_ANY(0으로 정의됨)  
값은 in6addr\_any(0으로 정의됨)로 변경
- IPv4만을 지원하는 주소 변환 함수를 IPv4/IPv6 지원 함수로 대체
- 데이터 전송 함수는 기존의 sendto( )/recvfrom( ) 함수를 변경 없이 그대로 사용



## 실습 7-2 P239~



```
C:\Windows\system32\cmd.exe

[보낼 데이터] 한국산업기술대학교
[UDP 클라이언트] 18바이트를 보냈습니다.
[UDP 클라이언트] 18바이트를 받았습니다.
[받은 데이터] 한국산업기술대학교

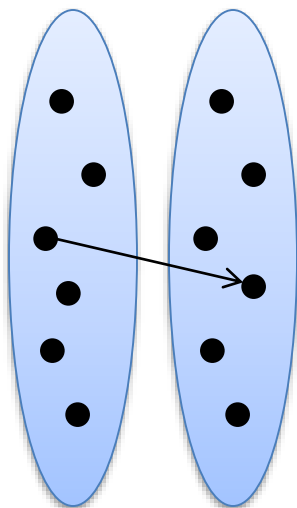
[보낼 데이터] 컴퓨터공학과
[UDP 클라이언트] 12바이트를 보냈습니다.
[UDP 클라이언트] 12바이트를 받았습니다.
[받은 데이터] 컴퓨터공학과

[보낼 데이터]

C:\Windows\system32\cmd.exe

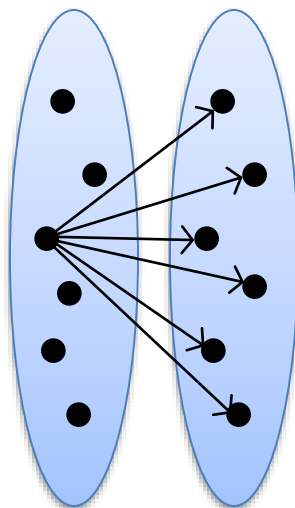
[UDP/[::1]:51287] 한국산업기술대학교
[UDP/[::1]:51287] 컴퓨터공학과
```

## ❖ 통신에 참여하는 개체 간 상호 작용



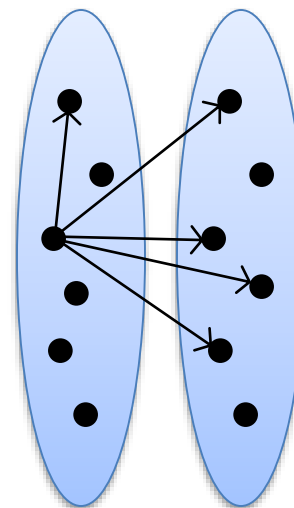
유니캐스팅

IPv4/IPv6



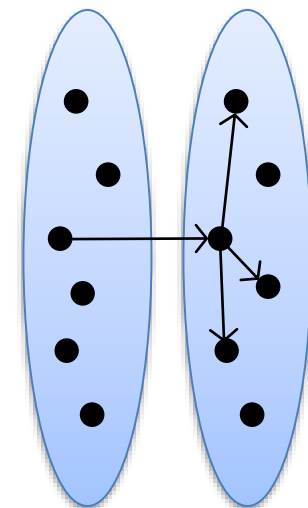
브로드캐스팅

IPv4



멀티캐스팅

IPv4/IPv6



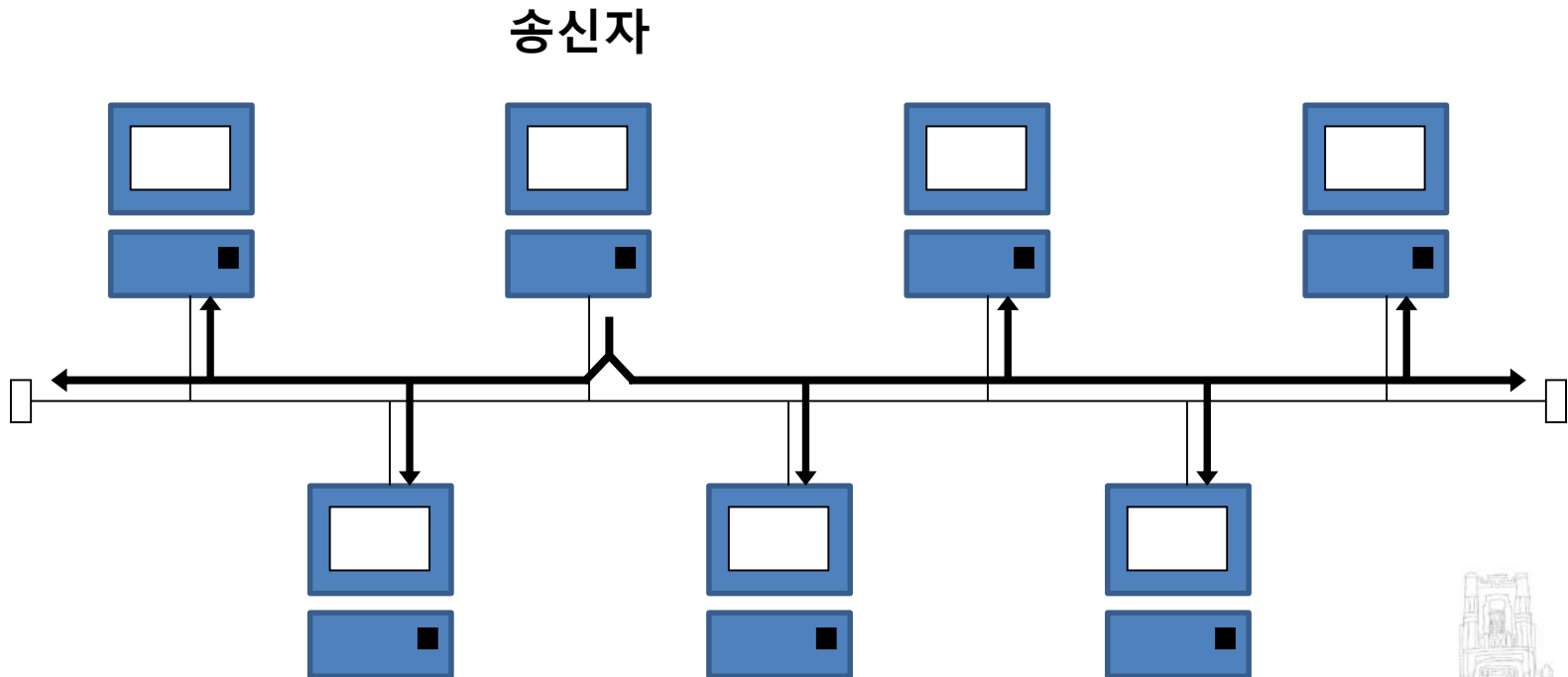
애니캐스팅

IPv6



## ❖ 브로드캐스팅 개념

- 송신자가 보낸 데이터 하나를 다수의 수신자가 받는 방식
  - 데이터 복사본을 여러 개 만들어 보내는 것이 아니므로 송신자 관점에서 보면 상당히 효율적인 기술임



## ❖ 브로드캐스트 데이터를 보내기 위한 절차

### ① 브로드캐스팅을 활성화함

```
BOOL bEnable = TRUE;
```

```
// SO_BROADCAST의 옵션정보를 TRUE로 변경하기 위한 변수
```

```
retval = setsockopt(sock, SOL_SOCKET, SO_BROADCAST,  
(char *)&bEnable, sizeof(bEnable));
```

```
if(retval == SOCKET_ERROR) err_quit("setsockopt()");
```



## ❖ 브로드캐스트 데이터를 보내기 위한 절차(계속)

### ② 브로드캐스트 주소로 데이터를 보냄

```
// 소켓 주소 구조체를 초기화한다.  
SOCKADDR_IN remoteaddr;  
ZeroMemory(&remoteaddr, sizeof(remoteaddr));  
remoteaddr.sin_family = AF_INET;  
remoteaddr.sin_addr.s_addr = inet_addr("255.255.255.255");  
remoteaddr.sin_port = htons(9000);  
// 송신용 버퍼를 선언하고 데이터를 넣는다.  
char buf[BUFSIZE];  
...  
// sendto() 함수로 데이터를 보낸다.  
retval = sendto(sock, buf, strlen(buf), 0,  
    (SOCKADDR *)&remoteaddr, sizeof(remoteaddr));  
if(retval == SOCKET_ERROR) 오류 처리;  
printf("%d바이트를 보냈습니다.\n", retval);
```





## ❖ 브로드캐스트 주소의 종류

호스트 ID

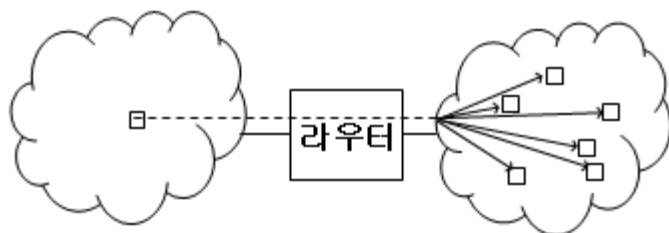
네트워크 ID	11 ..... 1	네트워크 브로드캐스트
---------	------------	-------------

네트워크 ID	서브넷 ID	11 ... 1	서브넷 브로드캐스트
---------	--------	----------	------------

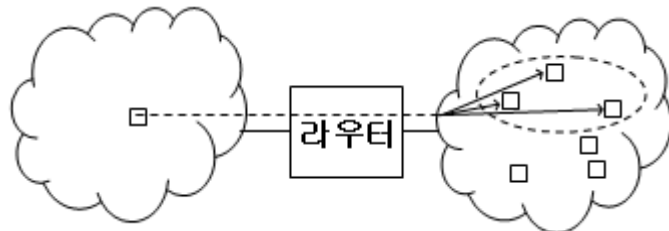
11 ..... 1	지역 브로드캐스트
------------	-----------

	From	To
Class A	<div>0.0.0.0</div> <div>Netid Hostid</div>	<div>127.255.255.255</div> <div>Netid Hostid</div>
Class B	<div>128.0.0.0</div> <div>Netid Hostid</div>	<div>191.255.255.255</div> <div>Netid Hostid</div>
Class C	<div>192.0.0.0</div> <div>Netid Hostid</div>	<div>223.255.255.255</div> <div>Netid Hostid</div>
Class D	<div>224.0.0.0</div> <div>Multicast Address</div>	<div>239.255.255.255</div> <div>Multicast Address</div>
Class E	<div>240.0.0.0</div> <div>Reserved</div>	<div>255.255.255.255</div> <div>Reserved</div>

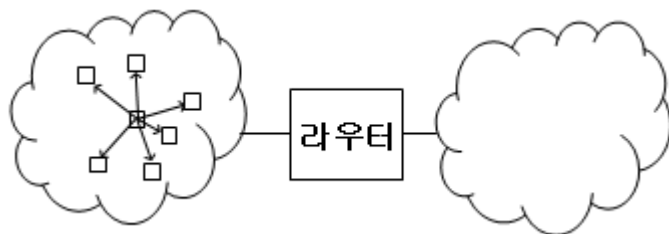
## ❖ 브로드캐스트 주소의 종류(계속)



네트워크 브로드캐스트



서브넷 브로드캐스트



지역 브로드캐스트



## 실습 7-3 P250~



The image shows two overlapping Windows command prompt windows. The background window has a title bar 'C:\Windows\system32\cmd.exe' and contains the following text:

```
[보낼 데이터] 한국산업기술대학교  
[UDP] 18바이트를 보냈습니다.  
  
[보낼 데이터]
```

The foreground window also has a title bar 'C:\Windows\system32\cmd.exe' and contains the following text:

```
[UDP/10.1.90.1:54324] 한국산업기술대학교  
[UDP/10.1.90.1:54327] 게임공학과  
[UDP/10.1.90.1:54327] 수고하세요
```



# Thank You !

[oasis01@gmail.com](mailto:oasis01@gmail.com) / [rhqudtn75@nate.com](mailto:rhqudtn75@nate.com)