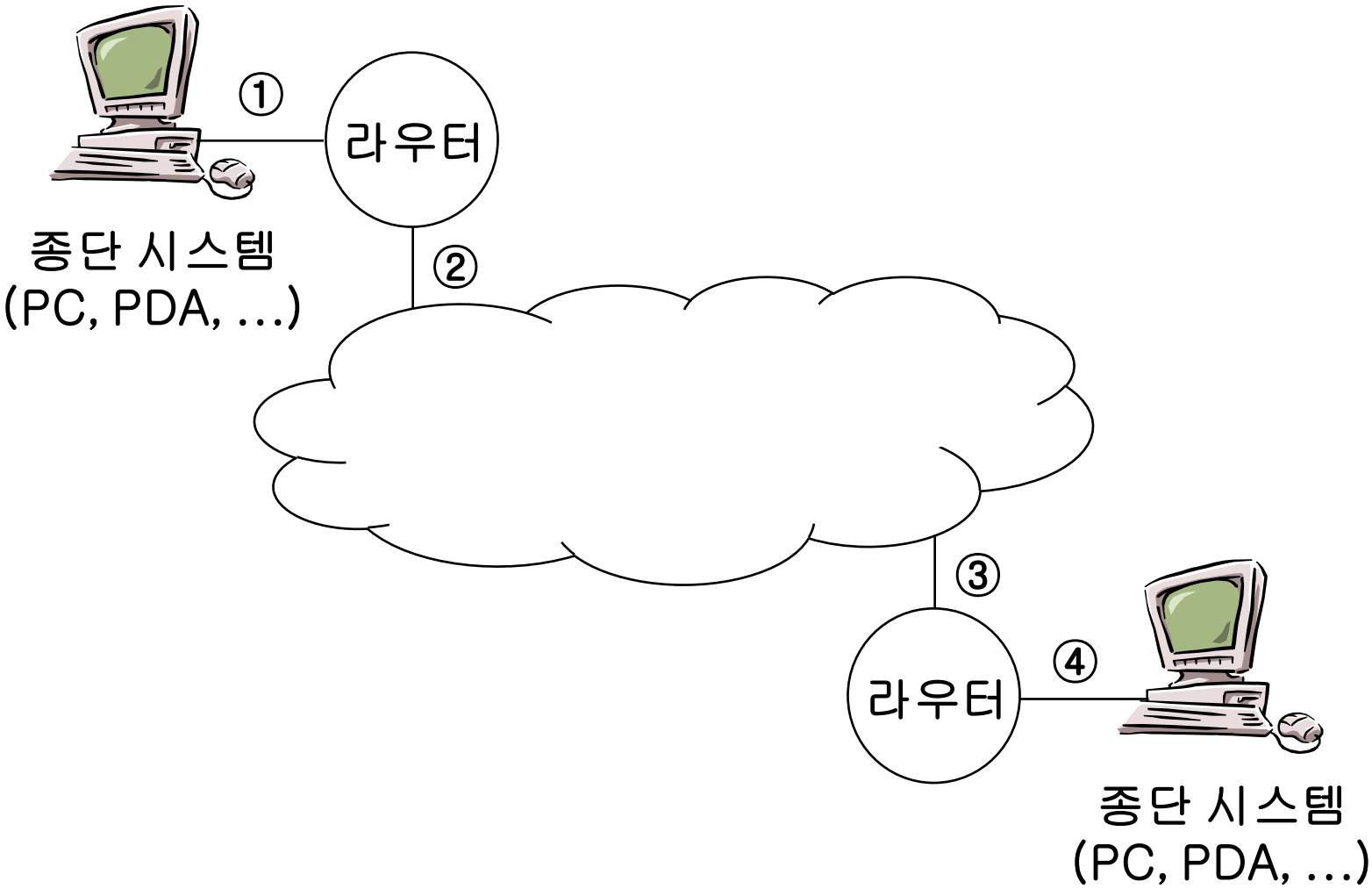
Network Programming - 01

afewhee@gmail.com

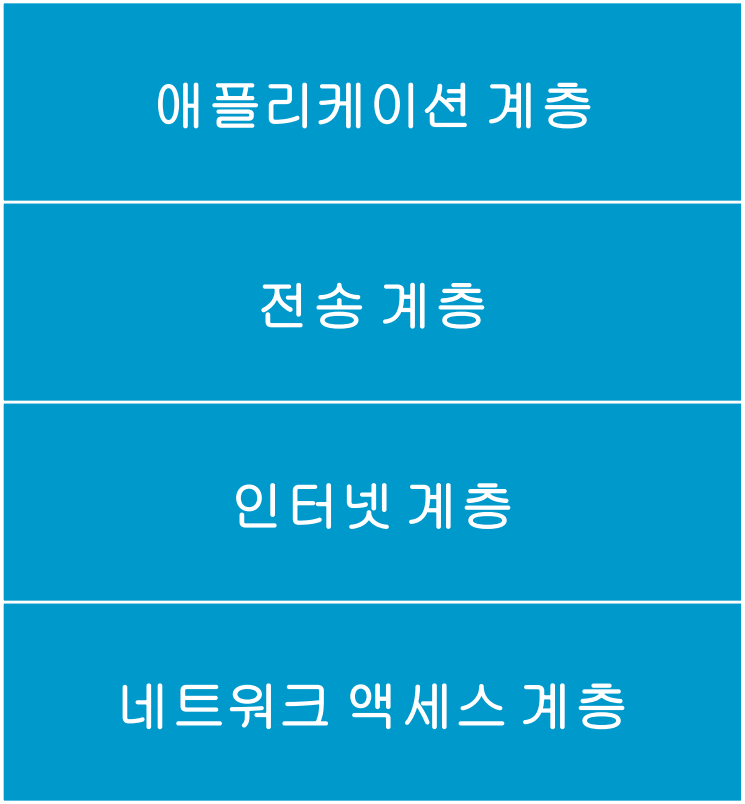
- 네트워크 기초
- 소켓
- TCP/IP
- 소켓 프로그래밍



- 종단 시스템(end-system)
  - 최종 사용자(end-user)를 위한 애플리케이션을 수행하는 주체
- 라우터(router)
  - 종단 시스템이 속한 네트워크와 다른 네트워크를 연결함으로써 서로 다른 네트워크에 속한 종단 시스템끼리 상호 데이터를 교환할 수 있도록 하는 장비
- 프로토콜(protocol)
  - 종단 시스템과 라우터간, 라우터와 라우터간, 그리고 종단 시스템과 종단 시스템간 통신을 수행하기 위한 정해진 절차와 방법

TCP/IP 프로토콜 구조

계층적 구조



TELNET, FTP, HTTP,
SMTP, MIME, SNMP, ...

TCP, UDP

IP

디바이스 드라이버
네트워크 하드웨어

- 네트워크 액세스 계층(network access layer)
  - 역할
    - 물리적 네트워크를 통한 실제적인 데이터 전송
  - 구성 요소
    - 네트워크 하드웨어 + 디바이스 드라이버
  - 주소 지정 방식
    - 물리 주소(physical address)
  - 예
    - 이더넷(Ethernet)

- 인터넷 계층(Internet layer)
  - 역할
    - 네트워크 액세스 계층의 도움을 받아, 전송 계층이 내려 보낸 데이터를 종단 시스템까지 전달
  - 구성 요소
    - 논리 주소 + 라우팅
  - 주소 지정 방식
    - IP 주소(Internet Protocol address)
  - 라우팅(routing)
    - 목적지까지 데이터를 전달하기 위한 일련의 작업
      - 라우팅을 위한 정보 획득
      - 라우팅 정보를 기초로 실제 데이터 전달(forward)

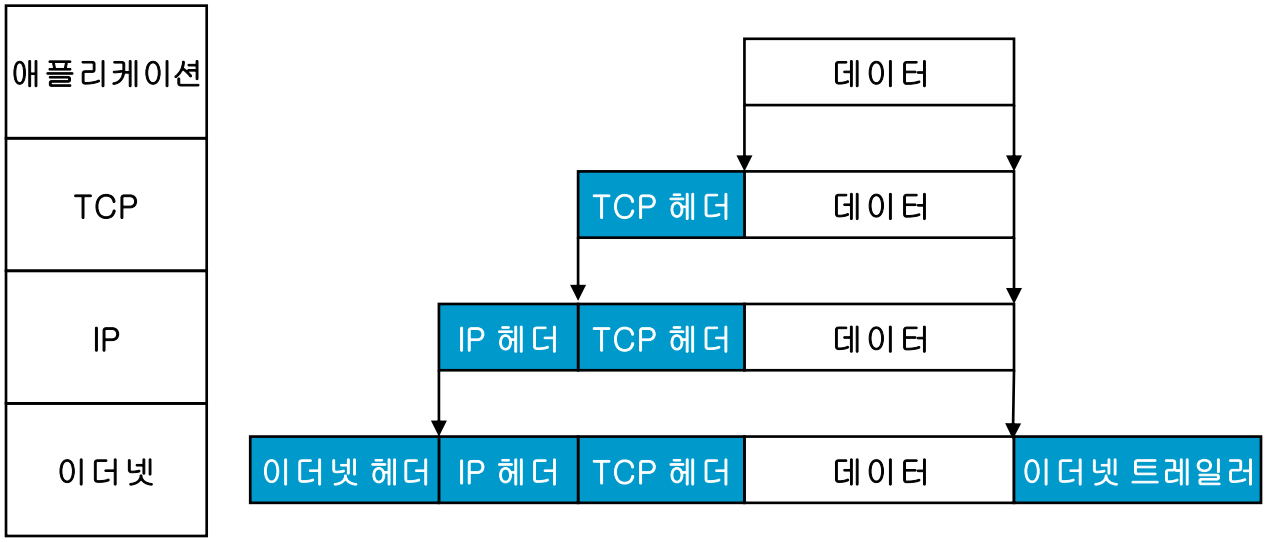
- 전송 계층(transport layer)
  - 역할
    - 최종적인 통신 목적지(프로세스)를 지정하고, 오류 없이 데이터를 전송
  - 주소 지정 방식
    - 포트 번호(port number)
  - 예
    - TCP(Transmission Control Protocol)
    - UDP(User Datagram Protocol)

TCP	UDP
연결형(connection-oriented) 프로토콜 - 연결이 성공해야 통신 가능	비연결형(connectionless) 프로토콜 - 연결 없이 통신 가능
데이터 경계를 구분하지 않음 - 바이트 스트림(byte-stream) 서비스	데이터 경계를 구분함 - 데이터그램(datagram) 서비스
신뢰성 있는 데이터 전송 - 데이터를 재전송함	비신뢰적인 데이터 전송 - 데이터를 재전송하지 않음
1 대 1 통신(unicast)	1 대 1 통신(unicast), 1 대 다 통신(broadcast), 다 대 다 통신(multicast)

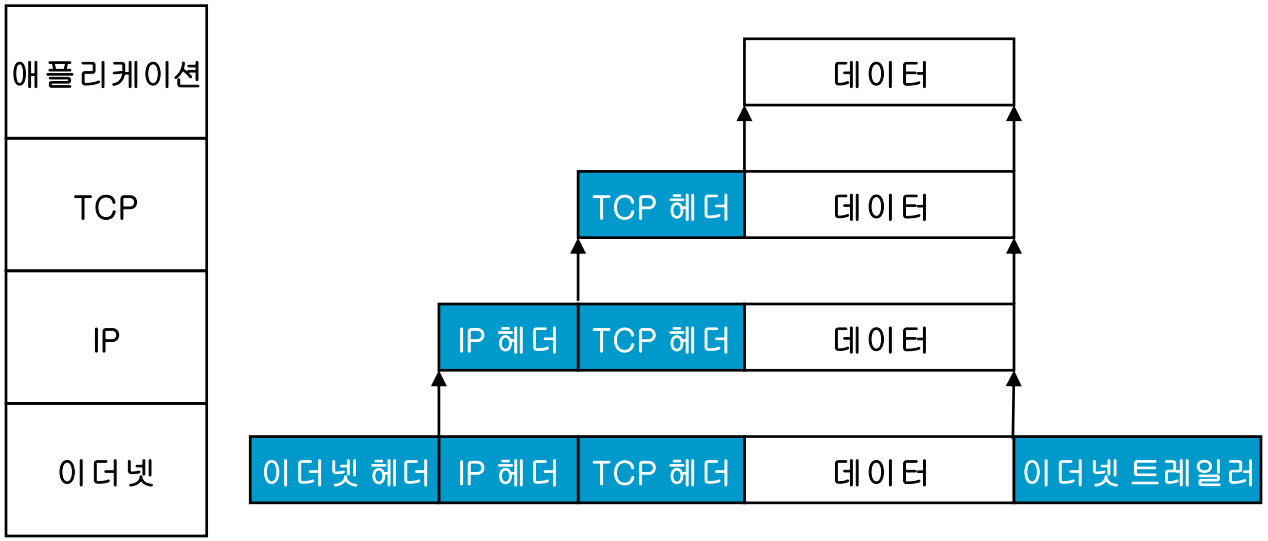
- 애플리케이션 계층(application layer)
  - 역할
    - 전송 계층을 기반으로 한 다수의 프로토콜과 이 프로토콜을 이용하는 애플리케이션을 포괄
    - 다양한 애플리케이션 서비스 제공
  - 예
    - Telnet, FTP, HTTP, SMTP 등
    - TCP/IP, UDP 기반 응용프로그램 들

- 패킷(packet)
  - 각각의 프로토콜에서 정의한 제어 정보(IP 주소, 포트 번호, 오류 체크 코드 등) + 데이터
  - 제어 정보의 위치에 따라, 앞쪽에 붙는 헤더(header)와 뒤쪽에 붙는 트레일러(trailer)로 구분

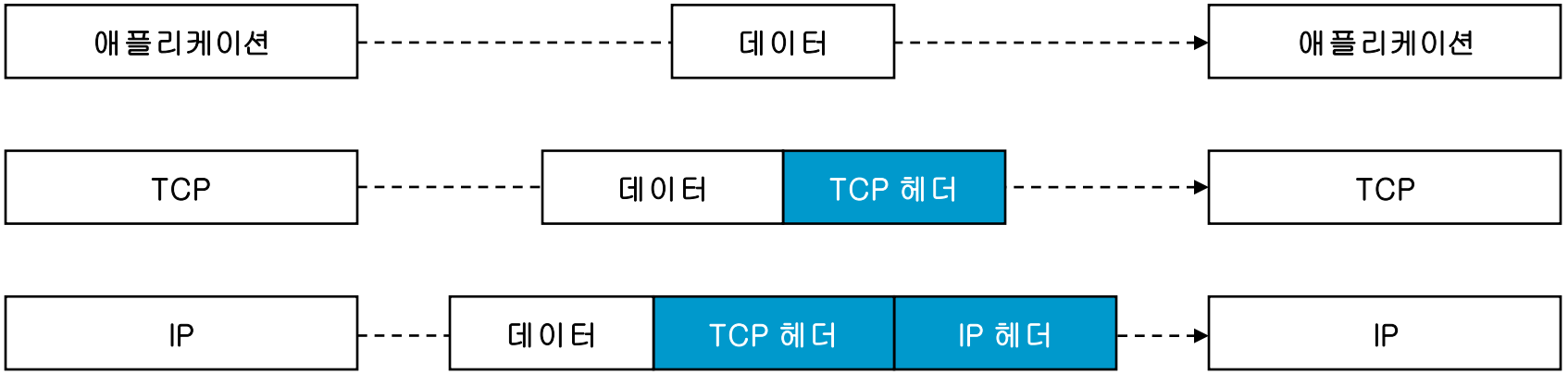
- 패킷 전송: 송신측



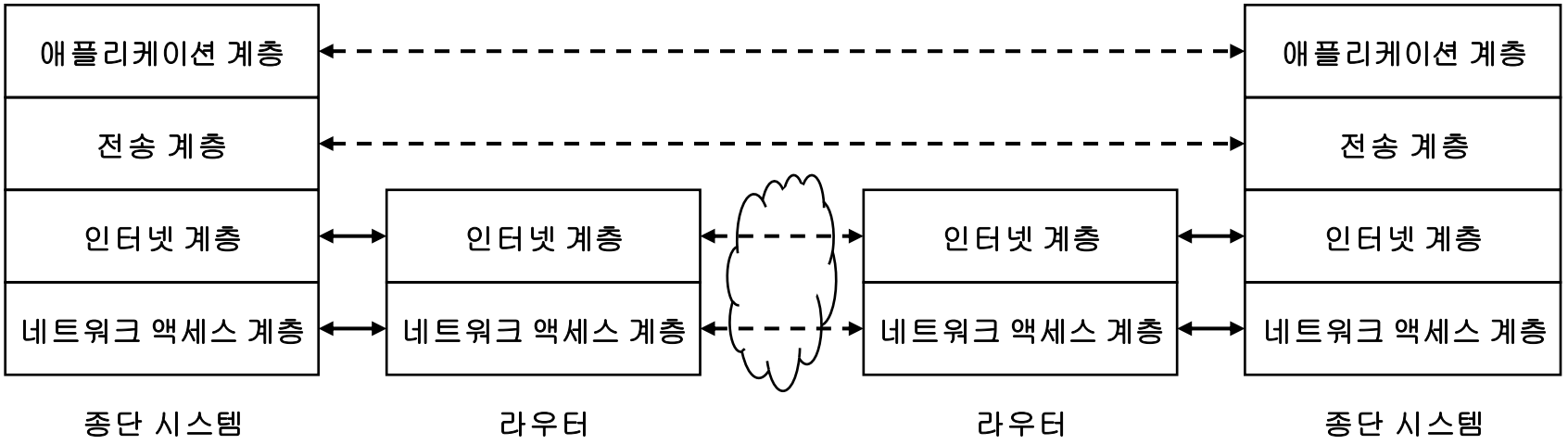
- 패킷 전송: 수신측



- 패킷 전송 형태:



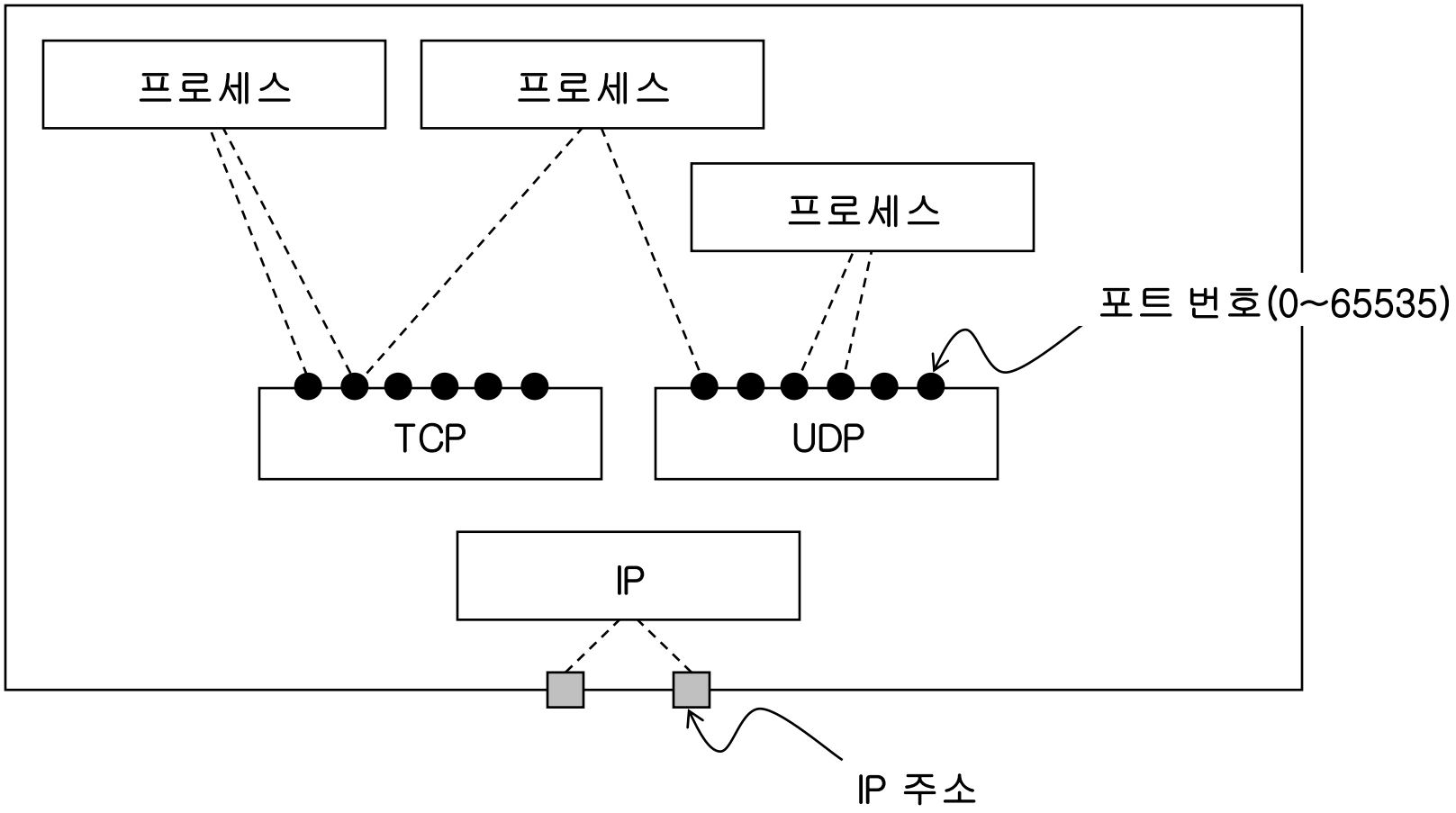
- TCP/IP 프로토콜을 이용한 패킷 전송



- IP 주소
  - 인터넷에 존재하는 호스트(종단 시스템, 라우터)를 유일하게 구별할 수 있는 식별자
  - IPv4는 32비트, IPv6는 128비트 사용
  - 8비트 단위로 구분하여 10진수로 표기(IPv4)
    - 예) 147.46.114.70
  - 루프백 주소(loop back address): 첫 번째 바이트가 127인 IP 주소

- 포트 번호
  - 통신 종착지(하나 혹은 여러 개의 프로세스)를 나타내는 식별자
  - 16 비트 논리적 할당(소켓에 할당)
  - Well-known ports : 0 ~ 1023

IP 주소와 포트 번호



- 세 가지 관점
  - ① 데이터 타입
  - ② 통신 종단점(communication end-point)
  - ③ 네트워크 프로그래밍 인터페이스

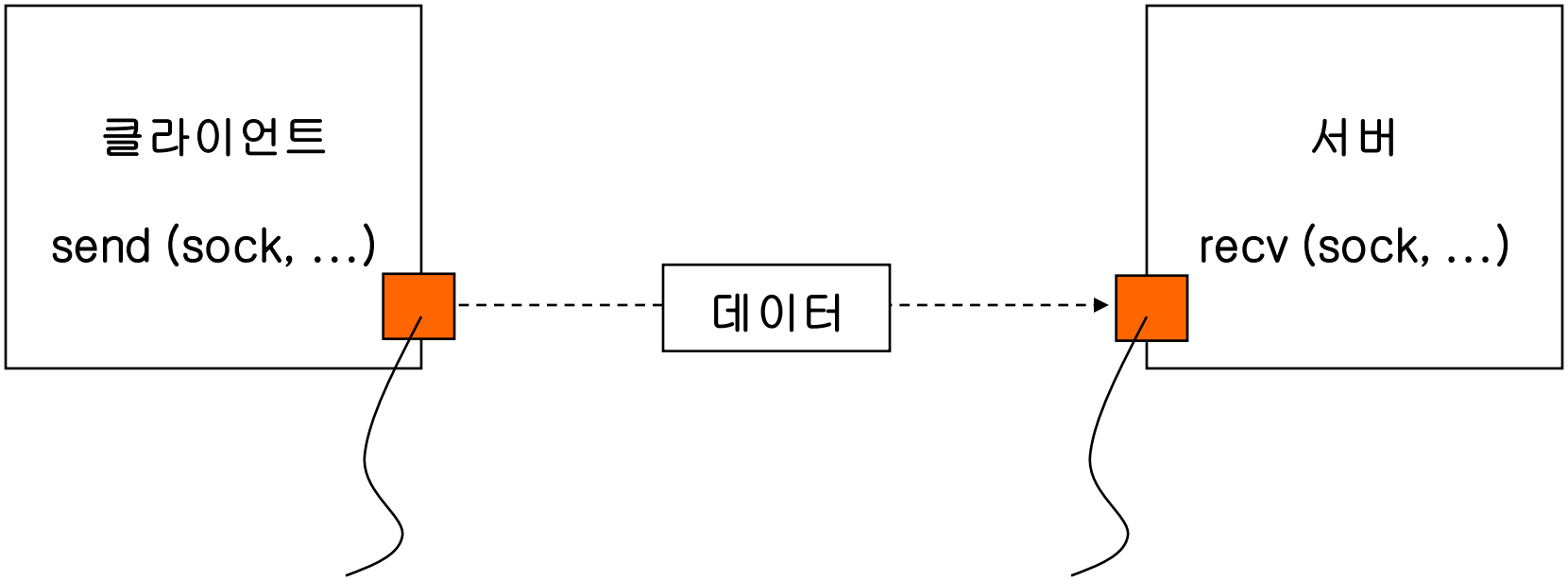
- 데이터 타입
  - 운영체제가 통신을 위해 관리하는 데이터를 간접적으로 참조할 수 있도록 만든 일종의 핸들(handle)
  - 생성과 설정 과정이 끝나면 이를 이용하여 통신과 관련된 다양한 작업을 할 수 있는 간편한 데이터 타입

```
// 파일 생성
int fd = open("myfile", ...);
...
read(fd, ...) // 읽기
write(fd, ...) // 쓰기
```

```
// 소켓 생성
SOCKET sock = socket(...);
...
recv(sock, ...) // 받기
send(sock, ...) // 보내기
```

- 통신 종단점
  - 소켓은 통신을 위해 필요한 여러 요소의 집합체
    - 사용할 프로토콜(TCP/IP, UDP/IP 등)
    - 송신측 IP 주소, 송신측 포트 번호
    - 수신측 IP 주소, 수신측 포트 번호
  - 애플리케이션은 자신의 소켓이 상대방의 소켓과 연결된 것으로 생각하고 데이터를 교환

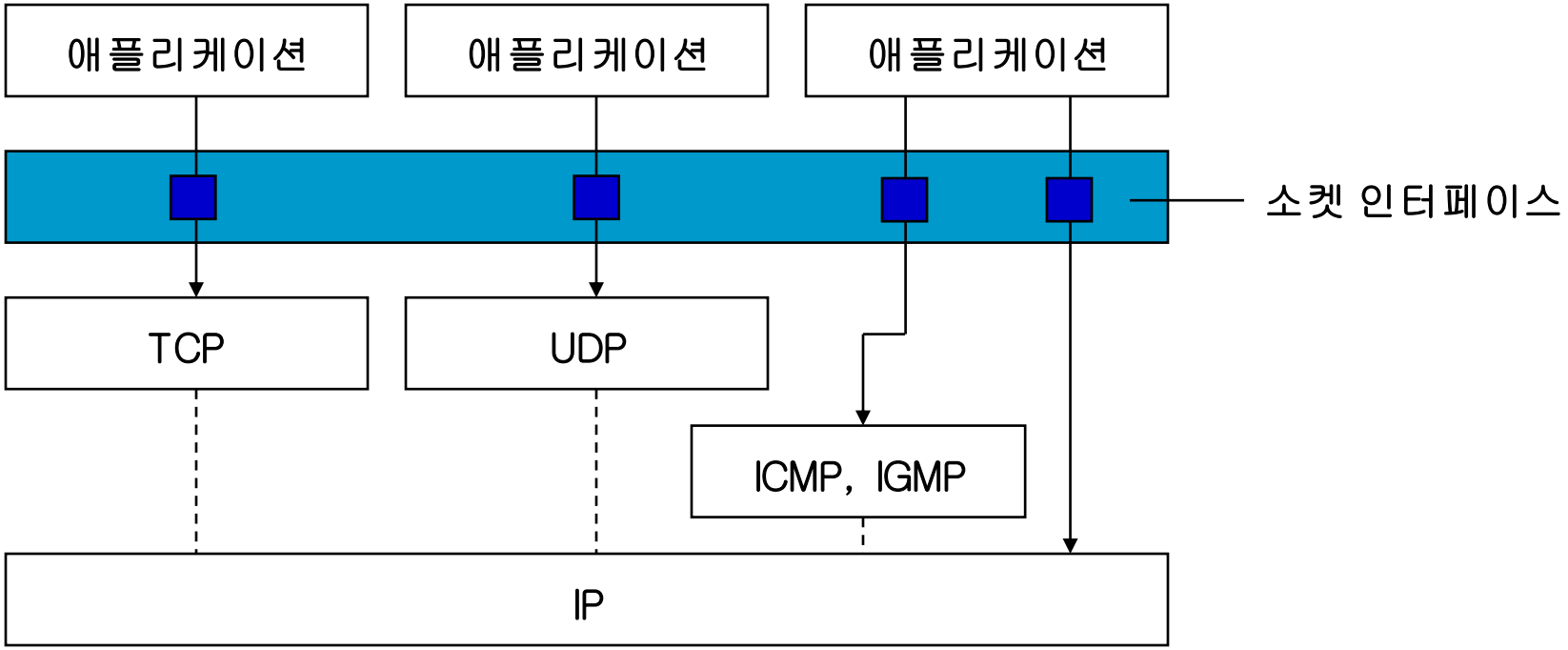
통신 종단점



<클라이언트 소켓>
- 프로토콜: TCP/IP
- IP 주소: 147.46.114.70
- 포트 번호: 12023

<서버 소켓>
- 프로토콜: TCP/IP
- IP 주소: 61.72.244.22
- 포트 번호: 9001

- 네트워크 프로그래밍 인터페이스
  - 통신 양단이 모두 소켓을 사용할 필요는 없음
  - TCP/IP 프로토콜 구조에서 (일반적으로) 애플리케이션 계층과 전송 계층 사이에 위치하는 것으로 간주



- 윈도우 소켓(Windows Sockets, Winsock)
  - 버클리 유닉스(Berkeley Software Distribution UNIX)에서 개발한 네트워크 프로그래밍 인터페이스를 윈도우 환경에서 사용할 수 있도록 한 것
  - 소스 코드 수준에서 호환성이 높음
  - 윈도우 95 버전부터 API(Application Programming Interface)에 정식으로 포함되어 제공

- 유닉스 소켓과의 차이점
  - 윈도우 소켓은 DLL을 통해 대부분의 기능이 제공되므로 DLL 초기화와 종료 작업을 위한 함수가 필요
  - 윈도우 애플리케이션은 대개 그래픽 사용자 인터페이스(GUI, Graphical User Interface)를 기반으로 하며, 메시지 구동 방식으로 동작하므로 이를 위한 확장 함수가 존재
  - 윈도우는 운영체제 차원에서 멀티스레드(multithread)를 지원하므로 멀티스레드 환경에서 안정적으로 동작하기 위한 구조와 이를 위한 함수가 필요

운영 체제 별 지원 사항

운영체제	윈속 버전
윈도우 95	1.1 (2.2)
윈도우 98/Me, 윈도우 NT/2000/XP/2003	2.2
윈도우 CE	1.1 (2.2)

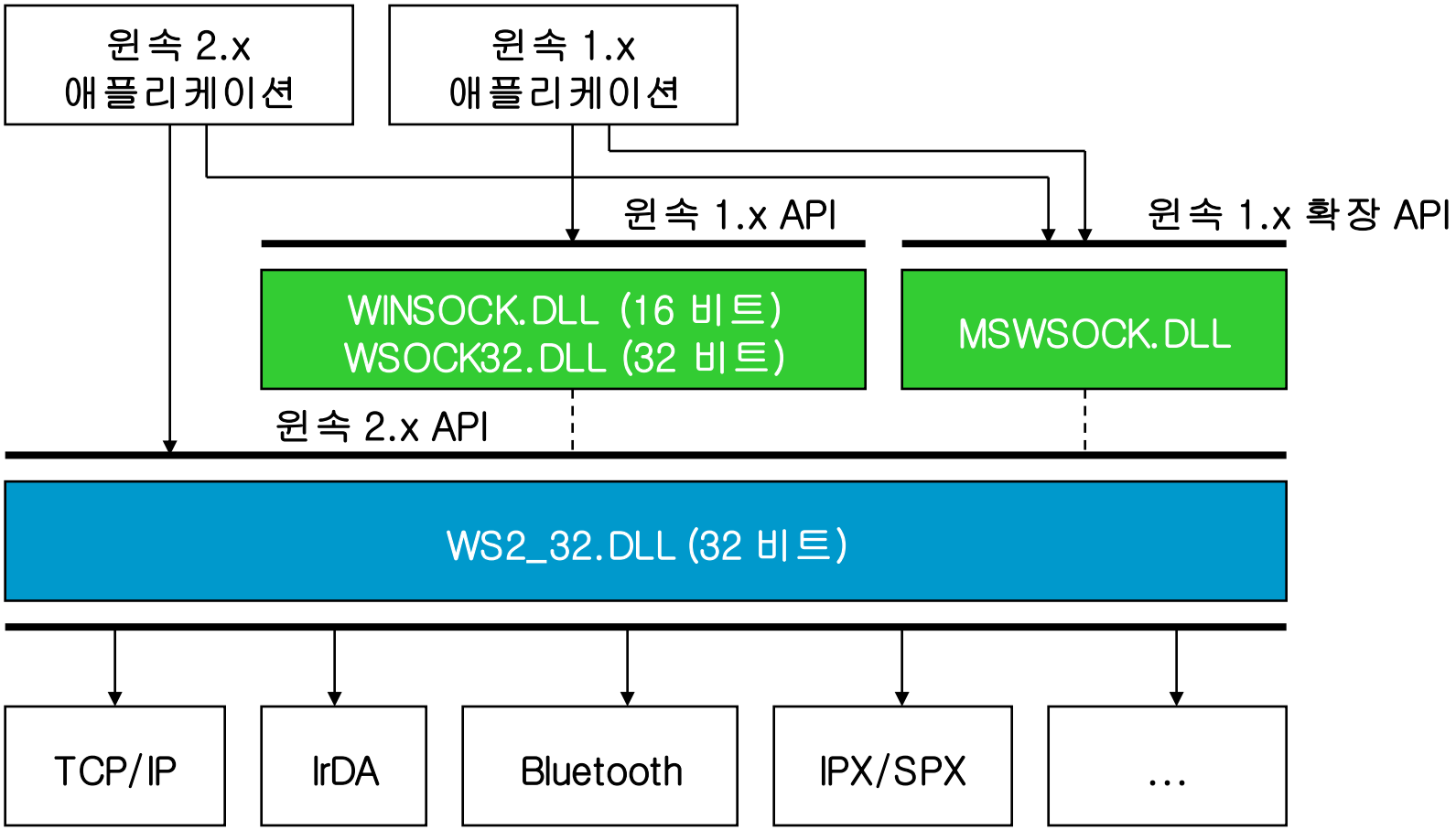
지원 프로토콜

- TCP/IP, IPv6(윈도우 XP 이상), IrDA(윈도우 98 이상), Bluetooth(윈도우 XP SP2 이상), IPX/SPX, ATM, DECNet, TP4(윈도우 2000부터 지원하지 않음), DLC(윈도우 XP부터 지원하지 않음), NetBEUI(윈도우 XP부터 지원하지 않음)

- 장점
  - 유닉스 소켓과 소스 코드 수준에서 호환성이 높으므로 기존 프로그램을 포팅 하기가 쉬움
  - 가장 널리 쓰이는 네트워크 프로그래밍 인터페이스이므로 한번 배워두면 여러 환경(윈도우, 유닉스, ...)에서 사용할 수 있음
  - TCP/IP 이외에도 다양한 종류의 프로토콜을 지원하므로 최소한의 코드 수정으로 애플리케이션에서 사용할 프로토콜을 변경 가능
  - 비교적 저 수준(low-level 혹은 mid-level)의 프로그래밍 인터페이스로, 세부적인 제어가 가능하므로 고성능의 네트워크 애플리케이션을 개발할 수 있음

- 단점
  - 애플리케이션 수준의 프로토콜을 프로그래머가 직접 설계해야 함
    - 데이터 포맷이나 전송 절차 등을 고려하여 프로그래밍해야 하므로 프로토콜을 변경할 경우 코드 수정이 불가피
  - 서로 다른 바이트 정렬(byte ordering) 방식을 사용하거나 데이터 처리 단위(32비트, 64비트, ...)가 서로 다른 종단 시스템간 통신을 할 경우, 애플리케이션 수준에서 데이터 변환을 처리해야 함

구조



- 윈속 라이브러리

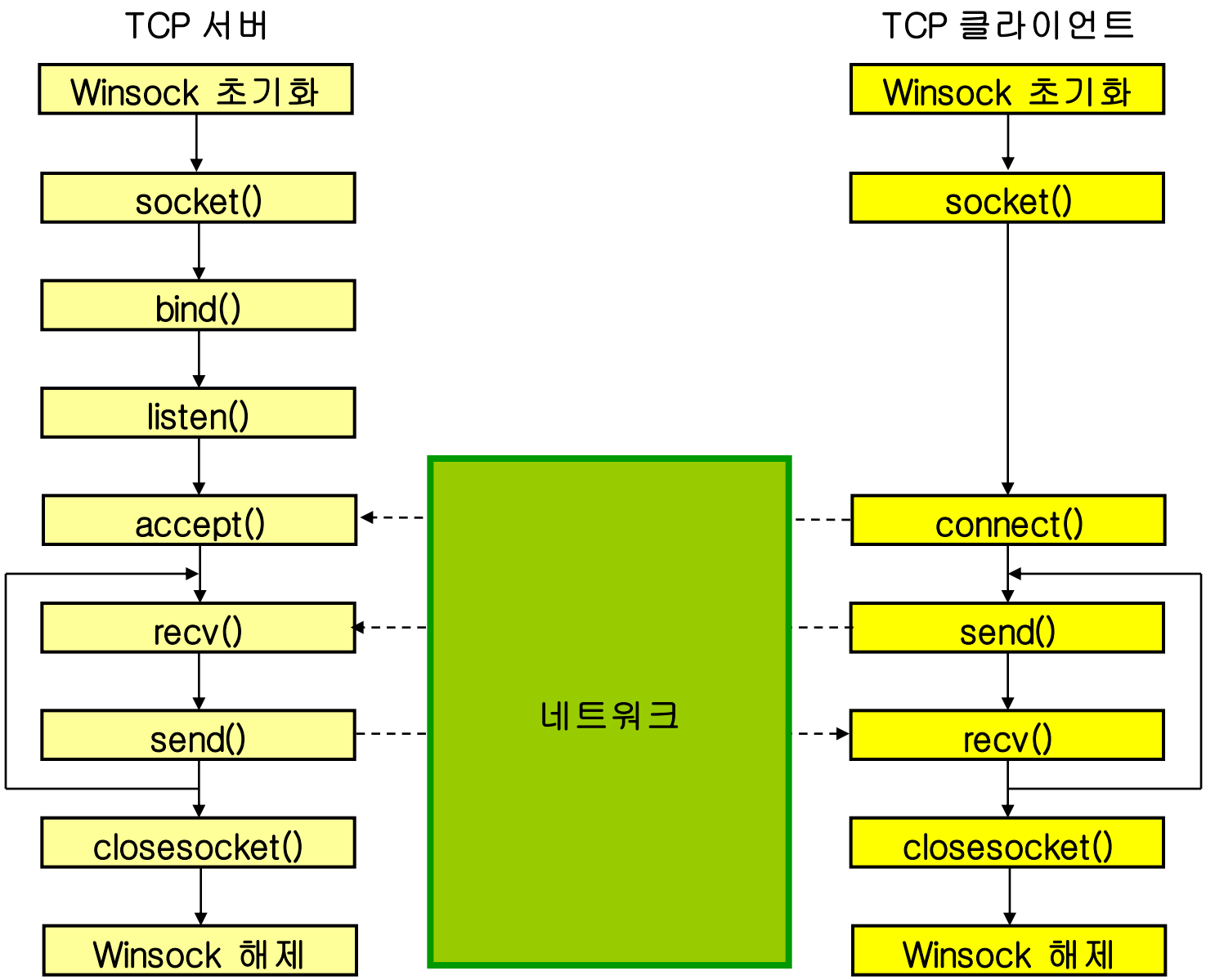
  #pragma comment(lib, “ws2_32.lib”)

- 윈속 헤더 파일

#include <winsock2.h>

winsock2.h는 Windows.h 보다 항상 앞서 있어야 함

- 윈속 응용 : TCP/IP, UDP, RAW, 무선 통신 등



TCP - 윈속 함수

- 윈속 초기화: WSAStartup() - 성공: 0, 실패: 오류 코드
- 윈속 종료: WSACleanup() - 성공: 0, 실패: SOCKET_ERROR
- 소켓 생성: socket() - 성공: 32Bit socket descriptor, 실패: INVALID_SOCKET
- 소켓 해제: closesocket() - 성공: 0, 실패: SOCKET_ERROR, 실패: INVALID_SOCKET

- 소켓 타입:
  - SOCK_STREAM: 신뢰성 있는 데이터 전송 기능. 연결형 프로토콜 ➔ TCP
  - SOCK_DGRAM: 신뢰성이 적은 데이터 전송 기능. 비연결형 프로토콜 ➔ UDP

- 주소 체계 : AF_INET
- 프로토콜: 프로토콜 명시적 지정
  - TCP ➔IPPROTO_TCP, UDP ➔ IPPROTO_UDP

인터넷 주소 초기화 예)

```
SOCKADDR_IN addr;
memset(&addr, 0, sizeof(SOCKADDR_IN));
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr(Server_IP);
// addr.sin_addr.s_addr = htons(INADDR_ANY);서버의 경우
addr.sin_port = htons(atoi("9190"));
```

- 오류 코드 얻기: int WSAGetLastError (void)

```
Ex)
hr = 소켓 처리 함수();

if(SOCKET_ERROR == hr)
{
    hr = WSAGetLastError();
```

- 오류 메시지를 문자열로 전환: FormatMessage(void)

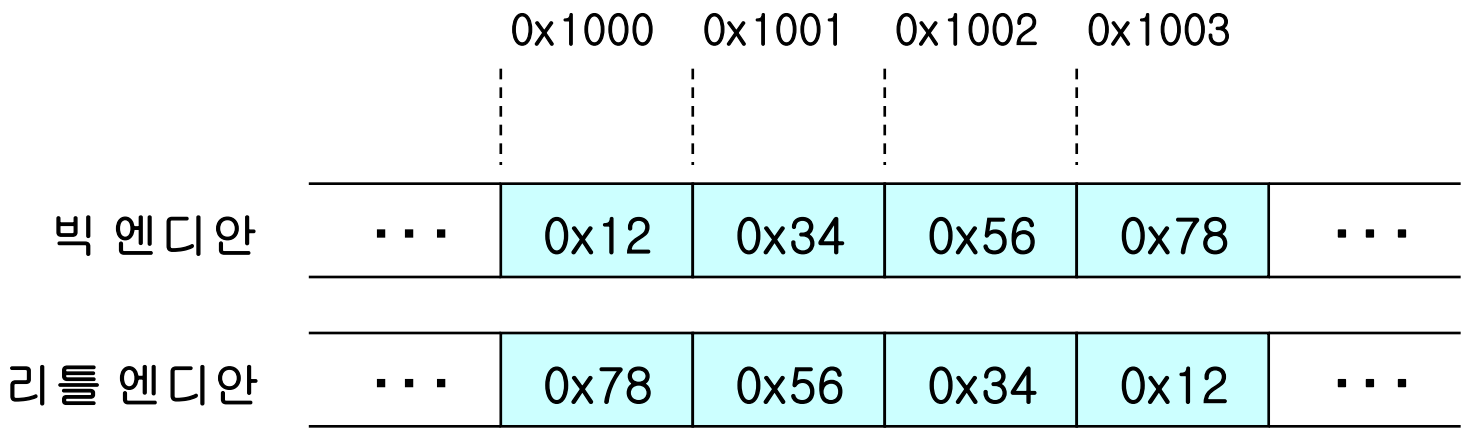
```
Ex)
hr = WSAGetLastError();

LPVOID lpMsgBuf;
FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS, NULL,
        hr,
        0,
        (LPTSTR) &lpMsgBuf, 0, NULL);

printf("Error:%s₩n", (LPCTSTR)lpMsgBuf);
LocalFree( lpMsgBuf );
```

- bind() - 서버
  - 성공: 0, 실패: SOCKET_ERROR
  - 서버의 지역 IP 주소와 지역 포트 번호를 결정
- listen() - 서버
  - 성공: 0, 실패: SOCKET_ERROR
  - 소켓과 결합된 TCP 포트 상태를 LISTENING으로 변경
- accept() - 서버
  - 성공: 새로운 소켓, 실패: INVALID_SOCKET
  - 접속한 클라이언트와 통신할 수 있도록 새로운 소켓을 생성하여 리턴
  - 접속한 클라이언트의 IP 주소와 포트 번호를 알려줌
- connect() - 클라이언트
  - 성공: 0, 실패: SOCKET_ERROR
  - 서버에게 접속하여 TCP 프로토콜 수준의 연결 설정

- send() 함수
  - 성공: 보낸 바이트 수, 실패: SOCKET_ERROR
  - 애플리케이션 데이터를 송신 버퍼에 복사함으로써 궁극적으로 하부 프로토콜(예를 들면, TCP/IP)에 의해 데이터가 전송되도록 함

- recv() 함수
  - 성공: 받은 바이트 수 또는 0(연결 종료시),
  - 실패: SOCKET_ERROR
  - 수신 버퍼에 도착한 데이터를 애플리케이션 버퍼로 복사
  - recvn(): 특정 길이 만큼 읽음

- 바이트 정렬(byte ordering)
  - 메모리에 데이터를 저장할 때의 바이트 순서
    - 빅 엔디안(big-endian), 리틀 엔디안(little-endian)
  - 네트워크의 IP, 포트 번호는 빅 엔디안



- 바이트 정렬 함수(유닉스 호환)

```
u_short htons (u_short hostshort); // host-to-network-short
u_long htonl (u_long hostlong); // host-to-network-long
u_short ntohs (u_short netshort); // network-to-host-short
u_long ntohl (u_long netlong); // network-to-host-long
```

- 바이트 정렬 함수(윈속 확장)

```
int WSAHtons (SOCKET s, u_short hostshort, u_short* lpnetshort);
int WSAHtonl (SOCKET s, u_long hostlong, u_long* lpnetlong);
int WSANtohs (SOCKET s, u_short netshort, u_short* lphostshort);
int WSANtohl (SOCKET s, u_long netlong, u_long* lphostlong);
```

- IP 주소 변환 함수
  - unsigned long inet_addr (const char* cp);
    - 문자열 형태로 IP 주소를 입력받아 32비트 숫자(네트워크 바이트 정렬)로 리턴
  - char* inet_ntoa (struct in_addr in);
    - network-to-ascii
    - 32비트 숫자(네트워크 바이트 정렬)로 IP 주소를 문자열 형태로 리턴
  - struct hostent* gethostbyname ();
    - 도메인 이름 ➡ IP 주소(네트워크 바이트 정렬)
  - struct hostent* gethostbyaddr (...);
    - IP 주소(네트워크 바이트 정렬) ➡ 도메인 이름