6. TCP프로토콜과 공격

41526: 네트워크보안

강문수
(mskang@chosun.ac.kr)

컴퓨터네트워크 연구실
(Computer Networks Lab.)

6장 목차

6.1 개요

- 인터넷 프로토콜 그룹의 핵심 프로토콜
- 브라우저, SSH, 텔넷과 전자우편과 같은 대부분의 응용 프로그램은 통신에 TCP를 사용
- TCP는 응용 프로그램에 호스트 간 통신 서비스를 제공하는 전송 계층이라는 계층에 있다
- 신뢰성과 순서화된 통신을 달성하기 위해 TCP는 연결을 유지하기 위해 통신의 양쪽 종단(end)이 필요하다
- 이 연결을 두 통신 응용 프로그램 사이의 파이프로 상상할 수 있다.

실습 환경

Network Security Labs

Packet Sniffing and Spoofing Lab
Sniffing packets sent over the local network and spoofing various types of packets using Python and C.

ARP Cache Poisoning Attack Lab
Lanuch ARP cache posining attacks; use this attack to conduct man-in-the-middle attacks.

ICMP Redirect Attack Lab
Attacks at the IP layer, ICMP redirect attack, and man-in-the-middle attack.

TCP Attacks Lab
Launching attacks to exploit the vulnerabilities of the TCP protocol, including session hijacking, SYN flooding, TCP reset attacks, etc.

The Mitnick Attack Lab
Launching the classic Mitnick attack to gain the unauthorized access to the target machine. This is a special case of TCP session hijacking.

실습 환경

TCP Attacks Lab

Overview

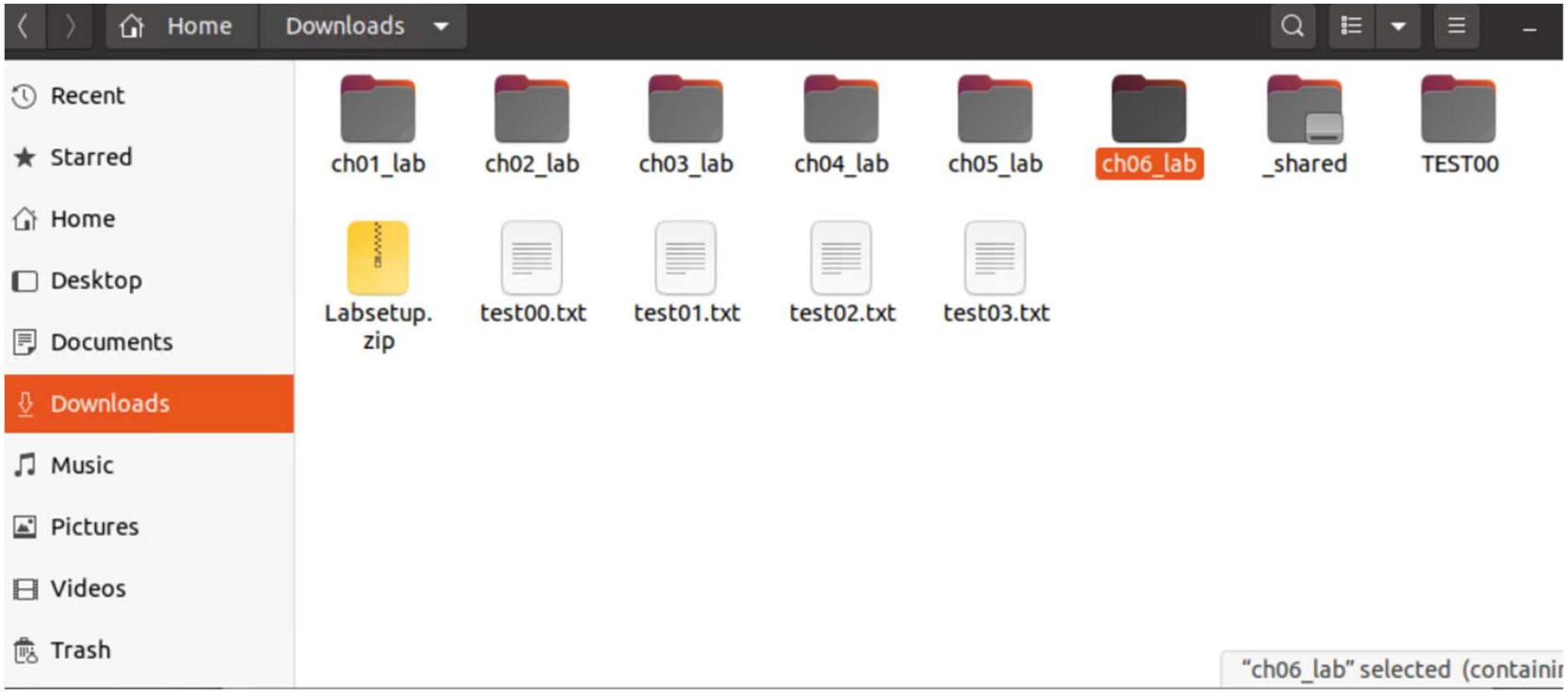
The learning objective of this lab is for students to gain first-hand experience on vulnerabilities, as well as on attacks against these vulnerabilities. Wise people learn from mistakes. In security education, we study mistakes that lead to software vulnerabilities. Studying mistakes from the past not only help students understand why systems are vulnerable, why a "seemly-benign" mistake can turn into a disaster, and why many security mechanisms are needed. More importantly, it also helps students learn the common patterns of vulnerabilities, so they can avoid making similar mistakes in the future. Moreover, using vulnerabilities as case studies, students can learn the principles of secure design, secure programming, and security testing.

The vulnerabilities in the TCP/IP protocols represent a special genre of vulnerabilities in protocol designs and implementations; they provide an invaluable lesson as to why security should be designed in from the beginning, rather than being added as an afterthought. Moreover, studying these vulnerabilities help students understand the challenges of network security and why many network security measures are needed.

Tasks (English) (Spanish)

- **VM version:** This lab has been tested on our `SEED Ubuntu-20.04 VM`
- **Lab setup files**
  - Labsetup.zip
  - Labsetup-arm.zip (for Apple Silicon machines)
- **Manual:**: Docker manual

실습 환경


Home
Downloads
Recent
Starred
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
ch01_lab
ch02_lab
ch03_lab
ch04_lab
ch05_lab
ch06_lab
_shared
TEST00
Labsetup.zip
test00.txt
test01.txt
test02.txt
test03.txt
"ch06_lab" selected (containi

실습 환경

docker-compose.yml에서,
attacker:, User1:, User2: 설정에 모두 volumes: 부분 추가

```
16      Victim:
17          image: handsonsecurity/seed-ubuntu:large
18          container_name: victim-10.9.0.5
19          tty: true
20          cap_add:
21                  - ALL
22          privileged: true
23          volumes:
24                  - ./volumes:/volumes
25          sysctls:
26                  - net.ipv4.tcp_syncookies=0
27
28          networks:
29              net-10.9.0.0:
30                  ipv4_address: 10.9.0.5
31
32          command: bash -c "
33                   /etc/init.d/openbsd-inetd start  &&
34                   tail -f /dev/null
35                   "
36
37      User1:
38          image: handsonsecurity/seed-ubuntu:large
```

실습 환경

■ 컨테이너 네트워크 실행

```
[06/03/24]seed@VM:~/.../ch06_lab$ ls
docker-compose.yml  volumes
[06/03/24]seed@VM:~/.../ch06_lab$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating user2-10.9.0.7  ... done
Creating victim-10.9.0.5 ... done
Creating user1-10.9.0.6  ... done
Creating seed-attacker   ... done
Attaching to seed-attacker, victim-10.9.0.5, user2-10.9.0.7, user1-10.9.0.6
user2-10.9.0.7 |  * Starting internet superserver inetd                 [ OK ]
victim-10.9.0.5 |  * Starting internet superserver inetd                 [ OK ]
user1-10.9.0.6 |  * Starting internet superserver inetd                 [ OK ]
```

6.2 TCP 프로토콜 동작 방식

■ TCP 동작 방식을 설명하기 위해 간단한 TCP 클라이언트와 서버 프로그램을 사용

TCP 클라이언트 프로그램: py_tcp_client.py

```
#!/bin/env python3
import socket

tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcp.connect(('10.0.2.69', 9090))

tcp.sendall(b"Hello Server!\n")
tcp.sendall(b"Hello Again!\n")

tcp.close()
```

TCP 서버 프로그램(Python) py_tcp_server.py

```
#!/bin/env python3
import socket

tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

tcp.bind(("0.0.0.0", 9090))

tcp.listen()
conn, addr = tcp.accept()

with conn:
    print('Connected by', addr)
    while True:
        data = conn.recv(1024)
        if not data:
            break
        print(data)
        conn.sendall(b"Got the data!\n")
```

6.2 TCP 프로토콜 동작 방식

■ TCP 동작 방식을 설명하기 위해 간단한 TCP 클라이언트와 서버 프로그램을 사용

TCP 클라이언트 프로그램: tcp_client.c

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <arpa/inet.h>

int main()
{
  // step 1: create a socket
  int sockfd = socket(AF_INET, SOCK_STREAM, 0);

  // step 2: set the destination information
  struct sockaddr_in dest;
  memset(&dest, 0, sizeof(struct sockaddr_in));
  dest.sin_family = AF_INET;
  dest.sin_addr.s_addr = inet_addr("10.9.0.5");
  dest.sin_port = htons(9090);

  // step 3: connect to the server
  connect(sockfd, (struct sockaddr *)&dest, siz

  // step 4: send data to the server
  char *buffer1 = "Hello Server!\n";
  char *buffer2 = "Hello Again!\n";
```

TCP 서버 프로그램: tcp_server.c, tcp_server_improved.c

```c

int main()
{
  int sockfd, newsockfd;
  struct sockaddr_in my_addr, client_addr;
  char buffer[100];

  // step 1: create a socket
  // AF_INET: IPv4 주소 체계
  // SOCK_STREAM: TCP 소켓 유형
  // 0: 기본 프로토콜 사용 (TCP의 경우 IPPROTO_TCP)
  // 서버 소켓, socket 함수로 생성
  // 클라이언트의 연결 요청을 수신 대기(listen)하고 수락(accept)하기 위해 사용
  sockfd = socket(AF_INET, SOCK_STREAM, 0);

  // step 2: bind a port number
  memset(&my_addr, 0, sizeof(struct sockaddr_in));
  my_addr.sin_family = AF_INET;
  my_addr.sin_port = htons(9090);
  my_addr.sin_addr.s_addr = INADDR_ANY; // 모든 인터페이스에서 수신 대기

  bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr_in));

```

6.2 TCP 프로토콜 동작 방식

■ TCP 연결, 송신/수신 버퍼

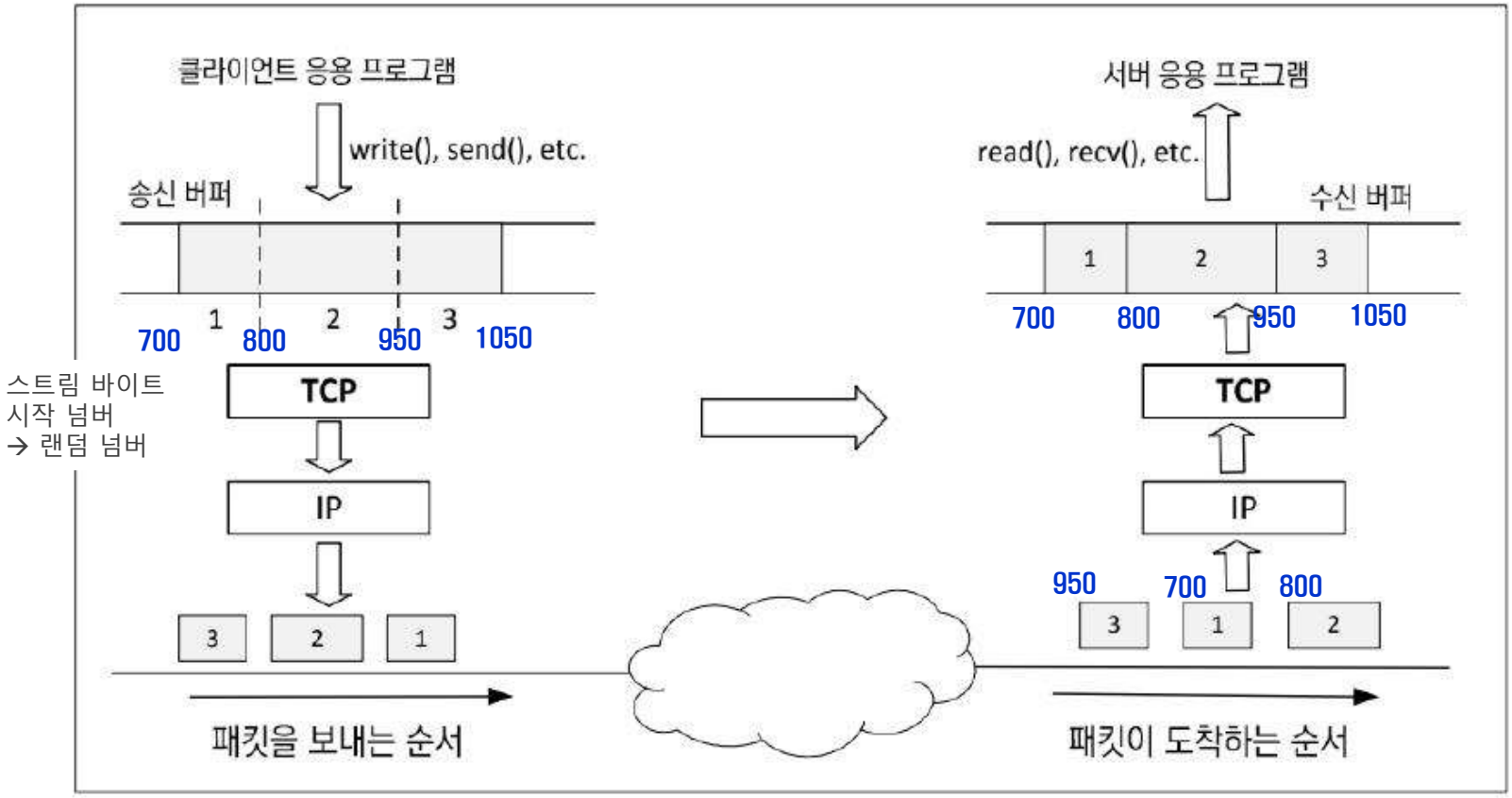


그림 6.1: TCP 데이터가 전송되는 방법

6.2 TCP 프로토콜 동작 방식

■ 순서 관리와 신뢰성

➲ 순서 번호(Sequence Number)와 확인응답(Acknowledgment)

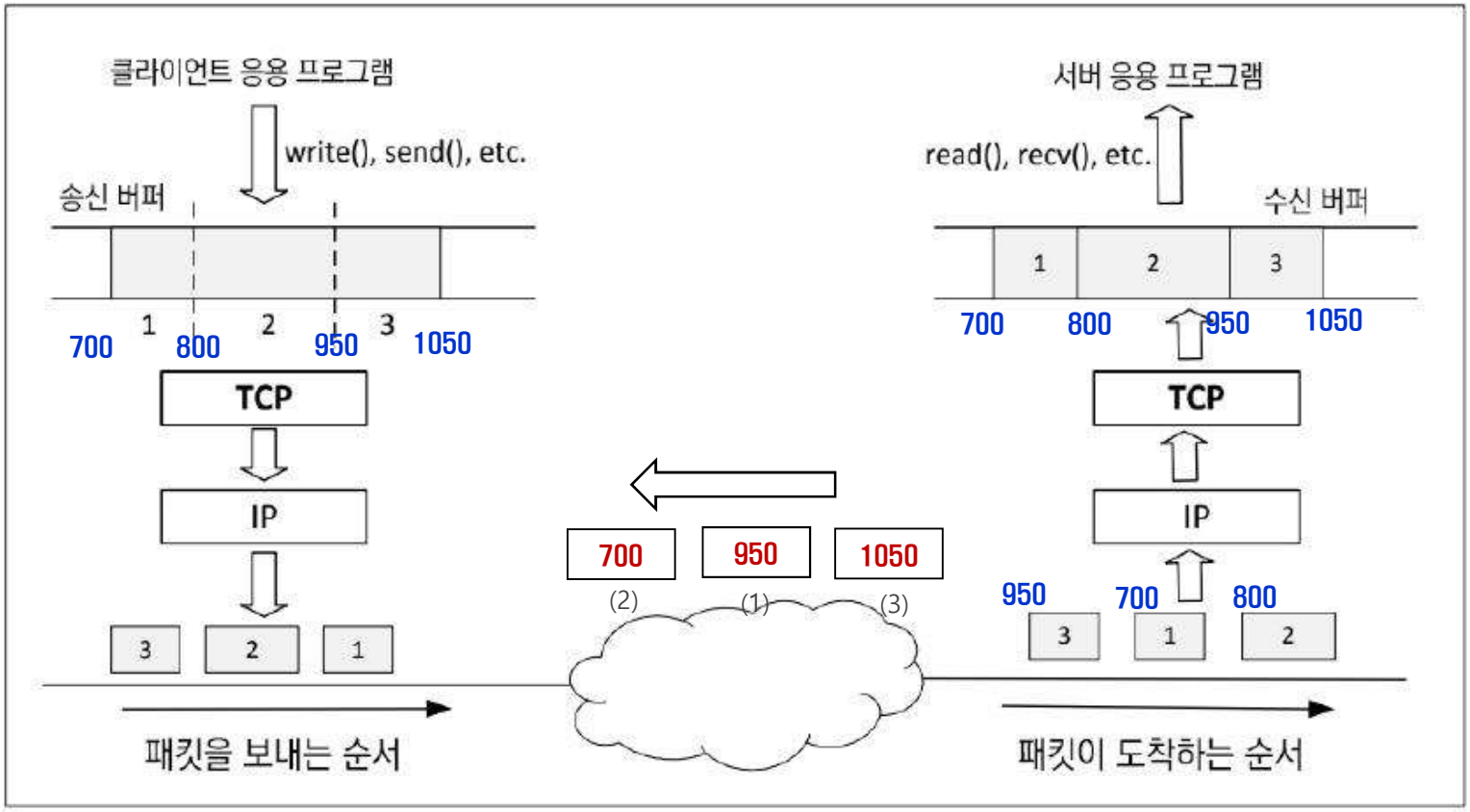


그림 6.1: TCP 데이터가 전송되는 방법

6.2 TCP 프로토콜 동작 방식

■ 속도 관리

➲ 흐름 제어(Flow Control)와 미닫이 창(Sliding Window)

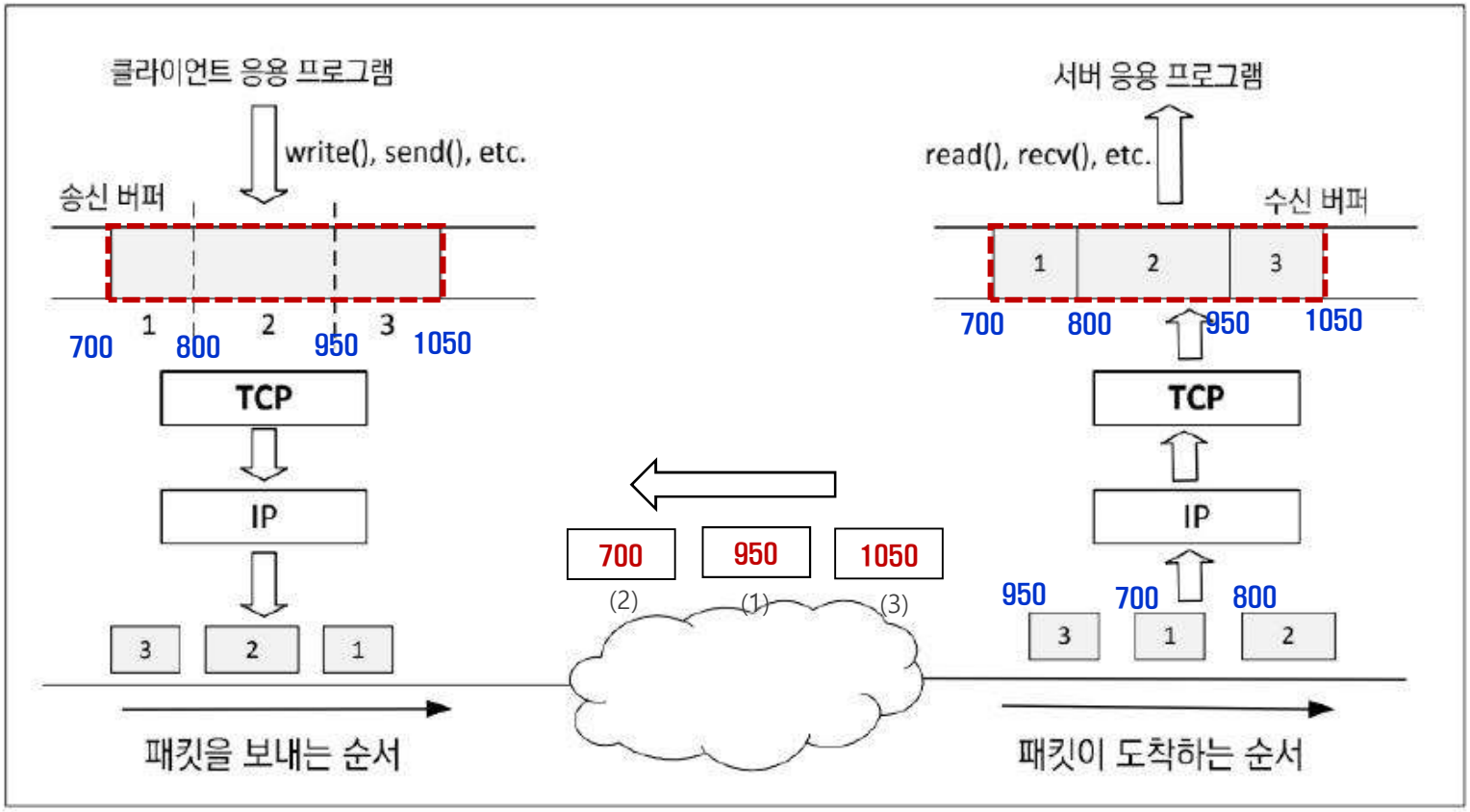


그림 6.1: TCP 데이터가 전송되는 방법

6.2 TCP 프로토콜 동작 방식

■ 속도 관리

➲ 흐름 제어(Flow Control)와 미닫이 창(Sliding Window)

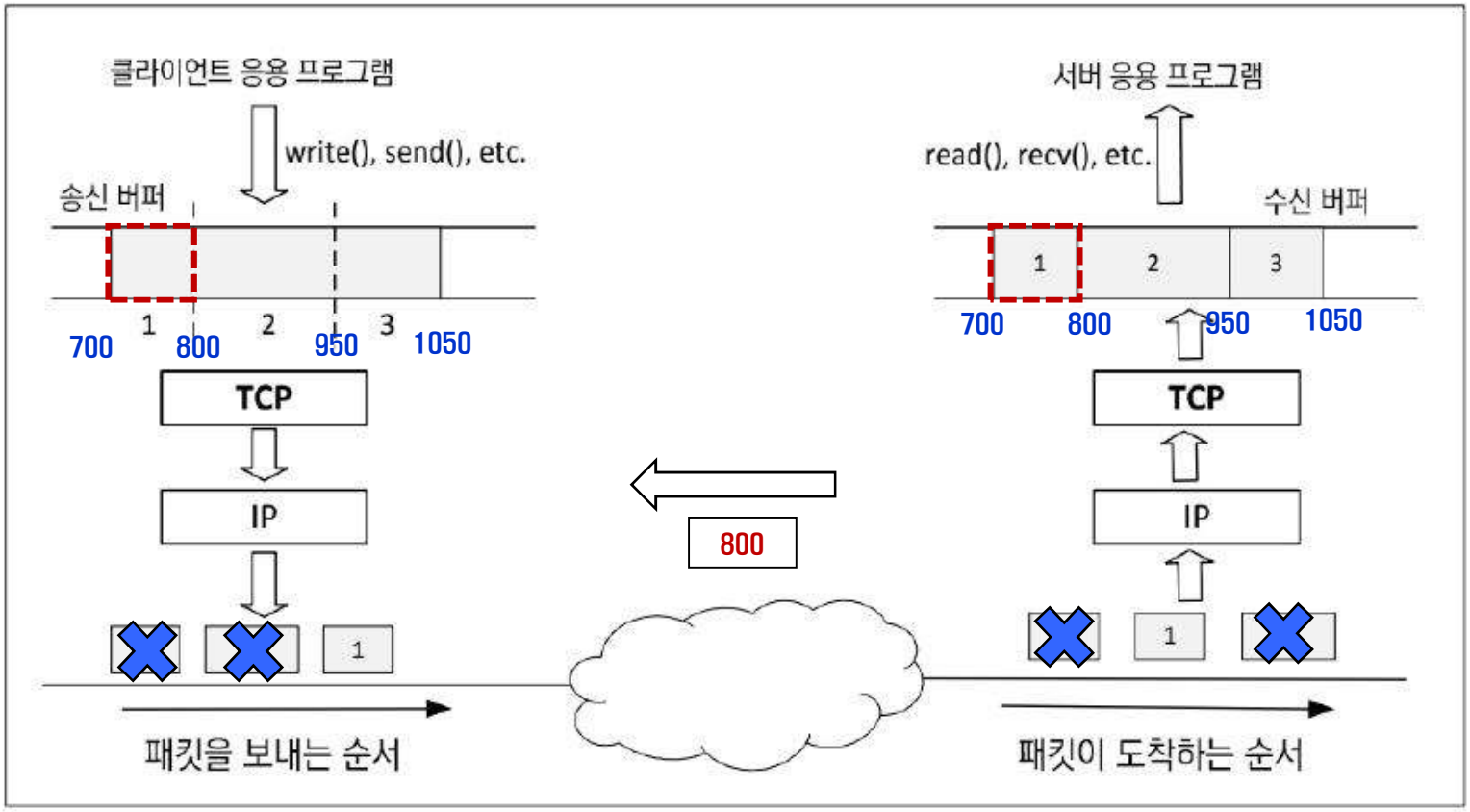


그림 6.1: TCP 데이터가 전송되는 방법

6.2 TCP 프로토콜 동작 방식

■ 속도 관리

➲ 흐름 제어(Flow Control)와 미닫이 창(Sliding Window)

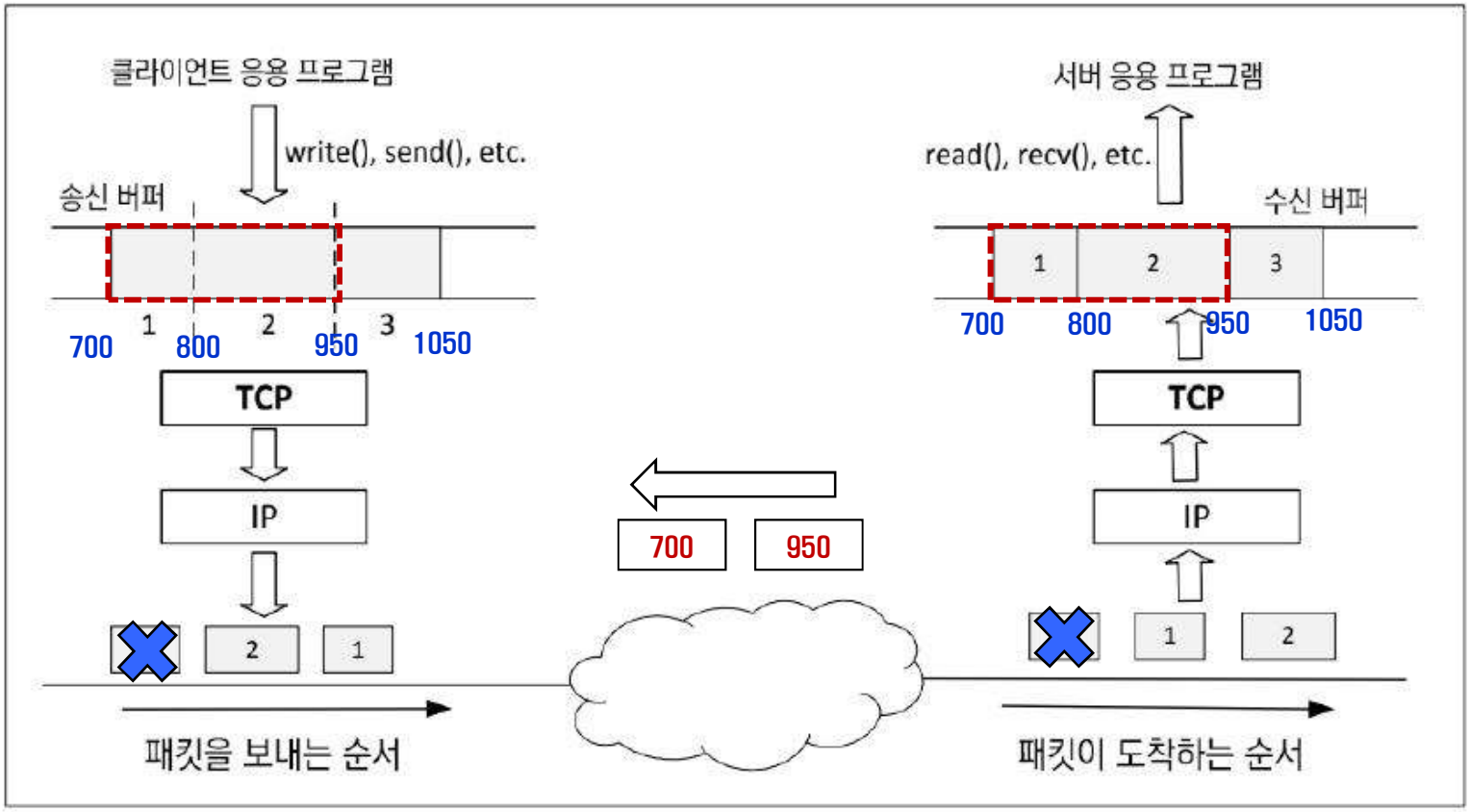


그림 6.1: TCP 데이터가 전송되는 방법

6.2.6 TCP 헤더

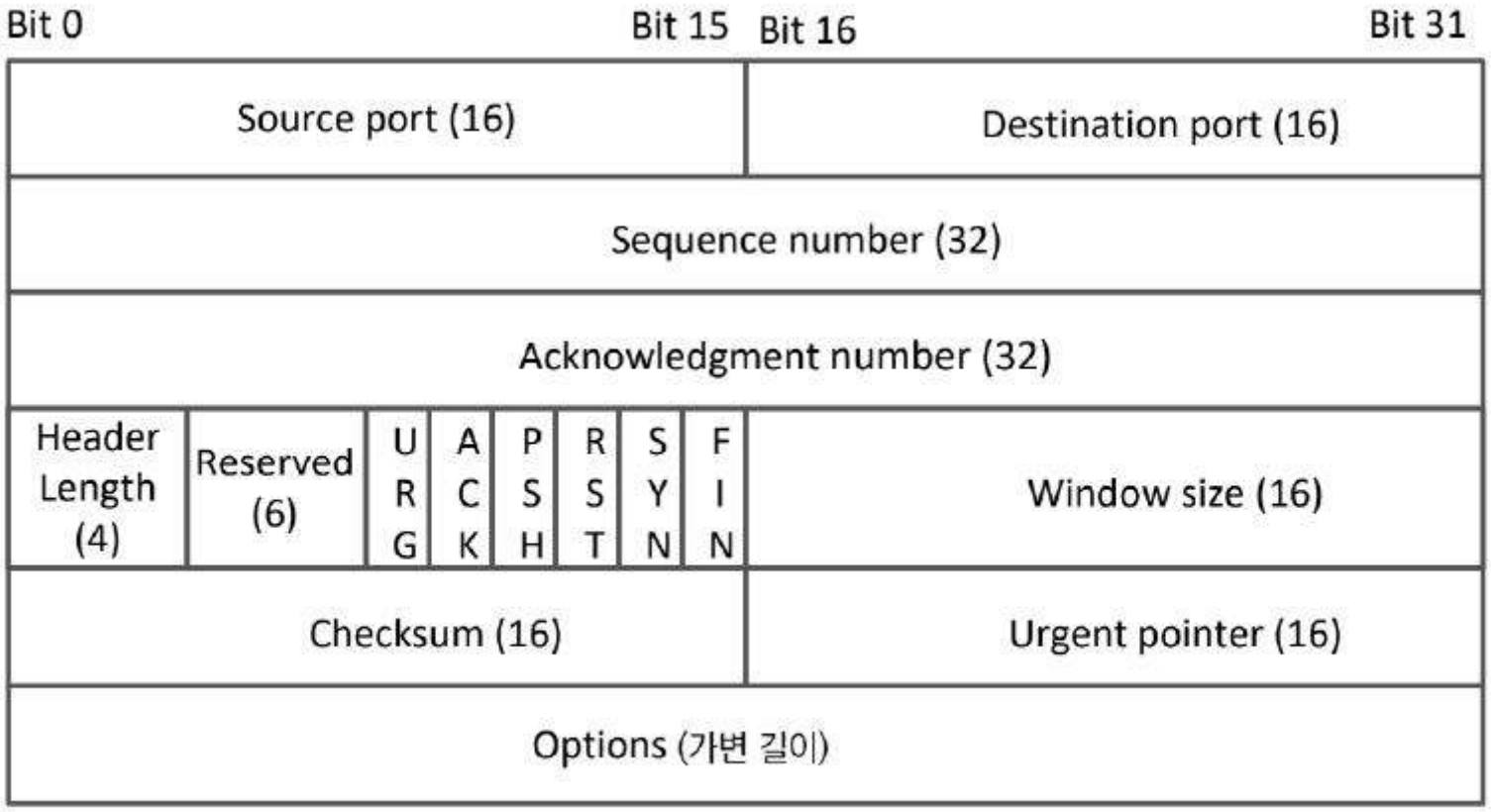
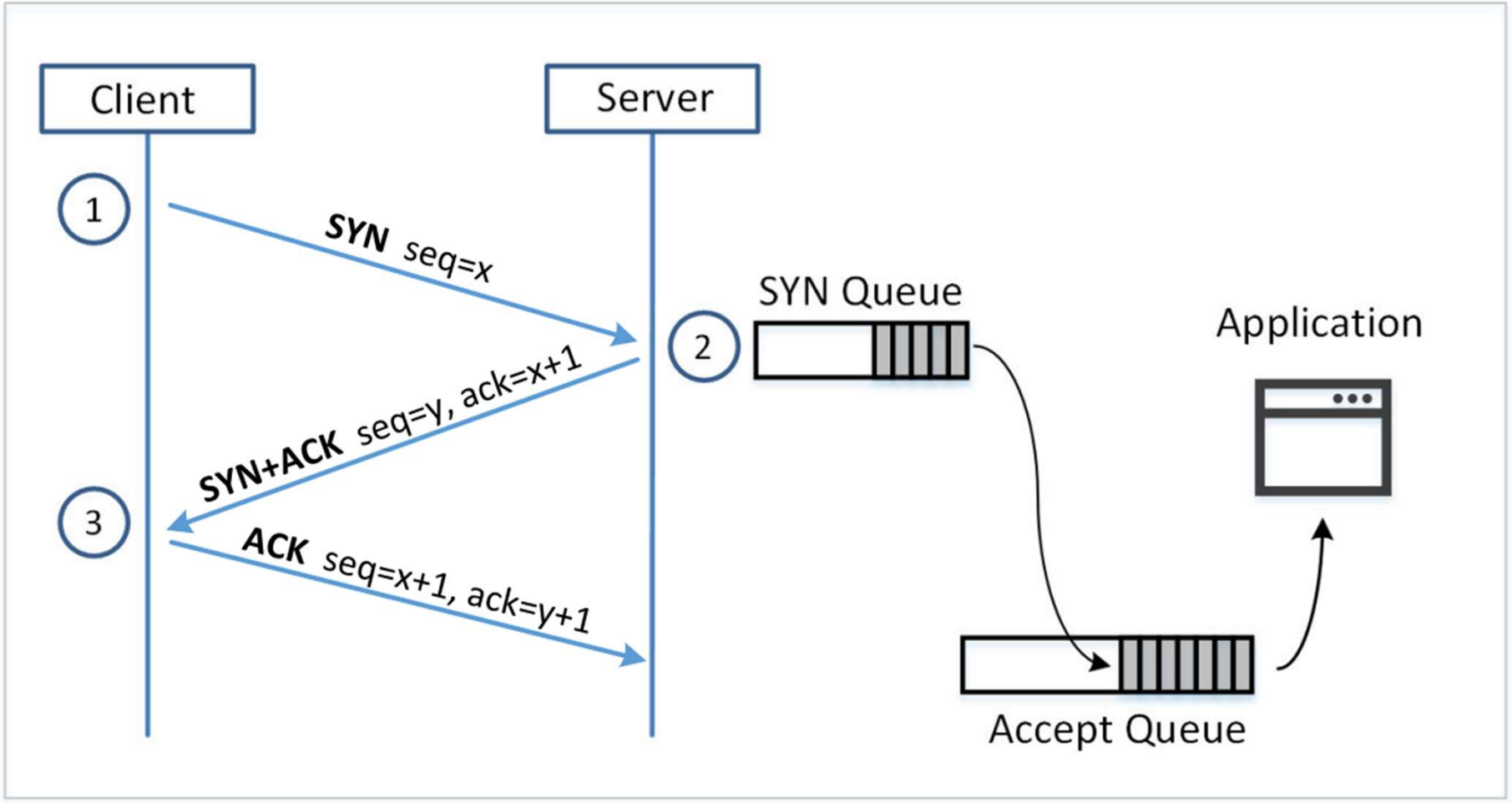


그림 6.2: TCP 헤더

6.3 syn 플러딩 공격

■ 연결 설정(Establishing Connections)



6.3.2 SYN 플러딩 공격