데이터통신과 네트워킹

Data Communication & Networking Ch. 14

동아대학교 컴퓨터AI 공학부
S03 301-01호실 (이양민 교수 연구실)
2024.12 (Kakao ID: yanwenry)

CHAPTER

14

TCP와 소켓 프로그래밍

Section

SECTION 01

전송 계층 관련 프로토콜 분석

1. TCP 헤더 분석

- 전송계층이 해야 할 일.
  - 다양한 응용 프로그램들이 인터넷을 사용할 수 있는 멀티 인터페이스를 제공.
  - 전송된 데이터의 무결성을 보장.
  - 네트워크의 상태는 매우 유동적이기 때문에 이에 맞추어 혼잡 제어.
  - 연결을 설정하는 문제와 해제하는 문제를 담당.

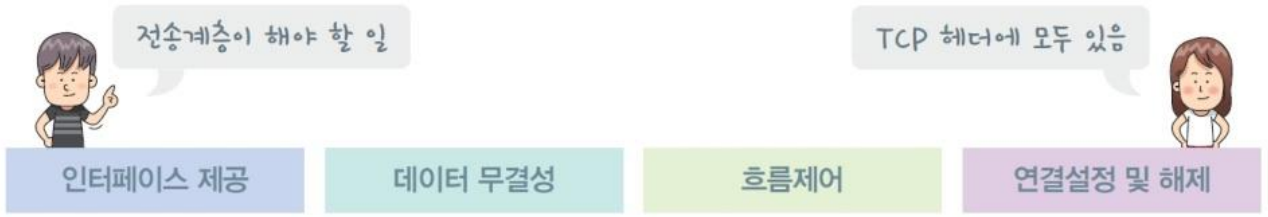


그림 14-1 전송 계층의 주요 작업

SECTION 01

전송 계층 관련 프로토콜 분석

- 전송계층이 해야 하는 대표적인 작업을 담당하는 프로토콜이 TCPTransmission Control Protocol
- 응용 프로그램이 소켓을 통해 인터넷으로 전달할 데이터를 보내면 TCP는 TCP 헤더를 붙여서 IP로 내려 보냄.
- IP는 IP 헤더를 붙여 이더넷으로 내려 보내면 이더넷이 데이터를 전송.

그림 14-2 인터넷 전송과 헤더들

전송 계층 관련 프로토콜 분석

- TCP 헤더는 IP헤더와 마찬가지로 32비트(4바이트)를 기준으로 나눔 -> TCP 헤더의 필수 부분은 5 x 4바이트 = 20바이트, 옵션은 없을 수 있으며, 있더라도 4바이트씩 증가.

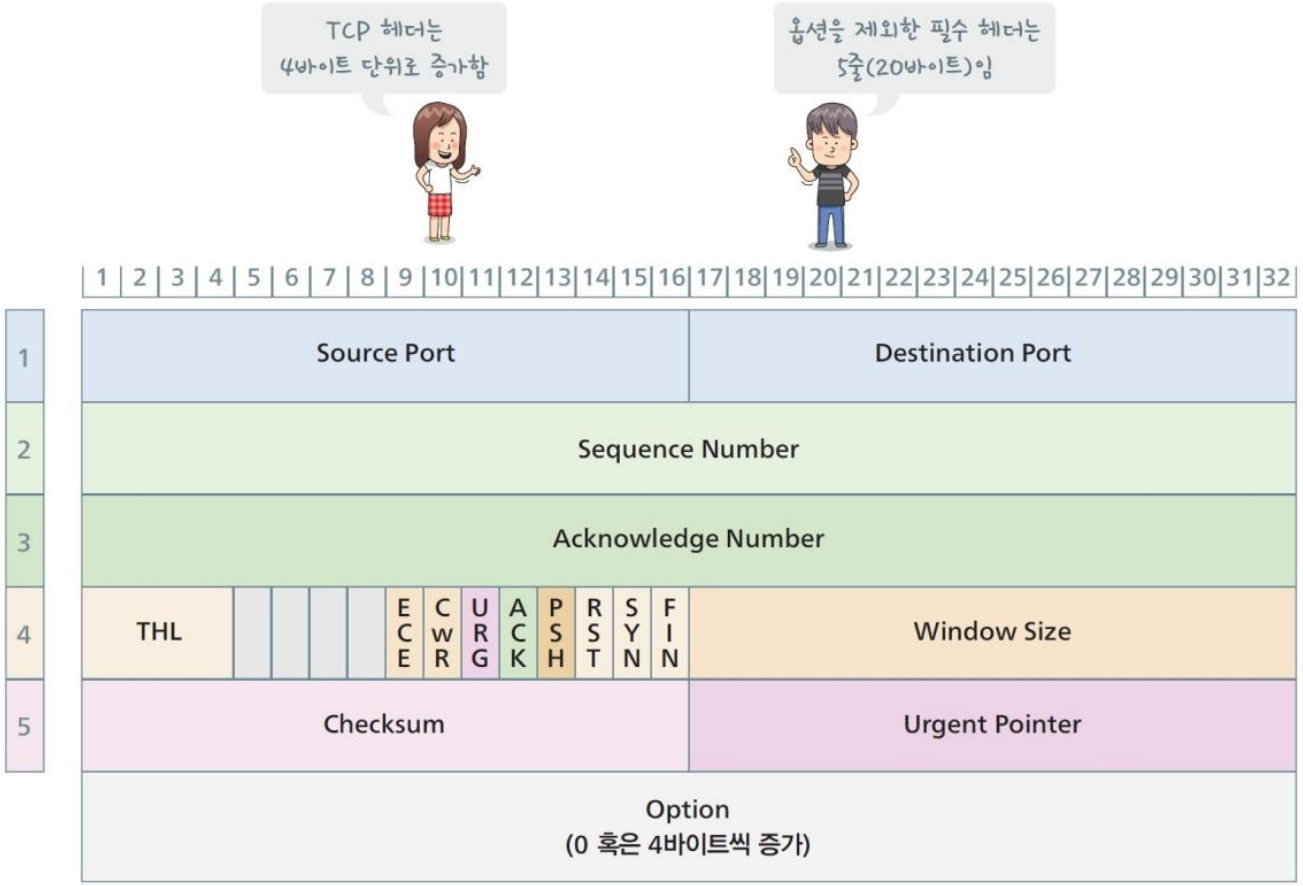
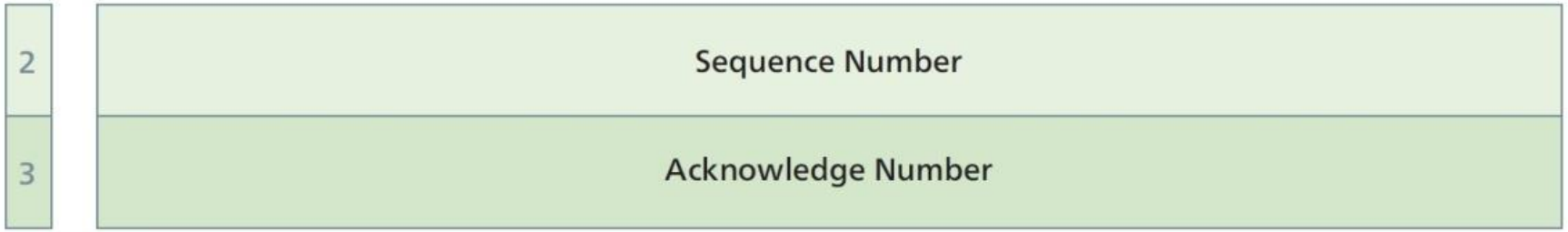


그림 14-3 TCP 헤더

전송 계층 관련 프로토콜 분석

- TCP를 사용하는 모든 응용 프로그램들이 포트를 사용하여 구분되기 때문에 보내는 쪽 포트(Source Port)와 받는 쪽 포트(Destination Port) 둘 다 표시하여 보냄.
- TCP의 구조상, 받는 쪽 포트를 알아야지만 데이터를 보낼 수 있음.
  - 받는 쪽의 응용 프로그램이 웹 데몬(HTTPD)와 같이 잘 알려진 소프트웨어라면 0에서 1023번까지의 well-known 포트 번호를 가지고 있음.
- 포트번호의 길이는 16비트이다. 따라서 0에서 65535까지의 포트번호를 가짐.
- 일반적인 응용 프로그램이라면 1024번 이상 65535이하의 임의의 값을 포트번호로 할당 받음.

전송 계층 관련 프로토콜 분석



- 네트워크에서 데이터를 전송하는 과정에서 순서가 뒤바뀌거나 사라지는 문제가 발생.
- 데이터 전송에는 2번줄 Sequence Number(일련번호)와 3번줄 Acknowledge Number(ACK 번호)를 사용.
- 일련번호와 ACK 번호는 연결을 설정하고 해제하는데도 사용.
- 컴퓨터가 켜진 후 꺼질 때 까지 무수히 많은 수의 데이터를 전송하기 때문에 일련번호와 ACK 번호는 충분히 커야 함.
- 일련번호와 ACK 번호에 32비트를 할당하였기 때문에 0부터 2^{32} - 1까지, 약 42억개의 번호를 사용 할 수 있음.

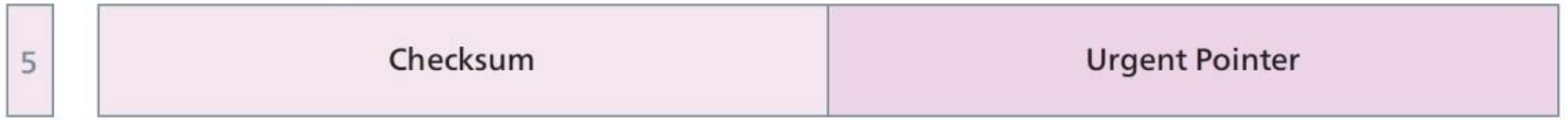
전송 계층 관련 프로토콜 분석

- THL: THL은 TCP Header Length의 약자. 옵션을 포함한 TCP 헤더 길이를 나타냄. 데이터는 포함되지 않음. THL 필드에 들어 있는 값은 4 바이트 단위의 줄의 길이를 의미. 따라서 옵션이 없는 경우에는 THL에는 5가 들어 있음.
- ECE : IP 헤더에서 ECN[Explicit Congestion Notification] 필드와 연동되는 필드가 ECE[Explicit Congestion notification Echo]. ECE는 ECN-Echo로 부름. IP 헤더에서 ECN은 혼잡제어에 사용되며, 네트워크에 혼잡이 발생하여 패킷이 제대로 전달되지 않을 경우 활성화 됨(ECE가 1). ECE를 ECN-Echo라 부르는 이유는 IP의 ECN이 네트워크가 혼잡하여 윈도우 크기를 줄인다고 결정하면, TCP의 ECE가 1로 바뀌기 때문. 윈도우 크기를 줄여 혼잡을 완화하는 작업은 TCP에서 이루어지지만 라우터의 혼잡한 상태를 확인 할 수 있는 것은 네트워크 계층.
- CWR : CWR은 Congestion Window Reduced의 약자로 ECE를 받아 윈도우 크기를 줄였다는 것을 확인하는 필드. 또한 ECE를 중복적으로 보내지 않아도 된다는 의미.

전송 계층 관련 프로토콜 분석

- URG : URG는 Urgent의 약자로 긴급하게 처리해야 할 데이터가 있는 경우 1이 됨.
- ACK : TCP 헤더의 ACK 번호가 의미가 있는 경우 ACK가 1이 됨. ACK가 0이면 ACK 번호는 무시.
- PSH : PSH는 push를 의미. PSH 필드가 1이 되면, 순서가 맞지 않아도 버퍼에 있는 데이터를 응용 계층으로 무조건 올려 보냄.
- SYN : SYN는 Synchronize를 의미하며 연결 설정에 사용되는 Connection Request(CR)을 의미. 연결을 설정할 때 SYN 필드가 1이 됨.
- FIN : FIN은 Finalize를 의미하며 연결 해제에 사용되는 Disconnection Request(DR)을 의미. 연결을 해제 할 때 FIN 필드가 1이 됨.
- RST : RST는 Reset을 의미한다. 연결 설정이나 연결 해제가 원만히 이루어지지 않을 경우 RST는 필드를 1로 만들면 리셋하자는 의미.
- Window Size: 슬라이딩 윈도우 프로토콜의 윈도우 크기를 나타냄. 윈도우의 크기 필드가 16비트이기 때문에 윈도우의 최대 크기(ACK 없이 보낼 수 있는 데이터의 개수)는 65536개.

전송 계층 관련 프로토콜 분석



- Checksum: 에러 검사 코드인 체크썸Checksum이 들어 있으며 크기는 16비트. TCP의 체크썸은 헤더와 데이터 모두 검사.
- Urgent Pointer: Urgent Pointer는 4번 줄 URG(Urgent)가 1이 되면 의미가 생기는 필드. 통신을 하던 중 긴급하게 처리해야 하는 데이터가 있다면, URG가 1이 됨. URG가 1이 된 경우, 데이터 중 긴급하게 처리 되어야 하는 위치를 Urgent Pointer가 가지고 있음.

전송 계층 관련 프로토콜 분석

2. UDP

- 화상회의나 유튜브와 같이 TCP의 작업은 부담스러운 경우를 위해 만든 프로토콜이 UDP, 영어로 User Datagram Protocol.
- 안정성보다는 속도를 요구하는 네트워크 통신에는 UDP/IP를 사용.
- UDP 헤더.
  - 1번 줄은 TCP 헤더와 마찬가지로 보내는 쪽 포트(Source Port)와 받는 쪽 포트(Destination Port).
  - 2번줄 UDP Length는 데이터를 포함한 UPD의 전체 길이를 나타냄. 표시 단위는 바이트.
  - 헤더를 포함한 UDP 전체에 대하여 에러검출 코드로 체크썸을 사용.

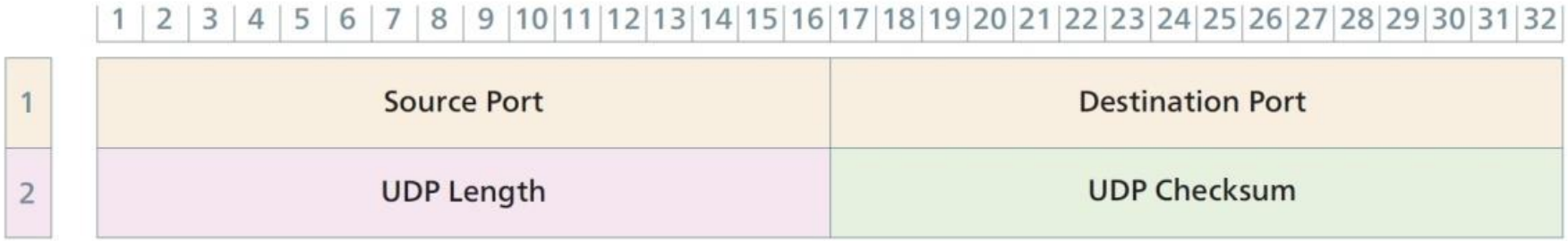


그림 14-4 UDP 헤더

전송 계층 관련 프로토콜 분석

3. RTP

- RTP는 스트리밍 데이터의 전송을 위해 설계된 실시간 전송 프로토콜Real-time Transport Protocol
- 주로 인터넷 전화(VoIPVoice over IP), 동영상 스트리밍, 화상통신과 같은 멀티미디어 통신에 사용
- TCP나 UDP는 운영체제가 인터넷을 사용하는 응용 프로그램에게 제공하는 통신 서비스
- RTP는 UDP를 사용하여 빠르게 데이터를 전송하도록 설계된 것으로 응용 프로그램이 자체적으로 사용
- RTP의 또 다른 특징은 다양한 비디오 및 오디오 포맷을 지원하기 위하여 헤더에서 기능을 추가하거나 뺄 수 있도록 설계.

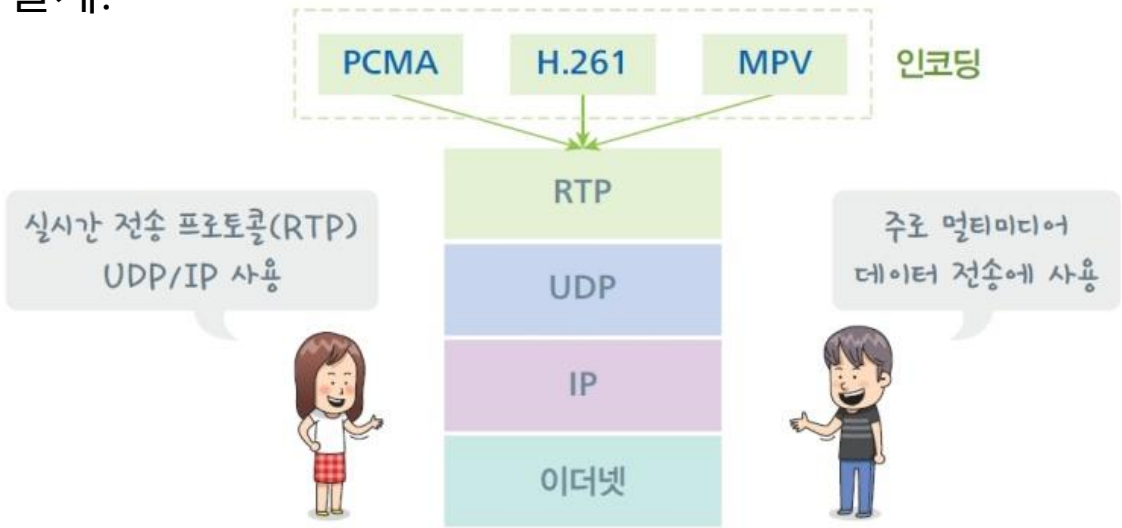


그림 14-5 실시간 전송 프로토콜(RTP) 구조

전송 계층 관련 프로토콜 분석

- RTP를 사용하는 경우, RTP 헤더 - UDP 헤더 - IP 헤더 - 이더넷 헤더들이 차례로 붙음.
- RTP는 지터 보상, 패킷 손실, 느린 전송 감지 기능들을 제공.
- RTP는 일 대 일 통신 뿐 아니라 멀티캐스트를 통해 여러 곳에 데이터를 전송할 수 있게 해 줌.

그림 14-6 실시간 전송 프로토콜(RTP) 구조

전송 계층 관련 프로토콜 분석

- 옵션을 제외한 RTP 헤더의 기본 크기는 3줄(12바이트)이다. 헤더의 옵션은 4바이트 단위로 증가.
- ver (version): RTP 버전 번호이며 현재는 2.
- P (padding): RTP 마지막에 패딩 데이터가 존재하는 경우 1이 되는 필드.
- x (extention): RTP 헤더의 마지막에 RTP Header extention이 존재하는 경우 1이 됨.

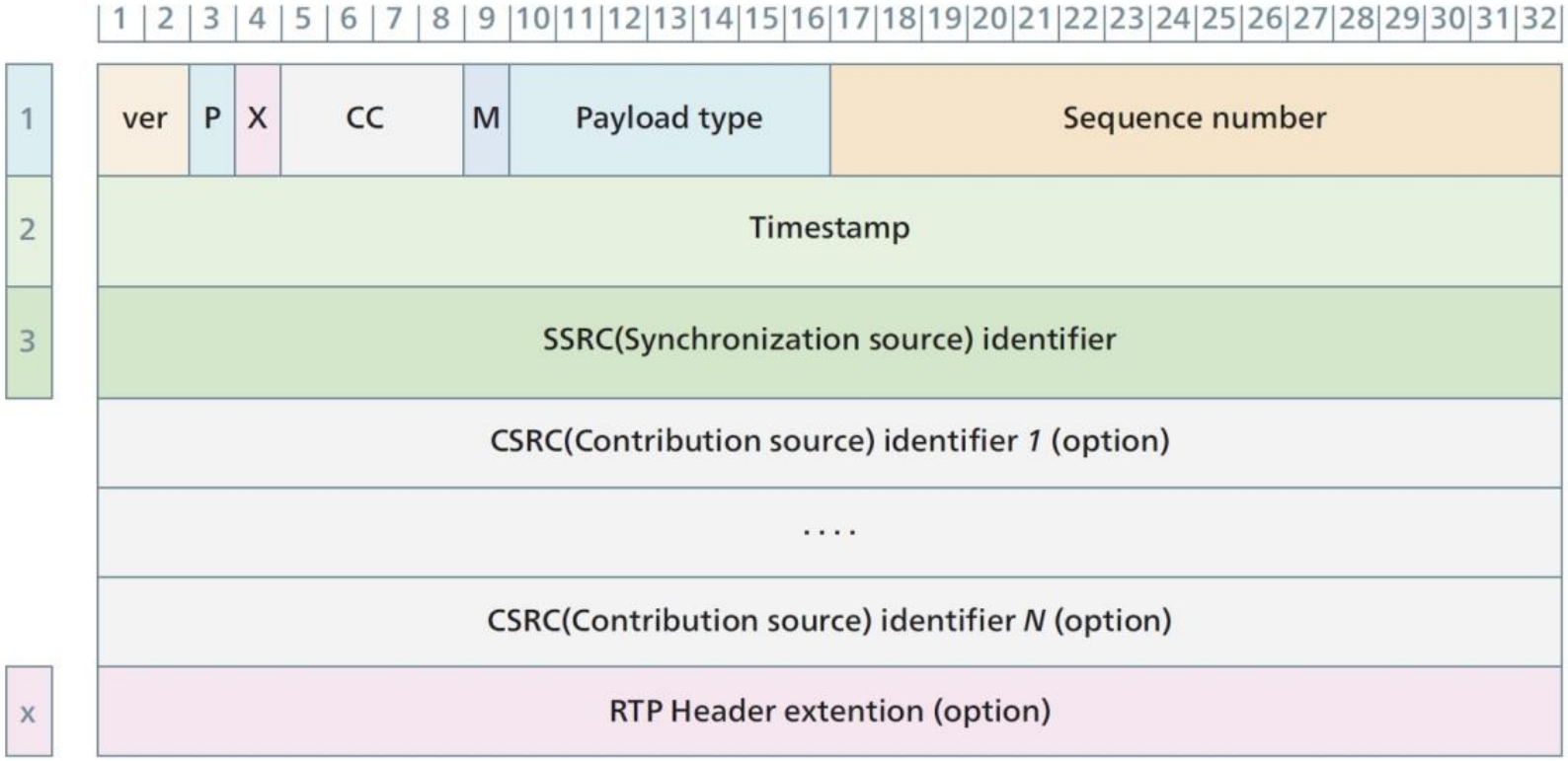


그림 14-7 실시간 전송 프로토콜(RTP) 헤더

전송 계층 관련 프로토콜 분석

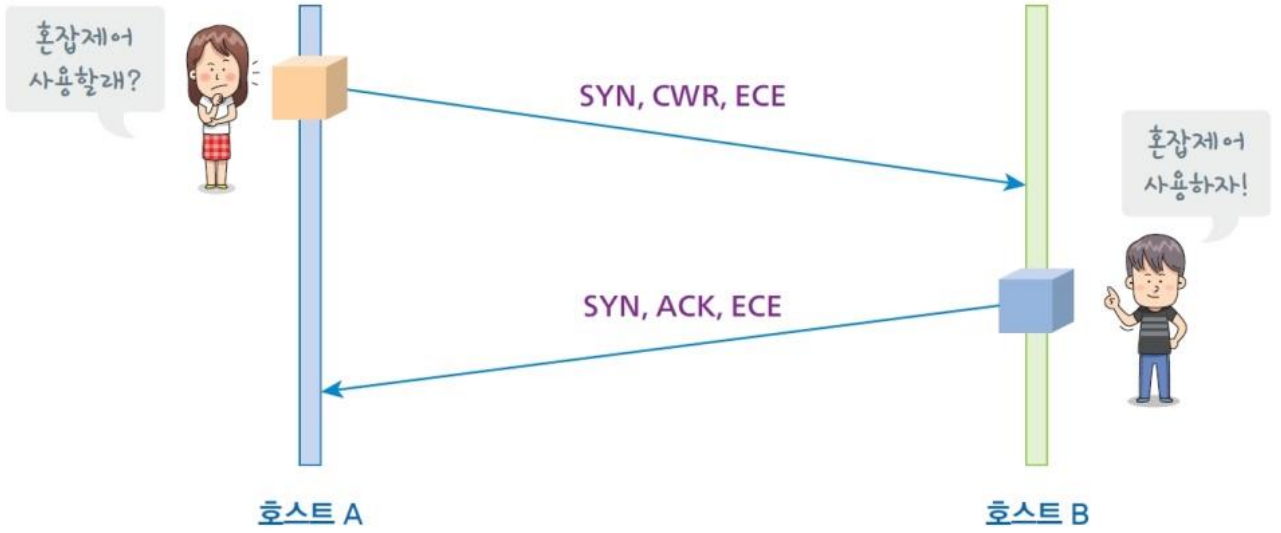
- CC (CSRC Count): RTP에서 발신자를 구분하는 고유번호는 3번 줄의 SSRC(Synchonization source)임. 발신자가 여러 개인 경우 CSRC(Contribution source)가 붙음. CSRC는 1개에서 N개까지 사용할 수 있는데 CC는 CSRC의 개수를 나타냄.
- M (marker): 마커는 응용프로그램이 필요할 때 사용할 수 있는 필드.
- payload type: 헤더 다음에 이어지는 데이터의 종류를 나타낸다. 사진, 음악, 동영상과 관련된 각종 인코딩 데이터가 표시 됨.
- Sequence number: RTP는 UDP를 기반으로 만들어졌기 때문에 패킷이 손실될 수 있음. Sequence number는 패킷의 손실이나 순서가 뒤바뀌는 경우를 확인하기 위해서 만들어진 필드.
- Timestamp: 데이터의 생성시간을 나타냄.

- RTP는 RTCP(RTP Control Protocol)와 결합하여 사용.

전송 계층 관련 프로토콜 분석

4. 혼잡제어

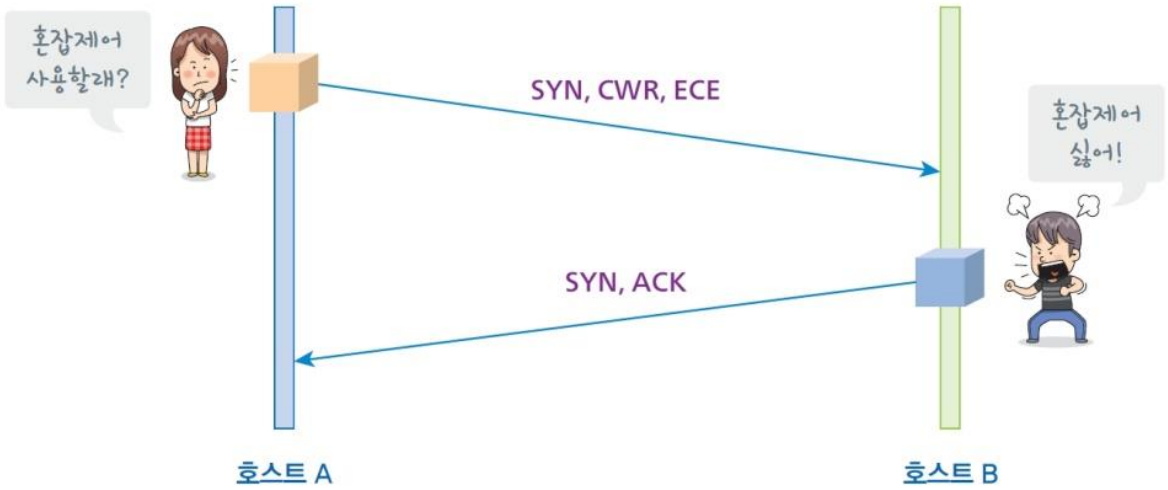
- 혼잡제어를 위해서 TCP에서는 ECE(Explicit Congestion notification Echo)와 CWR(Congestion Window Reduced) 필드가 사용되고 IP에서는 ECN(Explicit Congestion Notification)이 사용됨.
- 혼잡제어를 위해서는 TCP의 연결설정 단계에서 양쪽이 ECN을 사용할 것인지를 합의해야 함.
  - 혼잡제어를 사용을 원하는 호스트 A는 연결설정에서 SYN, CWR, ECE 필드를 1로 만들어 보냄 -> 이를 받은 호스트 B는 혼잡제어에 동의 할 경우 SYN, ACK, ECE 필드를 1로 만들어 보냄.



혼잡제어 사용을 합의하는 경우

전송 계층 관련 프로토콜 분석

- 만약 호스트 B가 혼잡제어가 필요 없다고 판단되는 경우에는 SYN, ACK만 보낸다. 이 경우 혼잡제어는 무시.



혼잡제어를 사용하지 않는 경우

그림 14-8 혼잡제어 사용 전 설정 단계

전송 계층 관련 프로토콜 분석

- 양쪽이 혼잡제어에 동의하는 경우.
- TCP의 윈도우 크기를 줄이도록 결정하는 것은 IP의 ECN 필드, ECN은 ECT(ECN Capable Transport) 비트와 CE(Congestion Experienced) 비트 두 개로 구성 -> ECN 필드가 (0, 0)으로 설정되면 혼잡제어를 안 한다는 의미 -> (0, 1) 혹은 (1, 0)으로 설정되면 라우터에게 혼잡제어를 하라는 의미(ECT 상태).
- 라우터에서 혼잡이 발견되면 라우터는 받는 쪽(호스트 B)로 ECN 필드를 (1, 1)로 만들어 보냄(CE 상태) -> 호스트 B는 TCP의 ECE를 1로 만들어 호스트 A에게 보냄 -> ECE를 받은 호스트 A는 슬라이딩 윈도우의 크기를 줄이고, ECE에 대한 확인으로 CWR을 1로 만들어 보냄.

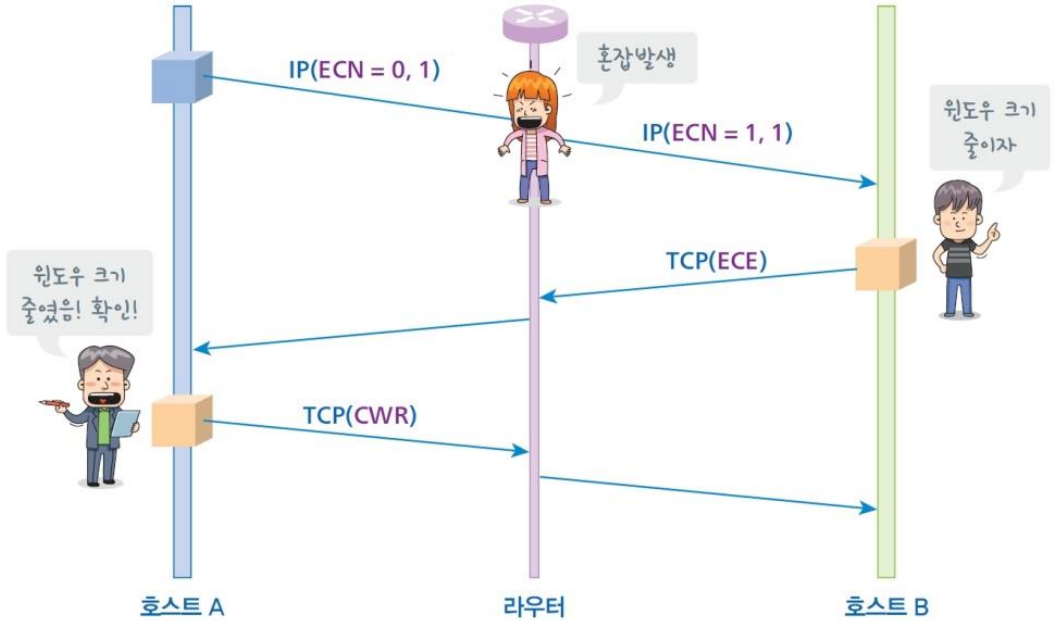


그림 14-9 혼잡제어가 이루어지는 과정

SECTION 02

소켓 프로그래밍

1. 소켓 프로그래밍 개요

- 여러개의 응용 프로그램들이 TCP를 사용하여 통신하는 과정.
  - 운영체제는 0에서 65535까지의 포트 번호를 중 비어있는 번호를 응용 프로그램에게 제공.
  - 서버쪽 포트에는 여러개의 소켓이 연결, 클라이언트들은 포트에 연결된 멀티 소켓에 하나씩 연결됨 -> **응용 프로그램들이 통신을 하려면 소켓에 접속해야 함.**
- 클라이언트 소켓은 connect()를 사용하여 서버 쪽 소켓과의 연결을 시도 -> connect()를 받아주는 것은 서버쪽의 accept() 함수.
- connect() 이전에 accept()가 준비되어야 함으로 서버쪽 코드를 실행 후 클라이언트 코드를 실행.

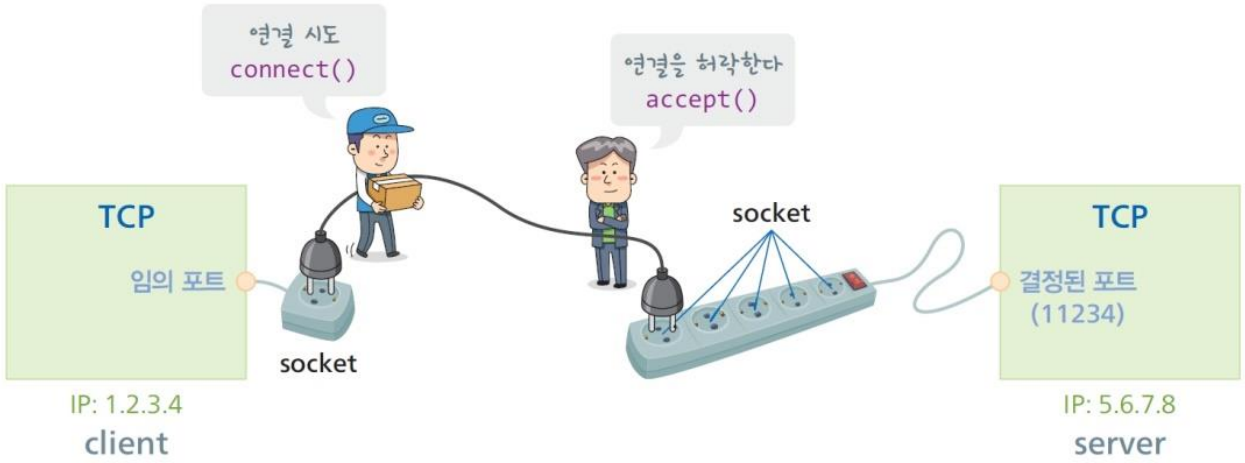


그림 14-10 소켓을 이용한 통신 구조

소켓 프로그래밍

- 소켓 사용방법은 파일에서 데이터를 읽거나 쓰는 작업과 유사 함.
- 파일을 사용하기 위해서는 해당파일이 존재하는지, 존재한다면 파일에 접근할 권한이 있는지를 먼저 살펴봐야 함.
- 파일을 사용하려면 준비단계가 필요한데, 이 함수가 open().
- fd = open("파일이름")이 성공적으로 끝났다면 파일에 접근 할 수 있는 열쇠(fd)를 줌 -> 열쇠를 이용하여 파일에 있는 데이터를 읽거나 쓸 수 있음.
- 작업을 한 후에는 열쇠를 반환해야 한다. 열쇠를 반환하는 작업이 close(fd).
- open() - read() / write() - close() 순으로 파일 작업이 이루어짐.

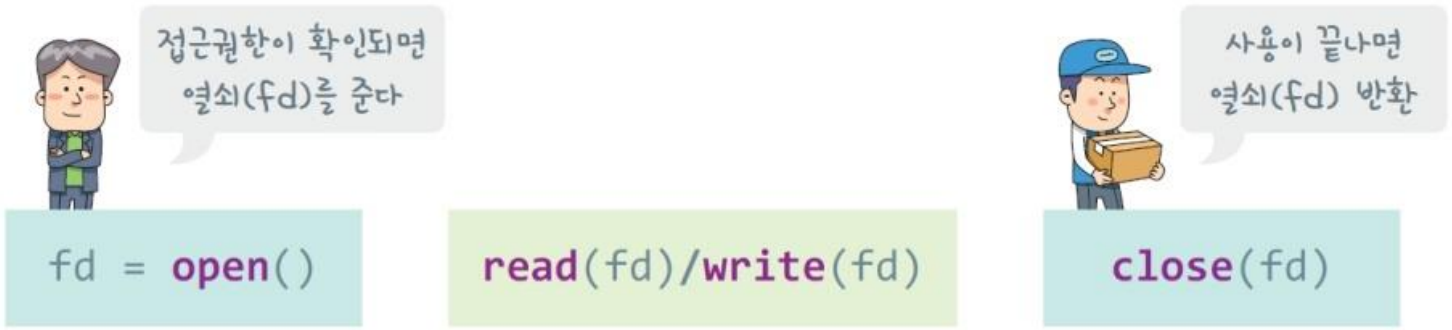


그림 14-11 파일에 접근하려면 열쇠가 있어야 한다

소켓 프로그래밍

- unix.txt 파일에 문자 데이터 Test를 쓰는 코드

코드 14-1 파일에 데이터 쓰기

```
01 #include <unistd.h>
02 #include <fcntl.h>
03
04 void main() {
05    int fd;
06
07    fd = open("unix.txt", O_RDWR);          // fd는 unix.txt에 대한 접근권한
08    write(fd, "Test", 5);                   // "Test"를 fp에 쓰기(write)
09    close(fd);                              // fd를 닫음
10 }
```

소켓 프로그래밍

- 소켓 통신 socket() - send() / recv() - close() 순.
- send(cs): 소켓에 쓰기write 연산을 하면 데이터를 보낸다는 의미. recv(cs): 소켓에서 읽기read 연산을 하면 데이터를 가져온다는 의미.
- cs = socket() 이 후에 클라이언트 소켓이 서버에 접속하는 과정(connect())과 서버가 이를 받아들이는 과정(accept())이 추가.

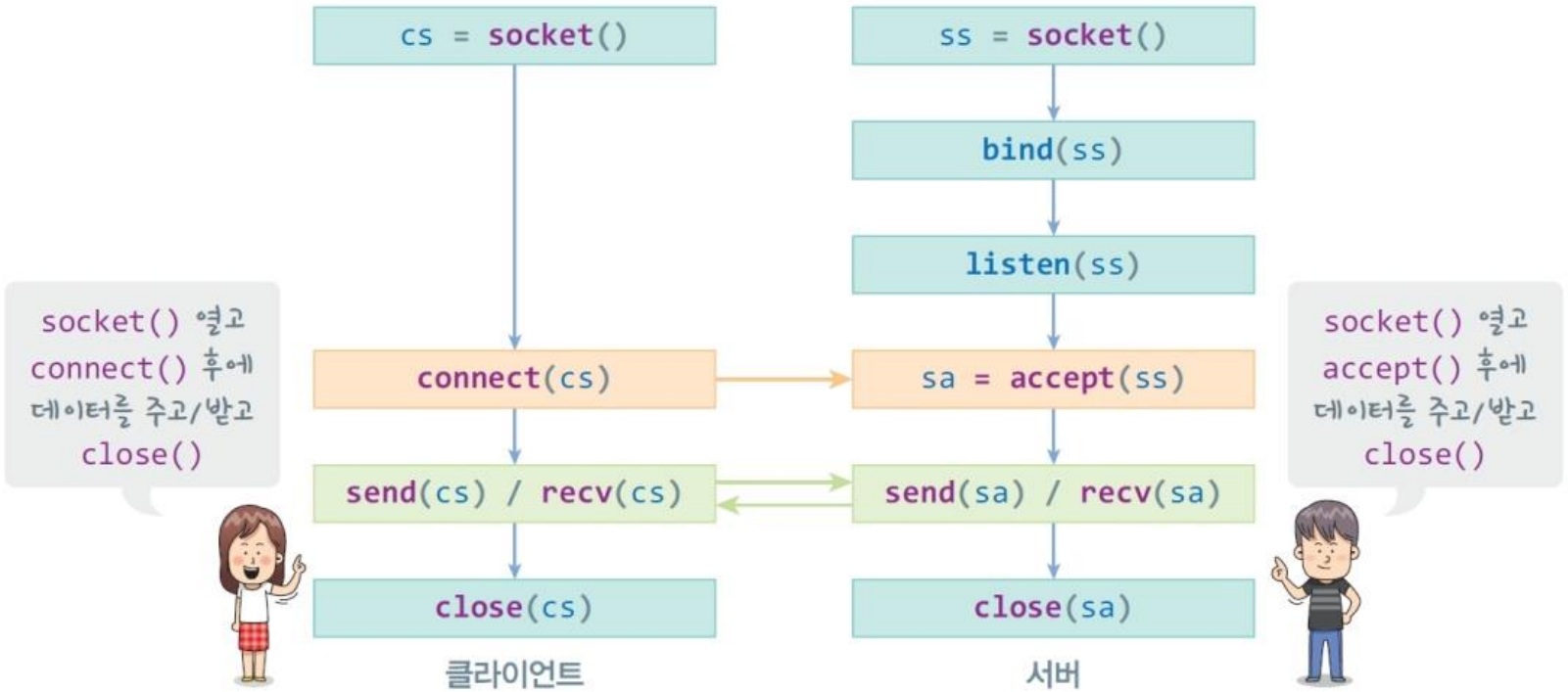


그림 14-12 클라이언트/서버 소켓의 통신 절차

소켓 프로그래밍

- 클라이언트 쪽의 경우, socket() - connect() - send() / recv() - close() 순으로 작업이 이루어짐.
- 먼저 실행되어야 할 서버 쪽의 경우 socket() - accept() - send() / recv() - close() 순으로 작업이 이루어짐.
- cs = socket()이 성공하면 열쇠(cs)를 주는데 이를 소켓 기술자socket descriptor라 부름 -> 이후의 모든 작업은 소켓 기술자(cs)를 통해 이루어짐.
- connect()가 서버와의 접속을 시도 함. 이후 서버에게 데이터를 보내는 경우 send()를 사용하고, 서버로부터 데이터를 받을 때는 recv()를 사용.
- 소켓 사용을 마치면 close()를 사용하여 소켓 기술자를 반환.
- 클라이언트와 달리 서버 쪽에는 bind()와 listen()이 추가.
  - 서버 쪽에는 포트 하나에 여러개의 소켓이 연결됨 -> 포트에 여러 개의 소켓을 연결하는 단계가 bind(). 한번만 실행하면 됨.
  - listen()은 여러개의 소켓들을 사용할 수 있는 상태로 만드는 함수 -> listen()은 accept()와 쌍으로 사용. 소켓을 듣고 있다(listen())가 클라이언트들이 접속을 시도하면 그 중의 하나를 받아들이는(accept()) 형태.

소켓 프로그래밍

코드 14-2 클라이언트 소켓 코드

```c
#include <stdio.h>                                  /* printf() */
#include <string.h>                                 /* memset() */
#include <unistd.h>                                 /* close() */
#include <sys/socket.h>                             /* socket library */
#include <arpa/inet.h>                              /* internet library */

void main()
{
   int cs;
   char buf[5];
   struct sockaddr_in csa;                          /* address struct */

   memset(&csa, 0, sizeof(csa));
   csa.sin_family = AF_INET;                        /* internet socket */
   csa.sin_addr.s_addr = inet_addr(127.0.0.1);      /* loop back addr. */
   csa.sin_port = htons(11234);                     /* PORT = 11234 */

   cs = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
   connect(cs, (struct sockaddr *) &csa, sizeof(csa));

   recv(cs, buf, 5, 0);                             /* receive 5 bytes */
   printf("Receive [%s]\n", buf);

   close(cs);
}
```

소켓 프로그래밍

- 소켓과 인터넷에 관련된 라이브러리는 4번줄 <sys/socket.h>와 5번줄 <arpa/inet.h>에 있음.
- 10번줄 sockaddr_in은 IP 주소나 포트번호와 같은 주소 정보를 가지는 구조체.
- 12번 줄 memset(&csa, 0, sizeof(csa))은 주소 구조체 csa를 초기화 함.
- 13번 줄 csa.sin_family = AF_INET는 인터넷을 사용하겠다는 의미.
- 14번 줄 csa.sin_addr.s_addr = inet_addr(127.0.0.1)는 서버의 IP 주소를 설정.
- 15번 줄의 csa.sin_port = htons(11234)은 서버의 포트 번호를 설정. 서버 포트번호는 11234임. 서버 IP 주소 127.0.0.1은 루프 백loop-back 주소라 불리는 특별한 IP 주소 -> 컴퓨터가 고정 IP 주소를 가진 것처럼 만들어 줌.
- 17번 줄 cs = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)에서 소켓을 open. socket() 매개변수 중 IPPROTO_TCP는 TCP를 사용하여 통신한다는 의미.
- 18번줄 connect(cs, (struct sockaddr *) &csa, sizeof(csa))에서 서버와 연결.
- 20번 줄 recv(cs, buf, 5, 0)에서 소켓으로부터 5바이트 문자를 받아 buf에 저장.
- 21번 줄에서 buf에 저장된 문자열을 출력.
- 23번 줄 close(cs)에서 소켓 기술자를 닫음.

소켓 프로그래밍

코드 14-3 서버 소켓 코드

```
01 #include <string.h>                                    /* memset() */
02 #include <unistd.h>                                    /* close() */
03 #include <sys/socket.h>                                /* socket library */
04 #include <arpa/inet.h>                                 /* internet library */
05
06 void main()
07 {
08    int ss, sa;
09    struct sockaddr_in ssa;                             /* address struct */
10
11    memset(&ssa, 0, sizeof(ssa));
12    ssa.sin_family = AF_INET;                           /* internet socket */
13    ssa.sin_addr.s_addr = htonl(INADDR_ANY);
14    ssa.sin_port = htons(11234);                        /* PORT = 11234 */
15
16    ss = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
17    bind(ss, (struct sockaddr *) &ssa, sizeof(ssa));    /* binding sockets */
18
19    while(1) {                                          /* 무한루프 - 데몬 */
20       listen(ss, 10);
21       sa = accept(ss, 0, 0);
22       send(sa, "test", 5, 0);                         /* send 5 bytes */
23       close(sa);
24    }
25 }
```

소켓 프로그래밍

- 12번 서버에서는 클라이언트의 주소를 알 수 없기 때문에 htonl(INADDR_ANY)라고 지정.
- 15번 줄 ss = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)에서 소켓을 open.
- 16번 줄 bind(ss, (struct sockaddr *) &ssa, sizeof(ssa))에서 소켓을 연결.
- 18번 줄 서버는 무한 루프를 돌면서(while(1)) 클라이언트의 요청이 있을 때마다 서비스를 해줌.
- 19번 줄 listen(ss, 10)에서 여러 소켓들을 활성화 함
- 20번 줄 sa = accept(ss, 0, 0)에서 소켓에 연결된 클라이언트를 받아들임. listen(ss, 10)에서는 소켓 기술자 ss가 사용됐지만, sa = accept()에서 소켓 기술자가 sa로 바뀌는 것에 주의. sa = accept()에 의해 실제로 받아들여진 소켓 기술자는 sa 임. listen(ss, 10)에서 10은 서버의 작업을 기다리는 클라이언트 소켓 대기열의 크기를 가리킴.
- 21번 줄 send(sa, "test", 5, 0)에서 클라이언트에게 문자열 "test"을 보냄. test는 4글자이지만 문자열 끝을 알리는 '₩n'가 추가되기 때문에 전체 크기는 5바이트 임.
- 22번 줄 close(sa)에서 클라이언트 소켓 sa를 닫음. 맨 처음 열린 서버 소켓 기술자 ss는 아직 살아있다기 때문에 클라이언트가 몇 번을 접속하더라도 문자열 "test"를 보낸다.

소켓 프로그래밍

- 유닉스 상에서 서버와 클라이언트 소켓 코드를 실행하는 방법.
  - 서버와 클라이언트 코드를 컴파일 함 -> gcc -o server server.c를 사용하여 server라는 실행 파일을 만들고, gcc -o client client.c를 사용하여 client 실행 파일을 만듦.
  - 서버는 백그라운드background로 실행.
  - 클라이언트 프로그램을 실행 시키면 서버로부터 "test"를 받아 Receive [test]가 출력됨.

```
$>gcc -o server server.c          서버 코드 컴파일
$>gcc -o client client.c          클라이언트 코드 컴파일
$>server &                        서버코드 백그라운드 실행(&)
$>client                          클라이언트 실행
$>Receive [test]                  서버로부터 test 수신
$>client                          클라이언트 다시 실행
$>Receive [test]                  서버로부터 test 수신
```

그림 14-13 유닉스 상에서 서버와 클라이언트 소켓 코드 실행 방법