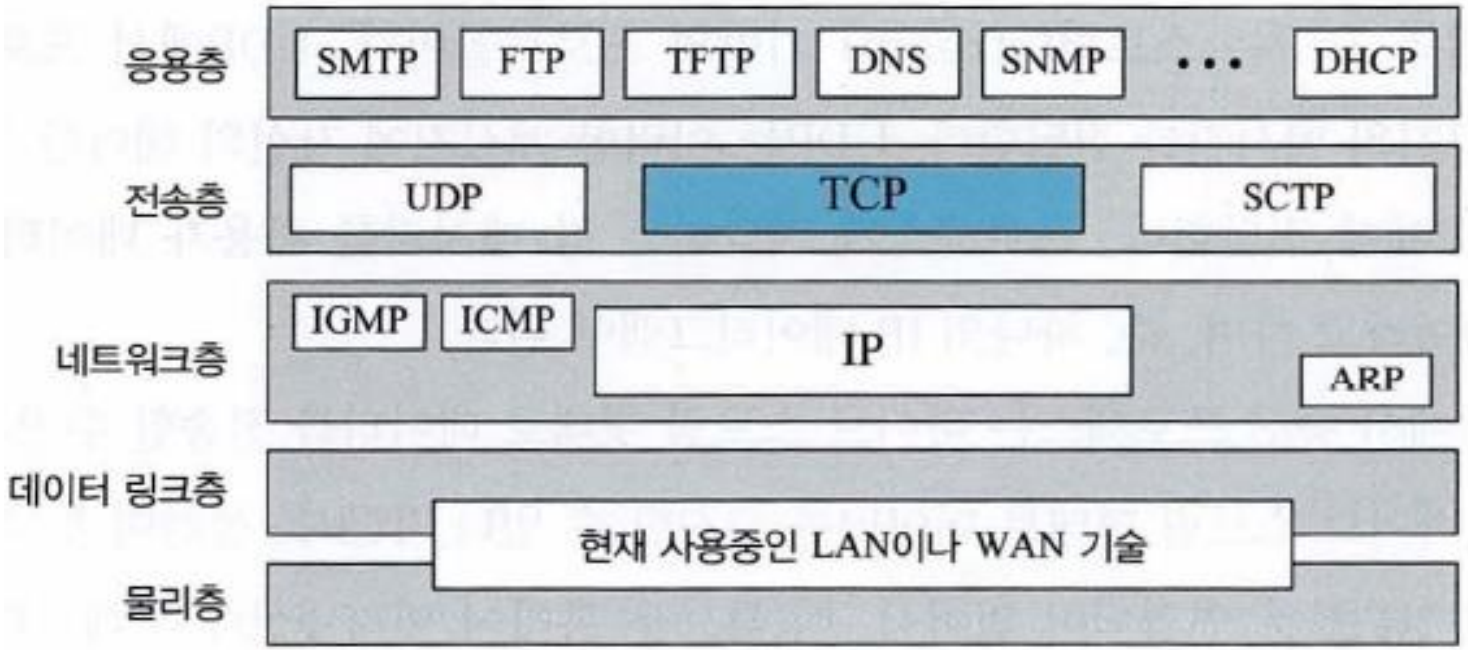
Chapter 12

전송 제어 프로토콜 (TCP)

1. TCP 서비스

- TCP(Transmission Control Protocol)는 응용층과 네트워크층 사이에 위치하고 있으며, 응용 프로그램과 네트워크 동작 사이의 매개체로서 사용된다.



1. TCP 서비스

- **프로세스-대-프로세스 통신**
  - **UDP와 같이,TCP도 포트 번호를 이용하여 프로세스-대-프로세스 통신을 제공한다.**
  - **TCP에서 사용하는 잘 알려진 포트**

포트	프로토콜	설명
7	Echo	메시지 전송장치에게 수신한 메시지를 다시 전송
9	Discard	접속 테스팅을 위한 Null 서비스
11	Users	Active users
13	Daytime	요청하는 호스트에게 날짜와 시간 정보를 보낸다.
17	Quote	오늘의 한마디를 연결된 호스트에게 보낸다.
19	Chargen	자막 생성 서비스 (Character Generation service); 끊임없이 문자 스트림을 보낸다
20, 21	FTP	파일 전송 프로토콜(File Transfer Protocol)
23	TELNET	암호화되지 않은 텍스트 통신
25	SMTP	간이 전자 우편 전송 프로토콜(Simple Mail Transfer Protocol, SMTP) - 인터넷에서 이메일을 보내기 위한 프로토콜

1. TCP 서비스

프로세스-대-프로세스 통신

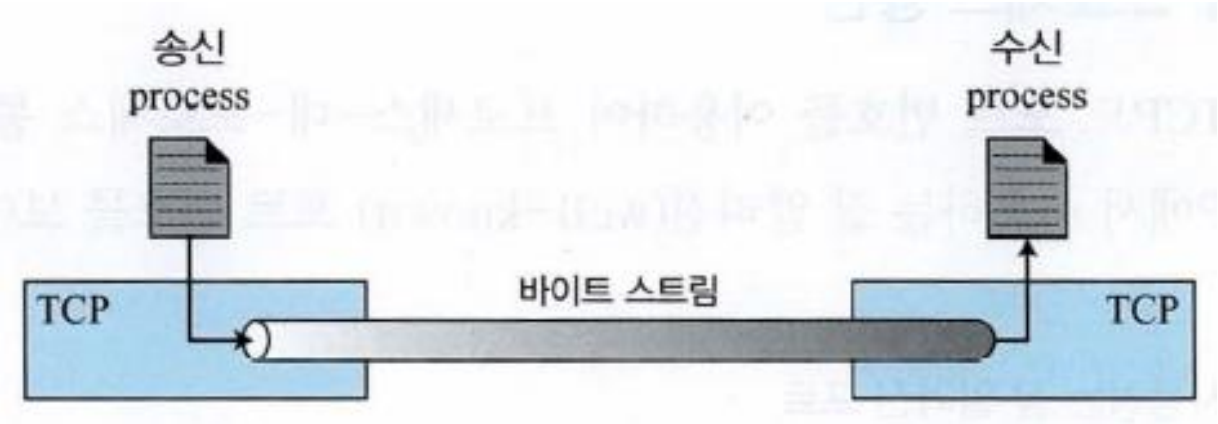
- UDP와 같이,TCP도 포트 번호를 이용하여 프로세스-대-프로세스 통신을 제공한다.
- TCP에서 사용하는 잘 알려진 포트

포트	프로토콜	설명
53	DNS	도메인 네임 시스템(Domain Name System, DNS)은 호스트의 도메인 이름을 호스트의 네트워크 주소로 바꾸거나 그 반대의 변환을 수행할 수 있도록 하기 위해 개발되었다. 특정 컴퓨터(또는 네트워크로 연결된 임의의 장치)의 주소를 찾기 위해, 사람이 이해하기 쉬운 도메인 이름을 숫자로 된 식별 번호(IP 주소)로 변환해 준다.
67	BOOTP	부트스트랩 프로토콜 서버. DHCP로도 사용
79	Finger	네트워크 상의 특정인이나 시스템의 상태를 제공하기 위한 프로토콜
80	HTTP	Hypertext Transfer Protocol. 브라우저가 인터넷의 웹 페이지에 연결할 수 있도록 하기위한 프로토콜.

1. TCP 서비스

스트림 전달 서비스

- UDP와 달리 TCP는 스트림(stream) 기반의 프로토콜이다.
- TCP에서 송신 프로세스는 바이트 스트림 형태로 데이터를 전송할 수 있으며, 수신 프로세스도 바이트 스트림 형태로 데이터를 수신할 수 있다.
- TCP를 통하여 두 개의 프로세스가 가상의 "튜브"로 연결되어 있어서 이 튜브를 통하여 인터넷상에서 데이터를 전송하는 것과 같은 환경을 제공한다.

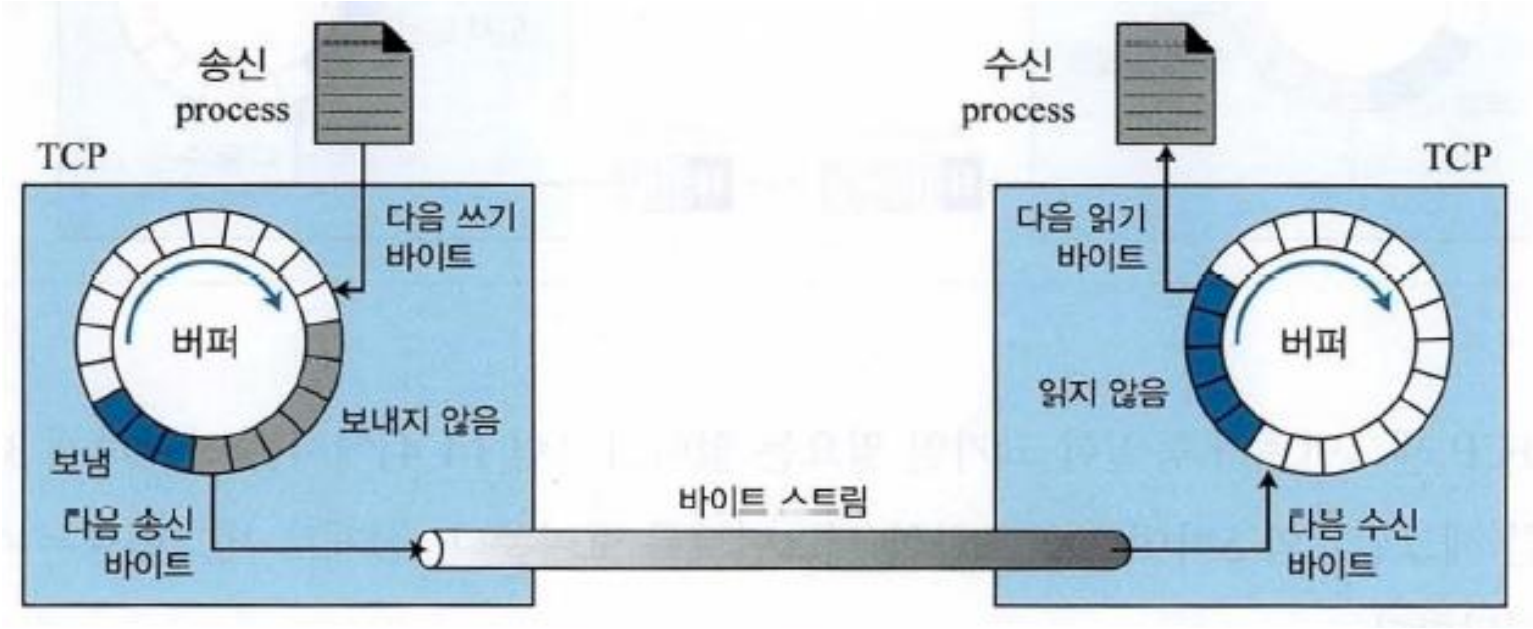


1. TCP 서비스

스트림 전달 서비스

송신 버퍼와 수신 버퍼

- 송신과 수신 프로세스가 동일한 속도로 데이터를 생성하고 소비하지 않을 수 있어서, TCP에는 데이터를 저장하기 위한 버퍼(buffer)가 필요하다.
- TCP에는 각 방향마다 송신 버퍼와 수신 버퍼가 있다.
- 이 버퍼들은 뒤에서 설명하는 흐름 제어와 오류 제어 메커니즘에서 사용된다.

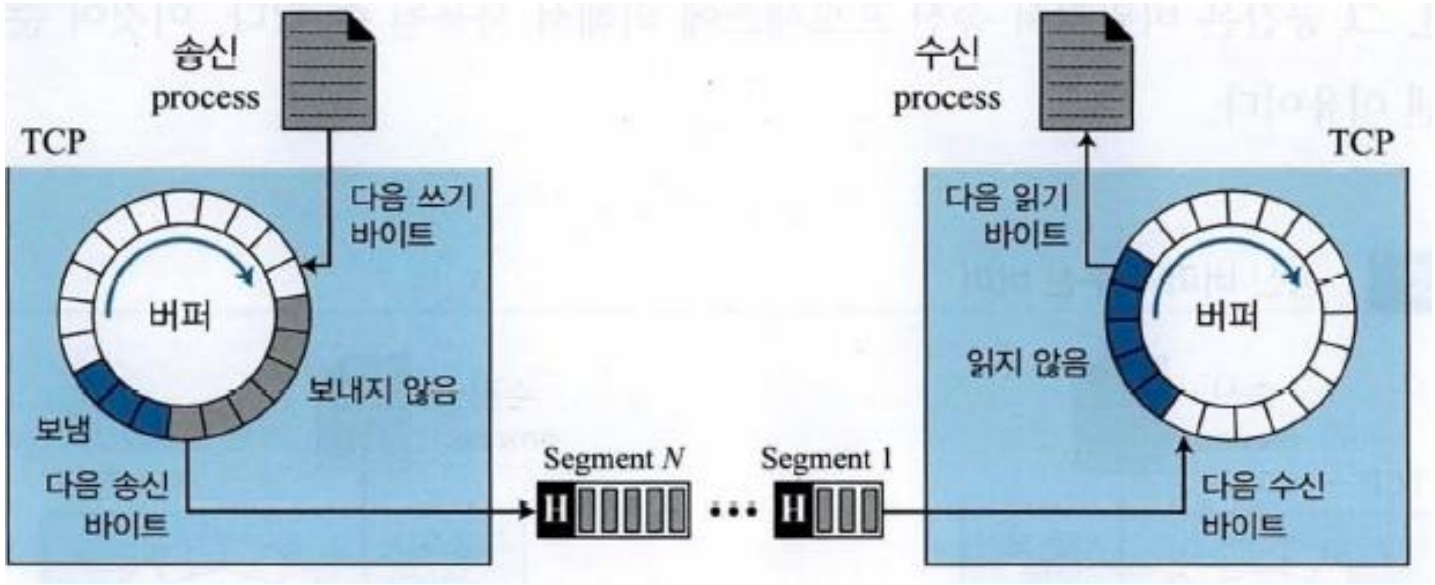


1. TCP 서비스

스트림 전달 서비스

세그먼트

- 버퍼를 이용하면 생산(즉, 송신) 프로세스와 소비(즉, 수신) 프로세스 간의 속도 차이를 처리하지만, 데이터를 전송하기 위해서는 하나의 단계가 더 필요하다.
- TCP에 대한 서비스 제공자로서 IP는 바이트 스트림 형태가 아닌 패킷 형태로 데이터를 전달한다.
- TCP는 일련의 바이트를 세그먼트(segment) 라는 패킷으로 그룹화한다.
- TCP는 각 세그먼트에 헤더를 붙이고(제어 목적으로), 전송을 위해 IP 계층으로 세그먼트를 전달한다.
- 이 세그먼트는 IP 데이터그램으로 캡슐화되어 전송된다.



1. TCP 서비스

전이중 통신

- TCP는 전이중(full-duplex) 서비스를 제공한다.
- 즉, 데이터를 동시에 양방향으로 전송할 수 있다.
- 각 TCP는 송신 버퍼와 수신 버퍼를 가지고 있으며, 세그먼트는 양방향으로 전송된다.

다중화와 역다중화

- UDP와 같이 TCP도 송신 측에서 다중화를 수행하고 수신 측에서 역디중화를 수행한다.
- 그렇지만, TCP는 연결형 프로토콜이기 때문에 해당 프로세스들 간에 연결이 설정되어야 한다.

1. TCP 서비스

연결형 서비스

- **TCP는 UDP와 다르게 연결형 프로토콜이다.**
- **A 노드에 있는 프로세스가 B 노드에 있는 프로세스와 데이터를 주고 받는 경우에는 다음과 같은 세 단계가 발생한다.**
  1. 두 TCP 간에 가상 연결이 설정된다.
  2. 양방향으로 데이터가 교환된다.
  3. 연결이 종료된다.

신뢰성 서비스

- **TCP는 신뢰성 있는 전송 프로토콜이다.**
- **TCP는 데이터가 안전하고 오류 없이 잘 도착했는지를 확인하기 위하여 확인응답 메커니즘을 이용한다.**

2. TCP 특징

번호화 시스템

- **TCP 소프트웨어는 어떤 세그먼트를 전송 또는 수신했는지를 기억하지만, 세그먼트 헤더에는 세그먼트 번호 값을 위한 필드가 없다.**
- **대신에 순서 번호(sequence number)와 확인응답 번호(acknowledgement number) 라는 두 개의 필드가 있다.**
- **이 두 개의 필드는 세그먼트 번호가 아닌 바이트 번호와 관련이 있다.**
- **바이트 번호**
  - TCP는 한 연결에서 전송되는 모든 데이터 바이트에 번호를 매긴다.
  - 이 번호는 각 방향에서 서로 독립적으로 매겨진다.
  - 번호는 0부터 시작할 필요는 없다.
  - 대신에 TCP는 0에서 2^{32} - 1 사이의 임의의 값을 선택하여 처음 바이트의 번호로 설정한다.
  - 예를 들어, 임의의 값이 1,057이고 전송하고자 하는 총 데이터가 6,000바이트라면, 1,057부터 7,056까지의 번호가 전송되는 각 바이트에 매겨진다.
  - 바이트 순서화는 흐름 제어와 오류 제어에서 사용된다.

2. TCP 특징

번호화 시스템

순서 번호

- 바이트에 번호가 매겨지면, TCP는 전송하고자 하는 세그먼트에 하나의 순서 번호를 할당한다.
- 각 세그먼트의 순서 번호는 그 세그먼트에 있는 첫 번째 바이트의 번호로 설정된다.
- 사용자 데이터를 전달하지 않는 세그먼트는 논리적으로 순서 번호를 가질 필요는 없다.
- 그렇지만 단지 제어 정보만을 포함하는 특별한 세그먼트는 순서 번호를 필요로 하며 수신 측으로부터 확인응답된다.
- 이러한 세그먼트는 연결 설정, 해제, 또는 중단을 위해서 사용된다.

2. TCP 특징

번호화 시스템

순서 번호

- 예제. TCP 연결을 통하여 5,000바이트 파일을 전송한다고 가정하자.
- 첫 번째 바이트는 10,001의 번호를 가지고 있다.
- 만일 각각 1.000바이트를 가지는 5 개의 세그먼트로 데이터가 전달된다면, 각 세그먼트의 순서 번호는 어떻게 되는가?

2. TCP 특징

번호화 시스템

확인응답 번호

- TCP에서 통신은 양방향으로 이루어진다.
- 연결이 설정되면 양측은 동시에 데이터를 송수신할 수 있다.
- 각 TCP는 자신이 바이트를 수신했다는 것을 확인해 주기 위하여 확인응답 변호를 이용한다.
- 확인응답 번호는 자신이 수신하기를 기대하는 다음 바이트의 번호를 나타낸다.
- 또한 확인응답 번호는 누적(cumulative) 된다.
- 여기서 누적이라는 단어의 의미는 예를 들어 수신 측에서 확인응답 번호로 5,643을 사용했다면, 처음부터 5,642까지 모든 바이트를 수신했다는 것을 의미한다.

2. TCP 특징

흐름 제어

- TCP는 UDP와 달리 흐름 제어 (flow control)를 제공한다.
- 송신 TCP는 송신 프로세스로부터 수신되는 데이터의 양을 조절하며 수신 TCP는 송신 TCP로부터 전송되는 데이터의 양을 조절한다.
- 이것은 수신 측에서 데이터가 과도하게 수신됨으로 인한 데이터의 손실을 방지하기 위한 것이다.

오류 제어

- 신뢰성 있는 서비스를 제공하기 위하여 TCP는 오류 제어 메커니즘을 구현한다.
- 오류 제어에서는 세그먼트가 (손실 또는 훼손된 세그먼트 등의) 오류 감지를 위한 데이터 단위이기는 하지만, 오류 제어는 바이트-단위로 동작한다.

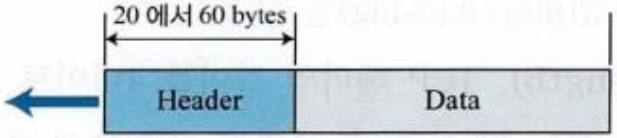
혼잡 제어

- TCP는 UDP와 다르게 네트워크의 혼잡을 고려한다.
- 송신 측에서 전송되는 데이터의 양은 (흐름 제어에 의해) 수신 측에 의해서 조절될 뿐만 아니라 네트워크의 혼잡 정도에 의해서도 결정된다.

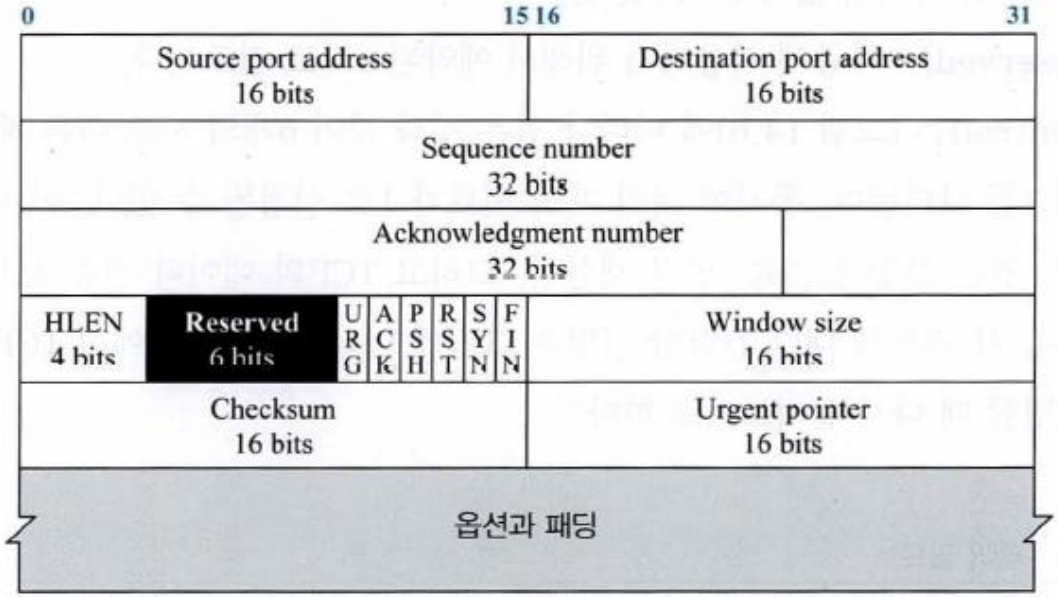
3. 세그먼트

형식

- 세그먼트는 20에서 60바이트 길이의 헤더와 응용 프로그램에서 생성되는 데이터로 구성되어 있다.
- 헤더는 옵션이 없는 경우에는 20바이트이고, 옵션을 포함하는 경우 최대 60바이트로 구성된다.



a. 세그먼트



b. 헤더

3. 세그먼트

형식

발신지 포트 주소(source port address)

- 이 필드는 세그먼트를 전송하는 호스트에 있는 응용 프로그램의 포트 번호를 지정하는 16 비트 필드이다.

목적지 포트 주소(destination port address)

- 이 필드는 세그먼트를 수신하는 호스트에 있는 응용 프로그램의 포트 변호를 지정하는 16비트 필드이다.

순서 번호(sequence number)

- 이 32 비트 필드는 세그먼트에 포함된 데이터의 첫 번째 바이트에 부여된 번호를 나타낸다.
- TCP는 신뢰성 있는 연결을 보장하기 위하여 전달되는 각 바이트마다 번호를 부여한다.
- 순서 번호는 목적지 TCP에게 세그먼트의 첫 번째 바이트가 이 번호에 해당하는 바이트라는 것을 알려준다.
- 연결 설정 단계에서 각 TCP는 난수 발생기를 이용하여 초기 순서 번호(ISN: Initial Sequence Number)를 만들며, 이때 사용되는 ISN은 각 방향에 따라 서로 다른 번호가 사용된다.

3. 세그먼트

형식

확인응답 번호(acknowledgement number)

- 이 32 비트는 세그먼트를 수신한 수신 노드가 상대방 노드로부터 수신하고자 하는 바이트의 번호를 나타낸다.
- 만일 세그먼트를 수신한 수신 노드가 상대방 노드로부터 바이트 번호 x를 성공적으로 수신했다면, 수신자는 확인응답 번호로 x + 1을 사용한다.
- 확인응답과 데이터는 함께 피기백킹 (piggybacking)될 수 있다.

헤더 길이 (header length)

- TCP 헤더의 길이를 4바이트 워드로 나타낸다.
- 헤더의 길이는 20에서 60바이트가 될 수 있다.
- 따라서 이 필드의 값은 5 (5 x 4 = 20)에서 15(15 x 4 = 60) 사이의 값이 될 수 있다.

예약(reserved)

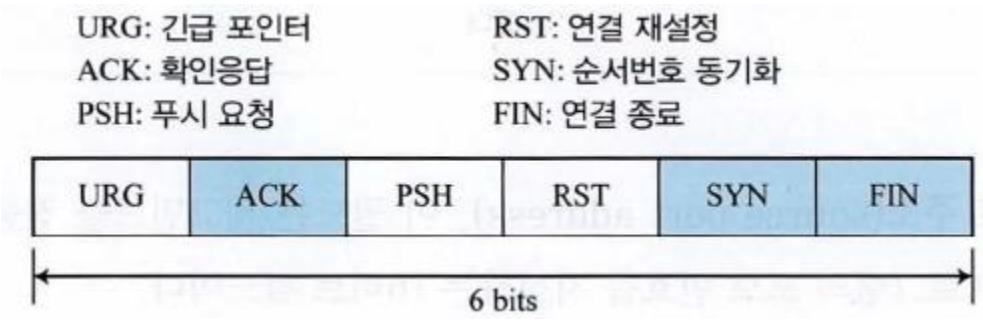
- 나중에 사용하기 위해서 예약된 6비트 필드이다.

3. 세그먼트

형식

제어 (control)

- 제어 필드는 6개의 서로 다른 제어 비트 또는 플래그를 나타낸다.
- 동시에 여러 개의 비트가 1로 설정될 수 있다.
- 이 비트들은 흐름 제어, 연결 설정과 종료, 연결 재설정, 그리고 TCP의 데이터 전송 모드를 위해서 사용된다.



창 크기 (window size)

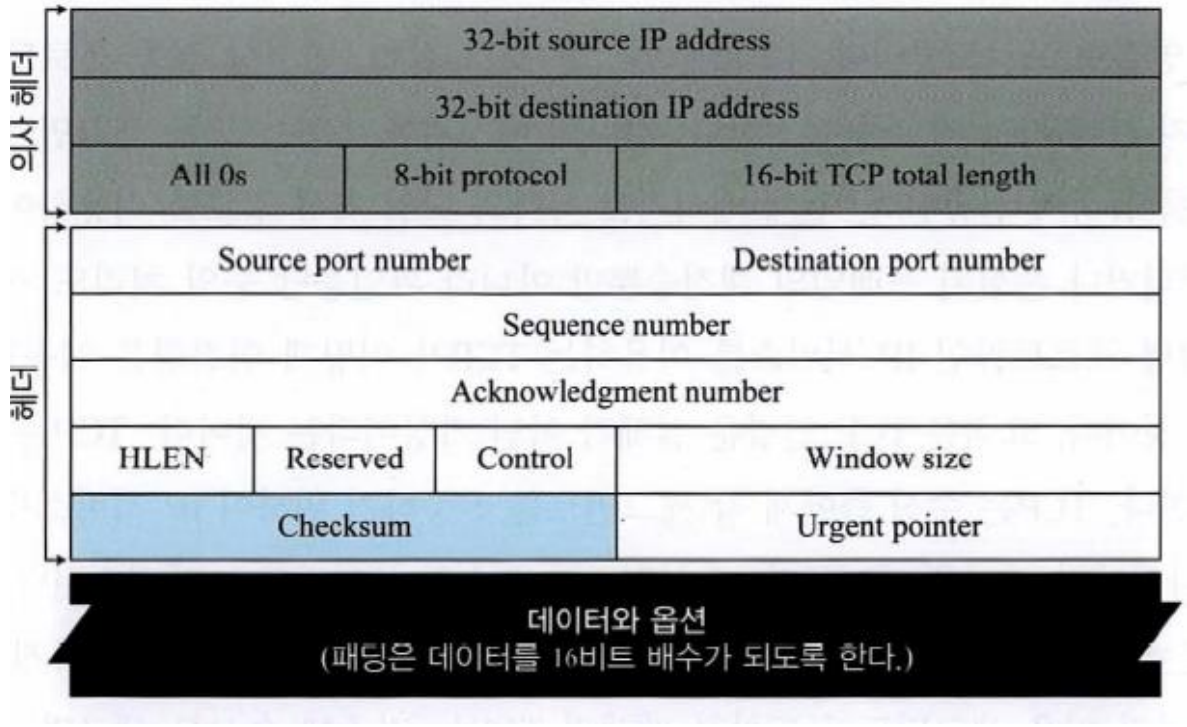
- 이 필드는 바이트 형태의 송신 TCP 창을 나타낸다.
- 이 필드의 길이가 16비트이기 때문에 창의 최대 크기는 65.535 바이트이다.
- 창 크기는 수신창(rwnd: receiving window) 이라고 하며 수신 측에 의해서 결정된다.

3. 세그먼트

형식

검사합(checksum)

- 이 16비트 필드는 검사합을 포함한다.
- TCP에서 검사합은 필수 사항이다.
- UDP와 같은 목적을 수행하기 위하여 동일한 의사 헤더가 검사합 계산을 위하여 세그먼트에 추가된다.
- TCP 의사 헤더의 프로토콜 필드의 값은 6 이다.



3. 세그먼트

형식

긴급 포인터 (urgent pointer)

- 긴급 플래그(urgent flag)가 1로 설정되어 있는 경우에만 유효한 이 16비트 필드는 세그먼트가 긴급 데이터를 포함하고 있을 때 사용된다.

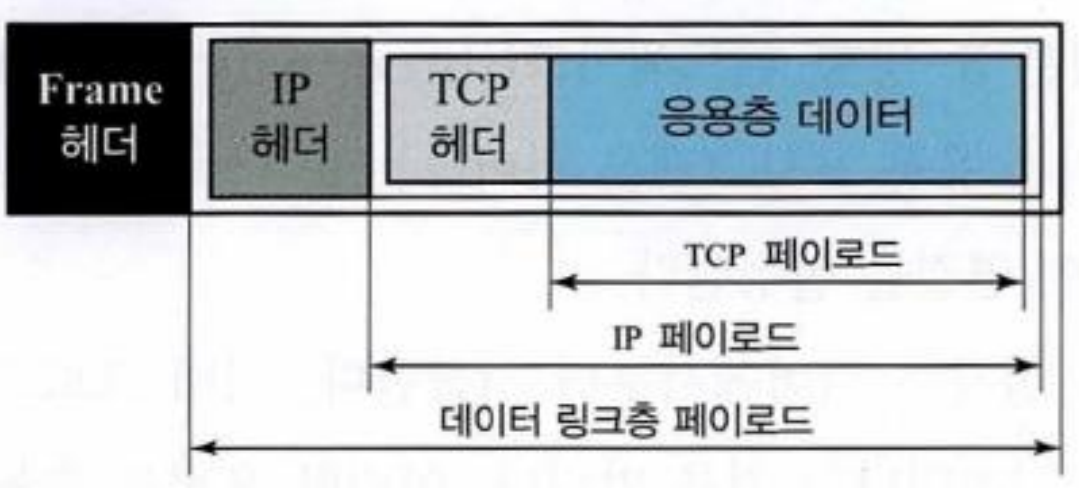
옵션(option)

- TCP는 최대 40바이트까지 옵션 정보가 있을 수 있다.

3. 세그먼트

캡슐화

- TCP 세그먼트는 응용층에서 온 데이터를 캡슐화한다.
- TCP 세그먼트는 IP 데이터그램에 캡슐화되며 IP 데이터그램은 다시 데이터 링크층의 프레임에 캡슐화된다.



4. TCP 연결

연결 설정

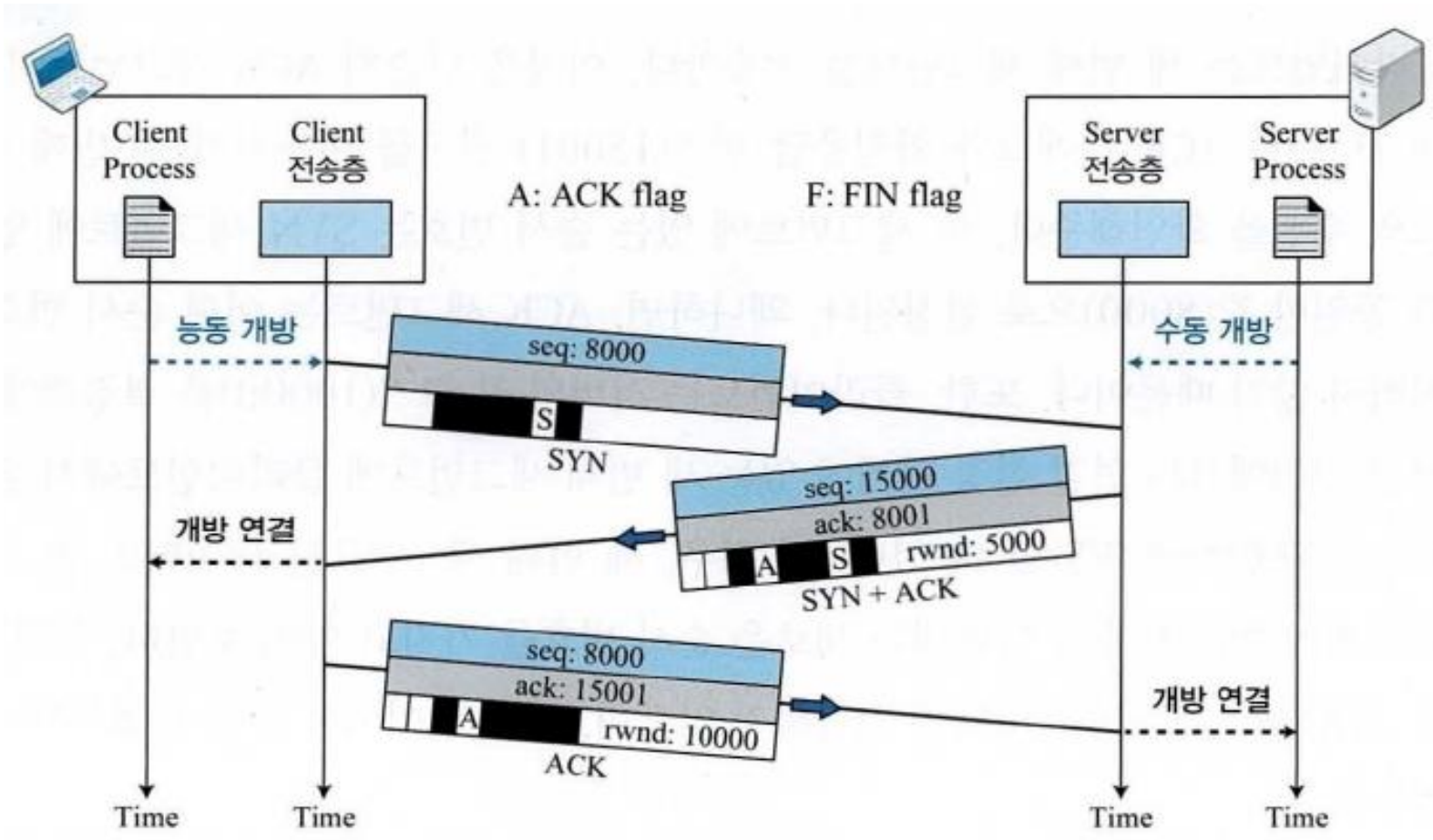
3-방향 핸드셰이크

- TCP에서 연결 설정은 3-방향 핸드셰이크(three-way handshaking) 라고 한다.
- 3-방향 핸드셰이크 절차는 서버에서부터 시작한다.
- 서버 프로그램은 자신의 TCP에게 연결을 수락할 준비가 되어있다는 것을 알린다.
- 이러한 요청을 수동 개방(passive open)이라고 한다.
- 클라이언트 프로그램은 능동 개방(active open)을 위한 요청을 실행한다.
- 수동 개방이 되어 있는 서버와 연결을 설정하고자 하는 클라이언트는 자신의 TCP에게 특정 서버와 연결을 설정한다는 것을 알린다.

4. TCP 연결

연결 설정

- 3-방향 핸드셰이크



4. TCP 연결

연결 설정

동시 개방

- 두 프로세스가 동시에 서로에게 능동 개방을 요구하는 상황이 일어날 수도 있다.
- 이 경우, 양쪽 TCP는 서로에게 SYN + ACK 세그먼트를 전송하게 되고 하나의 단일 연결이 두 TCP 사이에 설정된다.

SYN 플러딩 공격

- TCP의 연결 설정 과정은 SYN 플러딩 공격 (SYN flooding attack) 이라는 중요한 보안 문제에 노출되어 있다.
- 이 공격은 악의에 찬 공격자가 데이터그램의 발신지 IP 주소를 위조하여 서로 다른 클라이언트로 가장한 후에 많은 수의 SYN 세그먼트를 하나의 서버에 전송하는 경우에 발생한다.
- 이러한 SYN 플러딩 공격은 서비스 거부 공격 (denial of service attack) 이라는 보안 공격의 일종이며, 공격자가 시스템에서 많은 서비스 요구를 전송함으로써 시스템에 과부하가 걸리고 따라서 유효한 요청에 대해서 서비스를 거부하도록 함으로써 시스템을 독점하는 것이다.

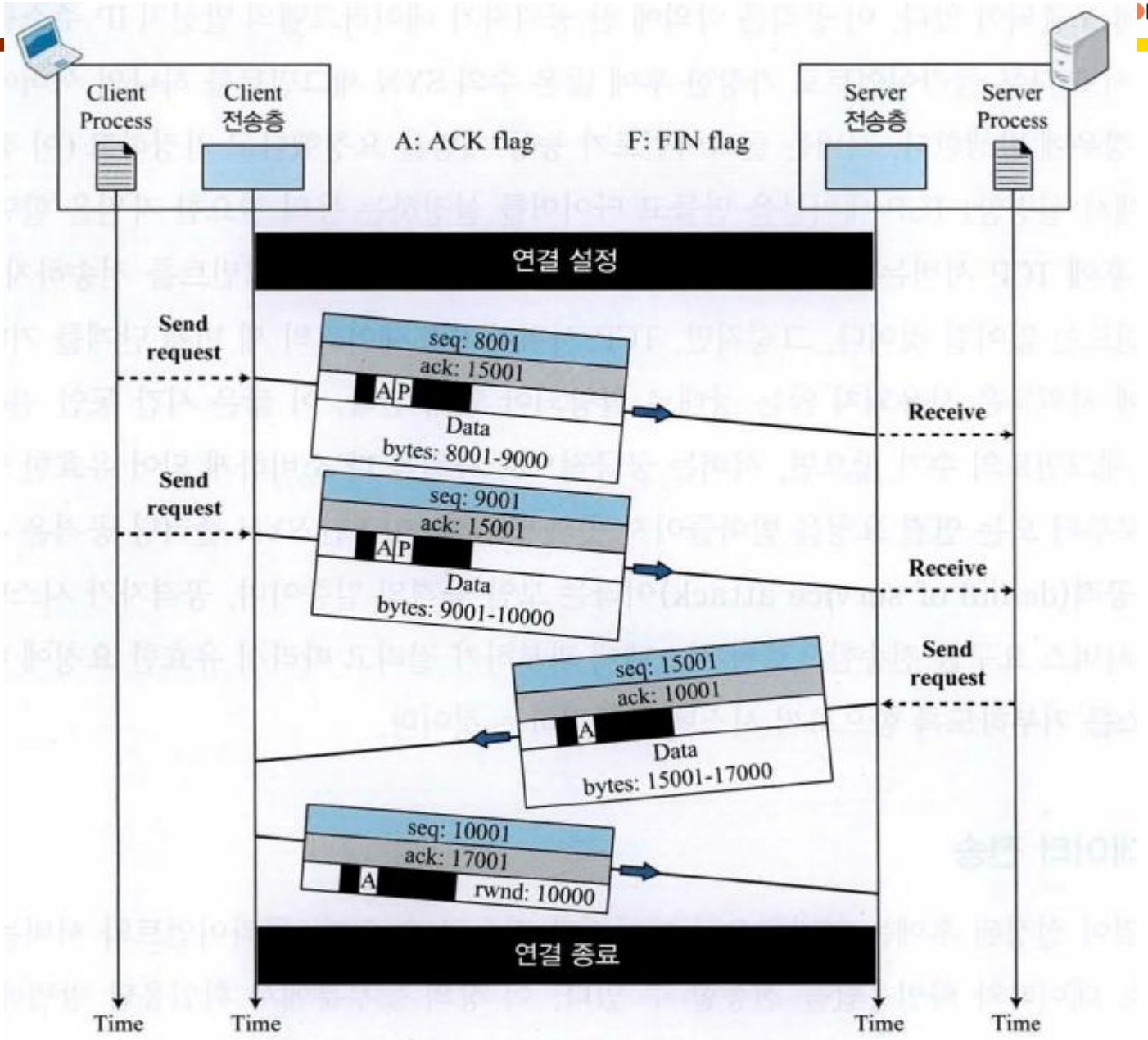
4. TCP 연결

데이터 전송

- 연결이 설정된 후에는 양방향으로 데이터가 전송될 수 있다.
- 클라이언트와 서버는 양방향으로 데이터와 확인응답을 전송할 수 있다.
- 동일한 방향으로 전송되는 데이터와 확인응답은 하나의 세그먼트로 전달될 수 있다.
- 즉, 확인응답은 데이터와 피기백 (piggyback) 된다.

4. TCP 연결

- 데이터 전송



4. TCP 연결

데이터 전송

푸시 데이터

- 송신 TCP는 송신 응용 프로그램에서 오는 데이터 스트림을 저장하기 위하여 버퍼를 이용한다.
- 송신 TCP는 세그먼트의 크기를 선택할 수 있다.
- 수신 TCP 역시 수신한 데이터를 저장하기 위한 버퍼를 가지고 있으며 응용 프로그램이 수신할 준비가 되어 있을 때 데이터를 응용 프로그램으로 전달한다.
- 이러한 유형의 신축성이 TCP의 효율을 향상시킨다.
- 그러나 때때로 이러한 신축성이 편리하지 않은 응용 프로그램이 있다.
- 이러한 응용 프로그램에서는 (송신 TCP에서 네트워크로) 데이터의 전송을 지연하거나 (수신 TCP에서 수신 응용 프로그램으로) 데이터의 전달을 지연하는 것은 받아들일 수 없을 것이다.
- TCP 에서는 이러한 상황을 처리할 수 있다.
- 전송 측에 있는 응용 프로그램은 푸시(push) 동작을 요구할 수 있다.
- 이것은 송신 TCP가 창이 다 찰 때까지 기다리지 않는다는 것을 의미한다.
- 송신 TCP는 세그먼트를 만들어서 즉시 전송해야 한다.
- 송신 TCP는 또한 푸시 비트를 1 로 설정하여 수신 TCP에게 세그먼트가 가능한 한 빨리 수신 응용 프로그램으로 전달해야 하는 데이터를 포함하고 있다는 것을 알려준다.

4. TCP 연결

데이터 전송

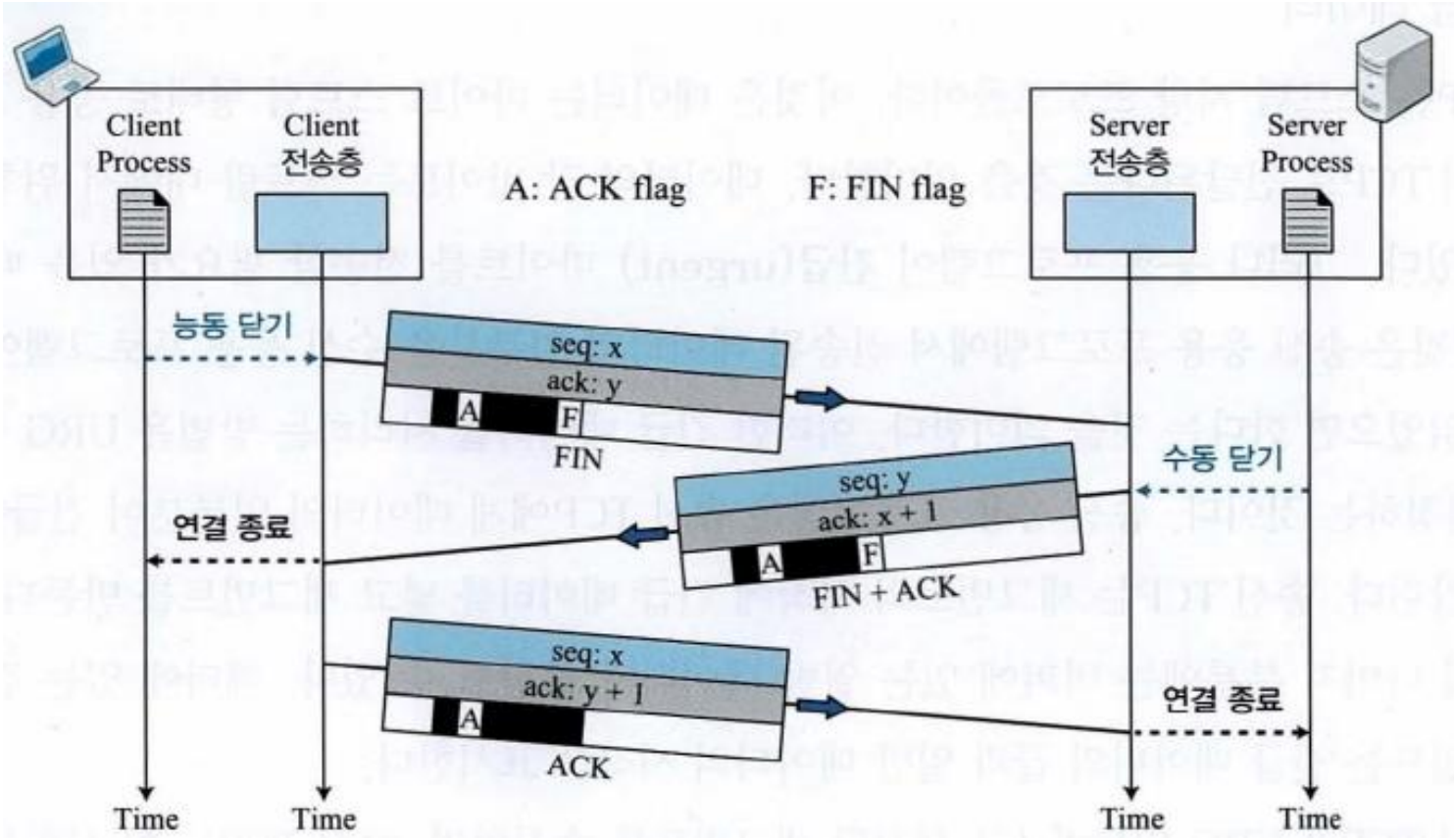
긴급 데이터

- TCP는 스트림 지향 프로토콜이다.
- 이것은 데이터는 바이트 스트림 형태로 응용 프로그램에서 TCP로 전달된다는 것을 의미한다.
- 데이터의 각 바이트는 스트림 내에서 위치를 가지고 있다.
- 그러나 응용 프로그램이 긴급(urgent) 바이트를 전송할 필요가 있을 때가 있다.
- 이러한 긴급 데이터를 처리하는 방법은 URG 비트를 1로 설정하는 것이다.
- 송신 응용 프로그램은 송신 TCP에게 데이터의 일부분이 긴급하다는 것을 알린다.
- 송신 TCP는 세그먼트의 시작에 긴급 데이터를 넣고 세그먼트를 만든다.
- 수신 TCP가 URG 비트가 1로 설정된 세그먼트를 수신하면 수신 TCP는 이 상황을 수신 응용에게 알린다.
- 수신 측에서 어떻게 처리할 것인가는 운영체제의 일이다.

4. TCP 연결

연결 종료

- 데이터를 교환하는 (클라이언트와 서버의) 어느 쪽도 연결을 종료할 수 있지만, 일반적으로는 클라이언트에서 종료를 시작한다.
- 3-방향 핸드셰이크



4. TCP 연결

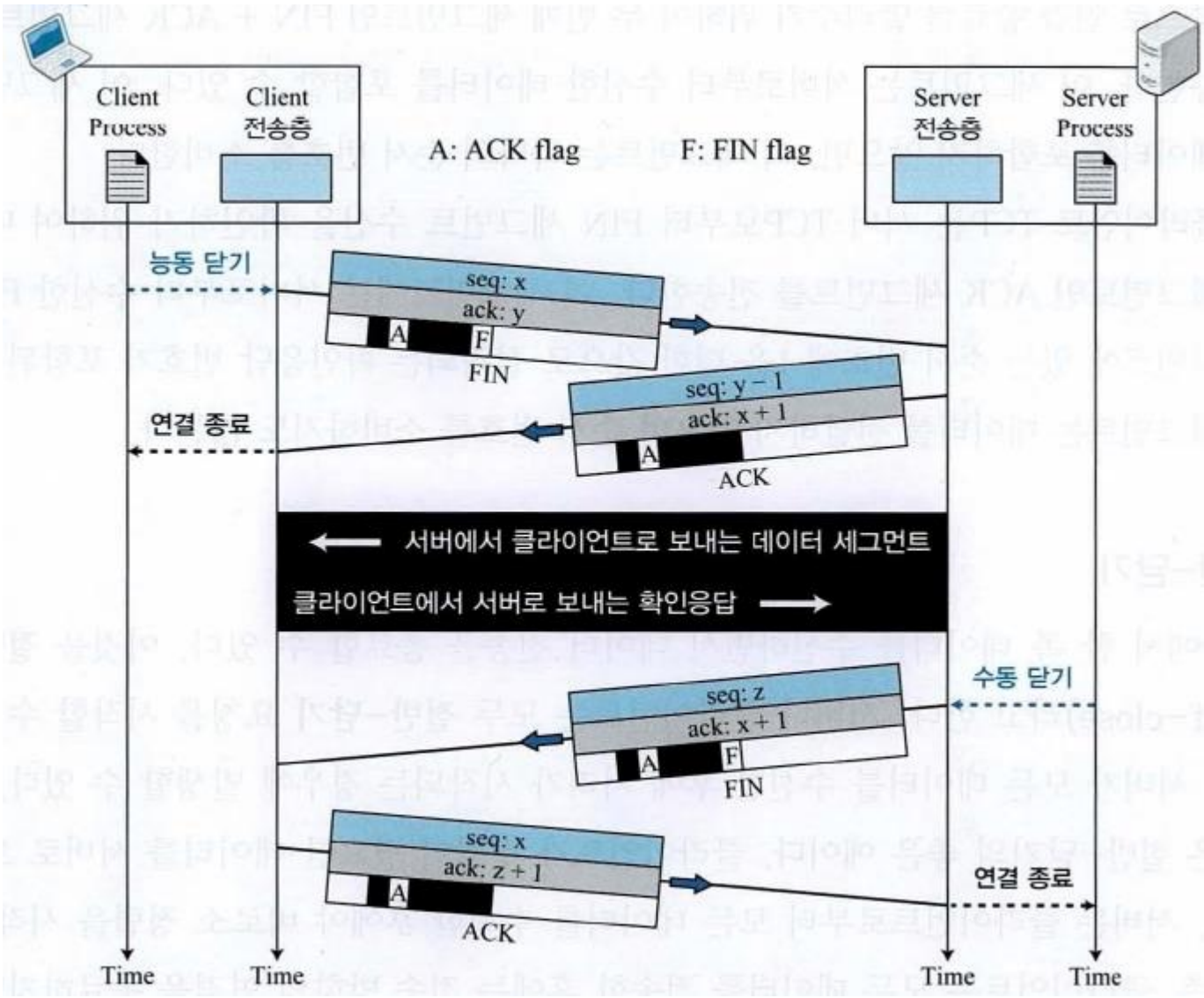
연결 종료

절반-닫기

- TCP에서 한 쪽 데이터를 수신하면서 데이터 전송을 종료할 수 있다.
- 이것을 절반-닫기 (half-close) 라고 한다.
- 서버나 클라이언트는 모두 절반-닫기 요청을 시작할 수 있다.
- 이것은 서버가 모든 데이터를 수신한 후에 처리가 시작되는 경우에 발생할 수 있다.

4. TCP 연결

- 연결 종료
  - 절반-닫기



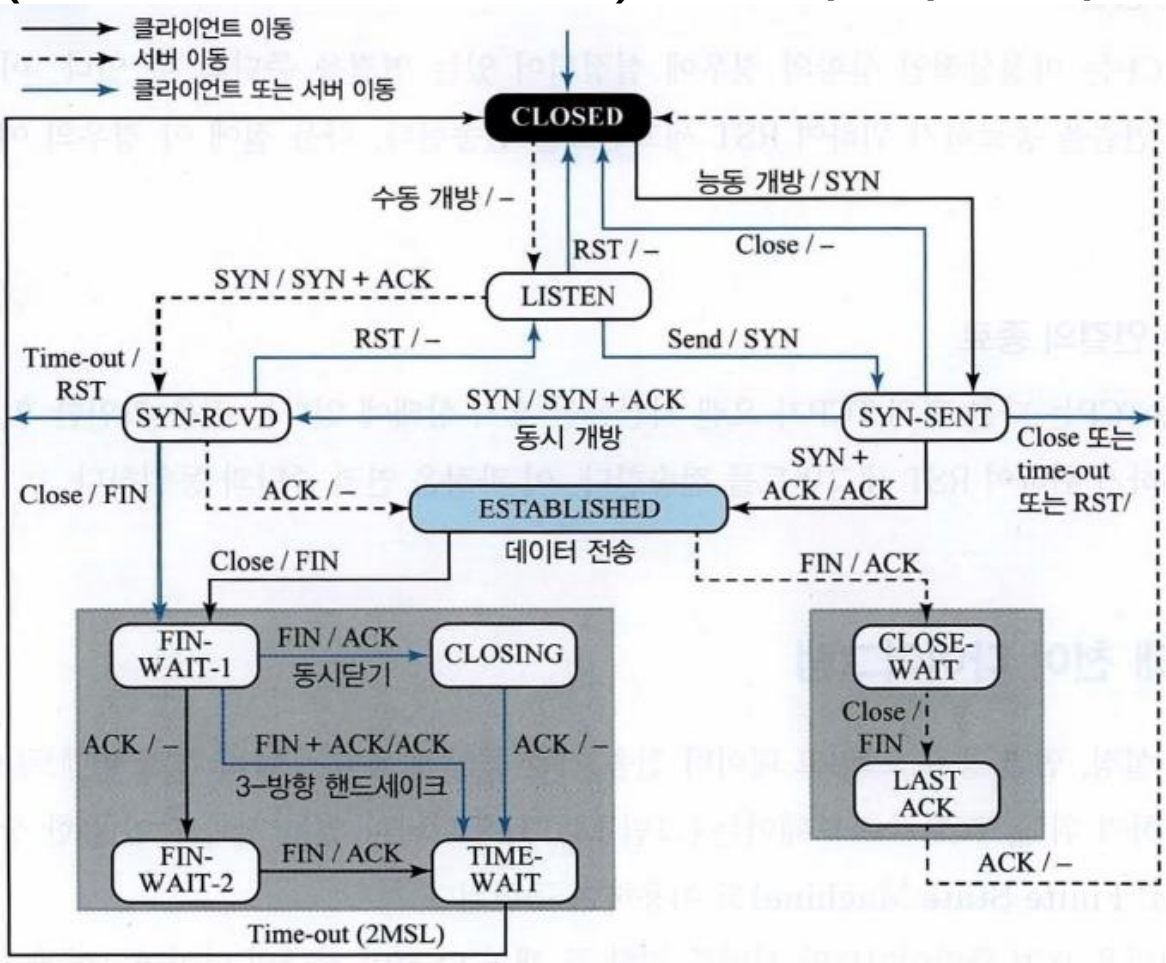
4. TCP 연결

연결 재설정

- **한쪽 편에 있는 TCP는 RST(reset) 비트를 1로 설정함으로써 연결 요청을 거절하거나, 연결을 중단하거나, 휴지 상태에 있는 연결을 종료할 수 있다.**
- **연결 거절**
  - 한쪽의 TCP가 존재하지 않은 포트로 연결을 요청하는 경우를 가정해보자.
  - 다른 쪽의 TCP는 요청을 거절하기 위하여 RST 비트를 설정한 세그먼트를 전송한다.
- **연결 중단**
  - 한 TCP는 비정상적인 상황의 경우에 설정되어 있는 연결을 중단할 수 있다.
  - 이 경우 TCP는 연결을 종료하기 위하여 RST 세그먼트를 전송한다.
- **휴지 연결의 종료**
  - 한 쪽 TCP는 다른 편의 TCP가 오랜 시간동안 휴지 상태에 있다는 것을 확인한 후, 연결을 해제하기 위하여 RST 세그먼트를 전송한다.
  - 이 과정은 연결 중단과 동일하다.

5. 상태 천이 다이어그램

- 연결 설정, 연결 종료, 그리고 데이터 전송 기간 동안 발생하는 여러 가지 이벤트(event)를 관리하기 위해, TCP 소프트웨어는 그림에 나타나 있는 것과 같이 유한 상태 기기 (FSM: Finite State Machine)를 이용하여 구현된다.



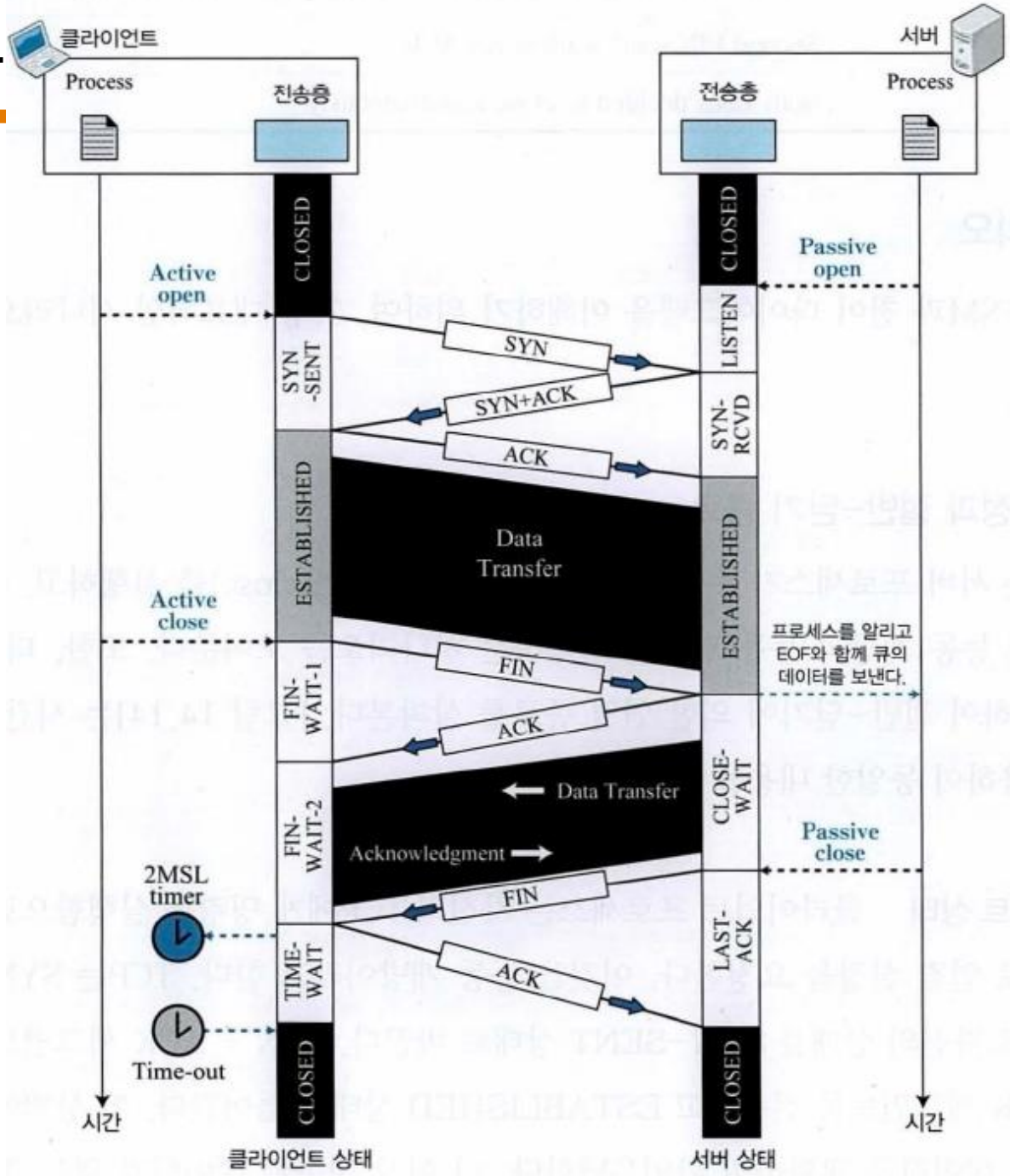
5. 상태 천이 다이어그램

TCP 상태

상태	설명
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN+ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN received ; waiting for second FIN
CLOSE-WAIT	First FIN received, ACK sent; waiting for application to close
TIME-WAIT	Second FIN received, ACK sent; waiting for 2MSL time-out
LAST-ACK	Second FIN sent; waiting for ACK
CLOSING	Both sides decided to close simultaneously

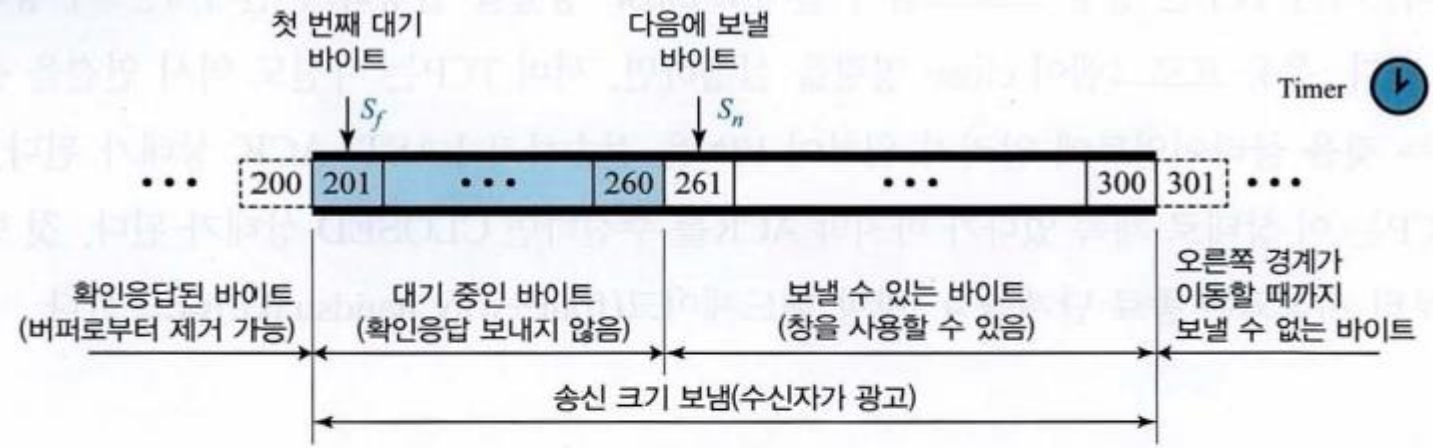
5. 상태 천이 다이어그

- 시나리오
  - 연결 설정과 절반-닫기 종료



6. TCP 창

■ 송신 창


첫 번째 대기 바이트
다음에 보낼 바이트
Timer
S_f
S_n
200
201
260
261
300
301
확인응답된 바이트 (버퍼로부터 제거 가능)
대기 중인 바이트 (확인응답 보내지 않음)
보낼 수 있는 바이트 (창을 사용할 수 있음)
오른쪽 경계가 이동할 때까지 보낼 수 없는 바이트
송신 크기 보냄(수신자가 광고)
a. 송신창
왼쪽 벽
닫기
축소됨
오른쪽 벽
열기
b. 열기, 닫기, 축소 송신창

6. TCP 창

송신 창

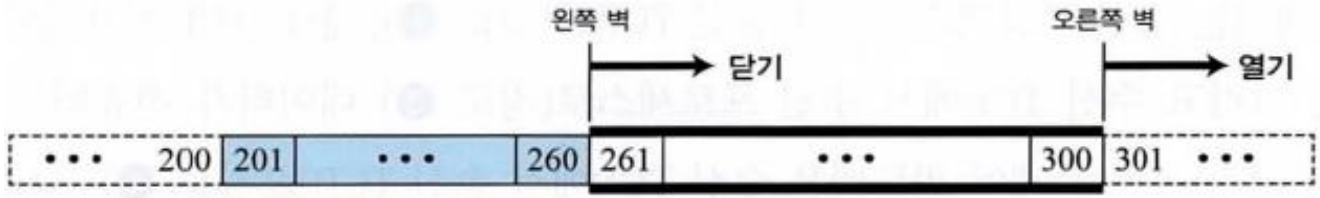
- **TCP의 송신 창은 선택적 반복 프로토콜에서 사용하는 것과 유사하지만, 다음과 같은 차이점이 있다.**

1. **하나의 차이점은 창과 관련된 개체 자체의 차이이다.**
   - 선택적 반복에서 창은 패킷의 번호를 나타내지만, TCP 창은 바이트의 변호를 나타낸다.
   - 비록 TCP에서는 세그먼트 단위의 전송이 이루어지지만 창을 제어하는 변수는 바이트로 표현된다.
2. **두 번째 차이점은 어떤 구현에서 TCP는 프로세스로부터 데이터를 수신하고 추후에 이를 전송하지만, 여기서는 송신 TCP가 프로세스로부터 데이터를 수신하자마자 데이터에 대한 세그먼트를 전송할 수 있다고 가정하였다.**
3. **또 다른 차이점은 타이머의 개수이다.**
   - 이론적으로 선택적 반복은 전송되는 각 패킷마다 타이머를 사용하지만 TCP 프로토콜은 단지 하나의 타이머만 사용한다.

6. TCP 창

■ 수신 창

a. 수신 창과 할당된 버퍼



b. 수신 창 열기, 닫기

6. TCP 창

수신 창

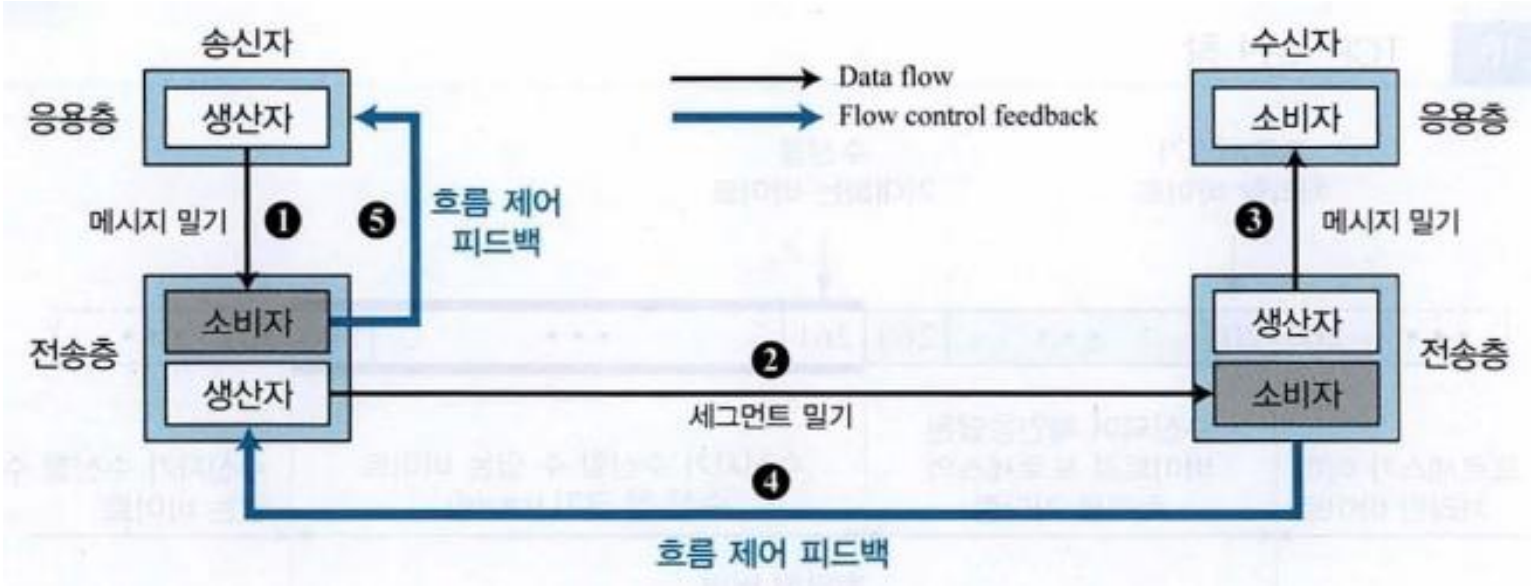
- **TCP 수신 창과 선택적 반복에서 사용되는 창과는 다음 두 가지 차이점이 있다.**

1. **첫 번째 차이점은 TCP에서 응용 프로세스가 자신의 속도로 데이터를 읽어갈 수 있다.**
   - 이것은 수신 측에 할당된 버퍼의 일부분은 수신되고 확인응답된 데이터로 채워져 있지만, 수신 프로세스에서 읽어가기 전까지는 이러한 데이터는 버퍼에 저장되어 있어야 한다는 것을 의미한다.
   - 그림에 나타나 있는 것과 같이 수신 창의 크기는 항상 버퍼의 크기보다 작거나 같다.
   - 수신 창의 크기는 수신 창이 송신 측으로부터 넘치지 않고 수신할 수 있는 바이트의 개수를 결정한다(흐름 제어).

2. **두 번째 차이점은 TCP 프로토콜에서 사용되는 확인응답 방법이다.**
   - 선택적 반복에서의 확인응답은 선택적이며 따라서 훼손되지 않고 수신된 패킷만을 정의한다는 것을 기억할 것이다.
   - TCP에서 주된 확인응답 메커니즘은 수신하기를 기대하는 다음 바이트를 알려주는 누적 확인응답 방법이다.

7. 흐름 제어

- 흐름 제어 (flow control)는 생산자가 데이터를 생성하는 속도와 소비자가 데이터를 처리하는 속도의 균형을 맞추는 것이다.



7. 흐름 제어

창열기와 닫기

- 비록 연결이 설정될 때 양쪽 끝에서 버퍼의 크기는 고정되지만 흐름 제어를 수행하기 위하여 TCP는 송신 측과 수신 측으로 하여금 자신의 창의 크기를 조정하도록 한다.
- 송신 측으로부터 여러 바이트가 들어오면 수신 창은 닫힌다.
- 또한, 수신 프로세스가 여러 바이트를 읽으면 수신 창은 열린다.
- 여기서는 축소(오른쪽 벽이 왼쪽으로 이동)되는 경우는 고려하지 않는다.
- 송신 창의 열기, 닫기, 그리고 축소는 수신 측에 의해서 조절된다.
- 송신 창은 새로운 확인응답이 도착하는 경우에는 닫힌다.
- 송신 창은 수신 측으로부터 광고되는 수신 창의 크기 (rwnd)에 의해서 열린다.

창 축소

- 수신 창은 축소될 수 없다.
- 그렇지만 수신 측에서 창의 축소를 야기하는 rwnd 값을 통보하는 경우에 송신 창은 축소될 수 있다.

7. 흐름 제어

창 폐쇄

- 수신 측은 rwnd 값을 0으로 해서 짧은 시간 동안 창을 폐쇄할 수 있다.
- 이것은 수신 측에서 잠시 송신 측으로부터 데이터를 수신하고 싶지 않을 때 발생한다.
- 이 경우 송신 측은 실제로는 창의 크기를 축소하지는 않지만 새로운 광고가 도착하기 전까지는 데이터 전송을 중단한다.
- 수신 측의 명령에 의해 창이 폐쇄되었지만, 송신 측에서는 한 바이트의 데이터를 포함하는 세그먼트를 언제든지 보낼 수 있다.
- 이것은 프로빙(probing)이라고 하며, 교착상태를 방지하기 위해서 사용된다.

7. 흐름 제어

- 어리석은 창 신드롬
  - 미닫이 창(sliding window) 운영에서는 전송 응용 프로그램이 데이터를 천천히 발생하거나, 또는 수신 응용 프로그램이 데이터를 천천히 소비하는 경우에 심각한 문제가 발생한다.
  - 이 경우에는 아주 적은 수의 데이터를 포함하는 세그먼트의 전송으로 인하여 운영의 효율이 감소하게 된다.
  - 이러한 비효율성은 데이터 링크층과 물리층의 오버헤드를 고려한다면 더욱 나빠질 것이다.
  - 이러한 문제를 어리석은 창 신드롬(silly window syndrome) 이라고 한다.

7. 흐름 제어

어리석은 창 신드롬

송신 측에서 발생하는 신드롬

- 송신 TCP가 한 번에 한 바이트씩 데이터를 천천히 생성하는 응용 프로그램을 다루는 경우에 어리석은 창 신드롬이 발생한다.
- 이러한 문제를 해결하는 한 가지 방법은 송신 TCP가 한 바이트 단위의 데이터를 전송하지 못하도록 하는 것이다.
- 전송 TCP는 데이터를 취합하여 가능한 한 큰 블록으로 데이터를 전송하도록 하는 것이다.
- 이 경우에는 충분한 데이터가 버퍼에 쌓이도록 전송 TCP가 일정 시간 기다려야 하는데, 얼마만큼 전송 TCP가 기다려야 하는데 만일 이 기간이 너무 길면 프로세스에서 지연이 생기게 된다.
- 만일 이 기간이 충분히 길지 않으면 역시 적은 데이터를 포함하는 세그먼트가 전송될 것이다.
- Nagle 알고리즘
  1. 송신 TCP는 단지 한 바이트라 하더라도 송신 응용 프로그램으로부터 수신한 첫 데이터를 세그먼트로 만든 후 전송한다.
  2. 첫 번째 세그먼트를 전송한 후에 , 송신 TCP은 수신 TCP로부터 확인응답을 수신하거나 또는 최대 크기의 세그먼트를 구성할 수 있을 정도로 충분한 데이터가 버퍼에 저장되기 전까지는 데이터를 출력 버퍼에 저장한다.
  3. 나머지 전송 기간 동안 2번째 단계를 반복한다.

어리석은 창 신드롬

수신 측에서 발생하는 신드롬

- Clark 해결 방법
  - 두 번째 방법은 확인응답의 전송을 지연하는 것이다.
  - 즉 세그먼트는 도착하더라도 즉시 확인 응답되지 않는다.
  - 지연된 확인응답(delayed acknowledgement)은 송신 TCP로 하여금 자신의 창을 진행하지 못하도록 한다.
  - 확인응답의 지연을 이용하면 다음과 같은 또 다른 장점을 얻을 수 있다.
  - 수신 측에서는 각 세그먼트에 대해서 확인응답을 전송할 필요가 없기 때문에, 수신 측에서 발생하는 트래픽이 감소된다.
  - 반면, 확인응답의 지연은 송신 측에서 확인응답을 받지 못한 세그먼트를 재전송할 가능성이 있기 때문에 단점도 있다.
  - TCP 프로토콜은 이러한 장점과 단점의 균형을 맞춘다.
  - 즉 TCP에서는 확인응답이 0.5초 이상 지연되지 않도록 규정되어 있다.

7. 흐름 제어

어리석은 창 신드롬

수신 측에서 발생하는 신드롬

- 수신 TCP가 한 번에 한 바이트씩 천천히 데이터를 소비하는 응용 프로그램에 서비스를 제공하는 경우에는 수신 TCP에 어리석은 창 신드롬 현상이 발생하게 된다.
- 송신 응용 프로그램이 4KB 블록의 데이터를 생성하지만, 수신 응용 프로그램이 한 번에 한 바이트씩 소비한다면 수신 버퍼는 모두 데이터로 차게 된다.
- 수신 응용 프로그램이 수신 TCP의 입력 버퍼로부터 첫 번째 바이트의 데이터를 읽으면 입력 버퍼에 한 바이트 정도의 공간이 생기게 된다.
- 수신 TCP는 한 바이트의 창 크기를 송신 TCP로 통보하며 데이터의 전송을 기다리던 송신 TCP는 이 통보를 좋은 소식이라고 간주하고 이후 단지 한 바이트의 데이터를 포함하는 세그먼트를 계속 전송한다.
- 따라서 효율성의 문제와 어리석은 창 신드롬 문제가 다시 발생하게 된다.
- 도착하는 속도보다 천천히 데이터를 소비하는 응용 프로그램에 의해 생기는 어리석은 창 신드롬 현상을 예방하기 위해서 다음의 두 가지 방법이 제시되었다.
- Clark 해결 방법
  - Clark 해결 방법은 데이터가 도착하자마자 확인응답을 전송하지만, 수신 버퍼에 최대 크기의 세그먼트를 수용할 수 있는 충분한 공간이 있거나 또는 적어도 수신 버퍼가 반 이상 비어 있기 전까지는 창의 크기를 0으로 통보하는 것이다.

8. 오류 제어

- TCP는 오류 제어를 이용하여 신뢰성을 제공한다.
- 오류 제어는 훼손 세그먼트의 감지 및 재전송, 손실 세그먼트의 재전송, 분실된 세그먼트가 도착하기 전까지 순서가 어긋난 세그먼트를 저장, 그리고 중복 세그먼트의 감지 및 폐기를 위한 메커니즘을 포함한다.
- TCP에서 오류 제어는 세 가지 간단한 도구인 검사합, 확인응답, 그리고 타임-이웃 등을 통하여 수행된다.

- 검사합
  - 각 세그먼트에는 검사합(checksum) 필드가 있으며, 이 필드는 세그먼트가 훼손되었는지를 검사하기 위하여 사용된다.
  - 만일 세그먼트의 확인한 검사합이 다르면, 목적지 TCP는 세그먼트가 손상된 것으로 간주하고 세그먼트를 폐기한다.
  - TCP는 모든 세그먼트에 필수사항인 16비트 검사합을 이용한다.

8. 오류 제어

확인응답

- **데이터를 포함하지는 않지만 하나의 순서 번호를 소비하는 제어 세그먼트도 역시 확인응답된다.**
- **ACK 세그먼트는 확인응답되지 않는다.**

- **확인응답 유형**
  - 누적 확인응답(ACK)
    - TCP는 원래 세그먼트의 수신을 누적해서 확인응답할 수 있도록 설계되었다.
    - 수신 측은 순서에 어긋나게 도착하고 저장된 모든 세그먼트는 무시하고 수신하고자 하는 다음 바이트를 알린다.
    - TCP 헤더에 있는 32 비트의 ACK 필드가 누적 확인응답(ACK: Accumulative Acknowledgment)을 위하여 사용되며 , 이 필드는 ACK 플래그 비트가 1 로 설정된 경우에만 유효하다.
  - 선택적 확인응답(SACK)
    - SACK는 ACK를 대치하는 것이 아니라 송신 측에 부가 정보를 알려주기 위해서 사용된다.
    - SACK는 순서에 어긋나게 들어온 데이터 블록과 중복 세그먼트 블록을 알려준다.
    - 그렇지만, TCP 헤더에는 이러한 유형의 정보를 추가할 수 있는 여분의 공간이 없어서 SACK는 TCP 헤더의 끝에 옵션의 형태로 구현된다.

확인응답

확인응답의 전송

1. 한 쪽에서 다른 쪽으로 데이터 세그먼트를 전송할 경우에는 수신하고자 하는 다음 순서 번호인 확인응답 번호(피기백킹)가 세그먼트에 포함되어야 한다.
   - 이 규칙은 필요한 세그먼트의 수를 줄임으로써 트래픽을 감소시킨다.

2. 수신 측에서 더 이상 보낼 데이터가 없고 (기대한 순서 번호를 가진) 순서에 맞는 세그먼트를 수신했으며 또한 이전에 수신한 세그먼트가 이미 확인응답된 경우에는, 수신 측에서는 또 다른 세그먼트가 도착하거나 또는 시간이 지나기 전(보통 500ms)까지는 ACK 세그먼트의 전송을 보류한다.
   - 이 규칙은 ACK 세그먼트 전송으로 인하여 별도의 트래픽이 발생하는 것을 방지하기 위한 것이다.

3. 수신 측에서 기대한 순서 번호를 가지는 세그먼트가 도착하고 또한 이전에 수신한 순서에 맞는 세그먼트가 아직까지 확인응답되지 않았다면, 수신 측은 즉시 ACK 세그먼트를 전송한다.
   - 이것은 망의 혼잡을 야기할 수 있는 세그먼트의 불필요한 재전송을 방지한다.

확인응답

확인응답의 전송

4. 기대한 것보다 더 큰 값을 가진 (즉, 순서에 어긋난) 순서 번호를 갖는 세그먼트가 도착하면, 수신 측은 ACK 세그먼트를 즉시 전송함으로써 자신이 수신하기를 기대하는 세그먼트의 순서 번호를 알린다.
   - 이것은 뒤에서 살펴볼 시나리오인 누락된 세그먼트의 빠른 재전송을 위한 것이다.

5. 누락된 세그먼트가 도착하면, 수신 측은 수신하고자 하는 다음 순서 번호를 알리기 위하여 ACK 세그먼트를 전송한다.
   - 이것은 누락으로 간주된 세그먼트를 수신하였다는 것을 알려주기 위하여 사용된다.

6. 중복 세그먼트가 도착하면, 수신 측은 세그먼트를 폐기하고, 수신하고자 하는 순서에 맞는 다음 세그먼트를 알리기 위하여 즉시 확인응답을 전송한다.
   - 이것은 ACK 세그먼트 자체가 손실되는 등의 문제를 해결한다

8. 오류 제어

재전송

- **전송된 세그먼트는 확인응답되기 전까지는 버퍼에 저장된다.**
- **재전송 타이머가 만료되거나 또는 송신 측에서 버퍼에 있는 첫 번째 세그먼트에 대한 3 개의 중복 ACK를 수신하는 경우에는 세그먼트가 재전송된다.**

- **RTO 이후의 재전송**
  - 송신 TCP는 각각의 연결마다 하나의 재전송 타임-아웃(RTO: Retransmission Time-Out) 타이머를 구동한다.
  - 타이머가 만료되어 타임-아웃이 발생하면, TCP는 버퍼 앞에 있는 세그먼트(즉, 가장 작은 순서 번호를 갖는 세그먼트)를 전송하고 타이머를 재구동한다.
  - TCP에서 RTO의 값은 가변적이며 세그먼트의 왕복 시간(RTT: Round Trip Time)을 기반으로 업데이트된다 RTT는 세그먼트를 목적지로 전송하고 그 세그먼트에 대한 확인응답을 수신하는데 걸리는 시간을 나타낸다.

8. 오류 제어

재전송

세 개의 중복 ACK 세그먼트 이후에 재전송

- RTO 값이 크지 않은 경우에는 앞에서 언급한 규칙을 이용하여 세그먼트를 재전송해도 충분하다.
- 송신측에서 타임-아웃을 기다리는 것보다 좀 더 빨리 재전송하도록 함으로써 처리율을 향상시키기 위하여 근래에 구현된 대부분의 TCP는 세 개의 중복 ACK 규칙을 따르고 또한 손실로 간주된 세그먼트를 즉시 재전송한다.
- 이러한 특징을 빠른 재전송이라고 하며 이러한 특징을 이용하는 TCP를 "Reno"라고 한다.
- 이러한 TCP에서 만일 하나의 세그먼트에 대한 세 개의 확인응답(즉, 원래의 ACK와 세 개의 정확하게 일치하는 사본)이 수신되면 타임-아웃을 기다리지 않고 다음 세그먼트가 즉시 재전송된다.

8. 오류 제어

- 순서가 어긋난 세그먼트
  - 현재의 TCP에서는 순서에 어긋나게 들어오는 세그먼트들을 버리지 않고, 일시적으로 그 세그먼트들을 저장하며, 손실된 세그먼트가 도착하기 전까지는 이 세그먼트들을 순서가 어긋난 세그먼트로 표시한다.
  - 그렇지만 순서가 어긋난 세그먼트들을 프로세스로 전달하지는 않는다.
  - TCP는 데이터가 순서에 맞게 프로세스에 전달되도록 한다.

9. 혼잡 제어

혼잡 창

- 수신자가 유일하게 송신자의 창 크기를 결정할 수 있다.
- 그렇지만 이것은 실제로 또 다른 개체인 네트워크를 전혀 고려하지 않았다.
- 만일 네트워크가 송신 측에서 발생하는 속도보다 늦게 데이터를 전달한다면, 네트워크는 송신 측으로 하여금 데이터 생성 속도를 늦추도록 해야 한다.
- 송신자는 수신자에 의해서 광고되는 창 크기와 혼잡 창 크기 정보를 가지고 있으며, 창의 실제 크기는 이 두 정보 중에 최솟값이다.

9. 혼잡 제어

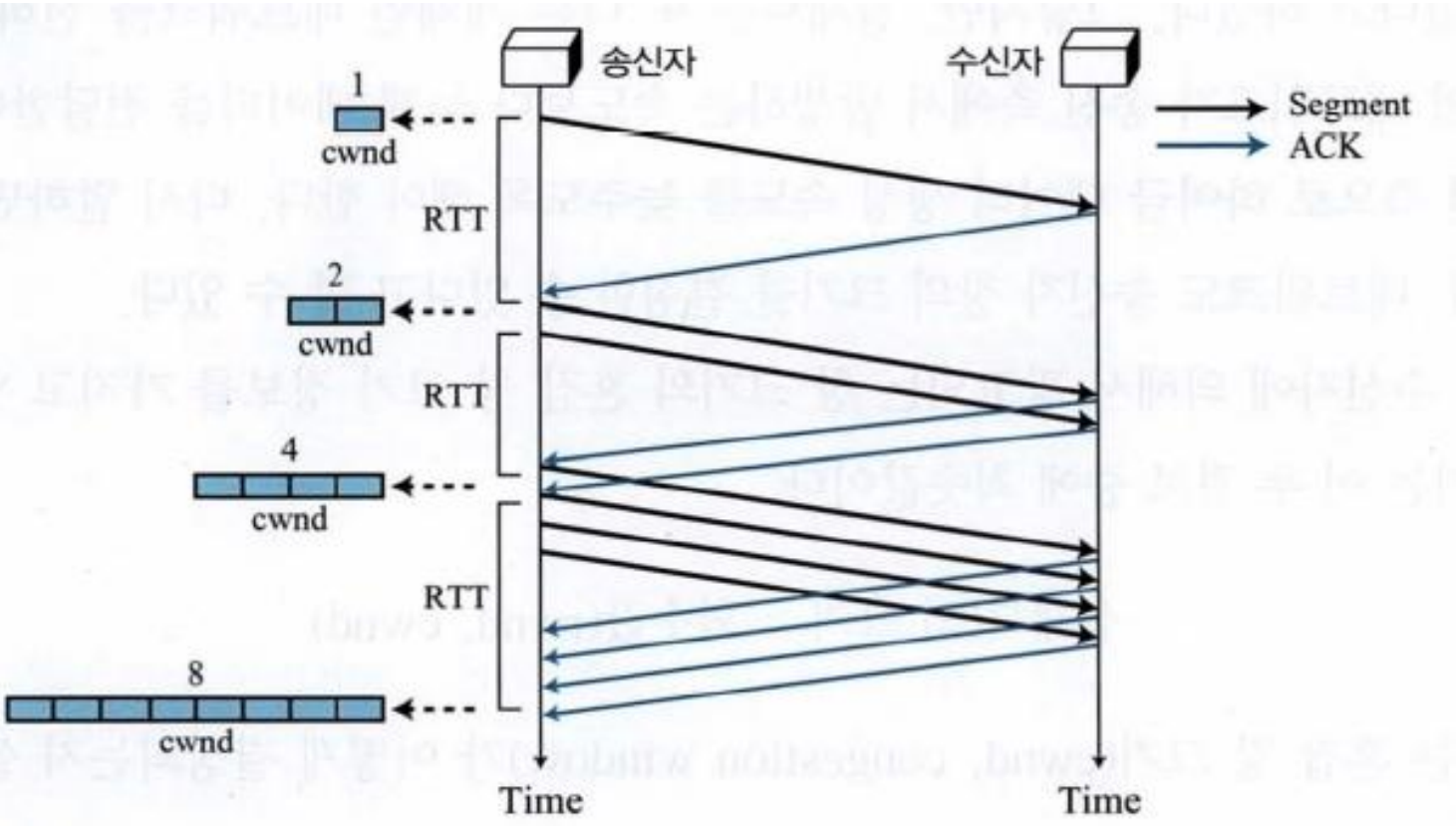
혼잡 제어 원칙

- **TCP에서 혼잡을 처리하는 일반적인 원칙은 느린 시작, 혼잡 회피, 그리고 혼잡 감지라는 세 단계를 기반으로 한다.**
- **느린 시작 단계에서 송신지는 매우 낮은 전송률로 시작하지만, 임계치에 도달할 때까지는 급격히 전송률을 증가시킨다.**
- **전송률이 임계치에 도달하면, 증가 속도를 감소시킨다.**
- **마지막으로 혼잡이 감지되면, 혼잡이 감지된 정도에 따라서 송신자는 느린 시작이나 혼잡 회피로 되돌아간다.**

- **느린시작: 지수증가**
  - 느린 시작(slow start) 알고리즘에서 혼잡 창(cwnd)의 크기는 최대 세그먼트 크기(MSS : Maximum Segment Size)에서부터 시작한다.
  - MSS는 동일한 이름을 가진 옵션을 이용하여 연결 설정 과정 동안에 결정된다.
  - 창의 크기는 하나의 확인응답이 도착할 때마다 하나의 MSS 만큼 증가한다.
  - 느린 시작이라는 이름에서 알 수 있듯이, 알고리즘은 느리게 시작하지만, 지수적으로 증가한다.

9. 혼잡 제어

- 혼잡 제어 원칙
  - 느린시작: 지수증가



9. 혼잡 제어

혼잡 제어 원칙

느린시작: 지수증가

- 왕복 시간의 관점에서 cwnd의 크기를 살펴보면, 다음과 같이 비율이 지수적으로 증가한다는 것을 알 수 있다.

$$\text{시작} \rightarrow \text{cwnd} = 1$$

$$1\ \text{RTT가 지난 후에} \rightarrow \text{cwnd} = 1 \times 2 = 2 \rightarrow 2^1$$

$$2\ \text{RTT가 지난 후에} \rightarrow \text{cwnd} = 2 \times 2 = 4 \rightarrow 2^2$$

$$3\ \text{RTT가 지난 후에} \rightarrow \text{cwnd} = 4 \times 2 = 8 \rightarrow 2^3$$

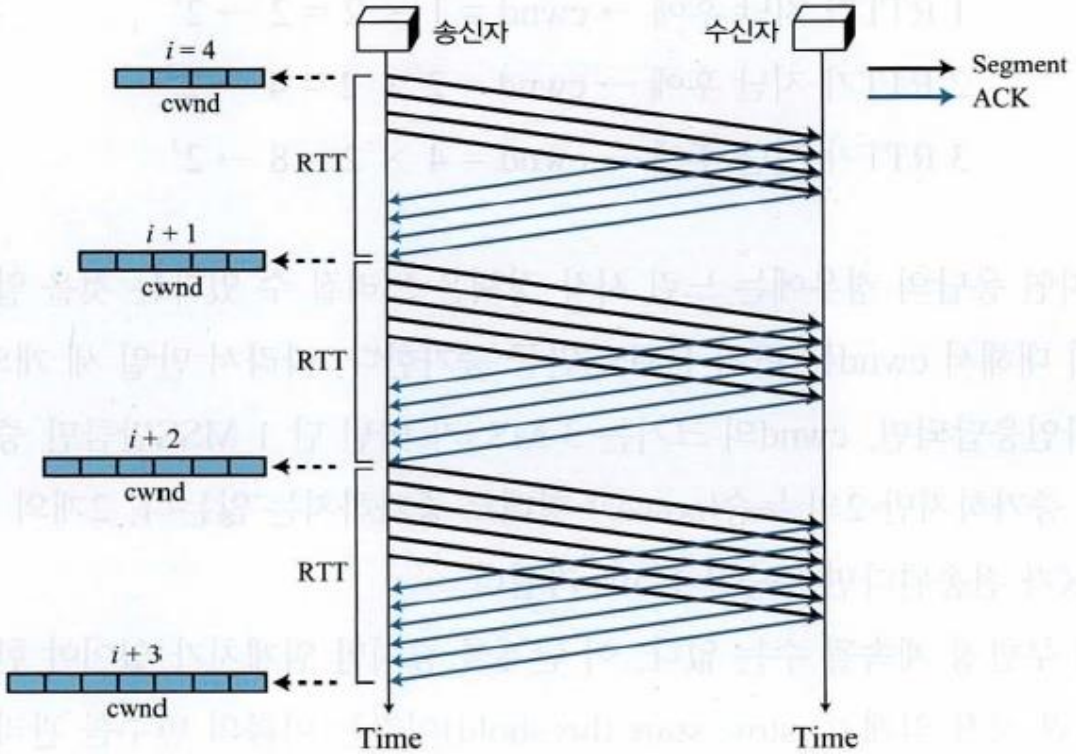
- 느린 시작이 무한정 계속될 수는 없다.
- 이 단계를 중지할 임계치가 있어야 한다.
- 송신자는 ssthresh(느린 시작 임계치 : slow start threshold)이라는 이름의 변수를 관리한다.
- 바이트 단위의 창 크기가 임계치에 도달하면, 느린 시작은 중지되고 다음 단계가 시작된다.

9. 혼잡 제어

혼잡 제어 원칙

혼잡회피: 가산증가

- 느린 시작 알고리즘에서 혼잡 창의 크기는 지수적으로 증가한다.
- 혼잡이 발생하기 전에 미리 혼잡을 예방하기 위해서는 이러한 지수 형태로 증가하는 속도는 늦춰져야 한다.
- TCP는 cwnd가 지수적이 아닌 가산적으로 증가하는 혼잡 회피 (congestion av이dance) 라고 하는 또 다른 알고리즘을 정의한다.



9. 혼잡 제어

혼잡 제어 원칙

혼잡회피: 가산증가

- 왕복 시간{RTT)의 관점에서 cwnd의 값을 살펴보면, cwnd의 증가율은 다음과 같이 가산적으로 증가한다(Additive Increase).

시작 → cwnd = i

1 RTT가 지난 후에 → cwnd = i +1

2 RTT가 지난 후에 → cwnd = i + 2

3 RTT가 지난 후에 → cwnd = I + 3

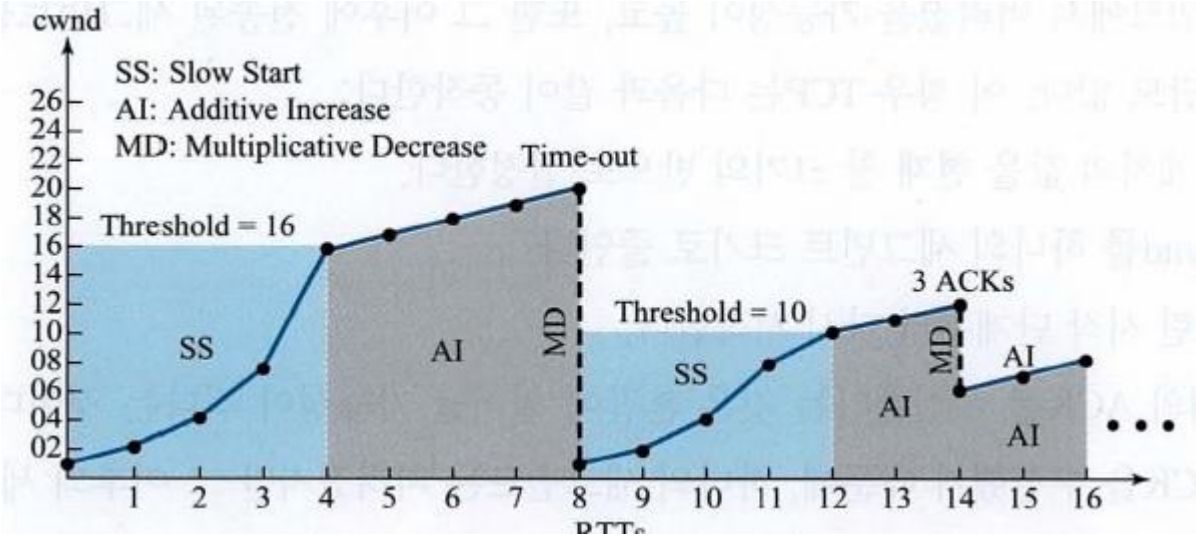
9. 혼잡 제어

혼잡 제어 원칙

혼잡감지: 지수감소

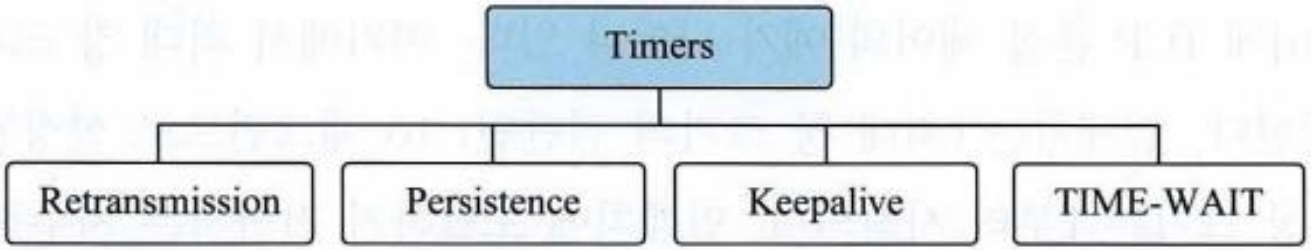
- 혼잡이 발생하면, 혼잡 창의 크기는 감소해야 한다.
- 송신자가 혼잡이 발생했다고 간주하는 유일한 방법은 세그먼트를 재전송할 필요가 있는 경우이다.
- 재전송은 라우터/네트워크에 과부하나 혼잡이 발생하여 라우터에 많은 패킷이 입력되어 라우터에서 버려지는(손실되는) 패킷을 복구하기 위하여 필요한 기능이다.
- 그렇지만 재전송은 RTO 타이머가 만료되거나 또는 세 개의 중복 ACK가 수신되는 경우에 발생한다.
- 두 경우 모두 임계치의 크기는 반으로 감소된다.

혼잡 제어 예



10. TCP 타이머

- TCP의 동작을 원활하게 수행하기 위하여, TCP는 4가지 타이머를 이용한다.



10. TCP 타이머

재전송 타이머

- **손실된 세그먼트의 재전송을 위하여, TCP는 한 세그먼트에 대한 확인응답을 기다라는 시간인 재전송 타임 아웃(RTO: Retransmission Time-Out) 값으로 설정되는 재전송 타이머를 가지고 있다.**
- **전송 타이머에 대해서 다음과 같은 규칙을 정의할 수 있다.**
  - TCP가 송신 버퍼의 맨 앞에 있는세그먼트를 전송할 때 타이머를 구동한다.
  - 타이머가 만료되면, TCP는 버퍼의 앞에 있는 첫 번째 세그먼트를 재전송하며 타이머를 다시 구동한다.
  - 누적 확인응답된 세그먼트들은 버퍼에서 삭제된다.
  - 버퍼가 빈 경우에는 TCP는 타이머를 정지하고, 그렇지 않으면 타이머를 다시 구동한다.

10. TCP 타이머

영속 타이머

- 창 크기 0을 통보하는 경우를 처리하기 위하여 TCP에는 또 다른 타이머가 필요하다.
- TCP에서 ACK 세그먼트는 확인응답되지 않고 또한 재전송되지도 않는다.
- 만일 이러한 확인응답이 손실되면 수신 TCP는 자신의 임무를 다했다고 간주하고 송신 TCP가 세그먼트를 전송하기를 기다린다.
- 확인응답만을 포함하는 세그먼트에 대해서는 재전송 타이머가 구동되지 않는다.
- 송신 TCP는 확인응답을 수신하지 못하고 수신 TCP가 창 크기를 알리는 확인응답을 전송하기만을 기다린다.
- 즉, 모든 TCP는 서로 기다리기를 계속한다{교착상태).
- 이러한 교착상태 (deadlock)를 해결하기 위하여 TCP는 각 연결마다 하나의 영속 타이머(persistence timer)를 사용한다.
- 송신 TCP가 창 크기 0을 가진 확인응답을 수신하면, 송신 TCP는 영속 타이머를 구동한다.
- 영속 타이머가 만료되면, 송신 TCP는 프로브(probe)라고 하는 특수한 세그먼트를 전송한다.
- 이 프로브는 수신 TCP에게 확인응답이 손실되었고 따라서 확인응답을 재전송하도록 알려준다.

10. TCP 타이머

킵얼라이브 타이머

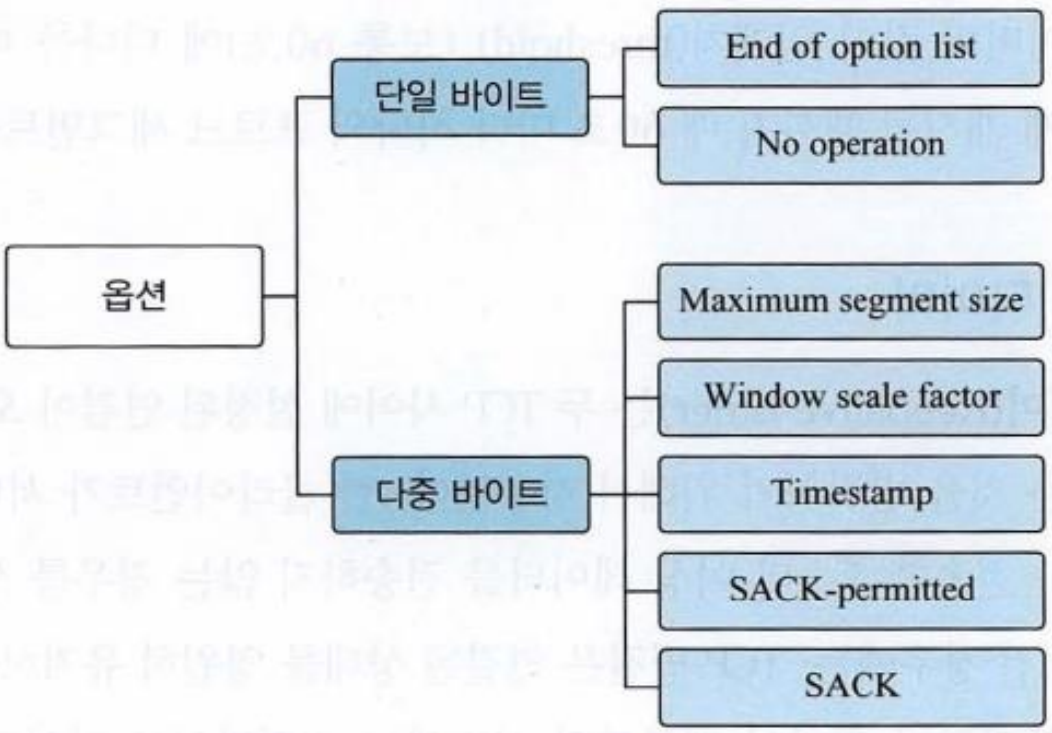
- 킵얼라이브 타이머(Keepalive timer)는 두 TCP 사이에 설정된 연결이 오랜 시간 동안 휴지(idle) 상태에 있는 것을 방지하기 위해서 사용된다.
- 한 클라이언트가 서버로 TCP 연결을 설정하고, 데이터를 전송한 후 더 이상 데이터를 전송하지 않는다면, 클라이언트에 이상이 생긴 경우에는 TCP 연결은 연결된 상태를 영원히 유지하게 된다.
- 이러한 상황을 해결하기 위하여 대부분의 서버에는 킵얼라이브 타이머가 구현되어 있다.
- 서버가 클라이언트로부터 세그먼트를 받을 때마다, 서버는 이 타이머를 초기화한다.

시간 대기 타이머

- 시간 대기(2MSL) 타이머는 연결 종료 동안에 사용된다.

11. 옵션

- TCP 헤더에는 최대 40바이트의 옵션 정보가 있을 수 있다.
- 이 옵션 정보들은 목적지에게 부가 정보를 전달하거나 또는 다른 옵션의 정렬을 맞추기 위하여 사용된다.



11. 옵션

옵션 종료(EOP)

- 옵션 종료(EOP: End-of-Option) 옵션은 옵션 구간의 끝에 패딩을 위하여 사용되는 단일 바이트 옵션이다.
- 이것은 마지막 옵션으로서만 사용될 수 있으며, 또한 단지 한 개의 옵션만 사용될 수 있다.
- 옵션 종료 옵션은 목적지에게 다음의 두 가지 정보를 제공한다:
  - 헤더에 더 이상의 옵션은 없다.
  - 응용 프로그램의 데이터는 다음 32 비트 워드의 처음부터 시작된다.

무동작(NOP)

- 무동작 (NOP: No-OPeration) 옵션은 옵션 사이의 채우기(filler)로서 사용되는 단일 바이트 옵션이다.
- 이 옵션은 일반적으로 다른 옵션의 앞에 위치해서 한 옵션이 4바이트의 배수가 되도록 한다.

11. 옵션

최대 세그먼트 크기

- 최대 세그먼트 크기(MSS: Maximum Segment Size) 옵션은 TCP 세그먼트의 목적지에서 수신할 수 있는 데이터 세그먼트의 최대 크기를 정의한다.
- 최대 세그먼트 크기라는 이름에도 불구하고, 이 옵션은 세그먼트의 최대 크기를 정의하는 것이 아니라 데이터 부분의 최대 크기를 정의한다.
- 이 필드의 길이는 16 비트이기 때문에 0 에서 65,535 바이트 사이의 값을 가질 수 있다.

창 확장 인자

- 헤더에 있는 창 크기 필드는 미닫이 창의 크기를 정의한다.
- 이 필드의 길이는 16 비트로서 창은 0에서 65,535바이트 범위를 가질 수 있다.
- 65,535는 창 크기로는 매우 큰 값처럼 보이지만, 전송되는 바이트마다 번호가 부여되기 때문에 특히 데이터가 고속의 처리율과 긴 지연시간을 가진 전송 매체를 통해서 전달될 때에는 이 크기가 충분하지 않을 수 있다.
- 창 크기를 증가시키기 위해 창 확장 인자(window scale factor)가 사용된다.

$$\text{새로운 창 크기} = \text{헤더에서 정의된 창 크기} \times 2^{\text{창 확장 인자}}$$

11. 옵션

창 확장 인자

- 창 확장 인자는 최대 255까지 가질 수 있지만 TCP/IP에서 설정될 수 있는 최댓값은 14이다.
- 창의 크기는 순서 변호의 최댓값을 초과할 수 없다.
- 창 확장 인지는 연결 설정 단계에서만 결정될 수 있다.

타임스탬프

- 타임스탬프(timestamp)는 10바이트 옵션이다.
- 능동 개방을 요청한 한쪽 종단은 연결 요청 세그먼트(즉, SYN 세그먼트) 내에 타임스탬프를 알린다.
- 다른 한쪽 종단으로부터 다음 세그먼트(즉, SYN + ACK 세그먼트)에 있는 타임스탬프를 수신하면, 타임스탬프를 사용해도 된다는 허락을 의미한다.
- 그렇지 않으면 더 이상 타임스탬프 옵션을 사용할 수 없다.
- 타임스탬프 옵션은 왕복 시간을 측정하거나 또는 순서 번호가 겹치는 것을 방지하는 응용에 사용된다.

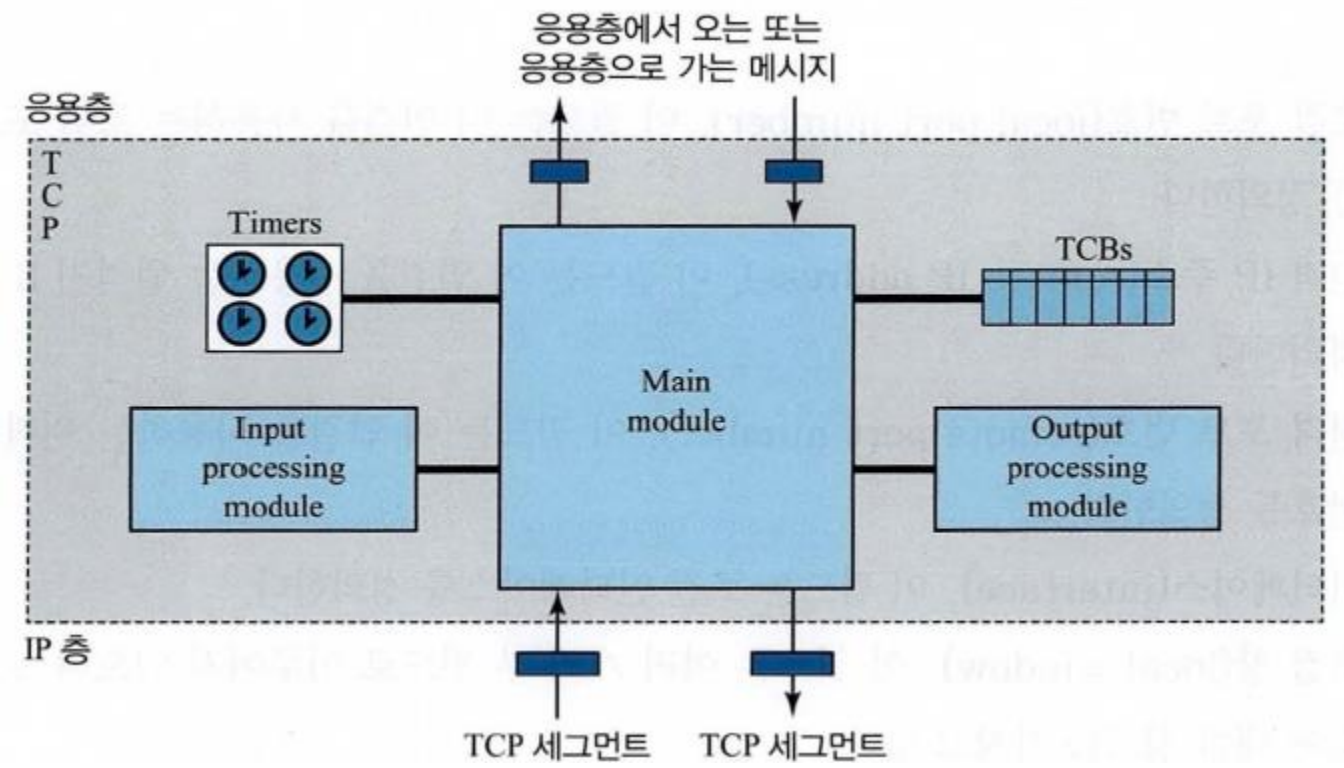
11. 옵션

SACK-허용 및 SACK 옵션

- TCP 세그먼트에 있는 확인응답 필드는 누적 확인응답 방식으로 설계되었다.
- 그렇지만, 이 방식에서 수신 측은 순서에 어긋나게 수신한 바이트에 대해서는 알려주지 않는다.
- 또한, 중복 세그먼트에 대해서도 세그먼트가 중복되었다는 것을 알려주지 않는다.
- 이것은 TCP의 성능에 나쁜 영향을 미칠 수 있다.
- TCP의 성능을 향상시키기 위하여, 선택적 확인응답(SACK: selective acknowledgment)이 제안되었다.
- 선택적 확인응답은 송신 측으로 하여금 실제로 어느 세그먼트가 없어졌으며 또한 어떤 세그먼트가 순서에 어긋나게 도착했는지에 대해서 더 자세하게 알 수 있도록 해준다.
- 이와 같은 새로 제안된 옵션에서는 중복 세그먼트에 대한 리스트도 포함한다.

12. TCP 패키지

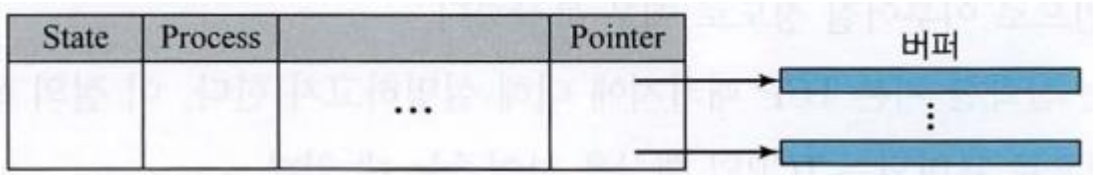
- TCP 패키지는 전송 제어 블록(TCB: transmission control block) 이라고 하는 테이블과 일련의 타이머, 그리고 메인 모듈, 입력 프로세스 모듈과 출력 프로세스 모듈로 이루어진 세 개의 소프트웨어 모듈로 이루어져 있다.



12. TCP 패키지

전송 제어 블록

- 연결 제어를 위하여, TCP는 각 연결에 대한 정보를 보관하기 위한 구조를 이용한다.
- 이러한 구조를 전송 제어 블록(TCB; Transmission Control Block) 이라고 한다.
- TCP에는 동시에 여러 개의 연결이 있을 수 있기 때문에, TCP는 테이블 형태로 TCB 배열을 유지한다.



- TCB의 가장 일반적인 필드는 다음과 같다.
  - 상태 (state) : 이 필드는 상태 천이 다이어그램에 따른 연결의 상태를 정의한다.
  - 프로세스(process) : 이 필드는 클라이언트 또는 서버로서 이 기기에서 연결을 사용하는 프로세스를 정의한다.
  - 로컬 IP 주소(local IP address) : 이 필드는 이 연결을 사용하는 로컬 IP 주소를 정의한다.
  - 로컬 포트 번호(local port number) : 이 필드는 이 연결을 사용하는 로컬 포트 번호를 정의한다.
  - 원격 IP 주소(remote IP address) : 이 필드는 이 연결을 사용하는 원격지 IP 주소를 정의한다.
  - 원격 포트 번호(remote port number) : 이 필드는 이 연결을 사용하는 원격지 포트 번호를 정의한다.

12. TCP 패키지

전송 제어 블록

TCB의 가장 일반적인 필드는 다음과 같다.

- 인터페이스(interface) : 이 필드는 로컬 인터페이스를 정의한다.
- 로컬 창(local window) : 이 필드는 여러 개의 부 필드로 이루어져 있으며 로컬 TCP 창에 대한 정보를 가지고 있다.
- 원격 창(remote window) : 이 필드는 여러 개의 부 필드로 이루어져 있으며 원격지 TCP 창에 대한 정보를 가지고 있다.
- 송신 순서 번호(sending sequence number) : 이 필드는 송신 순서 번호를 가지고 있다.
- 수신 순서 번호(receiving sequence number) : 이 필드는 수신 순서 번호를 가지고 있다.
- 송신 ACK 번호(sending ACK number) : 이 필드는 전송한 Aα 번호를 가지고 있다.
- 왕복 시간(round-trip time) : 여러 개의 필드가 RTT에 대한 정보를 보관하는 데 사용된다.
- 타임-아웃 값(time-out value) : 재전송 타임-아웃, 영속 타임-아웃, 킵얼라이브 타임-이웃 등과 같은 여러 개의 타임-아웃 값을 보관하기 위하여 여러 개의 필드가 사용된다.
- 버퍼 크기 (buffer size) : 이 필드는 로컬 TCP의 버퍼 크기를 정의한다.
- 버퍼 포인터 (buffer pointer) : 이 필드는 응용 프로그램에 의해 읽혀지기 전까지의 수신 데이터가 버퍼의 어느 곳에 있는지를 알려주는 포인터이다.

12. TCP 패키지

타이머

- TCP에는 동작을 처리하기 위한 여러 가지 타이머가 필요하다.

메인 모듈

- 메인 모듈은 TCP 세그먼트가 도착하거나 타임 아웃 이벤트가 발생하거나 또는 응용 프로그램으로부터 메시지가 들어오면 실행된다.

```
1  TCP_Main_Module (Segment)
2  {
3      Search the TCB Table
4      if (corresponding TCB is not found)
5          Create a TCB with the state CLOSED
6      Find the state of the entry in the TCB table
7      Switch (state)
8      {
```

12. TCP 패키지

메인 모듈

```
7        case CLOSED state:
8            if (“passive open” message received)
9                go to LISTEN state.
10           if (“active open” message received)
11           {
12               send a SYN segment
13               go to SYN-SENT state
14           }
15           if (any segment received)
16               send an RST segment
17           if (any other message received)
18               issue an error message
19           break
20
```

12. TCP 패키지

메인 모듈

```
21          case LISTEN state:
22              if("send data" message received)
23              {
24                  send a SYN segment
25                  Go to SYN-SENT state
26              }
27              if (any SYN segment received)
28              {
29                  Send a SYN + ACK segment
30                  Go to SYN-RCVD state
31              }
32              if (any other segments or message received)
33                  Issue an error message
34              break
35
```

12. TCP 패키지

메인 모듈

```
36          case SYN-SENT state:
37              if (time-out)
38                  Go to CLOSED state
39              if (SYN segment received)
40              {
41                  Send a SYN + ACK segment
42                  Go to SYN-RCVD state
43              }
44              if (SYN + ACK segment received)
45              {
46                  Send an ACK segment
47                  Go to ESTABLISHED state
48              }
49              if (any other segment or message received)
50                  Issue an error message
51              break
52
```

12. TCP 패키지

■ 메인 모듈

```
53          case SYN-RCVD state:
54              if (and ACK segment received)
55                  Go to ESTABLISHED state
56              if (time-out)
57              {
58                  Send an RTS segment
59                  Go to CLOSED state
60              }
61              if (“closed” message received)
62              {
63                  Send a FIN segment
64                  Go to FIN-WAIT-I state
65              }
66              if (RTS segment received)
67                  Go to LISTEN state
68              if (any other segment or message received)
69                  Issue an error message
70              break
```

12. TCP 패키지

■ 메

```
71
72          case ESTABLISHED state:
73              if (a FIN segment received)
74              {
75                  Send an ACK segment
76                  Go to CLOSED-WAIT state
77              }
78              if (“closed” message received)
79              {
80                  Send a FIN segment
81                  Go to FIN-WAIT-I state
82              }
83              if (a RTS or an SYN segment received)
84                  Issue an error message
85              if (data or ACK segment received)
86                  call the input module
87              if (“send” message received)
88                  call the output module
89              break
```

12. TCP 패키지

메인 모듈

```
90
91          case FIN-WAIT-I state:
92              if (a FIN segment received)
93              {
94                  Send an ACK segment
95                  Go to CLOSING state
96              }
97              if (a FIN + ACK message received)
98              {
99                  Send a ACK segment
100                 Go to FIN-WAIT state
101             }
102             if (an ACK segment received)
103                 Go to FIN-WAIT-2 state
104             if (any other segment or message received)
105                 Issue an error message
106             break
```

12. TCP 패키지

입력 처리 모듈

- TCP 설계에서, 입력 처리 모듈(input processing module)은 TCP가 ESTABLISHED 상태에 있을 때 수신하는 데이터나 확인응답을 처리하기 위하여 필요한 모든 기능을 담당한다.
- 이 모듈은 필요한 경우 ACK를 전송하고, 창 크기 알림을 처리하고, 오류 확인 등을 담당한다.

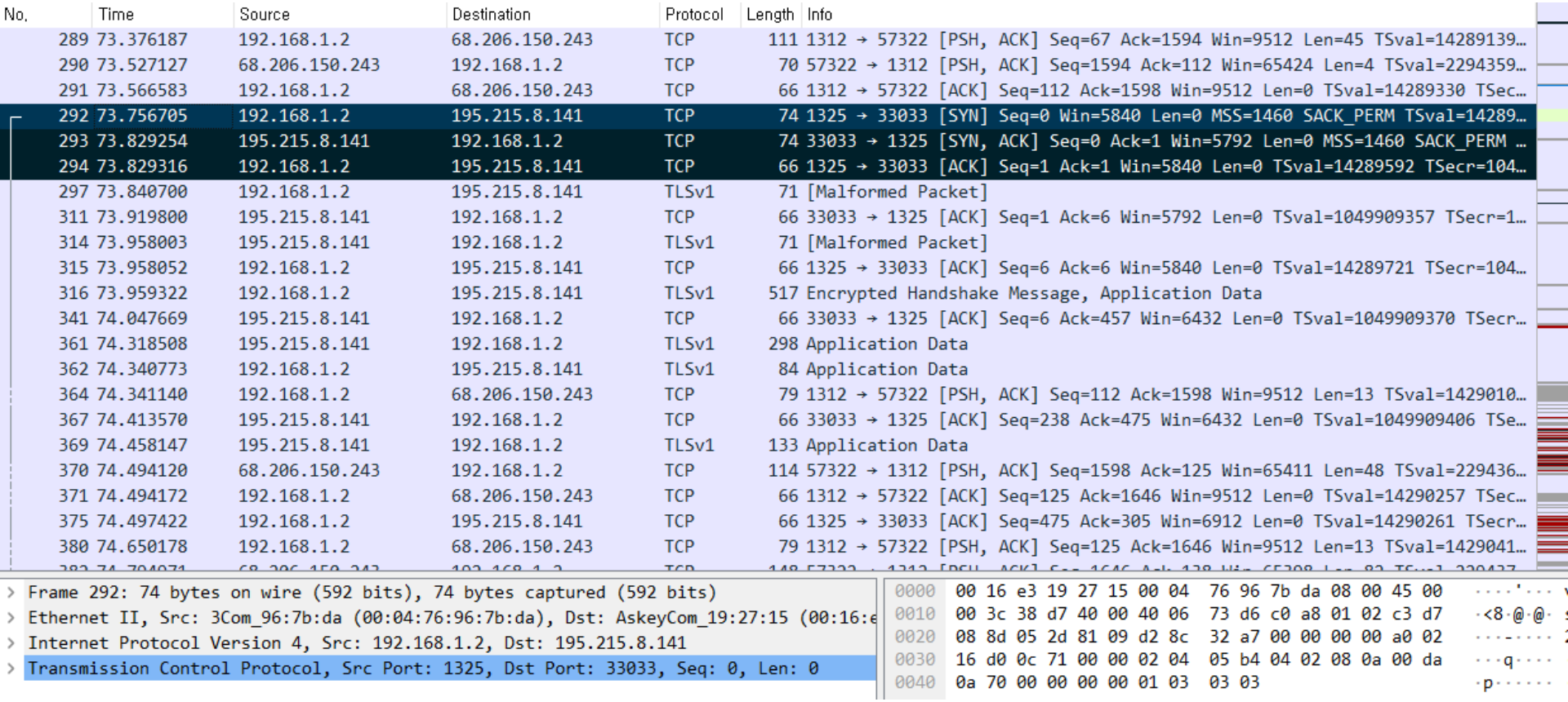
출력 처리 모듈

- TCP 출력에서, 출력 처리 모듈(output processing module)은 TCP가 ESTABLISHED 상태에 있을 때 응용 프로그램으로부터 수신한 데이터를 전송하기 위하여 필요한 모든 기능을 담당한다.
- 이 모듈은 재전송 타임 아웃, 영속 타임 아웃 등과 같은 기능을 담당한다.

13. TCP 덤프 분석

IT COOKBOOK

- 그림에 있는 292, 293, 294번 프레임을 확인해 보자.
- 이 300 ~ 302 번 프레임은 TCP 연결을 설정하는 3-방향 핸드쉐이크가 이루어진 패킷들이다.



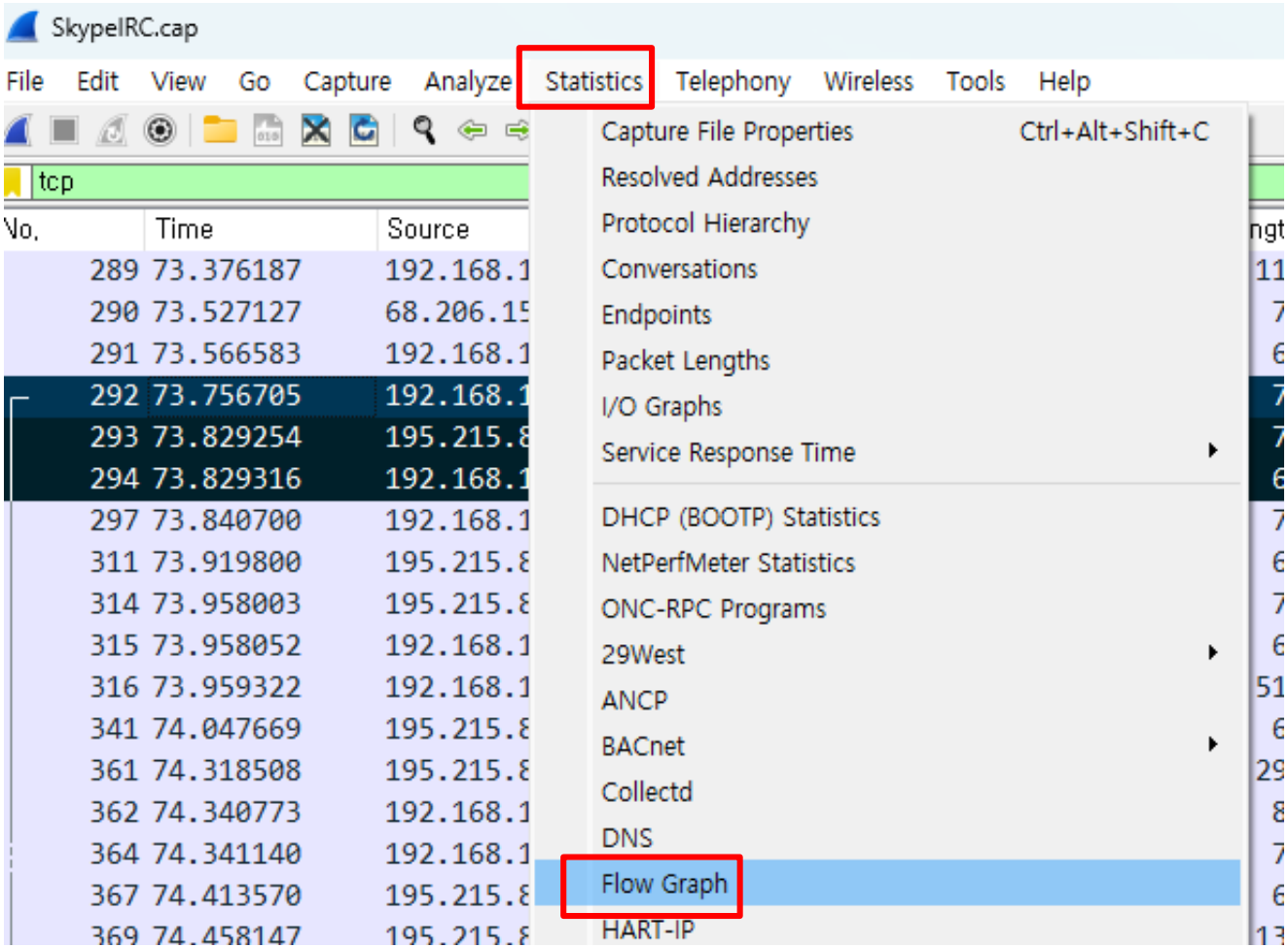
No.	Time	Source	Destination	Protocol	Length	Info
289	73.376187	192.168.1.2	68.206.150.243	TCP	111	1312 → 57322 [PSH, ACK] Seq=67 Ack=1594 Win=9512 Len=45 TSval=14289139…
290	73.527127	68.206.150.243	192.168.1.2	TCP	70	57322 → 1312 [PSH, ACK] Seq=1594 Ack=112 Win=65424 Len=4 TSval=2294359…
291	73.566583	192.168.1.2	68.206.150.243	TCP	66	1312 → 57322 [ACK] Seq=112 Ack=1598 Win=9512 Len=0 TSval=14289330 TSec…
292	73.756705	192.168.1.2	195.215.8.141	TCP	74	1325 → 33033 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM TSval=14289…
293	73.829254	195.215.8.141	192.168.1.2	TCP	74	33033 → 1325 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM …
294	73.829316	192.168.1.2	195.215.8.141	TCP	66	1325 → 33033 [ACK] Seq=1 Ack=1 Win=5840 Len=0 TSval=14289592 TSecr=104…
297	73.840700	192.168.1.2	195.215.8.141	TLSv1	71	[Malformed Packet]
311	73.919800	195.215.8.141	192.168.1.2	TCP	66	33033 → 1325 [ACK] Seq=1 Ack=6 Win=5792 Len=0 TSval=1049909357 TSecr=1…
314	73.958003	195.215.8.141	192.168.1.2	TLSv1	71	[Malformed Packet]
315	73.958052	192.168.1.2	195.215.8.141	TCP	66	1325 → 33033 [ACK] Seq=6 Ack=6 Win=5840 Len=0 TSval=14289721 TSecr=104…
316	73.959322	192.168.1.2	195.215.8.141	TLSv1	517	Encrypted Handshake Message, Application Data
341	74.047669	195.215.8.141	192.168.1.2	TCP	66	33033 → 1325 [ACK] Seq=6 Ack=457 Win=6432 Len=0 TSval=1049909370 TSecr…
361	74.318508	195.215.8.141	192.168.1.2	TLSv1	298	Application Data
362	74.340773	192.168.1.2	195.215.8.141	TLSv1	84	Application Data
364	74.341140	192.168.1.2	68.206.150.243	TCP	79	1312 → 57322 [PSH, ACK] Seq=112 Ack=1598 Win=9512 Len=13 TSval=1429010…
367	74.413570	195.215.8.141	192.168.1.2	TCP	66	33033 → 1325 [ACK] Seq=238 Ack=475 Win=6432 Len=0 TSval=1049909406 TSe…
369	74.458147	195.215.8.141	192.168.1.2	TLSv1	133	Application Data
370	74.494120	68.206.150.243	192.168.1.2	TCP	114	57322 → 1312 [PSH, ACK] Seq=1598 Ack=125 Win=65411 Len=48 TSval=229436…
371	74.494172	192.168.1.2	68.206.150.243	TCP	66	1312 → 57322 [ACK] Seq=125 Ack=1646 Win=9512 Len=0 TSval=14290257 TSec…
375	74.497422	192.168.1.2	195.215.8.141	TCP	66	1325 → 33033 [ACK] Seq=475 Ack=305 Win=6912 Len=0 TSval=14290261 TSecr…
380	74.650178	192.168.1.2	68.206.150.243	TCP	79	1312 → 57322 [PSH, ACK] Seq=125 Ack=1646 Win=9512 Len=13 TSval=1429041…

```
> Frame 292: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
> Ethernet II, Src: 3Com_96:7b:da (00:04:76:96:7b:da), Dst: AskeyCom_19:27:15 (00:16:e
> Internet Protocol Version 4, Src: 192.168.1.2, Dst: 195.215.8.141
> Transmission Control Protocol, Src Port: 1325, Dst Port: 33033, Seq: 0, Len: 0
```

```
0000  00 16 e3 19 27 15 00 04  76 96 7b da 08 00 45 00   ····'··· v·{···E·
0010  00 3c 38 d7 40 00 40 06  73 d6 c0 a8 01 02 c3 d7   ·<8·@·@· s·····
0020  08 8d 05 2d 81 09 d2 8c  32 a7 00 00 00 00 a0 02   ···-···· 2·····
0030  16 d0 0c 71 00 00 02 04  05 b4 04 02 08 0a 00 da   ···q····
0040  0a 70 00 00 00 00 01 03  03 03                     ·p·····
```

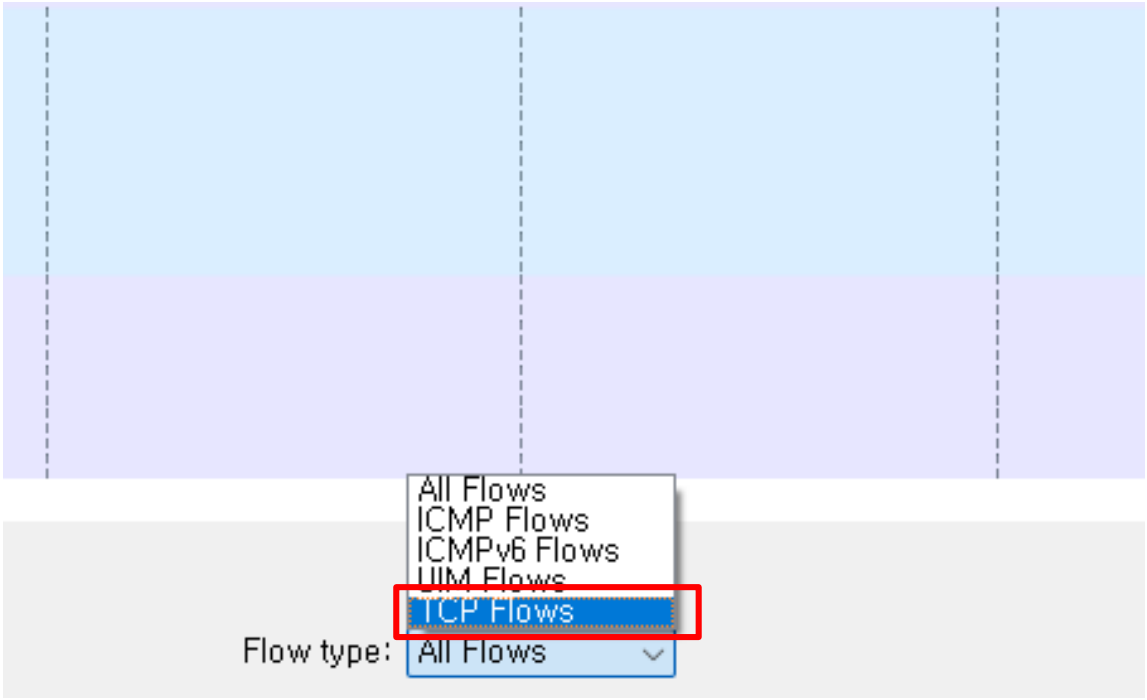
13. TCP 덤프 분석

- TCP 세그먼트 교환은 Flow Graph의 화살표 선으로 확인할 수 있다.
- 이 화면을 보려면 메뉴 바에서 [Statistics → Flow Graph]를 선택한다.

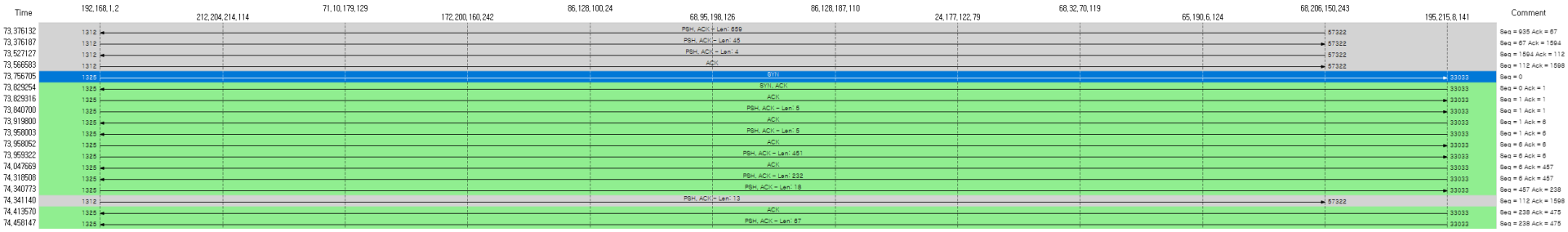


13. TCP 덤프 분석

- 그리고 [flow type]란에서 [TCP flow]를 선택한다.

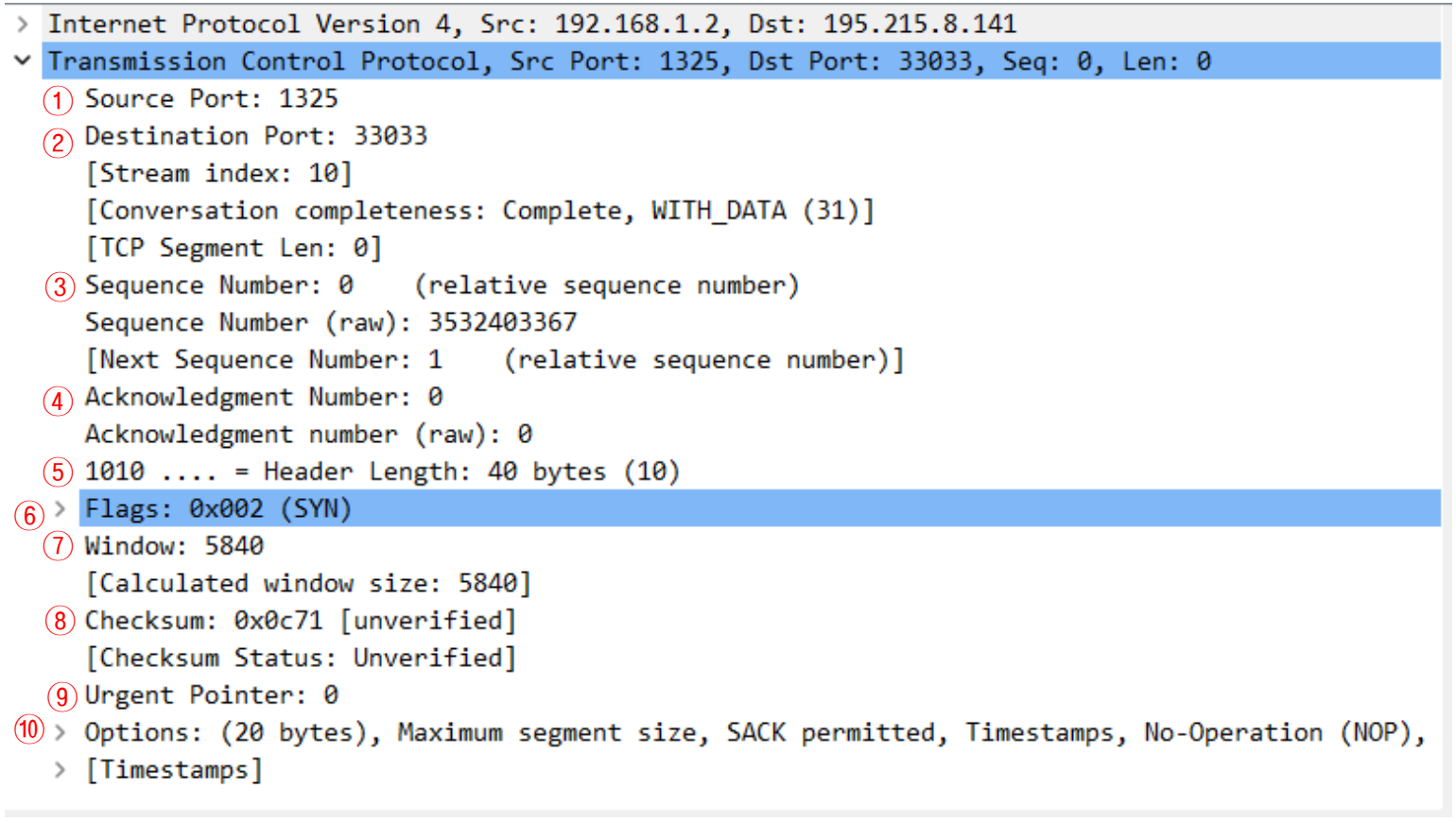


- 그러면 그림과 같이 Graph Analysis 화면이 나타난다.



13. TCP 덤프 분석

- 먼저 292번 프레임을 확인해 보자.
- 패킷 목록 정보에서 292번 프레임을 선택한 다음 패킷 상세 정보의 [Transmission Control Protocol] 앞의 [>]를 클릭한다.
- 그러면 그림과 같이 패킷 상세 정보 화면이 나타난다.



13. TCP 덤프 분석

- 그림을 보면 맨 먼저 이 패킷에서 발신지 포트 번호인 ① [Source port:] 필드에 클라이언트가 임시로 할당한 포트 번호가 1325로 나타나 있다.
- 다음은 목적지 포트 번호인 ② [Destination port:]는 [33033]번이 지정되어 있다.
- 또, ③ [Sequence number(순서 번호)]는 32 비트(4바이트) 필드로서 패킷의 순서 번호를 나타낸다.
- ④ Acknowledgement Number(확인응답 번호)는 아직 SYN/ACK 패킷을 수신하지 않았기 때문에 0으로 설정되었다.
- ⑤ [Header Length(헤더 길이)] 필드는 TCP 헤더의 길이를 나타낸다.
- 여기서는 [40]이란 값이 들어가 있다.
- 옵션이 없는 경우 TCP 헤더는 20바이트이므로 이는 TCP 옵션이 20바이트가 부가되어 있음을 알 수 있다.

13. TCP 덤프 분석

- 그림을 보면 맨 먼저 이 패킷에서 발신지 포트 번호인 ① [Source port:] 필드에 클라이언트가 임시로 할당한 포트 번호가 1325로 나타나 있다.
- 다음은 목적지 포트 번호인 ② [Destination port:]는 [33033]번이 지정되어 있다.
- 또, ③ [Sequence number(순서 번호)]는 32 비트(4바이트) 필드로서 패킷의 순서 번호를 나타낸다.
- ④ Acknowledgement Number(확인응답 번호)는 아직 SYN/ACK 패킷을 수신하지 않았기 때문에 0으로 설정되었다.
- ⑤ [Header Length(헤더 길이)] 필드는 TCP 헤더의 길이를 나타낸다.
- 여기서는 [40]이란 값이 들어가 있다.
- 옵션이 없는 경우 TCP 헤더는 20바이트이므로 이는 TCP 옵션이 20바이트가 부가되어 있음을 알 수 있다.

13. TCP 덤프 분석

- ⑥ [Flags] 앞의 [>]를 클릭하면 그림과 같이 나타난다.

```
Flags: 0x002 (SYN)
① 000. .... .... = Reserved: Not set
② ...0 .... .... = Accurate ECN: Not set
③ .... 0... .... = Congestion Window Reduced: Not set
④ .... .0.. .... = ECN-Echo: Not set
⑤ .... ..0. .... = Urgent: Not set
⑥ .... ...0 .... = Acknowledgment: Not set
⑦ .... .... 0... = Push: Not set
⑧ .... .... .0.. = Reset: Not set
⑨ .... .... ..1. = Syn: Set
⑩ .... .... ...0 = Fin: Not set
[TCP Flags: ··········S·]
```

- 플래그의 맨 앞은 ① 예약(Reserved) 부분이다.
- 이 부분은 [0]으로 설정되어 있고, 사용하지 않는 영역이다.
- ② Nonce 플래그는 ECN-Nonce라고도 하며, 이후의 ECN-Echo와 함께 네트워크가 혼잡할 때 일시적인 통지를 위해 사용된다.

13. TCP 덤프 분석

- ③ CWR(Congestion Window Reduced)은 혼잡 제어를 위해 사용되는 플래그로서 IP 프로토콜의 ECN 기능(DiffServ라는 새로운 우선순위 제어 방식으로 이용되는 기능)과 함께 이용된다.
- ④ ECN-Echo 플래그는 네트워크가 혼잡할 때에 수신 측 기기가 발신지에 혼잡을 통지할 경우에 ON 이 된다.
- ⑤ URG(URGent) 플래그는 [긴급 포인터 플래그]라고 하며 통신 도중에 중요한 데이터를 끼워 넣어 보낼 경우에 ON이 된다.
- ⑥ ACK(ACKnowledgement) 플래그는 확인응답 번호가 유효한 경우에 ON이 된다.
- ⑦ PSH(PuSH) 플래그는 푸시 기능에 의해 수신 창에 버퍼링된 세그먼트를 애플리케이션에 보낼 때 ON 이 된다.
- ⑧ RST(ReSeT) 플래그는 통신 도중에 강제 종료할 경우에 ON 이 되는 플래그이다.

13. TCP 덤프 분석

- ⑨ SYN(SYNchronize) 플래그는 통신 시작 시에 동기를 맞출 경우 ON이 된다.
- ⑩ FIN(FINish) 플래그는 통신 종료 시에 ON이 된다.
- 이 패킷은 3-방향 핸드쉐이크(3-way handshake)를 수행하기 위해 SYN 플래그만 ON이고 다른 플래그는 모두 OFF로 되어있다.

13. TCP 덤프 분석

- 다시 [Flags(플래그)] 필드 바로 아래에 나타난 ⑦ [Window] 필드를 살펴보자.
- [Window] 필드는 TCP 소켓의 수신 버퍼인 창 길이를 16비트로 나타낸다.
- ⑧ [Checksum] 필드는 16 비트로 TCP 세그먼트의 오류를 확인한다.
- 이 패킷에는 [Checksum : 0x0c71(unverified)], [Checksurn Status: Unverified]라고 나타나 있으며 처리가 이루어지지 않고 있음을 알 수 있다.
- 또한 그 다음 16 비트 긴급 포인터 필드 ⑨ [Urgent Pointer]가 이 부분에 포함된다.
- 일반적인 헤더는 여기까지이다.
- 이후에 ⑩ [Options(옵션)] 필드가 있는데, 이 부분을 전개해보자.

13. TCP 덤프 분석

- 옵션은 그림과 같이 구성되어 있다.

```
Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP),
① > TCP Option - Maximum segment size: 1460 bytes
② > TCP Option - SACK permitted
③ > TCP Option - Timestamps
④ > TCP Option - No-Operation (NOP)
⑤ > TCP Option - Window scale: 3 (multiply by 8)
```

- ① [Maximum segment size:] 필드는 TCP 세그먼트(TCP의 데이터 부분)의 최대 길이(MSS)를 나타낸다.
- ② [TCP Option - SACK permitted] 필드는 선택적 확인응답(Selective ACK)을 유효로 한다는 의미이다.
- ④ [No-Operation(NOP)] 필드는 [조작 없음]을 의미한다.
- ⑤ [Window scale: 3 (multiply by 8)]은 창 확장 옵션으로 수신 창 크기를 2 의 3승(8배)으로 하여 취급함을 의미한다.

13. TCP 덤프 분석

- 이제 다음으로 293번 프레임을 확인해 보자.
- 패킷 목록 정보에서 293번 프레임을 선택한 다음 패킷 상세 정보의 [Transmission Control Protocol] 앞의 [>]를 클릭하면 그림과 같이 나타난다.

```
v Transmission Control Protocol, Src Port: 33033, Dst Port: 1325, Seq: 0, Ack: 1, Len: 0
  ① Source Port: 33033
  ② Destination Port: 1325
    [Stream index: 10]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
  ③ Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 3824702383
    [Next Sequence Number: 1    (relative sequence number)]
  ④ Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 3532403368
    1010 .... = Header Length: 40 bytes (10)
  > Flags: 0x012 (SYN, ACK)
    Window: 5792
    [Calculated window size: 5792]
    Checksum: 0x3df2 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP),
  > [Timestamps]
  > [SEQ/ACK analysis]
```

13. TCP 덤프 분석

- ③ [Sequence number(순서 번호)] 필드는 [0]으로 되어있는데 실제로는 응답 측의 초기 순서 변호가 설정된다.
- 여기서는 [3532403368]이 설정되어 있다.
- ④ [Acknowledgment number(확인응답 번호)] 필드가 [1]로 나타나 있음을 확인하라.
- 이것은 클라이언트 PC 측이 보낸 순서 번호(상대적으로는 0)에 1을 더한 값(상대적으로는 1)이다.

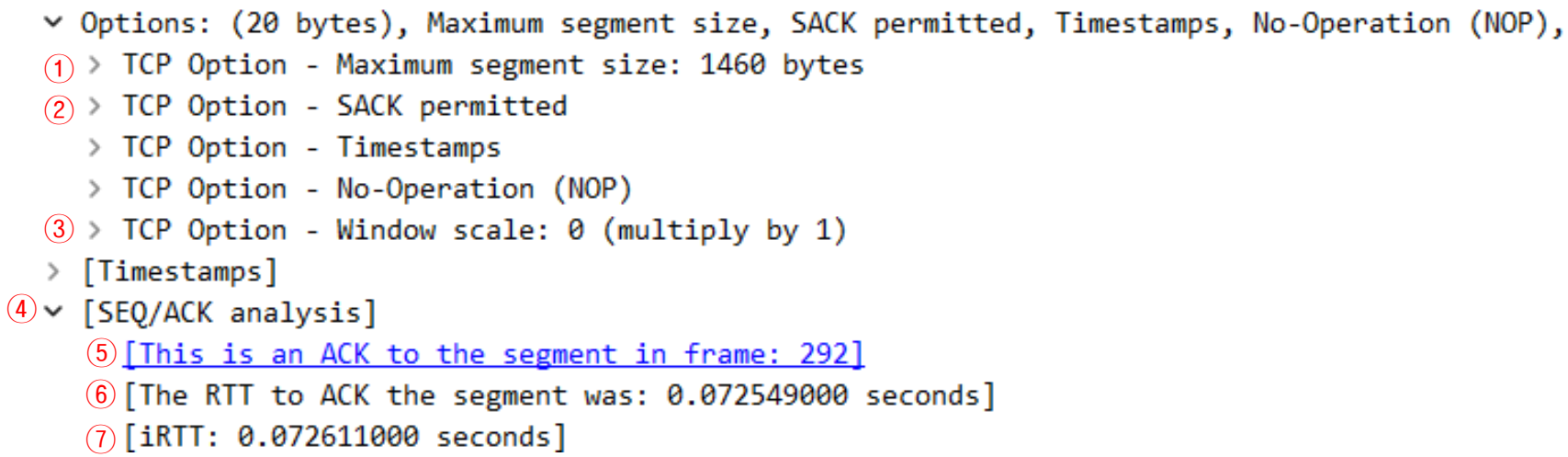
13. TCP 덤프 분석

- [Flags]를 펼쳐보면 그림과 같이 나타난다.

```
Flags: 0x012 (SYN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Accurate ECN: Not set
    .... 0... .... = Congestion Window Reduced: Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
①   .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
②   .... .... ..1. = Syn: Set
    .... .... ...0 = Fin: Not set
③   [TCP Flags: ·····A··S·]
④ Window: 5792
  [Calculated window size: 5792]
⑤ Checksum: 0x3df2 [unverified]
  [Checksum Status: Unverified]
⑥ Urgent Pointer: 0
  Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP),
```

13. TCP 덤프 분석

- 이번에는 접속 절차를 수행하기 위한 ② [Syn] 플래그가 ON 이 되어있으며 또한, 확인응답 번호를 유효로 하기 위해 ① [Acknowledgement] 플래그도 ON이 되어있음을 알 수 있다.
- 이어지는 ④ [Window(창 길이)]는 [5792]라고 지정되어 있다.
- 이후에 검사합 필드의 ⑤ [Checksum]과 긴급 포인터 필드인 ⑥ [Urgent Pointer] 필드가 이어진다.
- 여기서 [Options(옵션)과 [SEQ/ ACK analysis] 필드를 펼쳐 보면 그림과 같이 나타난다.



13. TCP 덤프 분석

- ① [Maximum segment size] 필드는 [1460]으로 되어있다.
- 또한, 앞에서와 마찬가지로 TCP 옵션의 길이를 조정하기 위하여 NOP 필드가 삽입되어 있다.
- 그리고 ② 선택적 확인 응답(SACK)을 유효로 함과 동시에 ③ 창 확장(Window Scale) 옵션을 2의 0승(2배)으로 지정함으로써 창 길이를 2배로 하여 취급함을 나타내고 있다.
- 와이어샤크가 접속을 분석한 ④ [SEQ/ ACK analysis]란 필드도 생성하고 있다.
- 구체적으로는 ⑤ [This is an ACK to the segment in frame : 300]이라 되어있고 이 패킷이 292번의 SYN 패킷에 대응한 ACK 패킷임을 나타내고 있다.
- 더욱이 ⑥ [The RTT to ACK the segment was 0.010550000 seconds]란 부분에서 시간의 차분인 RTT(Round Trip Time : 왕복 지연 시간)와 ⑦ iRTT(초기 왕복 지연 시간)도 확인할 수 있다.

13. TCP 덤프 분석

- 패킷 목록 정보에서 294번 프레임을 선택한 다음 패킷 상세 정보의 [Transmission Control Protocol] 앞의 [>]를 클릭하면 그림과 같이 나타난다.

```
▼ Transmission Control Protocol, Src Port: 1325, Dst Port: 33033, Seq: 1, Ack: 1, Len: 0
  ① Source Port: 1325
  ② Destination Port: 33033
    [Stream index: 10]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
  ③ Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 3532403368
    [Next Sequence Number: 1    (relative sequence number)]
  ④ Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 3824702384
    1000 .... = Header Length: 32 bytes (8)
  ⑤ > Flags: 0x010 (ACK)
  ⑥ Window: 730
    [Calculated window size: 5840]
    [Window size scaling factor: 8]
  ⑦ Checksum: 0x8035 [unverified]
    [Checksum Status: Unverified]
  ⑧ Urgent Pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  > [Timestamps]
  ▼ [SEQ/ACK analysis]
    ⑨ [This is an ACK to the segment in frame: 293]
    ⑩ [The RTT to ACK the segment was: 0.000062000 seconds]
    [iRTT: 0.072611000 seconds]
```

13. TCP 덤프 분석

- ③ [Sequence nurnber(순서 번호)] 필드는 상대적인 번호로서 [1]로 표시되어 있는데 실제로는 클라이언트 PC의 초기 순서 번호에 1을 더한 값이 된다.
- 더욱이 ④ [Acknowledgement number(확인응답 번호)] 필드가 [1]로 나타나 있음을 확인하기 바란다.
- 이것은 서버의 초기 순서 번호에 1을 더한 값이다.
- 이처럼 3-방향 핸드셰이크의 마지막 세그먼트에는 순서 번호, 확인응답 번호 모두 초기 값에 1을 더한 값이 된다.
- ⑤ [Flags] 필드를 펼치면 확인응답 번호를 유효로 하기 위해 ACK 플래그가 ON이 되어 있음을 알 수 있다.
- 더욱이 ⑥ [Window(창 길이)]인 [730]은 얼핏 수신 버퍼에 730바이트 밖에 없는 것처럼 생각되지만 실제로는 창 확장 인수에 의해 5840바이트란 의미가 된다.

13. TCP 덤프 분석

- ⑦ [Checksum(검사합)] 필드는 무효로 설정되어 있다.
- 따라서 [Checksum Status : Unverified]라고 나타난다.
- 또한 ⑧ 긴급 포인터는 [0]이 된다.
- ⑨ [This is an ACK to the segment in frame: 301] 이라 되어있으며, 이 패킷이 293번의 SYN/ACK 패킷에 대한 ACK 패킷임을 나타내고 있다.
- 더욱이 ⑩ [The RTT to ACK the segment was : 0.000062000 seconds]이며 왕복 지연시간도 확인할 수 있다.
- 이렇게 세 개의 세그먼트를 주고 받으며 3-방향 핸드셰이크가 종료된다.

Thank You