컴퓨터 네트워크

Socket 프로그래밍

Ducsun Lim
Mobile&Network Intelligence Lab.
Hanyang University
2019/11/19

네트워크 프로그래밍과 소켓에 대한 이해

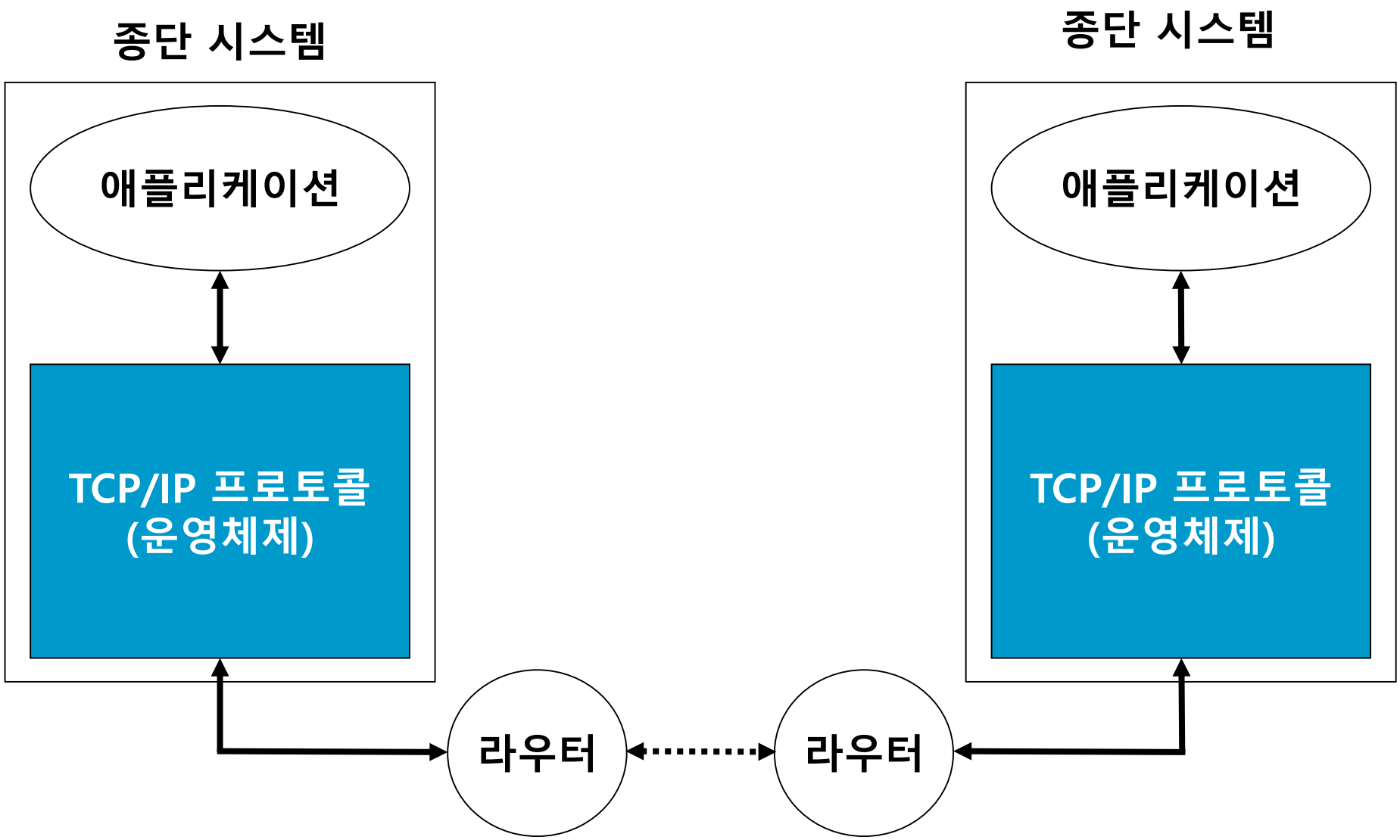
❑ 네트워크 프로그래밍

- 소켓이라는 것을 기반으로 프로그래밍을 하기 때문에 소켓 프로그래밍이라고도 함
- 네트워크로 연결된 둘 이상의 컴퓨터 사이에서의 데이터 송수신 프로그램의 작성을 의미함

❑ 소켓에 대한 간단한 이해

- 네트워크(인터넷)의 연결도구
- 운영체제에 의해 제공이 되는 소프트웨어적인 장치
- 소켓은 프로그래머에게 데이터 송수신에 대한 물리적, 소프트웨어적 세세한 내용을 신경 쓰지 않게 함

TCP/IP 프로토콜



❑ TCP/IP 프로토콜 구조

– 계층적 구조

TCP/IP 프로토콜 구조

❑ 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)

TCP/IP 프로토콜 구조

❑ 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/IP 프로토콜 구조

❑ TCP와 UDP

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)

TCP/IP 프로토콜 구조

❑ application layer

- 역할
  - 전송 계층을 기반으로 한 다수의 프로토콜과 이 프로토콜을 이용하는 애플리케이션을 포괄
  - 다양한 애플리케이션 서비스 제공
- 예
  - Telnet, FTP, HTTP, SMTP 등

패킷 전송 원리

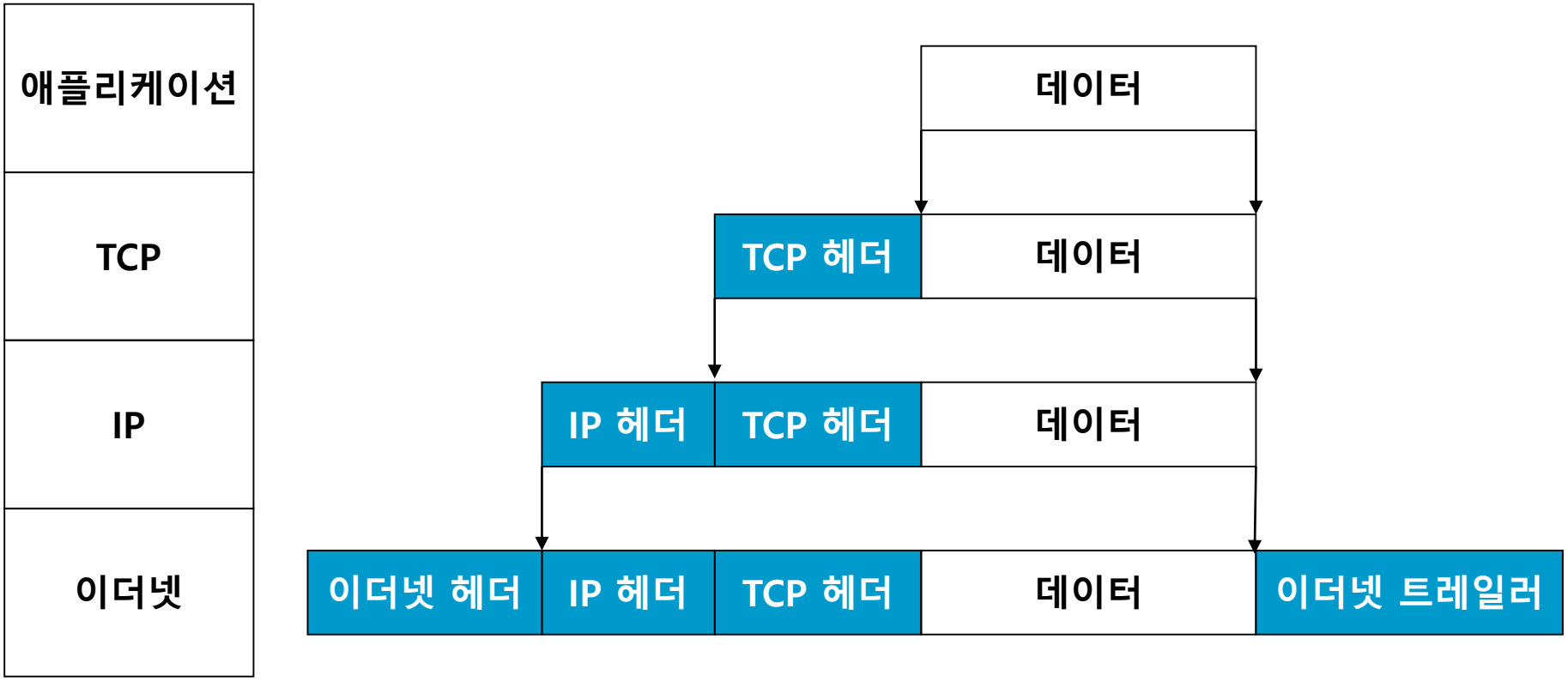
❑ packet

- 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 원리

❑ 패킷 전송 형태

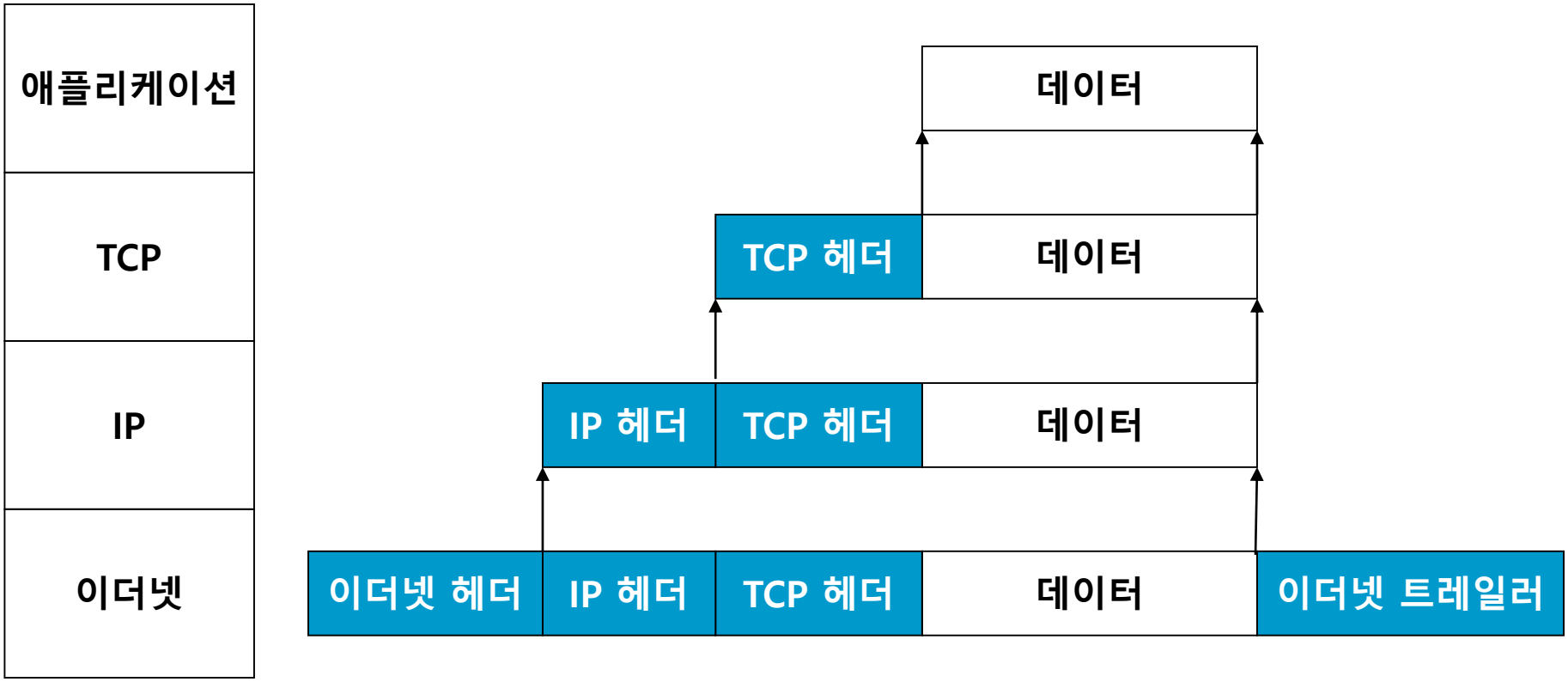
– 송신측



패킷 전송 원리

❑ 패킷 전송 형태

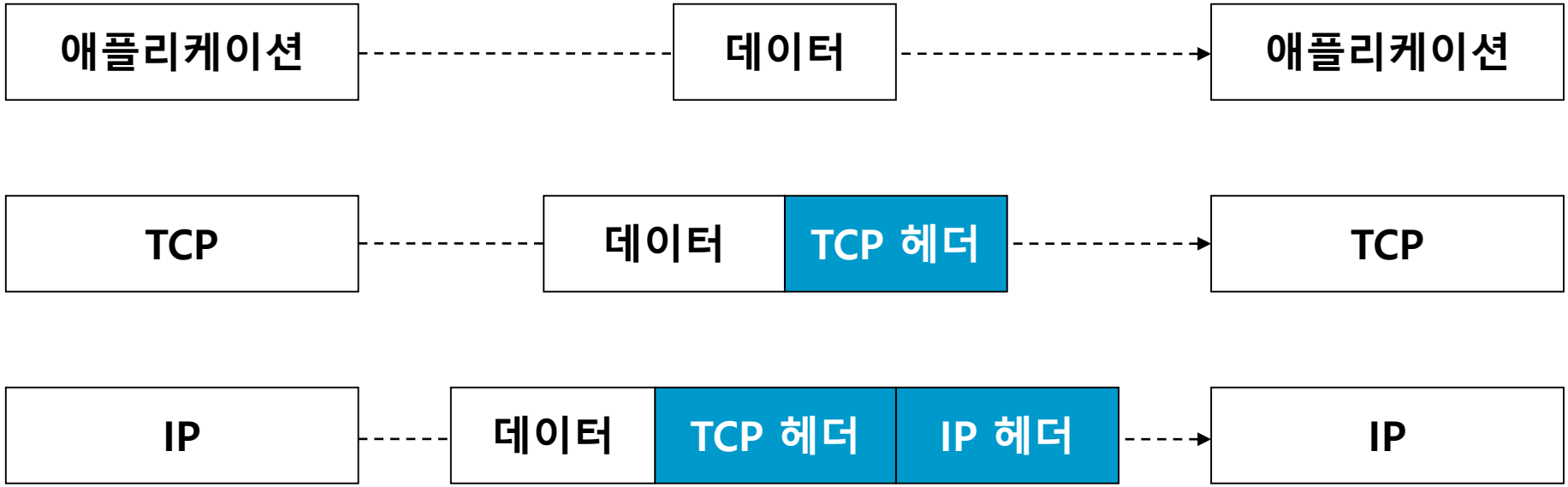
– 수신측



패킷 전송 원리

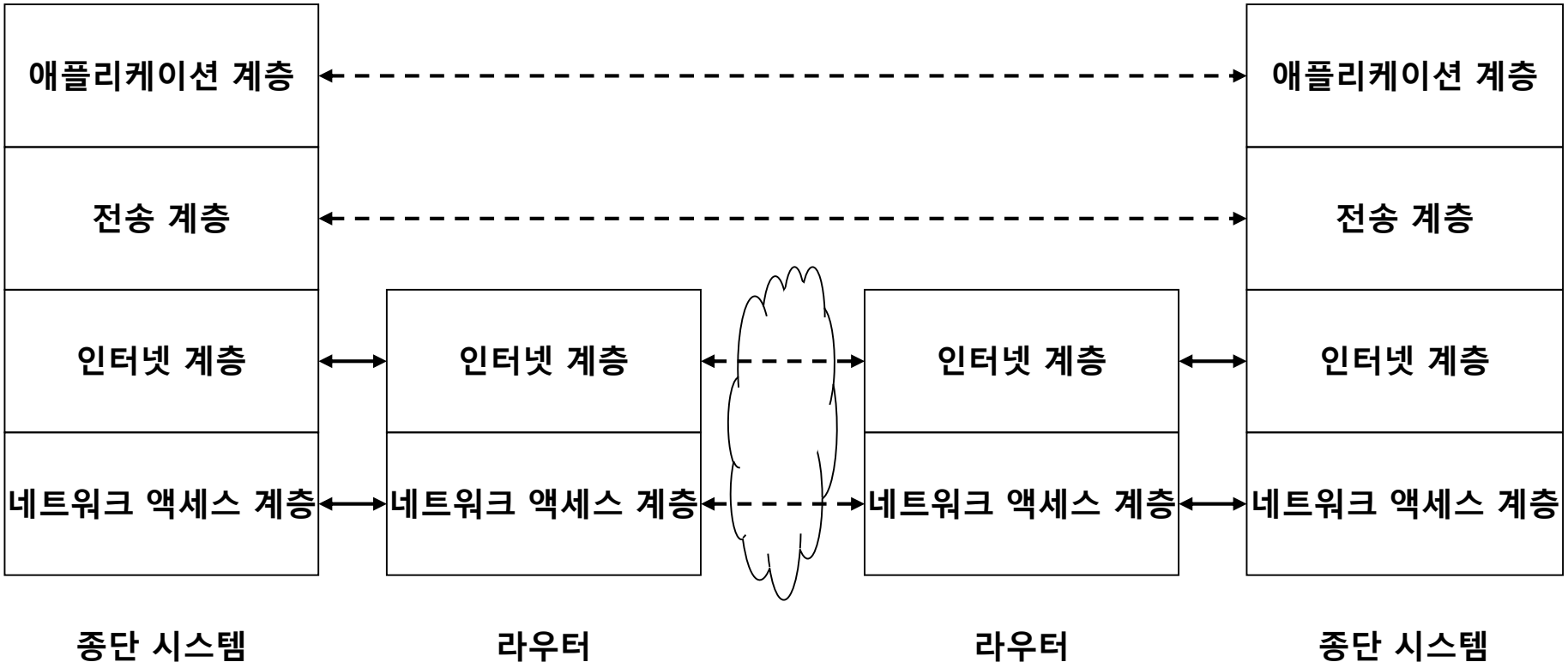
❑ 패킷 전송 형태

– 계층별



패킷 전송 원리

❑ TCP/IP 프로토콜을 이용한 패킷 전송



IP 주소와 포트 번호

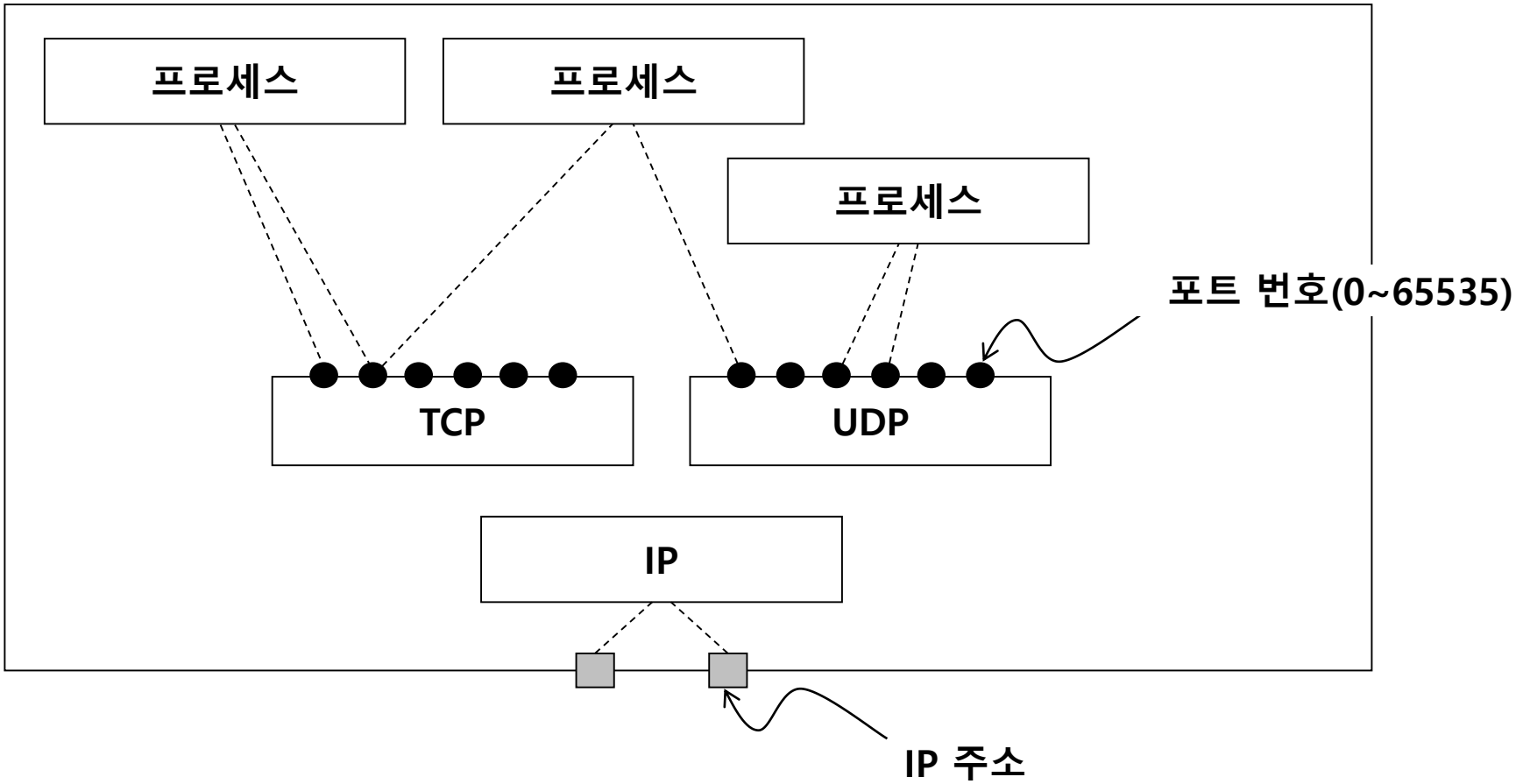
❑ IP 주소

- 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

❑ 포트 번호

- 통신 종착지(하나 혹은 여러 개의 프로세스)를 나타내는 식별자

❑ IP 주소와 포트 번호



IP 주소와 포트 번호

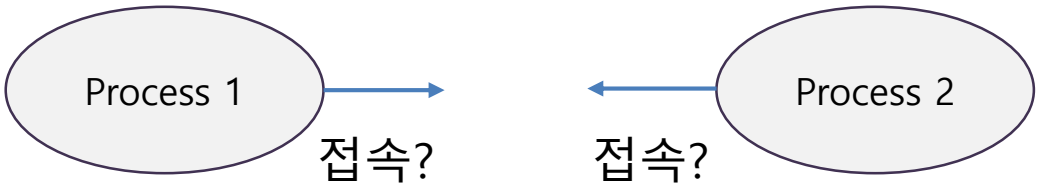
❑ domain name

- IP 주소는 사람이 기억하고 사용하기에 불편한 점이 많으므로 domain name을 사용하는 경우가 많음
- 실제 통신을 하기 위해서는 IP 주소로 변환해야 함
  - 네트워크 프로그래밍을 할 때 127.x.x.x 형태가 있는데, 이를 루프백 주소(loopback address)라 부름
  - Loopback address는 인터넷에 존재하는 호스트의 고유한 주소로는 할당되지 않으며, 컴퓨터 자신을 나타내는 의미로 내부적으로만 사용됨

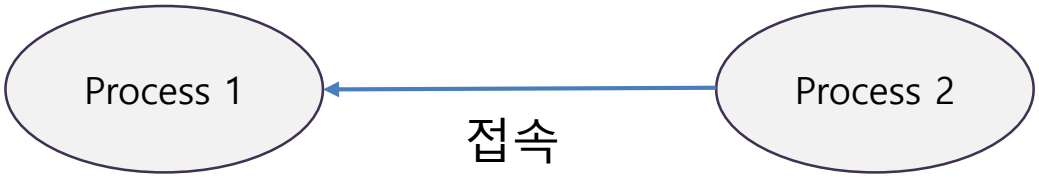
client/server

❑ client/server model

- 네트워크 프로그램은 일반적으로 client/server model로 작성함
- 실행되는 두 프로그램간에 C/S 모델이 적용될 경우, 두 프로그램은 프로스세 간 통신 (IPC, Inter-Process Communication) 기법을 통해 상호 정보를 교환함



두 프로세스의 통신-동시접속



두 프로세스의 통신-서버와 클라이언트

소켓의 생성

❑ 소켓의 비유와 분류

- TCP 소켓은 전화기에 비유될 수 있음
- 소켓은 socket 함수의 호출을 통해서 생성함
- 단, 전화를 거는 용도의 소켓과 전화를 수신하는 용도의 소켓 생성 방법에는 차이가 있음

```
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

➡ 성공 시 파일 디스크립터, 실패 시 -1 반환

소켓의 생성은 전화기의 장만에 비유할 수 있음

전화번호의 부여, 연결

❑ 소켓의 주소 할당 및 연결

- 전화기에 전화번호가 부여되듯이 소켓에도 주소 정보가 할당됨
- 소켓의 주소정보는 IP와 PORT 번호로 구성이 됨

```
#include <sys/socket.h>
int bind(int sockfd, struct sockaddr *myaddr, socklen_t addrlen);
```

➜ 성공 시 0, 실패 시 -1 반환

전화기 연결

❑ 연결요청이 가능한 상태의 소켓

- 연결요청이 가능한 상태의 소켓은 걸려오는 전화를 받을 수 있는 상태에 비유
- 전화를 거는 용도의 소켓은 연결요청이 가능한 상태의 소켓이 될 필요가 없음

  이는 걸려오는 전화를 받는 용도의 소켓에서 필요한 상태임

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

➜ 성공 시 0, 실패 시 -1 반환

소켓에 할당된 IP와 port 번호로 연결요청이 가능한 상태가 됨

수화기를 드는 상황

❑ 연결요청의 수락

- 걸려오는 전화에 대해서 수락의 의미로 수화기를 드는 것에 비유할 수 있음
- 연결요청이 수락되어야 데이터의 송수신이 가능함
- 수락된 이후에 데이터의 송수신은 양방향으로 가능함

연결요청 가능한 상태로 변경

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

➜ 성공 시 파일 디스크립터, 실패 시 -1 반환

accept 함수호출 이후에는 데이터의 송수신이 가능함

단, 연결요청이 있을 때에만 accept 함수가 반환을 함

소켓의 개념

❑ 세 가지 관점에서의 소켓의 개념

- 데이터 타입
- 통신 종단점(communication end-point)
- 네트워크 프로그래밍 인터페이스

소켓의 개념

❑ 데이터 타입

– file descriptor 혹은 handle과 유사한 개념

```
int fd = open("my life", ...); //파일생성
...
read(fd, ...) //읽기
write(fd, ...) //쓰기
```

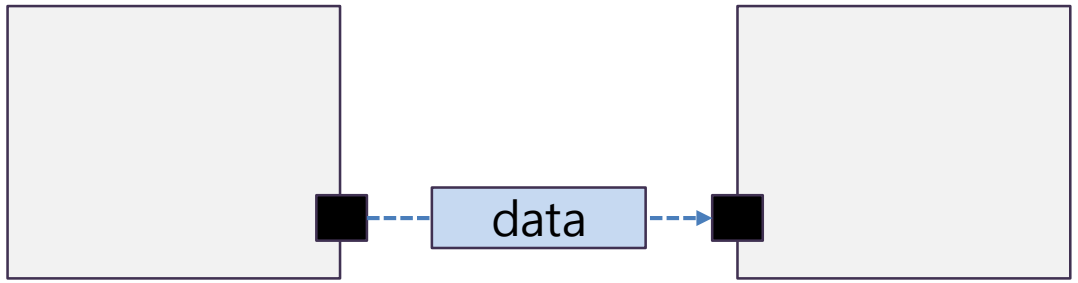
```
SOCKET sock = socket(...); //소켓 생성
...
recv(sock, ...) //받기
send(sock, ...) //보내기
```

소켓의 개념

❑ 통신 종단점(communication end-point)

– TCP/IP protocol을 이용하여 애플리케이션이 통신을 하기 위해서는 다음과 같은 요소가 결정되어야 함

- 사용할 protocol(TCP/IP, UDP/IP, ...)
- 송신측 IP
- 송신측 포트 번호
- 수신측 IP
- 수신측 포트번호



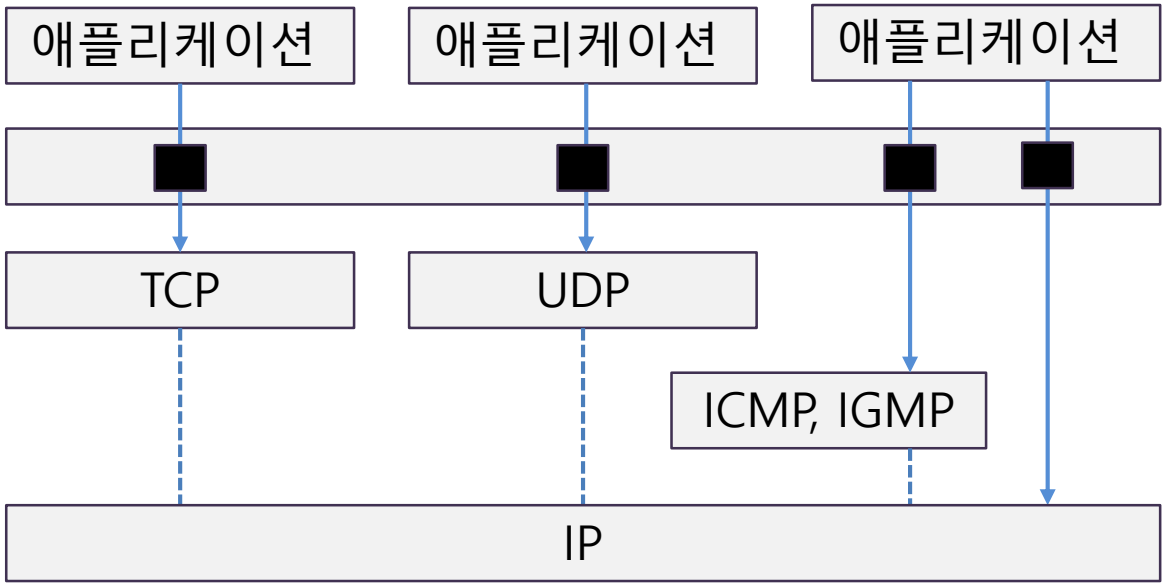
<클라이언트 소켓>
- 프로토콜: TCP/IP
- IP 주소: 166.104.114.70
- 포트 번호: 12023

<서버 소켓>
- 프로토콜: TCP/IP
- IP 주소: 61.72.224.22
- 포트 번호: 9001

소켓의 개념

❑ 네트워크 프로그래밍 인터페이스

- TCP/IP 프로토콜 관점에서 소켓은 하나의 네트워크 프로그래밍 인터페이스에 불과함
- 애플리케이션이 통신을 하기 위해 양쪽 모두 소켓을 사용해야 하는 것은 아님
- 양쪽 모두 동일한 프로토콜을 사용하고, 정해진 형태와 절차에 따라 데이터를 주고 받으면 됨



Socket 프로그래밍

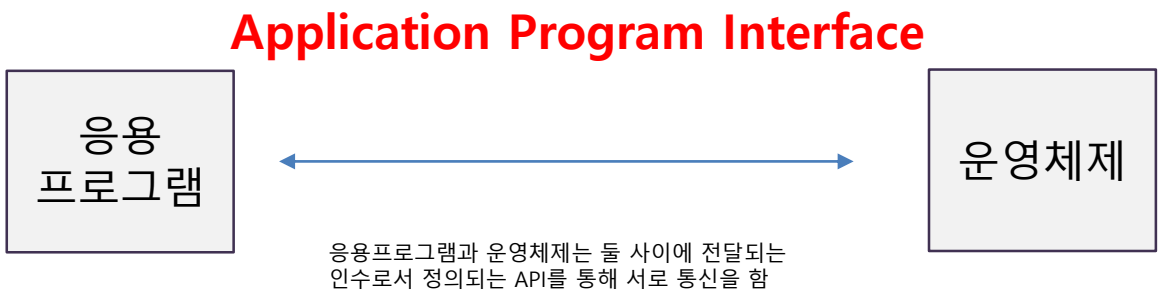
– TCP/IP 프로그래밍

- **연결(Association)**
  - Client와 Server간의 정보 교환이 이루어지기 전에 요구되는 정보
    - ➔ 프로토콜 식별자
    - ➔ 로컬 인터넷 주소
    - ➔ 로컬 포트 번호
    - ➔ 원격 인터넷 주소
    - ➔ 원격 포트 번호
- **Transport layer protocol port**
  - TCP/IP 전송 계층 프로토콜의 서비스를 사용하는 각 프로세스는 하나의 포트 번호가 할당 됨
    - ➔ 포트 0 : 사용되지 않음
    - ➔ 포트 1-255 : 잘 알려진(well-known) 포트 번호
    - ➔ 포트 256-1023 : 예약된(reserved) 포트 번호
    - ➔ 포트 1024-4999 : 잠시 사용하는 클라이언트 포트 번호
    - ➔ 포트 5000-65535 : 사용자 정의 서버 포트 번호

Socket 프로그래밍

– 통신 응용 프로그램 인터페이스(API)

- Socket 시스템 호출
- RPC(Remote Procedure Call)
- TLI(Transport Layer Interface)
- Socket 인터페이스는 보통 두 개의 응용 프로그램이 다른 컴퓨팅 시스템에서 실행 될 때 이들 간에 클라이언트-서버 관계를 구현하기 위해 사용
- Socket API는 TCP/IP 이외의 프로토콜에 대한 접근을 제공하기 위해 사용



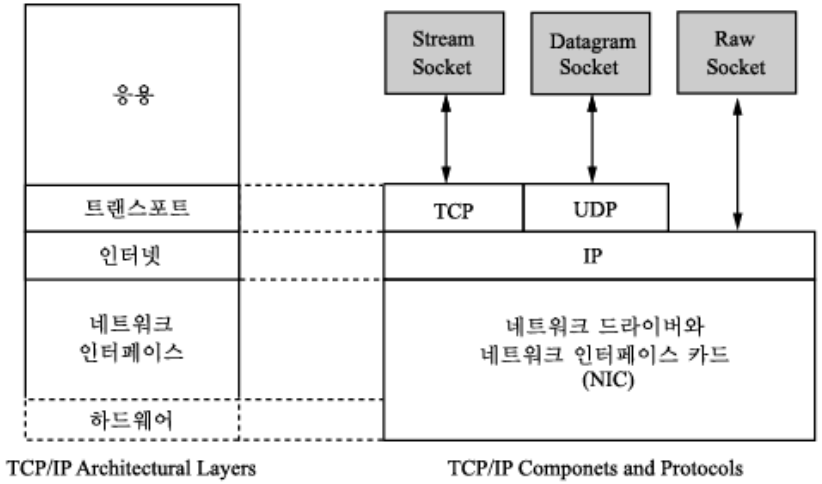
Socket 프로그래밍

– Socket의 형태

- **stream socket**
  - TCP Transport Layer Protocol을 사용하여 통신하는 소켓
  - 연결-지향 형태를 지원
- **Datagram socket**
  - UDP 전송 계층 프로토콜을 사용하여 통신하는 소켓
  - 신뢰적이지 못한 데이터그램 형태를 지원
- **Raw socket(원시 소켓)**
  - 하위의 IP와 ICMP로의 접근을 제공하는 소켓
  - 특정 목적을 위해 사용

Socket 프로그래밍

– 소켓 간의 관계



– socket 주소

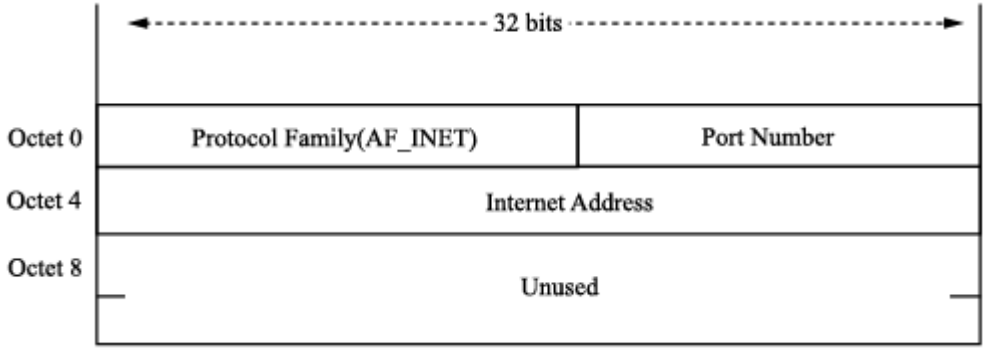
- 많은 소켓 시스템 호출은 소켓 주소 정보를 포함하는 자료 구조에 대한 포인터를 참조함
- UNIX 환경에서 다음 include는 응용 프로그램에서 소켓 식별자와 자료 구조를 이용할 수 있도록 함

```
#include<sys/types.h>
#include<sys/socket.h>
```

Socket 프로그래밍

– 소켓 주소 자료 구조는 아래와 같은 정보를 포함함

- Protocol Family : 사용될 Protocol을 식별하는 16비트 정수값
- Port Number : 프로세스에 할당된 포트 번호를 식별하는 16비트 정수값
- Address : 프로세스가 실행되는 호스트의 인터넷 주소를 포함하는 32비트 정수값

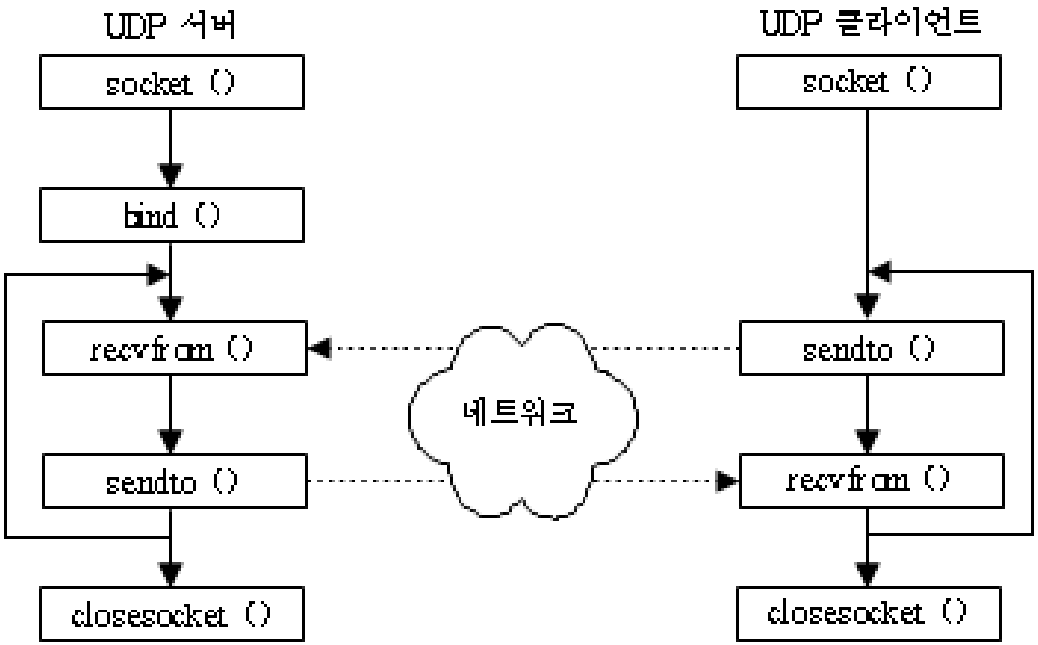


소켓 주소 자료 구조와 자료형

Socket 프로그래밍

– 응용 프로토콜

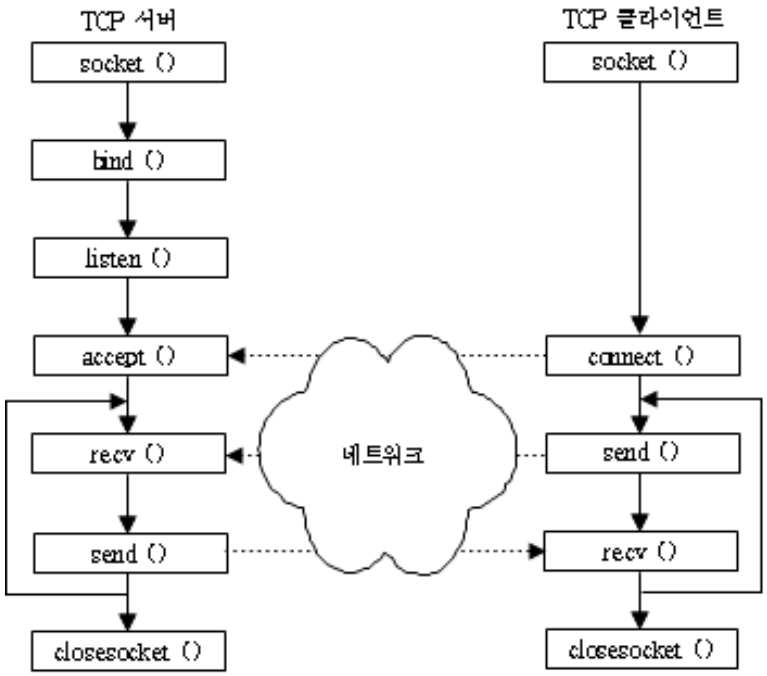
- **비연결(Connectionless) 응용 프로토콜**
- 비연결 응용 프로토콜을 구현하는 클라이언트-서버 응용은 UDP를 이용하여 통신하는 datagram socket을 사용



Socket 프로그래밍

– 응용 프로토콜

- **연결-지향 (Connection-oriented) 응용 프로토콜**
- 연결-지향 응용 프로토콜을 구현하는 클라이언트-서버 응용은 TCP 연결 상에서 통신하기 위하여 stream socket을 이용



Socket 프로그래밍

– Socket 시스템 호출 성명

- **socket 시스템 호출**
  - Socket 호출은 socket을 초기화하기 위해 사용
  - 네트워크 통신을 하기 위해 프로세스가 해야 할 것은 원하는 통신
  - 프로토콜의 형태를 명시하는 socket 시스템을 호출

- **bind 시스템 호출**
  - 이름이 없는 소켓에 이름을 부여함
  - 비연결 응용 프로토콜에서 bind를 사용

- **connect 시스템 호출**
  - 연결-지향 응용 프로토콜을 사용할 때 클라이언트 프로세스가 연결을 하고자 할 때 호출
  - Client-Server간의 연결을 하기 위해 필요한 정보교환을 하기 위해 3-way handshake 절차 동작

Client > Server : TCP SYN
Server > Client : TCP SYN ACK
Client > Server : TCP ACK

Socket 프로그래밍

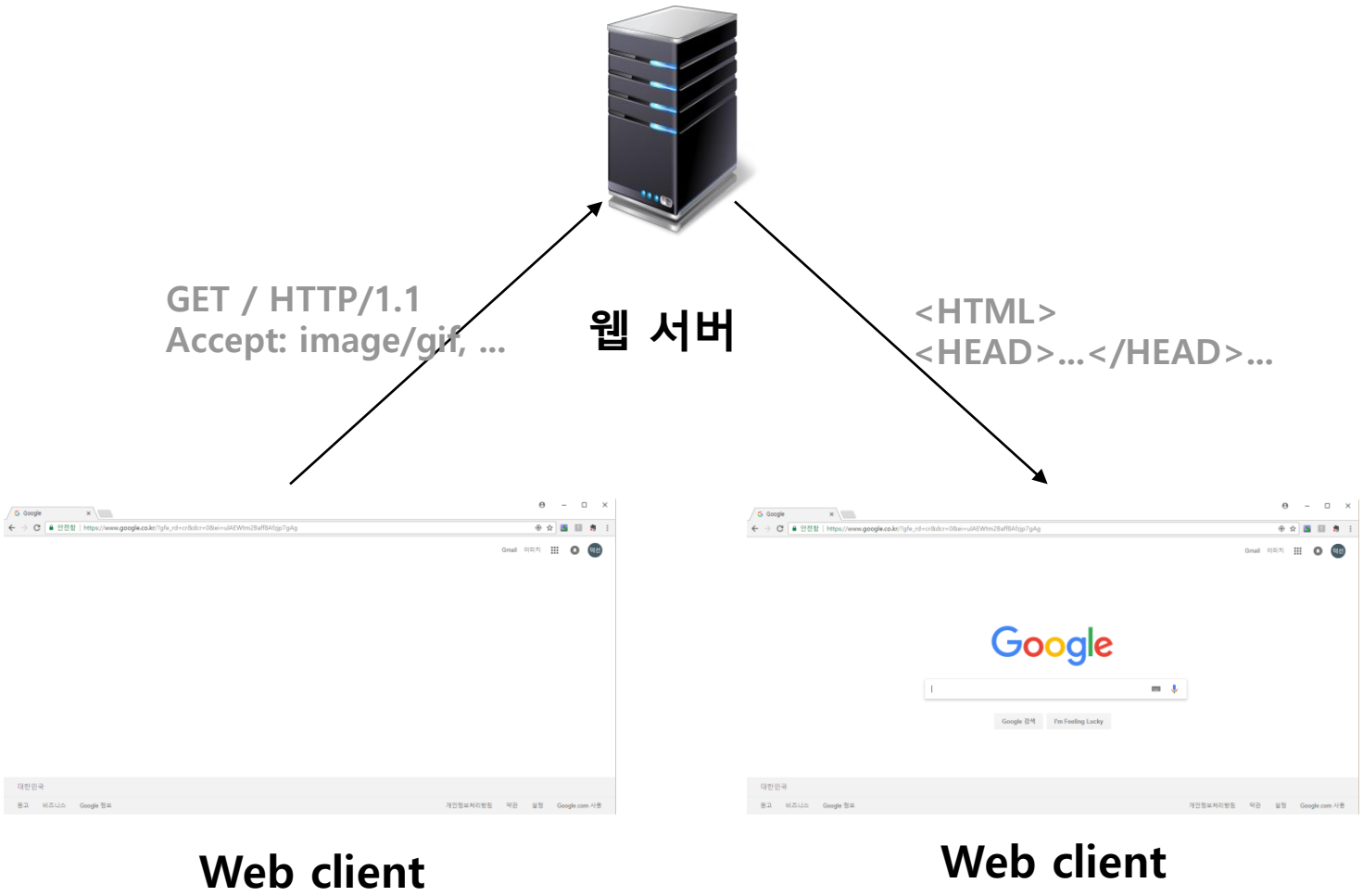
- **listen 시스템 호출**
  - TCP를 사용하는 연결-지향 응용 프로토콜에서의 서버 프로세스가 연결을 받아들이겠다는 것을 나타내기 위한 호출
- **accept 시스템 호출**
  - 연결-지향 응용 프로토콜에서 서버 프로세스가 listen 시스템 호출을 수행한 후 accept 시스템 호출을 수행함으로써 클라이언트 프로세스로부터 실제적 연결을 기다림
- **send, sendto, recv, recvfrom 시스템 호출**
  - 데이터를 보내거나(send, sendto) 받기 위해(recv, recvfrom) 사용 되는 시스템 호출
  - 표준 read, write 시스템 호출과 유사하지만 추가적으로 다른 parameter를 요구
- **close 시스템 호출**
  - 소켓을 닫기 위해 사용

Windows socket TCP server/client

Socket 프로그래밍

– TCP 서버/클라이언트 동작 원리 (1/6)

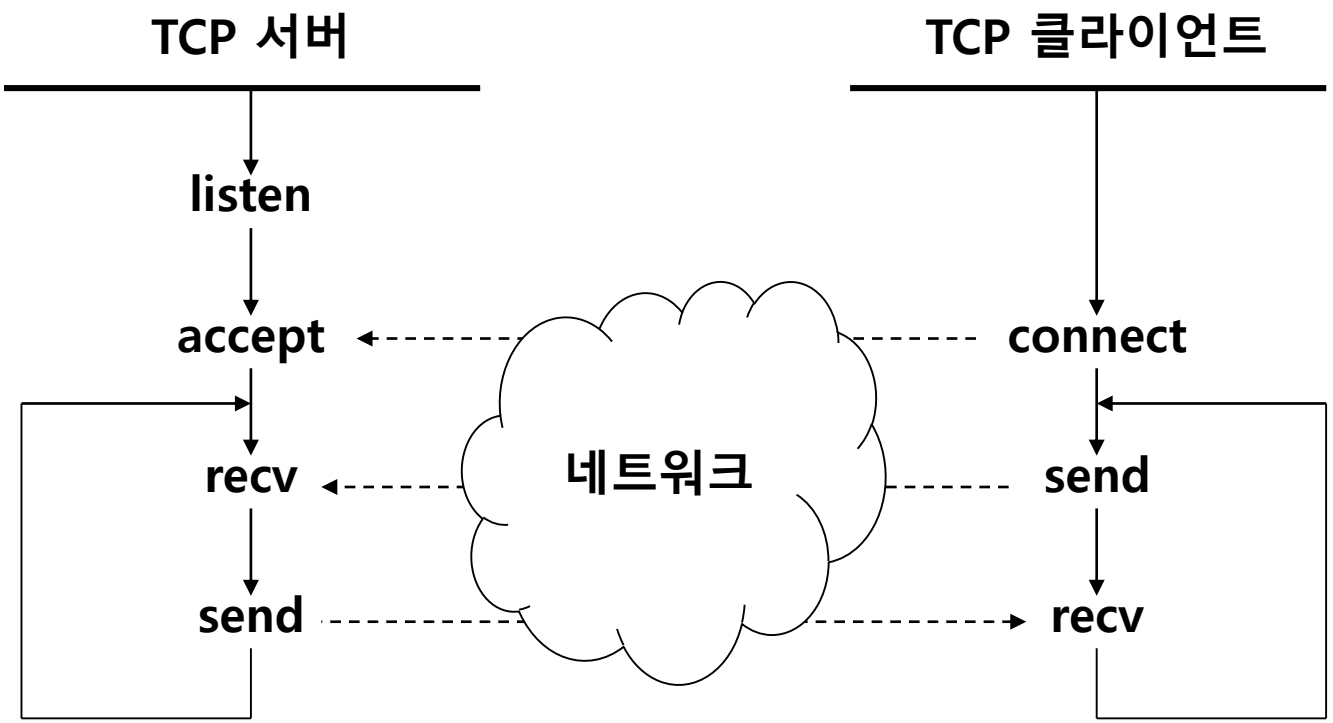
- TCP 서버/클라이언트 예



Socket 프로그래밍

– TCP 서버/클라이언트 동작 원리 (2/6)

- TCP server/client 동작 방식



– TCP 서버/클라이언트 동작 원리 (3/6)

- TCP server/client 동작 방식

① 서버는 먼저 실행하여 클라이언트가 접속하기를 기다림(listen)

② 클라이언트가 서버에게 접속(connect)하여 데이터를 보냄(send)

③ 서버는 클라이언트 접속을 수용하고(accept), 클라이언트가 보낸 데이터를 받아서(recv) 처리

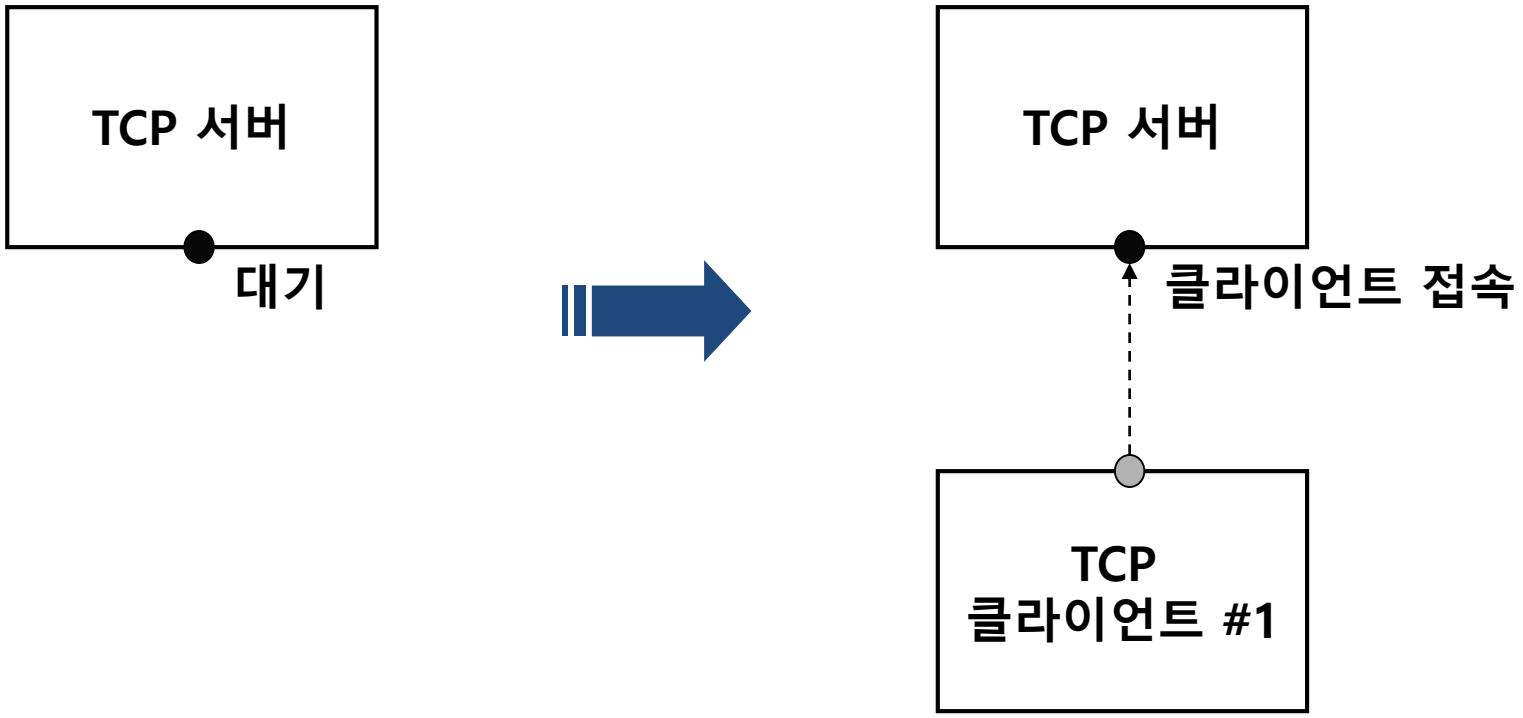
④ 서버는 처리한 데이터를 클라이언트에게 보냄(send)

⑤ 클라이언트는 서버가 보낸 데이터를 받아서(recv) 자신의 목적에 맞게 사용함

Socket 프로그래밍

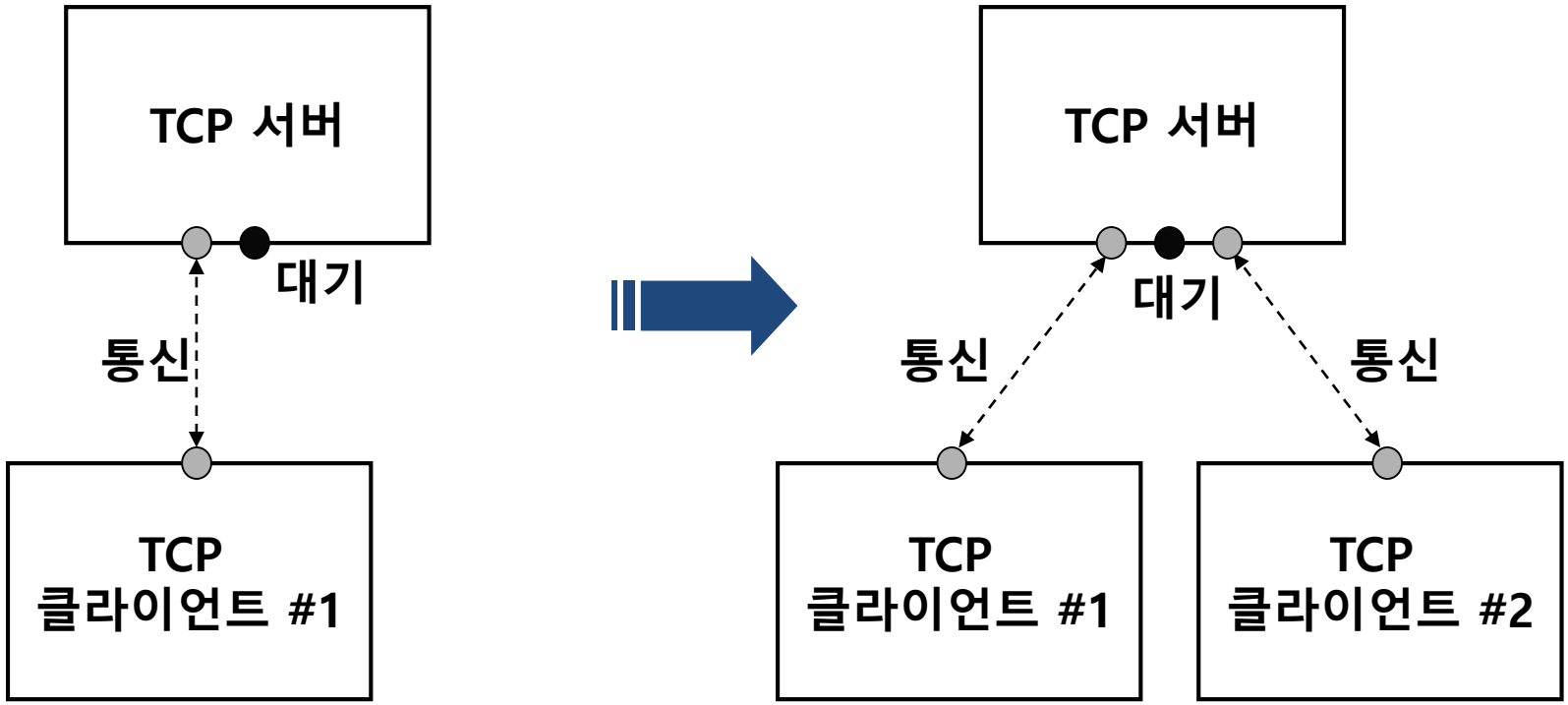
– TCP 서버/클라이언트 동작 원리 (4/6)

- TCP server/client 동작 방식



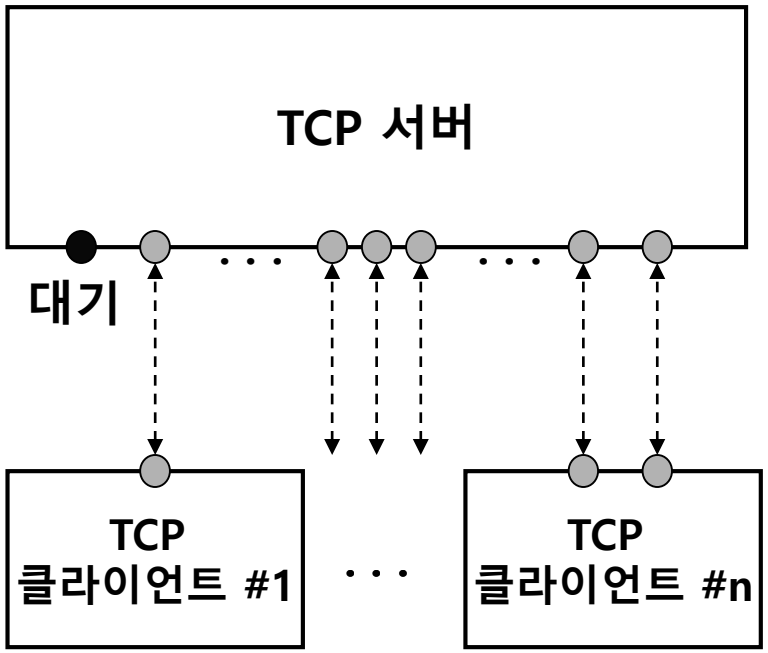
– TCP 서버/클라이언트 동작 원리 (5/6)

- TCP server/client 동작 원리



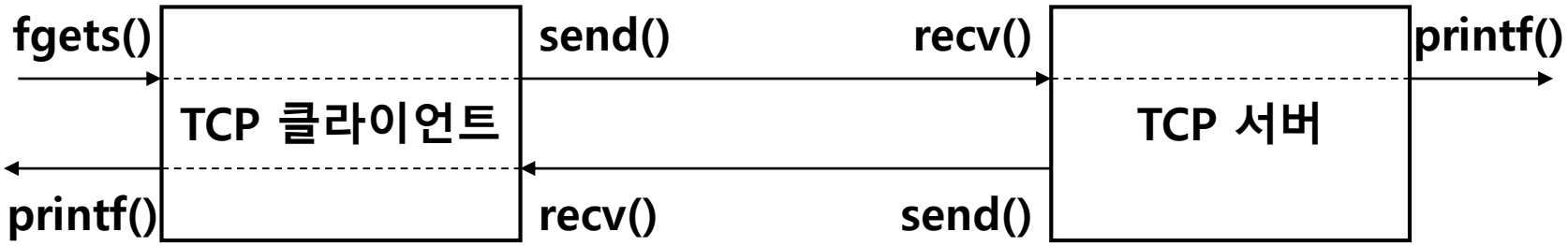
– TCP 서버/클라이언트 동작 원리 (6/6)

- TCP server/client 동작 원리



TCP 서버/클라이언트 예제

– 동작 방식



– TCP 서버/클라이언트 분석 (1/2)

- TCP/IP 소켓 통신을 위해 필요한 요소
  - Protocol

    ➔ 소켓을 생성할 때 결정
  - 지역(local) IP 주소와 지역 포트 번호

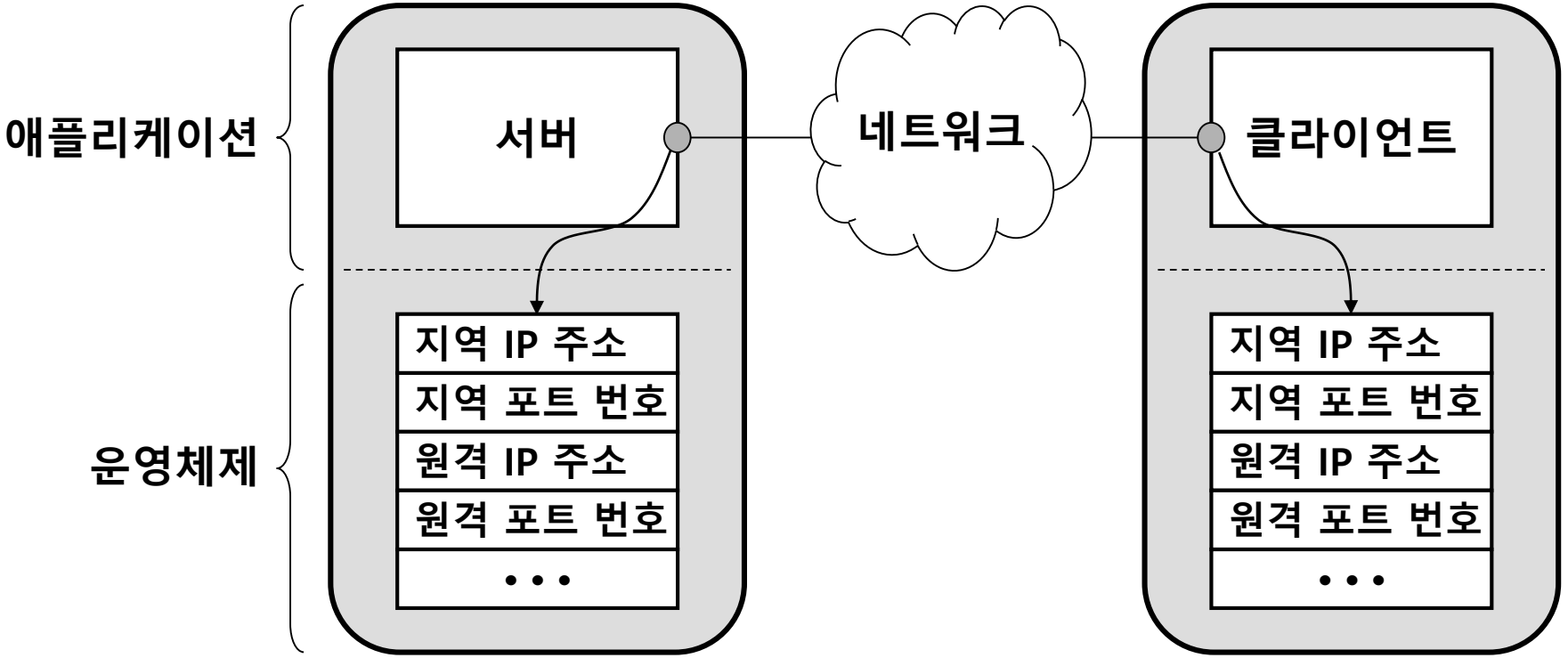
    ➔ 서버 또는 클라이언트 자신의 주소
  - 원격(remote) IP 주소와 원격 포트 번호

    ➔ 서버 또는 클라이언트가 통신하는 상대방의 주소

Socket 프로그래밍

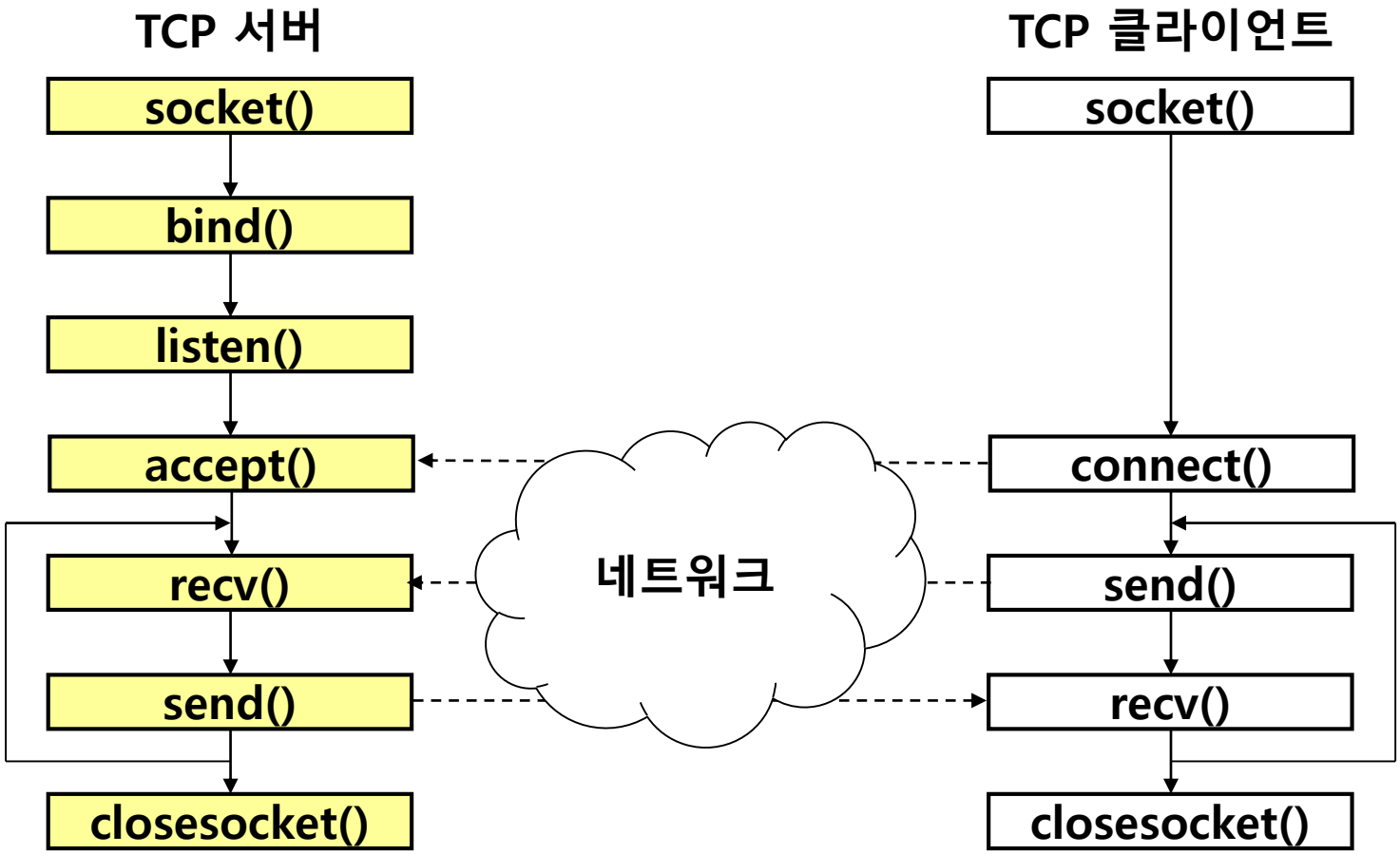
– TCP server/client 분석 (2/2)

- Socket data structure



Socket 프로그래밍

– TCP server function(1/8)



– **TCP server function(2/8)**

- bind() 함수
- 서버의 지역 IP 주소와 지역 포트 번호를 결정

```
int bind (
    SOCKET s,
    const struct sockaddr* name,
    int namelen
) ;
```

성공: 0, 실패: SOCKET_ERROR

– **TCP server function(3/8)**

- bind() 함수 사용 예

```
050     SOCKADDR_IN serveraddr;
051     ZeroMemory(&serveraddr, sizeof(serveraddr));
052     serveraddr.sin_family = AF_INET;
053     serveraddr.sin_port = htons(9000);
054     serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
055     retval = bind(listen_sock, (SOCKADDR *)&serveraddr,
            sizeof(serveraddr));
056     if(retval == SOCKET_ERROR) err_quit("bind()");
```

Socket 프로그래밍

– TCP server function(4/8)

- listen() 함수
- 소켓과 결합된 TCP 포트 상태를 LISTENING으로 변경

int listen (int sock, int backlog) ;
성공: 0, 실패: SOCKET_ERROR 반환

- sock 연결요청 대기상태에 두고자 하는 소켓의 파일 디스크립터 전달, 이 함수의 인자로 전달된 디스크립터의 소켓이 서버소켓(리스닝 소켓)이 됨
- backlog 연결요청 대기 큐(Queue)의 크기정보 전달, 5가 전달되면 큐의 크기가 5가 되어 클라이언트의 연결 요청을 5개까지 대기시킬 수 있음

연결요청도 일종의 데이터 전송임
따라서 연결요청을 받아들이기 위해서도 하나의 소켓이 필요함
그리고 이 소켓을 가리켜 서버소켓 또는 리스닝 소켓이라 함
listen 함수의 호출은 소켓을 리스닝 소켓이 되게 함

– **TCP server function(5/8)**

- listen() 함수 사용 예

```
059     retval = listen(listen_sock, SOMAXCONN);
060     if(retval == SOCKET_ERROR) err_quit("listen()");
```

– TCP server function(6/8)

- accept() 함수
  - 접속한 클라이언트와 통신할 수 있도록 새로운 소켓을 생성하여 리턴
  - 접속한 클라이언트의 IP 주소와 포트 번호를 알려줌

SOCKET accept (int sock, struct sockaddr* addr, socklen_t * addrlen) ;
성공: 새로운 소켓, 실패: INVALID_SOCKET

- sock 서버 소켓의 파일 디스크립터 전달
- addr 연결요청 한 클라이언트 주소정보를 담을 변수의 주소 값 전달, 함수호출이 완료되면 인자로 전달된 주소의 변수에는 클라이언트 주소정보가 채워짐
- addrlen 두번째 매개변수 addr에 전달된 주소의 변수 크기를 바이트 단위로 전달, 단 크기 정보를 변수에 저장한 다음에 변수의 주소 값을 전달함 그리고 함수호출이 완료되면 크기 정보로 채워져 있던 변수에는 클라이언트 주소정보 길이가 바이트 단위로 계산되어 채워짐

Socket 프로그래밍

– **TCP server function(7/8)**

• accept() 함수 사용 예

```
062     // 데이터 통신에 사용할 변수
063     SOCKET client_sock;
064     SOCKADDR_IN clientaddr;
065     int addrlen;
...
068     while(1){
069         // accept()
070         addrlen = sizeof(clientaddr);
071         client_sock = accept(listen_sock, (SOCKADDR *)&clientaddr, &addrlen);
072         if(client_sock == INVALID_SOCKET){
073             err_display("accept()");
074             continue;
075         }
```

연결요청 정보를 참조하여 클라이언트 소켓과의 통신을 위한 별도의 소켓을 추가로 하나 더 생성함
그리고 이렇게 생성된 소켓을 대상으로 데이터의 송수신이 진행됨 실제로 서버의 코드를 보면 실제로 소켓이 추가로 생성되는 것을 확인할 수 있음

Socket 프로그래밍

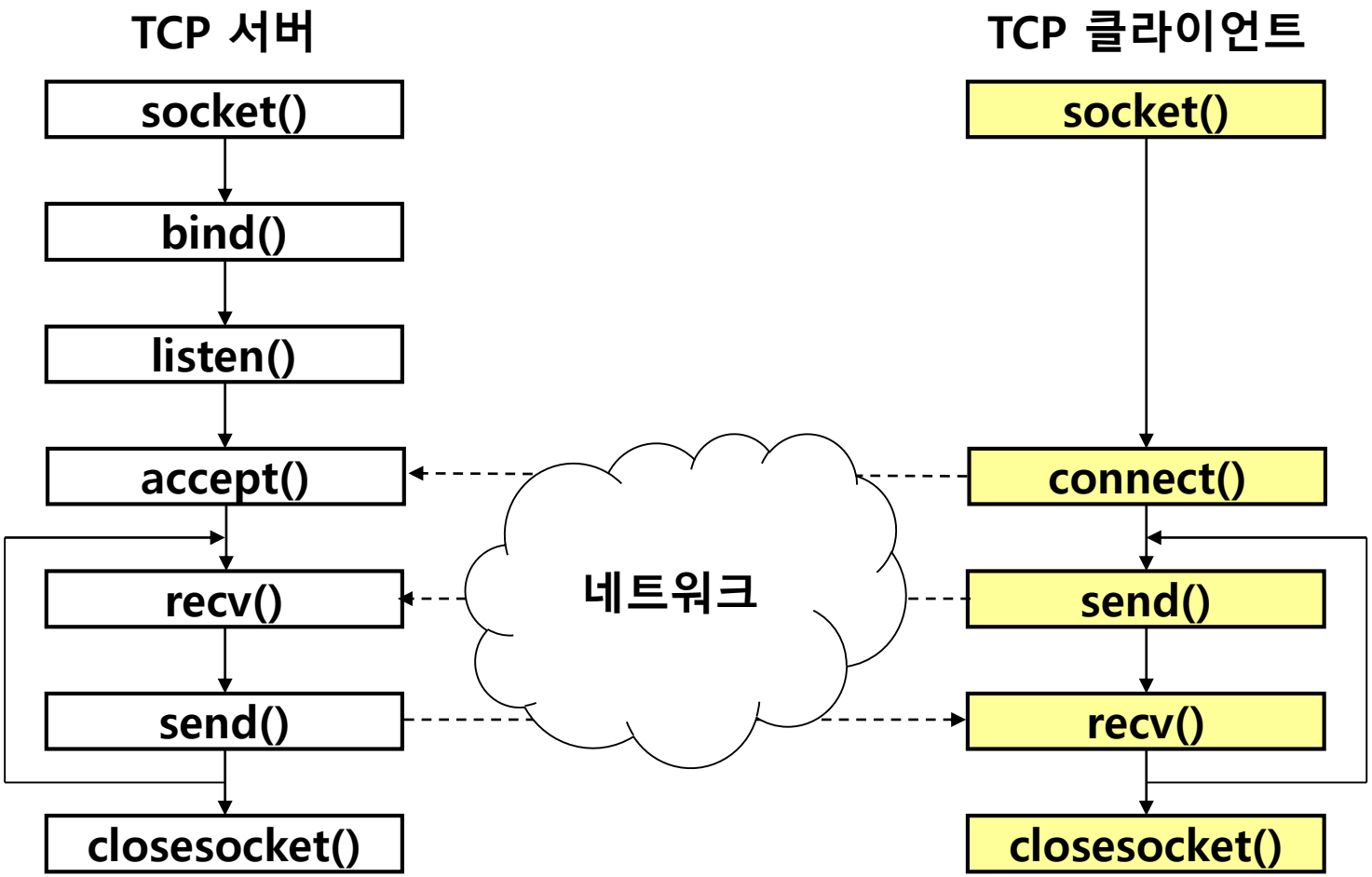
– **TCP server function(8/8)**

- accept() 함수 사용 예

```
076         printf("₩n[TCP 서버] 클라이언트 접속: IP 주소=%s, 포트 번호=%d₩n",
077             inet_ntoa(clientaddr.sin_addr), ntohs(clientaddr.sin_port));
078
079         // 클라이언트와 데이터 통신
080         while(1){
...
101         }
102
103         // closesocket()
104         closesocket(client_sock);
105         printf("[TCP 서버] 클라이언트 종료: IP 주소=%s, 포트 번호=%d₩n",
106             inet_ntoa(clientaddr.sin_addr), ntohs(clientaddr.sin_port));
107     }
```

Socket 프로그래밍

– TCP 클라이언트 함수(1/3)



– TCP 클라이언트 함수(2/3)

– connect() 함수

- 서버에게 접속하여 TCP 프로토콜 수준의 연결 설정

int connect (int sock, const struct sockaddr * servaddr, socklen_t addrlen) ;
성공: 0, 실패: SOCKET_ERROR

- sock 서버 소켓의 파일 디스크립터 전달
- addr 연결요청 한 클라이언트 주소정보를 담을 변수의 주소 값 전달, 함수호출이 완료되면 인자로 전달된 주소의 변수에는 클라이언트 주소정보가 채워짐
- addrlen 두 번째 매개변수 seraddr에 전달된 주소의 변수 크기를 바이트 단위로 전달, 단 크기 정보를 변수에 저장한 다음에 변수의 주소 값을 전달함 그리고 함수호출이 완료되면 크기 정보로 채워져 있던 변수에는 클라이언트 주소정보 길이가 바이트 단위로 계산되어 채워짐

– TCP 클라이언트 함수(3/3)

- connect() 함수 사용 예

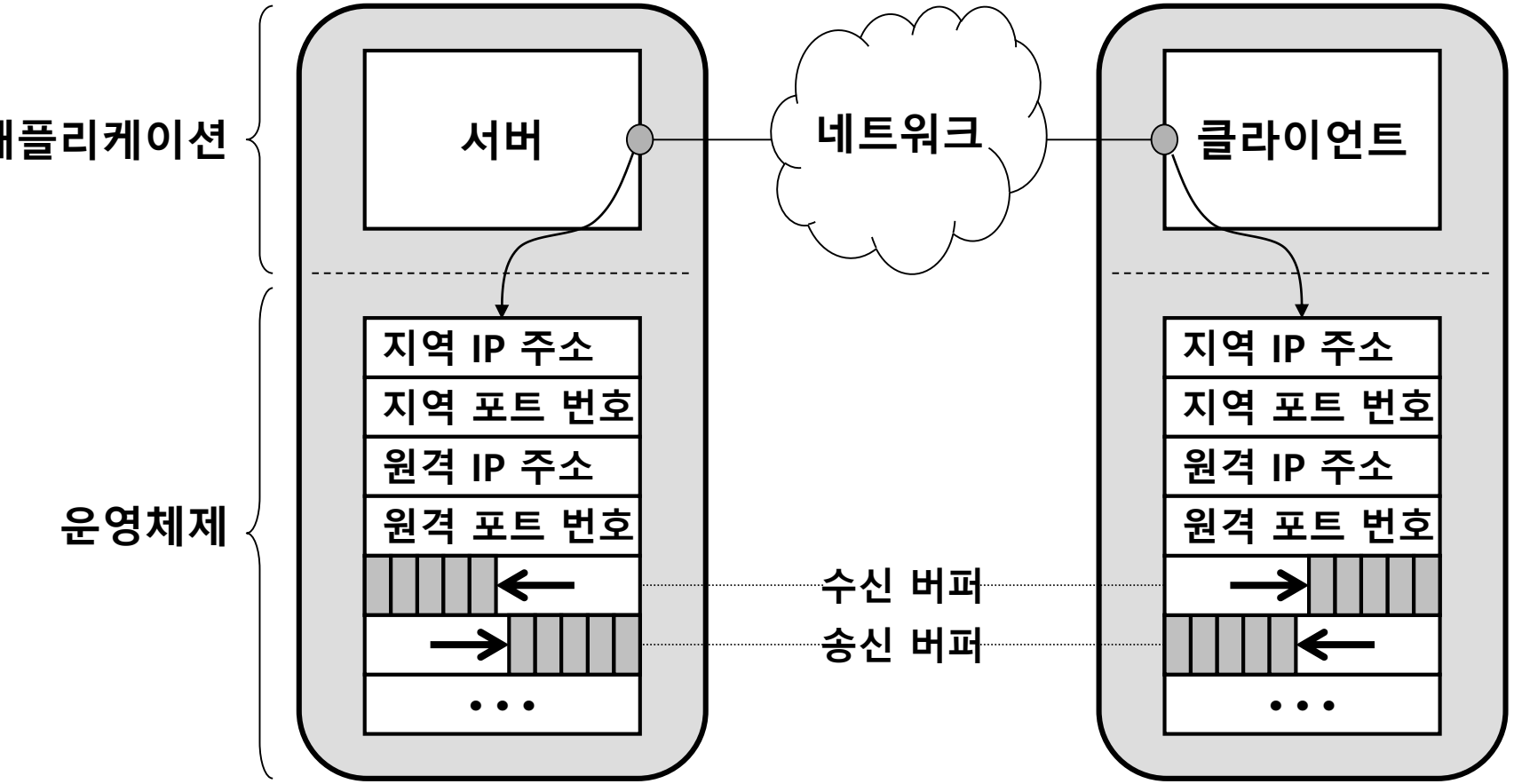
클라이언트의 경우 소켓을 생성하고, 이 소켓을대상으로 연결의 요청을 위해서 connect 함수를 호출하는 것이 전부임 그리고 connect 함수를 호출할때 연결할 서버의 주소정보도 함께 전달함

```
070     SOCKADDR_IN serveraddr;
071     serveraddr.sin_family = AF_INET;
072     serveraddr.sin_port = htons(9000);
073     serveraddr.sin_addr.s_addr = inet_addr("127.0.0.1");
074     retval = connect(sock, (SOCKADDR *)&serveraddr,
            sizeof(serveraddr));
075     if(retval == SOCKET_ERROR) err_quit("connect()");
```

Socket 프로그래밍

– 데이터 전송 함수(1/10)

- 소켓 데이터 구조체



– 데이터 전송 함수(2/10)

- send() 함수
  - 애플리케이션 데이터를 송신 버퍼에 복사함으로써 궁극적으로 하부 프로토콜 (예TCP/IP)에 의해 데이터가 전송되도록 함

```
int send (
    SOCKET s,
    const char* buf,
    int len,
    int flags
);
```

성공: 보낸 바이트 수, 실패: SOCKET_ERROR

❑ 데이터 전송 함수(3/10)

– recv() 함수

- 수신 버퍼에 도착한 데이터를 애플리케이션 버퍼로 복사

```
int recv (
    SOCKET s,
    char* buf,
    int len,
    int flags
);
```

성공: 받은 바이트 수 또는 0(연결 종료시), 실패: SOCKET_ERROR

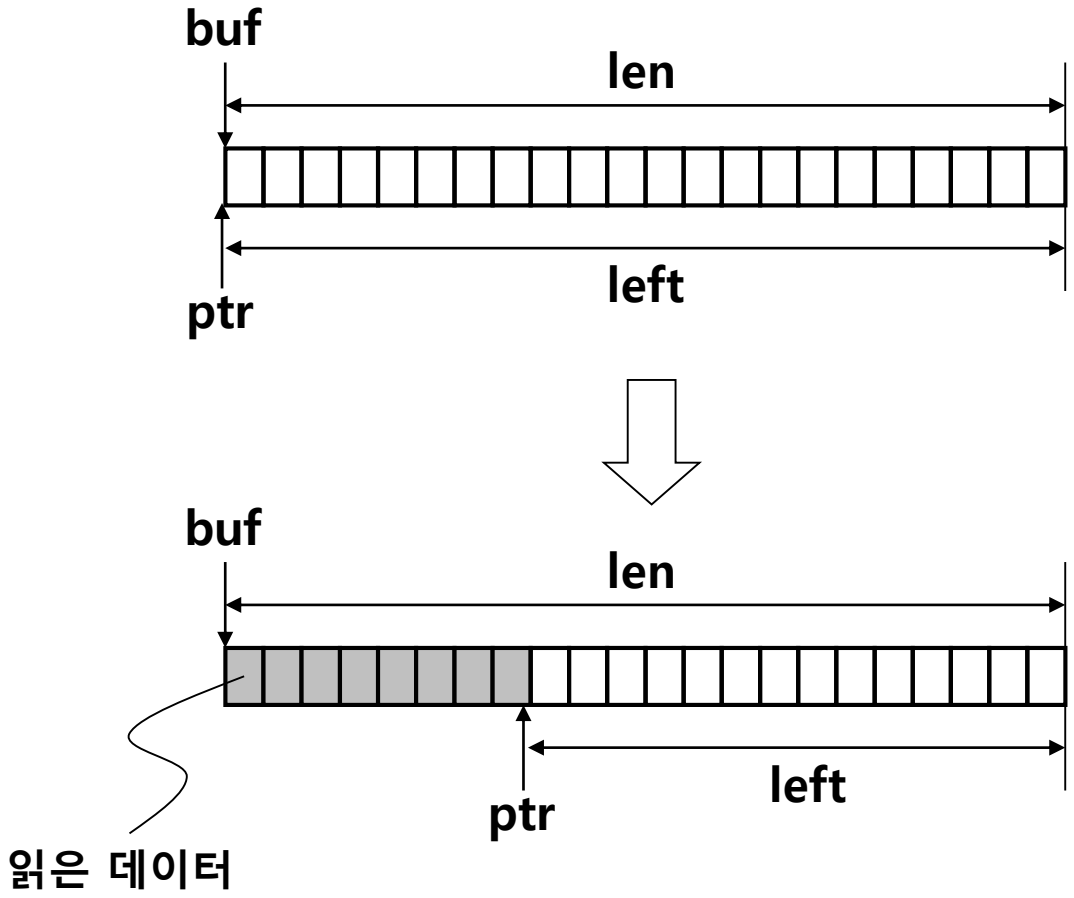
– 데이터 전송 함수(4/10)

- recvn() 함수

```
037  int recvn(SOCKET s, char *buf, int len, int flags)
038  {
039      int received;
040      char *ptr = buf;
041      int left = len;
042
043      while(left > 0){
044          received = recv(s, ptr, left, flags);
045          if(received == SOCKET_ERROR)
046              return SOCKET_ERROR;
047          else if(received == 0)
048              break;
049          left -= received;
050          ptr += received;
051      }
052
053      return (len - left);
054  }
```

– 데이터 전송 함수(5/10)

- recvn() 함수 동작 원리



– 데이터 전송 함수(6/10)

- 데이터 전송 함수 사용 예 – TCP client

```
078     char buf[BUFSIZE+1];
079     int len;
...
082     while(1){
083         // 데이터 입력
084         ZeroMemory(buf, sizeof(buf));
085         printf("₩n[보낼 데이터] ");
086         if(fgets(buf, BUFSIZE+1, stdin) == NULL)
087             break;
088
089         // '₩n' 문자 제거
090         len = strlen(buf);
091         if(buf[len-1] == '₩n')
092             buf[len-1] = '₩0';
093         if(strlen(buf) == 0)
094             break;
```

– 데이터 전송 함수(7/10)

- 데이터 전송 함수 사용 예 – TCP client

```
096         // 데이터 보내기
097         retval = send(sock, buf, strlen(buf), 0);
098         if(retval == SOCKET_ERROR){
099             err_display("send()");
100             break;
101         }
102         printf("[TCP 클라이언트] %d바이트를 보냈습니다.₩n", retval);
103
104         // 데이터 받기
105         retval = recvn(sock, buf, retval, 0);
106         if(retval == SOCKET_ERROR){
107             err_display("recv()");
108             break;
109         }
110         else if(retval == 0)
111             break;
```

– 데이터 전송 함수(8/10)

- 데이터 전송 함수 사용 예 – TCP client

```
113         // 받은 데이터 출력
114         buf[retval] = '₩0';
115         printf("[TCP 클라이언트] %d바이트를 받았습니다.₩n", retval);
116         printf("[받은 데이터] %s₩n", buf);
117     }
```

– 데이터 전송 함수(9/10)

- 데이터 전송 함수 사용 예 – TCP server

```
066     char buf[BUFSIZE+1];
...
080         while(1){
081             // 데이터 받기
082             retval = recv(client_sock, buf, BUFSIZE, 0);
083             if(retval == SOCKET_ERROR){
084                 err_display("recv()");
085                 break;
086             }
087             else if(retval == 0)
088                 break;
089
090             // 받은 데이터 출력
091             buf[retval] = '₩0';
092             printf("[TCP/%s:%d] %s₩n", inet_ntoa(clientaddr.sin_addr),
093                 ntohs(clientaddr.sin_port), buf);
```

Socket 프로그래밍

– 데이터 전송 함수(10/10)

- 데이터 전송 함수 사용 예 – TCP server

```
095             // 데이터 보내기
096             retval = send(client_sock, buf, retval, 0);
097             if(retval == SOCKET_ERROR){
098                 err_display("send()");
099                 break;
100             }
101         }
```

❑ 파일 전송 서버

```
...
    int serv_sd;
    int clnt_sd;
    int fd;
    char buf[BUFSIZE];

    struct sockaddr_in serv_addr;
    struct sockaddr_in clnt_addr;
    int clnt_addr_size;
    int len;

    fd = open("file_server.c", O_RDONLY); //파일오픈

   serv_sd = socket(PF_INET, SOCK_STREAM, 0);

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(atoi(argv[1]));

    bind(serv_sd, (struct sockaddr *)&serv_addr,
                sizeof(serv_addr));
    listen(serv_sd, 5);

    clnt_addr_size = sizeof(clnt_addr);
    clnt_sd = accept(serv_sd,
                (struct sockaddr *)&clnt_addr,
                 &clnt_addr_size);

    // 파일의 내용을 클라이언트에게 전송
    while ((len = read(fd, buf, BUFSIZE)) != 0) {
        write(clnt_sd, buf, len);
    }

    // 파일전송을 마치고 출력스트림을 닫는다.
    if (shutdown(clnt_sd, SHUT_WR) == -1) {
        error_handling("shutdown() error");
    }

    // 클라이언트로부터 “Thank you”메시지 수신
    len = read(clnt_sd, buf, BUFSIZE);
    write(1, buf, len); // 표준 출력(모니터)으로 출력

    close(fd);
    close(clnt_sd);
...
```

❑ 파일 수신 클라이언트

```
...
    int fd;
    int sd;
    char buf[BUFSIZE];
    struct sockaddr_in serv_addr;
    int len;

    // 서버로부터 수신한 파일의 내용을 저장할 파일
    fd = open("receive.dat",
              O_WRONLY | O_CREAT | O_TRUNC);
    sd = socket(PF_INET, SOCK_STREAM, 0);

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
    serv_addr.sin_port = htons(atoi(argv[2]));

    connect(sd, (struct sockaddr *)&serv_addr,
                sizeof(serv_addr));

    while ((len = read(sd, buf, BUFSIZE)) != 0) {
        write(fd, buf, len);
    }

    // 파일수신이 끝나면 서버에 “Thank you”로 응답
    write(sd, "Thank you₩n", 10);

    close(fd);
    close(sd);
...
```

❑ 윈도우 기반의 파일 전송 서버

```
…
// 클라이언트에게 보내줄 파일 오픈
fp = fopen("file_server_win.c", "r");
if (fp == NULL)
    ErrorHandling("File open error!");

hServSock = socket(PF_INET, SOCK_STREAM, 0);
…
hClntSock = accept(hServSock,
                (SOCKADDR *)&clntAddr,
                &clntAddrSize);
…
// 클라이언트가 연결하면 파일의 내용을 읽어서 전송
while (1) {
    len = fread(buf, sizeof(char), BUFSIZE, fp);

    send(hClntSock, buf, len, 0);
    if (feof(fp))
        break;
}
```

```
…
// 파일 전송이 끝나면 소켓의 출력스트림을 닫음
if (shutdown(hClntSock, SD_SEND) ==
                SOCKET_ERROR)
    ErrorHandling("shutdown() error!");

//클라이언트로부터 "Thank you" 메시지 수신 및 출력
len = recv(hClntSock, buf, BUFSIZE-1, 0);
buf[len] = 0;
fputs(buf, stdout);

fclose(fp);
closesocket(hClntSock);
WSACleanup();
…
```

❑ 윈도우 기반의 파일 전송 서버

```
…
// 서버로부터 수신한 데이터를 저장할 파일
fp = fopen("receive.dat", "w");
if (fp == NULL)
    ErrorHandling("File open error!");

hSocket = socket(PF_INET, SOCK_STREAM, 0);

…

// 서버에 연결
if (connect(hSocket, (struct sockaddr *)&servAddr,
                sizeof(servAddr)) == SOCKET_ERROR)
    ErrorHandling("connect() error!");

…
```

```
…
// 서버로부터 데이터를 수신하여 파일에 저장
while((len = recv(hSocket, buf, BUFSIZE, 0)) != 0) {
    fwrite(buf, sizeof(char), len, fp);
}

// 파일 수신이 끝나면 "Thank you" 메시지 전송
send(hSocket, "Thank you₩n", 10, 0);

fclose(fp);
closesocket(hSocket);
WSACleanup();
…
```

❑ 실행결과

```
C:₩socket₩Debug>file_server.exe 50000
Thank you

C:₩socket₩Debug>
```

```
C:₩socket₩Debug>file_client.exe 127.0.0.1 50000

C:₩socket₩Debug>_
```