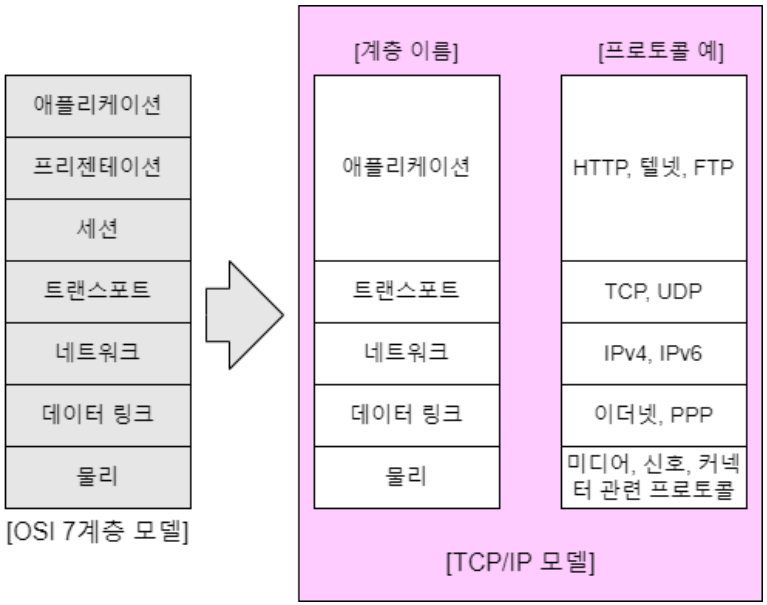
Chapter 03

TCP/IP

1. 인터넷 통신 모델

TCP/IP 통신 모델

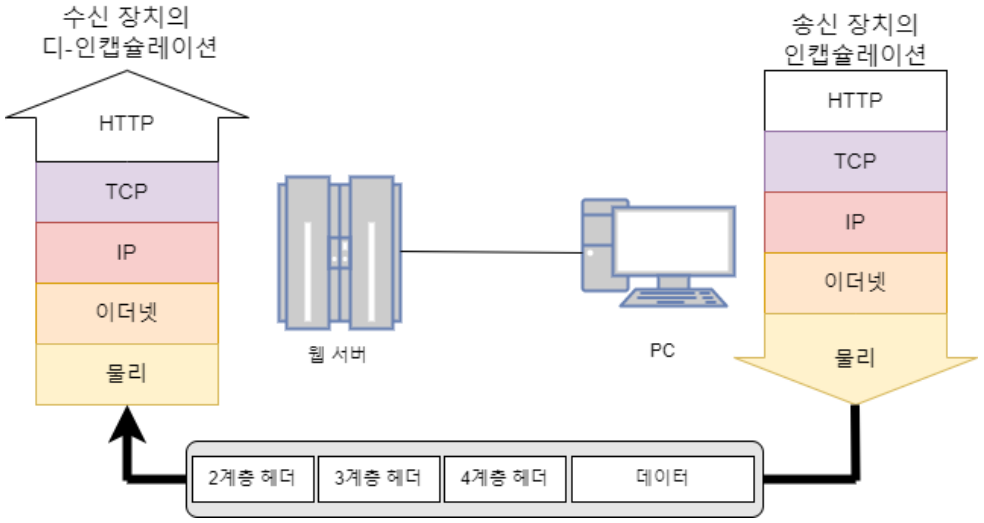
- 통신 모델마다 구체적인 기능을 구현한 프로토콜들을 정의하기 때문에 모델이 많을수록 구성은 복잡해지고 비용은 증가.
- 따라서 네트워크 관리자와 소유자의 고민은 '어떻게 하면 통신 모델을 줄여서 비용을 줄이는가?' 이다.
- 현재는 거의 TCP/IP 모델만 사용.



1. 인터넷 통신 모델

TCP/IP 통신 모델

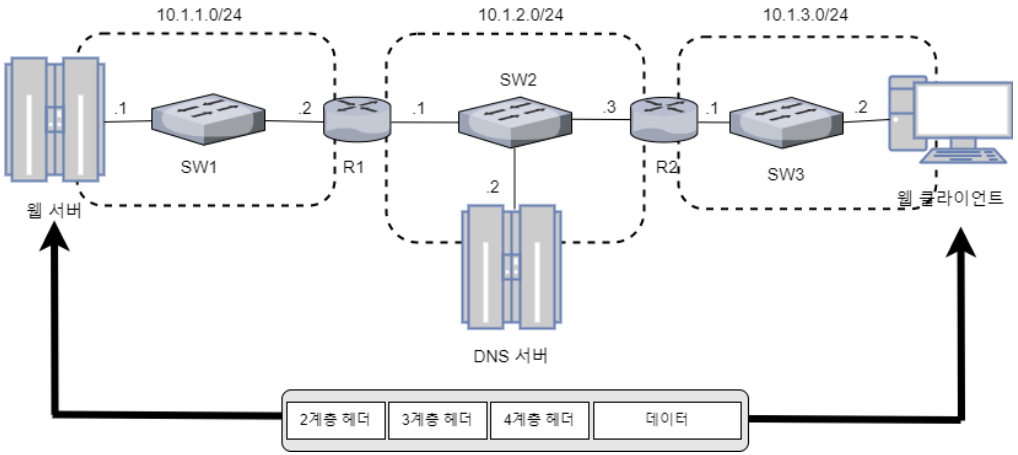
- 패킷은 송신 장치에서 그림과 같이 7계층에서 1계층으로 내려오면서 옷, 즉 헤더를 추가하는 인캡슐레이션(encapsulation) 과정을 거친다.
- 수신 장치에서는 1계층에서 7계층으로 올라가면서 옷, 즉 헤더를 벗기고 데이터만 추출하는 디인캡슐레이션(de-encapsulation) 과정을 거친다.
- 4계층 옷까지 입은 데이터 덩어리를 세그먼트(segment), 3계층 옷까지 입은 데이터 덩어리를 패킷(packet), 2계층 옷까지 입은 2계층 데이터 덩어리를 '프레임(frame)'이라고 부른다.



1. 인터넷 통신 모델

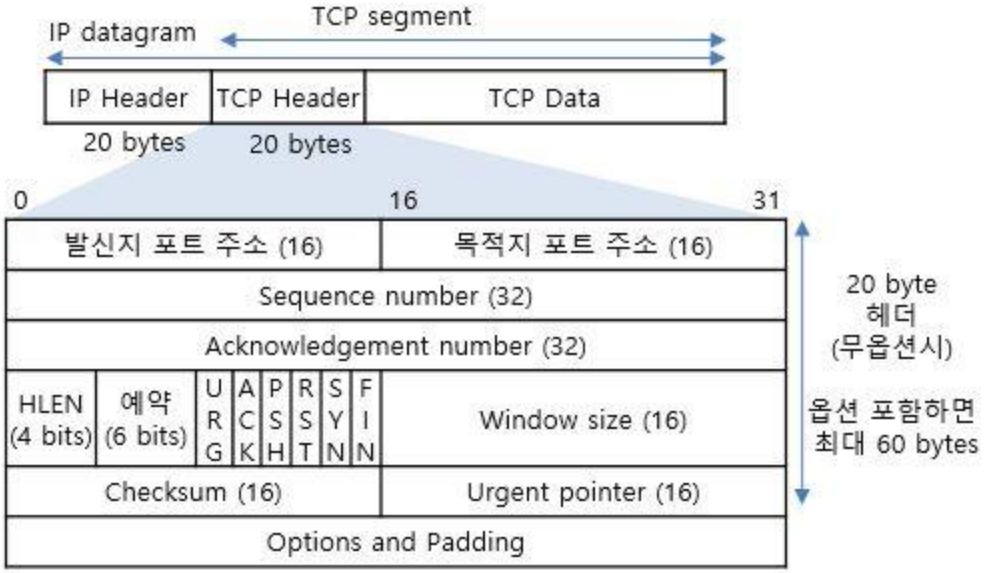
TCP

- 4계층 옷은 3계층 장치인 라우터나 2계층 장치인 스위치가 보거나 처리할 수 없다.
- 4계층 옷인 TCP(Transmission Control Protocol) 헤더를 처리할 수 있는 장치는 7계층 장치인 PC와 서버와 같은 단말뿐이다.



TCP

TCP 헤더의 포맷



TCP

TCP 헤더

- 목적지 포트(Destination Port)
  - 데이터 자리에 어떤 7계층 프로토콜이 정의하는 메시지가 실려 있는지를 표시
  - 계층별 프로토콜들의 번호

계층	필드	구분	번호
4	목적지 포트	HTTP	80
		FTP	20/21
		텔넷	23
		DNS	53
		TFTP	69
3	프로토콜	TCP	6
		UDP	17
2	타입	IPv4	0x0800
		IPv6	0x86DD

  - 이러한 필드들이 없다면 하나의 하드웨어 서버는 계층별로 하나의 프로토콜만 지원할 수 있다.
  - 따라서 HTTP-TCP-IPv4-이더넷, DNS-UDP-IPv6-이더넷 등 조합 수만큼 별도의 하드웨어를 필요로 하기 때문에 구축 비용이 대폭 상승하게 될 것

TCP

TCP 헤더

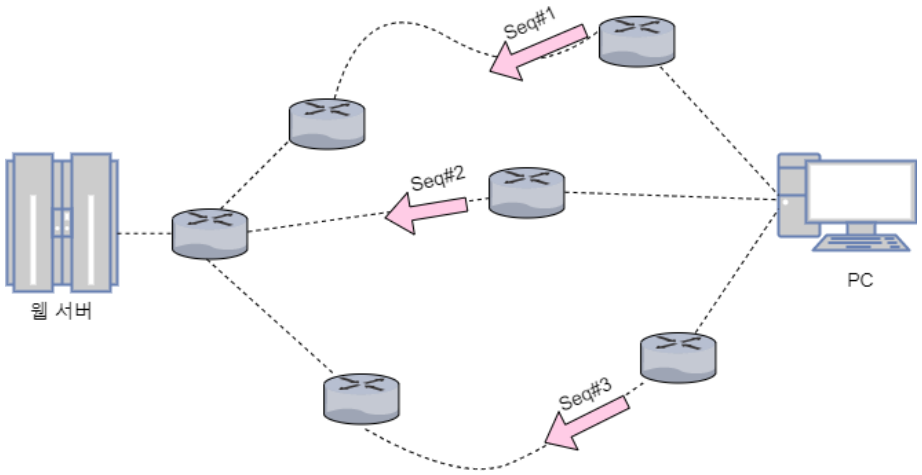
- 출발지 포트(Source Port)
  - 통신 커넥션이 시작될 때, 보통 1025~65535 범위 중 하나가 랜덤하게 선택.
  - 통신 커넥션은 출발지 IP 주소, 목적지 IP 주소, 출발지 포트 번호, 목적지 포트 번호에 의해 구분.
  - 이 중 하나라도 다르면 다른 TCP 커넥션에 속한다.
  - 예를 들어, 내 PC에서 동일한 웹 서버(예를 들어, www.naver. com 주소를 갖는 네이버 웹 서버)에 대해 2개의 웹 페이지 창을 열게 하는 것은 출발지 포트 덕분

- 플래그
  - 플래그 필드는 SYN, ACK, FIN, RST 비트를 포함한다.
  - TCP는 데이터 전송 전에 플래그 필드를 활용해 데이터가 전송 가능한지를 확인한 후 커넥션을 오픈하고, 커넥션을 유지하고 전송 완료 후에는 커넥션을 종결한다.
  - 이와 같이 커넥션 중심의 프로토콜을 커넥션-오리엔티드(connection-oriented) 프로토콜이라 부른다.

TCP

TCP 헤더

- 순서 번호(Sequence number)
  - 상위 계층에서 내려온 데이터 덩어리를 TCP는 자르고 UDP는 자르지 않는다.
  - 자르는 것은 출발지 장치에서 일어나고, 조립은 목적지 장치의 몫이다.
  - 그런데 그림과 같이, 세그먼트들은 다양한 경로를 통해 목적지에 도착하기 때문에 도착 순서가 아니라 원래의 순서대로 조립해야 한다.
  - 따라서 순서 번호가 필요하다.

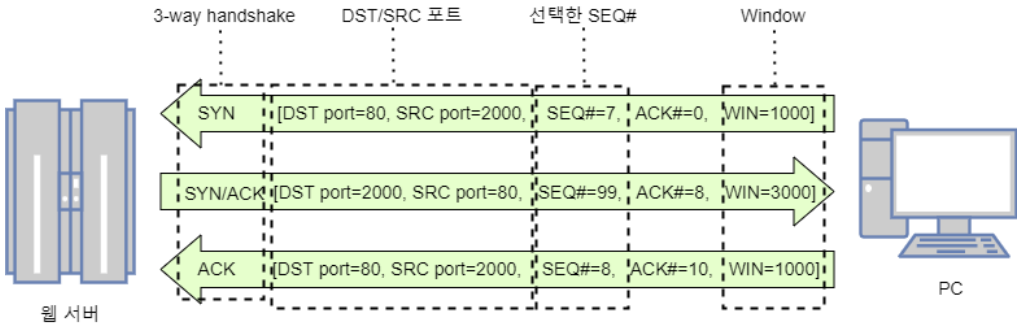


TCP

TCP 헤더

- 플래그
  - 플래그 필드는 SYN, ACK, FIN, RST 비트를 포함한다.
  - TCP는 데이터 전송 전에 플래그 필드를 활용해 데이터가 전송 가능한지를 확인한 후 커넥션을 오픈하고, 커넥션을 유지하고 전송 완료 후에는 커넥션을 종결한다.
  - 이와 같이 커넥션 중심의 프로토콜을 커넥션-오리엔티드(connection-oriented) 프로토콜이라 부른다.

  - 커넥션 오픈
    - » 최초에 PC는 그림과 같이 TCP SYN 세그먼트(플래그의 S에 비트 = 1인 세그먼트)를 보내 원하는 서비스를 제공하는지 묻는다.
    - » 서버는 자신이 HTTP 서비스를 지원할 수 있고, 새로운 커넥션을 위한 메모리(버퍼) 자원 이 있을 때, TCP SYN/ACK 세그멘트(플래그의 SYN과 ACK 비트 = 1인 세그멘트)를 보낸다.
    - » 또한 두 장치는 1부터 시작하는 SEQ#를 사용하는 대신, 커넥션 설정 과정에서 선택한 SEQ#를 사용한다.



TCP

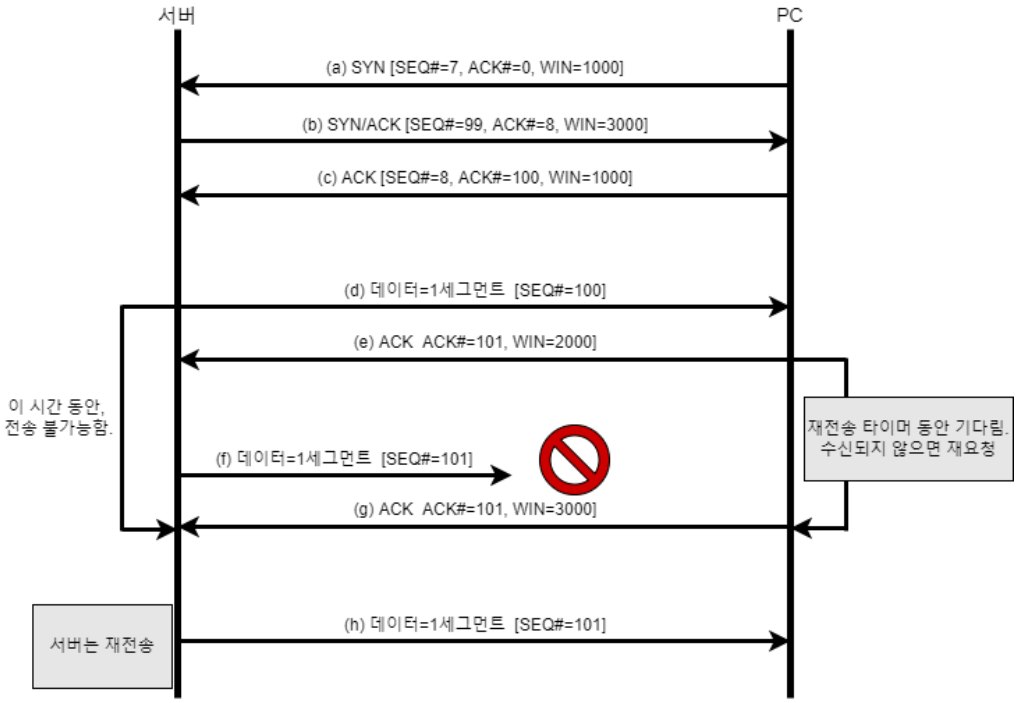
TCP 헤더

- 플래그
  - 커넥션 종료 과정
    - » 커넥션을 시작할 때 커넥션 설정 과정을 거치는 것과 마찬가지로 통신 종료 시에는 FIN과 ACK 세그먼트를 교환해 커넥션 종료 과정을 거친다.
    - » 이를 통해 커넥션에 할당한 메모리를 회수한다.
    - » FIN 세그멘트는 정상적인 종료를 위해 사용하지만, RST 세그멘트는 비정상적인 종료를 위해 사용한다.

TCP

TCP 헤더

- Acknowledgement #(플래그의 ACK비트와 다름)
  - TCP는 SEQ#와 ACK#를 활용해 송신 실패 시 재전송한다.
  - 따라서 TCP를 믿음직한(reliable) 프로토콜이라 한다.
  - TCP의 오류 복구 과정은 그림과 같다.

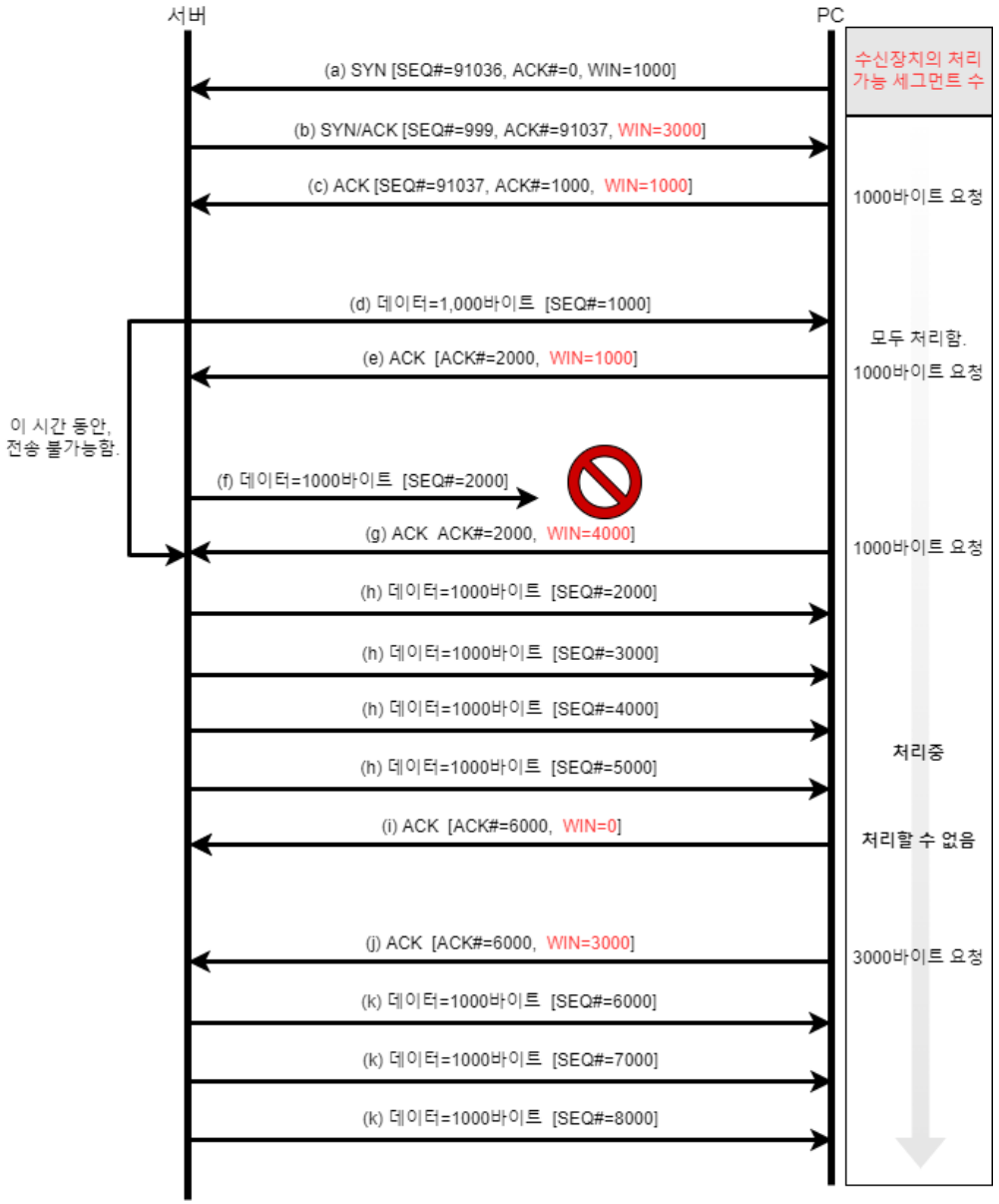


1. 인터넷 통신 모델

TCP

TCP 헤더

- Window size
  - 이 필드를 활용해 한꺼번에 수신할 수 있는 데이터 양을 송신 장치에게 알려준다.
  - 데이터량은 수신 장치의 여건에 따라 바뀌는데, 이를 플로우 컨트롤 기능이라 한다.
  - TCP 플로우 컨트롤 과정은 그림과 같다.



TCP

TCP 헤더

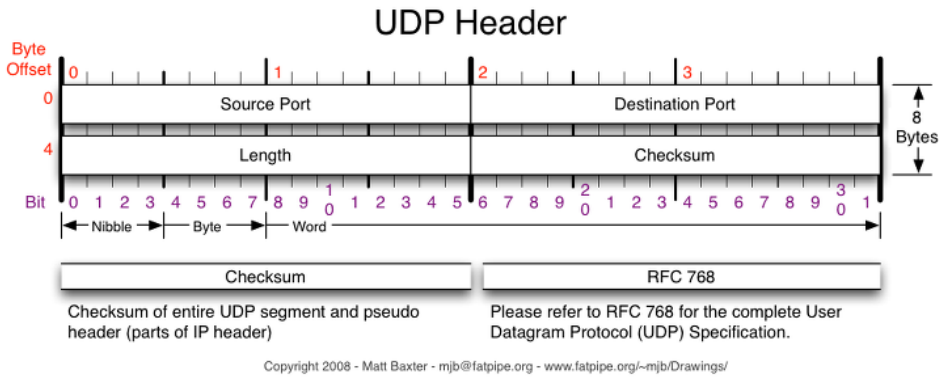
- Checksum
  - 이 필드는 수신한 데이터의 완전성을 체크하기 위해 사용한다.
  - 체크섬 값은 데이터 필드를 입력 값으로 계산되기 때문에 송신 장치와 수신 장치가 계산한 체크섬 값은 동일해야 한다.
  - 수신 장치에서 계산한 체크섬 값이 체크섬 필드에 표시된 체크섬 값과 일치하지 않는다면 세그먼트는 깨진 것이므로 버려지고 ACK 응답을 보내 주지 않는다.
  - 송신 장치는 ACK 응답을 받지 못하므로 재전송해준다.

- Urgent pointer
  - 중요 데이터의 위치를 표시합니다.

1. 인터넷 통신 모델

UDP

- 7계층의 DNS, RTP 등의 메시지들은 코딩 후에 4계층으로 내려간다.
- DNS, RTP 등의 메시지는 TCP 대신 UDP를 사용한다.



UDP

TCP와 UDP 비교

비교 필드	TCP	UDP	설명
SYN/ACK/FIN	O	X	이 필드가 있는 TCP는 커넥션-오리엔티드 프로토콜, 이 필드가 없는 UDP는 커넥션리스 프로토콜입니다.
Sequence # Acknowledgement #	O	X	TCP는 이 필드를 활용해 에러 복구(error control) 기능을 제공합니다. 따라서 TCP는 신뢰성이 있습니다(reliable). 이 필드가 없는 UDP는 에러 복구 기능을 제공할 수 없습 니다. 따라서 UDP는 신뢰성이 없습니다(unreliable).
윈도우 사이즈 (Window)	O	X	TCP는 이 필드를 활용해 플로우 컨트롤 기능을 제공합니다. 이 필드가 없는 UDP는 플로우 컨트롤 기능을 제공할 수 없습니다.
DST/SRC 포트	O	O	앞에서 설명한 이 필드는 반드시 필요한 필드들로 TCP와 UDP도 이 필드들을 가집니다. 따라서 TCP와 UDP 모두 동일 장치에 동일 서비스를 중복적으로 사용할 수 있습니다.

UDP

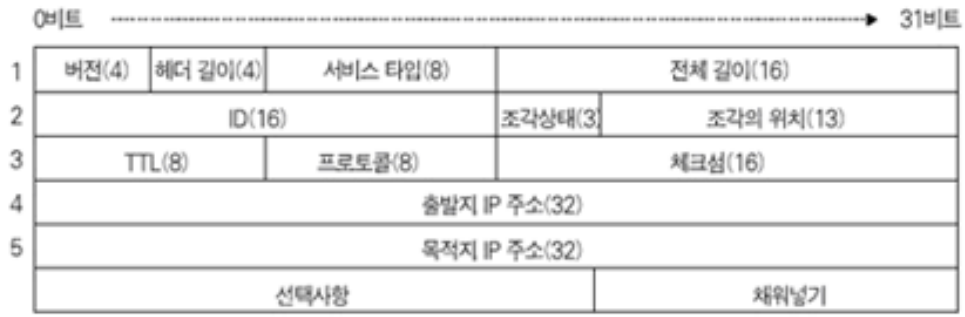
TCP와 UDP를 선택하는 이유

비교 필드	TCP	UDP
① 메시지 길이	긴 경우	짧은 경우
	HTTP/FTP/SMTP	DHCP/SNMP/SYSLOG
② 통신 단말 간의 거리	긴 경우	짧은 경우
	HTTP/FTP/SMTP/텔넷	DHCP/SNMP/SYSLOG/DNS
③ 실시간/비실시간	비실시간	실시간/비실시간
	HTTP/FTP/SMTP	VoIP(Voice over IP)
④ 유니캐스트/멀티캐스트/브로드캐스트	유니캐스트만	유니캐스트/멀티캐스트/브로드캐스트
	HTTP/FTP/SMTP 등 대부분의 애플리케이션 계층 프로토콜	멀티캐스트/브로드캐스트 애플리케이션 계층 프로토콜

1. 인터넷 통신 모델

IP

IP 헤더



- 버전
  - IPv4 인지, IPv6인지를 구분합니다.
- ID, 조각의 위치(Fragmentation Offset)
  - IP 패킷 사이즈가 MTU(Max Transmission Unit)를 초과할 경우에 자른다.
  - ID와 프래그멘테이션 옵셋(Fragmentation Offset) 필드는 순서 번호를 표시하기 위해 사용.
  - TCP와 IP는 자르는 프로토콜이다.
  - TCP 분할은 7계층 장치인 송신 장치에서 일어난다.
  - 중간에는 3계층 장치인 라우터와 2계층 장치인 스위치뿐이기 때문이다.
  - IP 분할은 중간에 거치는 라우터에서 일어날 수 있다.
  - 반면, TCP나 IP 조립은 항상 목적지 장치에서 일어난다.

IP

IP 헤더

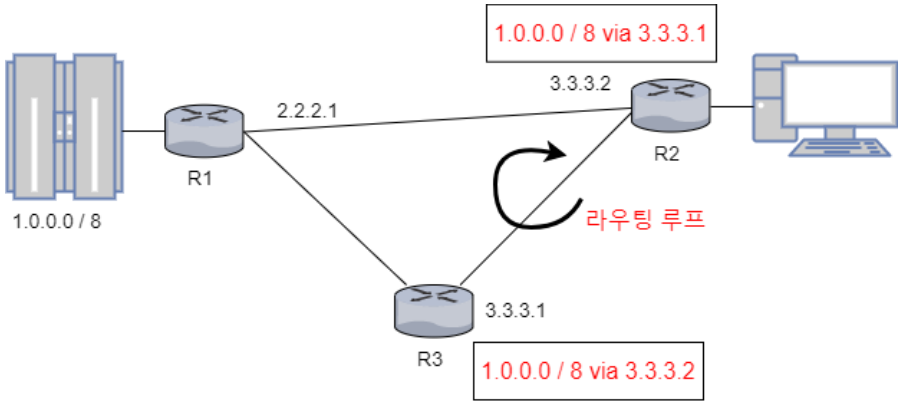
- 조각상태
  - DF와 MF를 포함하는 필드를 플래그 필드라고 한다.

플래그 필드	명칭	설명
첫 번째 필드	항상 '0'	현재는 사용하지 않음
두 번째 필드	DF(Don't Fragment)	'1'로 세팅되면 이 패킷을 분할할 수 없고, 이와 반대로 '0'이면 분할할 수 있음을 의미합니다
세 번째 필드	MF(More Fragment)	'1'이면 마지막 프래그먼트(조각)가 아님 을 '0'이면 마지막 프레그먼트임을 표시 합니다.

IP

IP 헤더

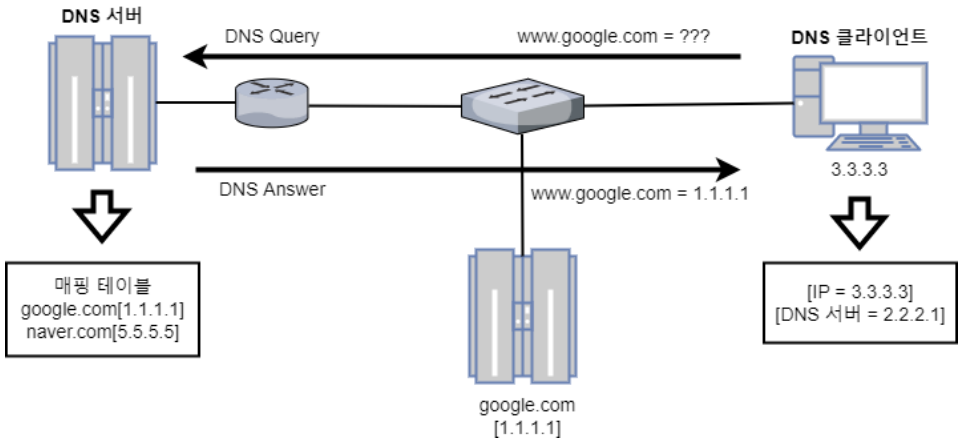
- TTL(Time To Live)
  - TTL 필드는 라우터를 통과할 때마다 '1'씩 카운트 다운된다.
  - 디폴트 값은 OS에 따라 다른데, 라우터는 '255'이다.
  - 그림과 같은 라우팅 루프(routing loop)가 일어나면 두 라우터 사이에서 영원히 순환하면서 CPU, 대역폭과 같은 네트워크 자원을 소모할 것.
  - 이 문제는 네트워크의 성능을 떨어뜨리는 원인이 된다.
  - 이 문제를 해결하는 솔루션이 바로 TTL 필드이다.



IP

IP 헤더

- 프로토콜 ID
  - 상위 계층의 프로토콜을 구분하는 번호.
  - 17이면 UDP, 6이면 TCP를 의미한다.
- 헤더 체크섬
  - 패킷 헤더의 완전성을 검사.
- 출발지 IP 주소
  - 패킷의 출발지 IP 주소가 입력.
- 목적지 IP 주소
  - PC는 사용자가 입력한 도메인 이름(예를 들어, www.google.com)만 안다.
  - 이 문제를 해결하는 서비스가 DNS(Domain Name Service)이다.
  - PC는 그림 66과 같이 DNS 서버에게 DNS 쿼리(query) 패킷을 보내 www.google.com의 IP 주소를 묻고, DNS 서버는 www.google.com의 IP 주소(1.1.1.1)를 포함한 DNS 앤서(answer) 패킷으로 응답한다.
  - 즉, PC는 DNS 서비스에 의해 목적지 IP 주소 자리를 입력할 수 있다.



IP

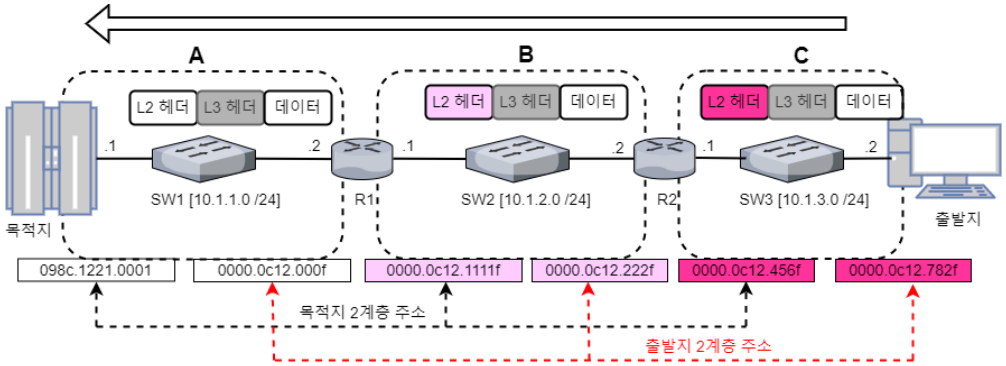
IP 헤더

- ToS(Type of Service)
  - 패킷의 중요도를 표시.
  - 라우터의 인터페이스는 인풋 큐와 아웃풋 큐를 가집니다.
  - 수신 패킷은 인바운드 인터페이스의 인풋 큐에 쌓인다.
  - 인풋 큐에서 대기하던 패킷은 매인 메모리(main memory)로 이동해 라우팅과 옷을 갈아입는 과정을 거친다.
  - 이 후에 송신을 위해 아웃바운드 인터페이스의 아웃풋 큐에서 대기한다.
  - 이때 ToS 값이 높은 패킷이 도착하면 줄의 앞에 새치기를 시켜준다.
  - 이 새치기 서비스를 통해 패킷은 우선 처리되므로 지연을 줄일 수 있다.
  - 이러한 서비스를 QoS(Quality of Service)라 하고, VoIP(Voice of IP)와 같은 실시간 통신이 필요한 서비스에 적용한다.

1. 인터넷 통신 모델

이더넷

- 네트워크마다 다른 2계층 프로토콜을 적용해도 괜찮다.
- 또한 모든 네트워크에 동일한 2계층 프로토콜을 적용한다하더라도 다른 네트워크의 2계층 프로토콜과 독립적으로 동작한다.
- 2계층 주소는 네트워크 내부의 장치들을 구분하기 위해 사용한다.
- 그림은 A, B, C 네트워크 모두에 이더넷 프로토콜을 적용한 경우이다.
- 2계층 헤더는 하나의 네트워크를 통과하기 위한 것이므로 네트워크를 통과할 때마다 갈아입어야 한다.



1. 인터넷 통신 모델

이더넷용 MAC 헤더 생성

- IP 헤더를 만들었으면 IP 담당 부분의 앞에 MAC 헤더(그림)을 붙인다.
- IP 헤더의 수신처 IP 주소에 패킷을 전달하는 목적지가 쓰여있으므로 이것을 보면 패킷을 어디로 운반해야 하는지 판단할 수 있는데, 이더넷에는 TCP/IP 개념이 통용되지 않는다.
- 이더넷은 TCP/IP와 다른 구조로 패킷의 수신처를 판단하며, 이 구조를 따르지 않으면 이더넷 패킷을 운반할 수 없기 때문이다.
- 이더넷의 수신처 판단 구조로 사용하는 것이 MAC 헤더이다.

이더넷 프레임 최소 64바이트 ~ 최대 1,518 바이트

이더넷 헤더 14바이트 | 이더넷 트레일러 4바이트

수신처 MAC주소	송신처 MAC 주소	타입	페이로드	FCS
48비트	48비트	16비트	368~12,000비트	32비트

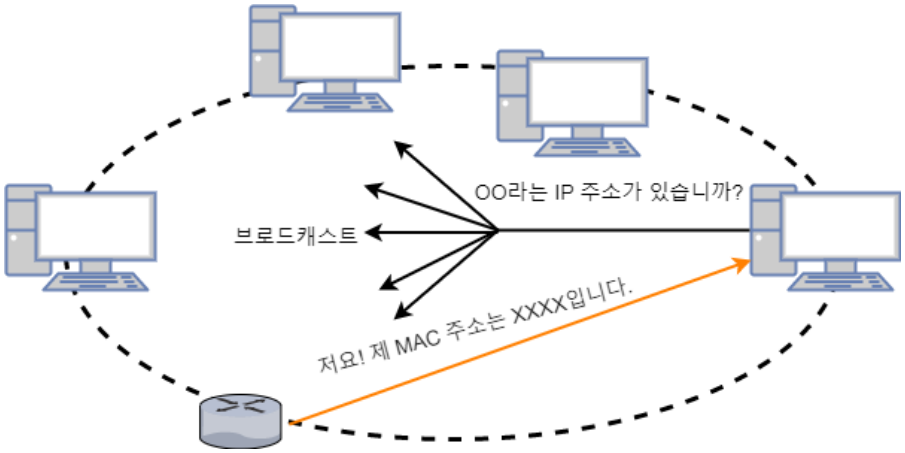
이더넷용 MAC 헤더 생성

MAC 헤더를 만들 때는 이 세 가지 항목에 값을 설정하기만 한다.

- 이더 타입
  - IP 프로토콜을 나타내는 0800(16진 표기)이라는 값을 설정.
- 송신처 MAC 주소
  - 여기에는 자체의 LAN 어댑터의 MAC 주소를 설정.
- 수신처 MAC 주소
  - 패킷을 건네주는 상대의 MAC 주소를 설정하여 이더넷에 의뢰한 후 상대에게 패킷이 전달되므로 여기에는 패 킷을 건네주는 상대의 MAC 주소를 기록해야 한다.
  - 그러나 이 시점에서는 아직 누구에게 패킷을 건네주어야 할지 모르기 때문에 우선 패킷을 건네줄 상대가 누구인지를 조사하는데, 이것은 경로표에 기록되어 있다.
  - 경로표에서 찾은 일치하는 행의 'Gateway' 항목에 기록되어 있는 IP 주소의 기기가 패킷을 건네줄 상대가 된다.
  - 상대의 MAC 주소는 모르기 때문에 IP 주소에서 MAC 주소를 조사하는 동작을 실행한다.

ARP로 수신처 라우터의 MAC 주소 조회

- ARP의 동작방식



- 패킷을 보낼 때마다 이 동작을 하면 ARP의 패킷이 불어나기 때문에 한 번 조사한 결과는 ARP 캐시라는 메모리 영역에 보존하여 다시 이용
- ARP 캐시를 사용해서 ARP의 패킷을 줄일 수 있지만, ARP 캐시에 저장된 MAC 주소를 언제까지나 계속 사용하면 문제가 발생할 수 있기 때문에, ARP 캐시에 저장된 값은 시간이 되면 삭제.

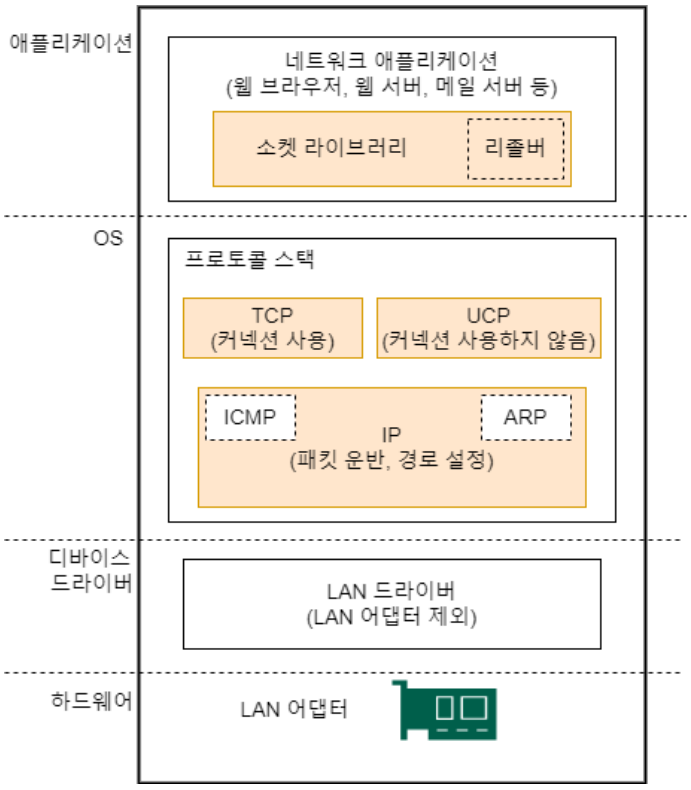
ARP로 수신처 라우터의 MAC 주소 조회

- MAC 헤더를 IP 헤더의 앞에 붙이면 패킷이 완성된다.
- 이렇게 해서 패킷을 만들기까지가 IP 담당 부분의 역할이다.
- MAC 헤더는 이더넷에서 사용하는 것이므로 IP 담당 범위 밖이라고 생각할 수도 있지만, 현실적으로 생각하면 MAC 헤더를 붙여 패킷을 완성하기까지 IP에서 담당한 쪽이 더 좋은 방법이다.
- LAN 어댑터에 건네주기 전에 IP 담당 부분에서 패킷을 완성하면 LAN 어댑터는 완성된 패킷만 송신하면 되기 때문에, IP 이외의 특수한 패킷도 한 개의 LAN 어댑터로 대응할 수 있다.

2. 소켓 생성

프로토콜 스택의 내부 구성

- 프로토콜 스택의 내부는 그림과 같이 역할이 서로 다른 몇 부분으로 나뉘어져 있다.
- 애플리케이션에 따라 송 수신하는 데이터의 내용은 다르지만, 데이터를 송 · 수신할 때의 동작은 공통이므로 애플리케이션에 따른 차이는 없다. 애플리케이션의 아랫부분에는 소켓 라이브러리가 있으며, 그 안에는 리졸버가 내장되어 있다.
- 이것이 DNS 서버에 조회하는 동작을 실행한다.
- 브라우저나 메일 등의 일반적인 애플리케이션은 TCP를 사용하여 데이터를 송 · 수신한다.
- DNS 서버에 대한 조회 등에서 짧은 제어용 데이터를 송 · 수신하는 경우에는 UDP를 사용한다.
- 인터넷에서 데이터를 운반할 때는 데이터를 작게 나누어 패킷이라는 형태로 운반하는데, 이 패킷을 통신 상대까지 운반하는 것이 IP의 주 역할이다.
- LAN 어댑터가 실제 송 · 수신 동작, 즉 케이블에 대해 신호를 송 · 수신하는 동작을 실행



2. 소켓 생성

소켓의 실체

- 프로토콜 스택은 내부에 제어 정보를 기록하는 메모리 영역을 가지고 있으며, 여기에 통신 동작을 제어하기 위한 제어 정보를 기록한다.
- 본래 소켓은 개념적인 것이어서 실체가 없으므로 굳이 말하자면 이 제어 정보가 소켓의 실체라고 할 수 있다.
- 프로토콜 스택은 이 소켓을 참조하면서 동작한다.
- 예를 들어 데이터를 송신할 때는 소켓에 기록되어 있는 상대측의 IP 주소나 포트 번호를 보고 그 IP 주소와 포트 번호를 대상으로 데이터를 송신한다.
- 이외에도 소켓에는 통신 동작을 제어하기 위한 여러 가지 제어 정보가 기록되어 있다.
- 프로토콜 스택은 이것을 참조하여 다음에 무엇을 해야 하는지를 판단하는데, 이것이 소켓의 역할이다.

2. 소켓 생성

Socket을 호출했을 때의 동작

- 애플리케이션이 socket 함수를 호출하여 소켓을 만들 것을 의뢰하면 프로토콜 스택은 의뢰에 따라 한 개의 소켓을 생성.
- 소켓은 작성된 직후에는 아직 송 수신 동작이 시작되지 않은 초기 상태이므로 초기 상태임을 나타내는 제어 정보를 소켓의 메모리 영역에 기록하는데 이 과정을 통해 소켓이 만들어진다.
- 소켓이 만들어지면 소켓을 나타내는 디스크립터를 애플리케이션에 알려준다.
- 디스크립터는 프로토콜 스택의 내부에 있는 다수의 소켓 중 어느 것을 가리키는지를 나타내는 번호표와 같은 정보이다.
- 디스크립터를 받은 애플리케이션은 이후 프로토콜 스택에 데이터 송 · 수신 동작을 의뢰할 때 디스크립터를 호출한다.

3. 서버 접속

접속의 의미

- 소켓을 만들면 애플리케이션(브라우저)은 connect 함수를 호출한다.
- 그러면 프로토콜 스택은 자기쪽의 소켓을 서버측 소켓에 연결한다.
- 이더넷이나 통신 회선은 항상 케이블이 연결되어 있으므로 언제나 신호를 보낼 수 있다.
- 그러므로 데이터를 신호로 변환하여 송신하기만 하면 언제든지 통신이 가능하다.
- 그러나 이 시점에서 즉 소켓을 만든 직후에 애플리케이션에서 데이터 송신 의뢰가 오면 프로토콜 스택은 어떻게 될까?
- 소켓을 만든 직후는 아직 거기에 아무것도 기록되어 있지 않으므로 통신 상대가 누구인지도 모른다.
- 그러므로 서버의 IP 주소나 포트 번호를 프로토콜 스택에 알리는 동작이 필요한데, 이것이 접속 동작의 한 가지 역할이다.
- 그럼 서버 측은 어떨까?
- 서버 측에도 소켓이 만들어졌지만 서버 측의 프로토콜 스택도 클라이언트측과 마찬가지로 소켓을 만드는 동작만으로는 통신상대를 알 수 없다.
- 따라서, 클라이언트측에서 서버 측에 통신 동작의 개시를 전달하는 것도 접속 동작의 역할 중 하나이다.

3. 서버 접속

접속의 의미

- 접속 동작의 첫 번째 동작은 통신 상대와의 사이에 제어 정보를 주고받아 소켓에 필요한 정보를 기록하고 데이터 송 · 수신이 가능한 상태로 만드는 것이다.
- 데이터 송 · 수신 동작을 실행할 때는 송 · 수신하는 데이터를 일시적으로 저장하는 메모리 영역이 필요한데 이 메모리 영역을 '버퍼 메모리'라고 부른다.
- 버퍼 메모리의 확보도 접속 동작을 할 때 실행되는데, 이것이 '접속' 한다는 동작의 의미이다.

3. 서버 접속

TCP/IP 헤더

- 접속 동작뿐만 아니라 데이터를 송 · 수신하는 동작이나 연결을 끊는 동작도 포함하여 통신 동작 전체에서 어떤 정보가 필요한지 검토하여 내용을 TCP 프로토콜의 사양으로 규정하고 있다.
- 이 제어 정보를 저장하는 것이 헤더이다.
- 제어 정보는 소켓에 기록하여 프로토콜 스택의 동작을 제어하기 위한 정보가 더 있다.
- 소켓에 기록한 제어 정보는 상대측에서 볼 수 없다.
- 왜냐하면 규칙에 따라 헤더에 제어 정보를 기록하여 대화하면 그것으로 클라이언트와 서버가 서로 연락을 취하기 때문이다.
- 소켓에 기록하는 제어 정보는 프로토콜 스택을 만드는 사람에 따라 달라지므로 간단히 설명할 수 없다.
- 그러나 소켓의 제어 정보 중에서 중요한 것은 명령에 의해 표시할 수 있는데, 이것은 어느 os의 프로토콜 스택에도 공통이다.

3. 서버 접속

접속동작의 실제

- 애플리케이션이 Socket 라이브러리의 connect를 호출하는 것부터 접속 동작이 시작된다
  - connect(<디스크립터>, <서버측의 IP 주소와 포트 번호>, ...)
- 서버 측의 IP 주소와 포트 번호를 쓰면 명령이 프로토콜 스택의 TCP 담당 부분에 전달된다.
- 그러면 TCP 담당 부분은 IP 주소로 표시된 상대, 즉 서버의 TCP 담당 부분과의 사이에 제어 정보를 주고받는다.
- 이를 위해, 먼저 데이터 송 · 수신 동작의 개시를 나타내는 제어 정보를 기록한 헤더를 만든다.
- 헤더에는 다수의 항목이 있는데, 여기에서 중요한 것은 송신처와 수신처의 포트 번호이다.
- 이를 통해 송신처가 되는 클라이언트측의 소켓과 수신처가 되는 서버측의 소켓을 지정할 수 있다.

3. 서버 접속

접속동작의 실제

- 이렇게 해서 TCP 헤더를 만들면 이것을 IP 담당 부분에 건네주어 송신하도록 의뢰한다.
- 그러면 IP 담당 부분이 패킷 송신 동작을 실행하고 네트워크를 통해 패킷이 서버에 도착하면 서버측의 IP 담당 부분이 이것을 받아 TCP 담당 부분에 건네준다.
- 이후 서버측의 TCP 담당 부분이 TCP 헤더를 조사하여 기록되어 있는 수신처 포트 번호에 해당하는 소켓을 찾아낸다.
- 해당하는 소켓이 발견되면 여기에 필요한 정보를 기록하고 접속 동작이 진행중이라는 상태가 된다.
- 이 과정이 끝나면 서버의 TCP 담당 부분은 응답을 돌려 보낸다.
- 클라이언트와 마찬가지로 송신처와 수신처의 포트 번호나 SYN 비트 등을 설정한 TCP 헤더를 만든다.
  - 그리고 응답을 돌려보낼 때 ACK라는 컨트롤 비트도 1로 만든다.
  - 이것은 패킷을 받은 것을 알리기 위한 동작이다.
- 그리고 TCP 헤더를 IP 담당부분에 건네주어 클라이언트에 반송하도록 의뢰한다.
- 그러면 패킷이 클라이언트에 돌아오고 IP 담당 부분을 경유하여 TCP 담당 부분에 도착한다.
- 이때 TCP 헤더를 조사하여 서버측의 접속 동작이 성공했는지 확인한다.

3. 서버 접속

접속동작의 실제

- 이로써 클라이언트측은 끝나지만 한 가지 작업이 더 남아있다.
- 패킷이 도착한 것을 서버에 알리기 위해 ACK 비트를 1로 만든 TCP 헤더를 반송한다.
- 그리고 이것이 서버에 도착하면 접속 동작의 대화가 끝난다.
- 이로써 소켓은 데이터를 송 · 수신할 수 있는 상태가 된다.
- 이때 파이프와 같은 것으로 소켓이 연결되었다고 생각할 수 있다.
- 실제로는 무언가로 연결되어 있지 않지만, 이렇게 생각하는 것이 네트워크 업계의 관습이다.
- 이 파이프와 같은 것을 커넥션이라고 한다.
- 또한 커넥션은 데이터 송 · 수신 동작을 계속하고 있는 동안, 즉 close 함수를 호출하여 연결을 끊을 때까지 계속 존재한다.
- 이렇게 해서 커넥션이 이루어지면 프로토콜 스택의 접속동작이 끝나므로 connect 함수의 실행이 끝나면서 애플리케이션을 제어할 수 있게 된다.

4. 데이터 송·수신

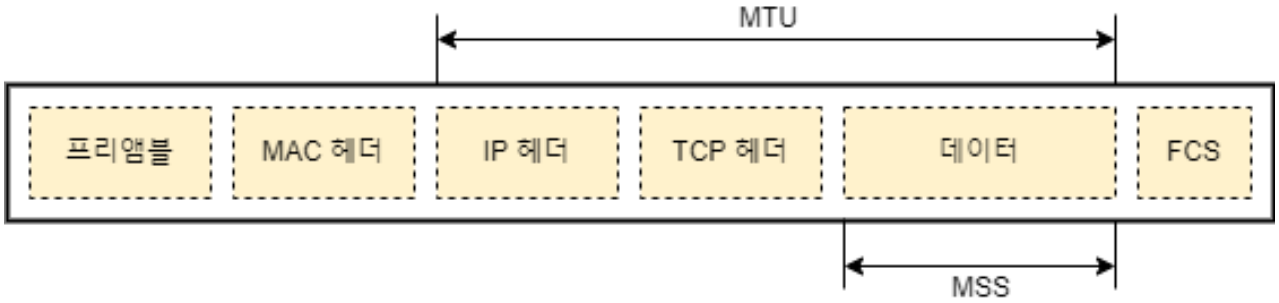
프로토콜 스택에 HTTP 요청 메시지 전달

- 데이터 송 · 수신 동작은 애플리케이션이 write 함수를 호출하여 송신 데이터를 프로토콜 스택에 건네주는 것부터 시작된다.
- 이것을 받은 프로토콜 스택이 송신 동작을 실행한다는 순서로 진행되는데, 이 동작에는 몇 가지 중요한 점이 있다.
- 먼저 프로토콜 스택은 받은 데이터의 내용에 무엇이 쓰여있는지 알지 못한다.
- write 함수를 호출할 때 송신 데이터의 길이를 지정하지만, 프로토콜 스택은 해당 길이만큼만 바이너리 데이터가 1바이트씩 차례로 나열되어 있다고 인식할 뿐이다.
- 프로토콜 스택은 받은 데이터를 곧바로 송신하는 것이 아니라 일단 자체의 내 부에 있는 송신용 버퍼 메모리 영역에 저장하고, 애플리케이션이 다음 데이터를 건네주기를 기다린다.
- 데이터를 일단 저장하는 데는 이유가 있다.

4. 데이터 송·수신

프로토콜 스택에 HTTP 요청 메시지 전달

- 송신을 의뢰할 때 애플리케이션에서 프로토콜 스택에 건네주는 데이터의 길이는 애플리케이션의 종류나 만드는 방법에 따라 결정된다.
- 한 번의 송신 의뢰에서 건네주는 데이터의 길이는 애플리케이션의 사정에 따라 결정되며, 프로토콜 스택에서 제어할 수 없다.
- 이러한 상황에서 받은 데이터를 곧바로 보내는 단순한 방법이라면 작은 패킷을 많이 보낼 수도 있다.
- 그렇다면 네트워크의 이용 효율이 저하되므로 어느 정도 데이터를 저장하고 나서 송 · 수신 동작을 한다.
- 프로토콜 스택은 MTU라는 매개변수를 바탕으로 저장할 데이터의 길이를 판단한다.
- MTU는 한 패킷으로 운반할 수 있는 디지털 데이터의 최대 길이로, 이더넷에서는 보통 1,500바이트가 된다(그림).



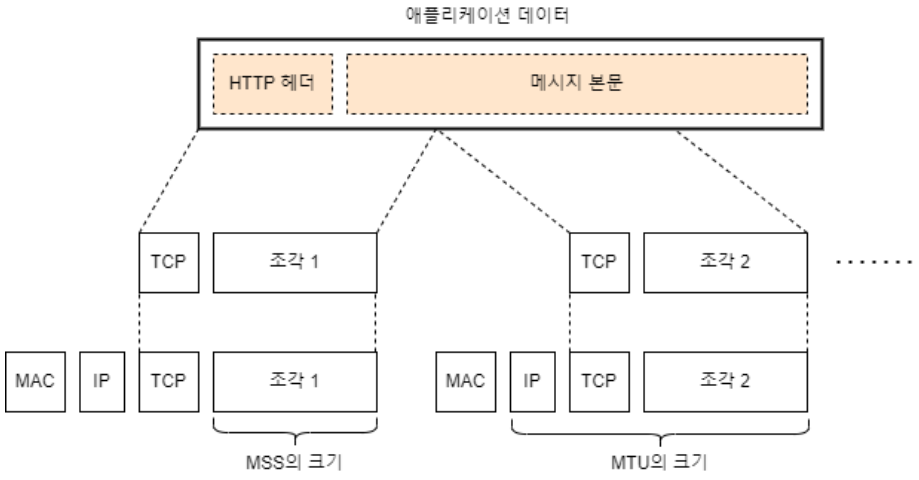
4. 데이터 송·수신

프로토콜 스택에 HTTP 요청 메시지 전달

- 저장할 데이터의 길이를 판단하는 또 한가지 요소는 타이밍이다.
- 애플리케이션의 송신 속도가 느려지는 경우 MSS에 가깝게 데이터를 저장하면 여기에서 시간이 걸려 송신 동작이 지연되므로 버퍼에 데이터가 모이지 않아도 적당한 곳에서 송신 동작을 실행해야 한다.
- 따라서 프로토콜 스택은 내부에 타이머가 있어서 이것으로 일정시간 이상 경과하면 패킷을 송신한다.
- 프로토콜 스택에만 맡긴다면 좋지 않은 일이 생길 수도 있으므로 애플리케이션측에서 송신의 타이밍을 제어하는 여지도 남겨두었다.
- 즉, 데이터 송신을 의뢰할 때 옵션을 지정할 수 있으며 여기에서 '버퍼에 머물지 않고 바로 송신할 것'이라고 지정하면 프로토콜 스택은 버퍼에 머물지 않고 송신 동작을 실행한다.

데이터의 분할 전송

- HTTP 요청 메시지는 보통 그다지 길지 않으므로 한 개의 패킷에 들어가지만, 폼을 사용하여 긴 데이터를 보낼 경우 등 한 개의 패킷에 들어가지 않을 만큼 긴 것도 있다.
- 이 경우 송신 버퍼에 저장된 데이터는 MSS의 길이를 초과하므로 다음 데이터를 기다릴 필요가 없다.
- 따라서 송신 버퍼에 들어있는 데이터를 맨 앞부터 차례대로 MSS의 크기에 맞게 분할하고 분할한 조각을 한 개씩 패킷에 넣어 송신한다.
- 이렇게 해서 송신 버퍼에 저장한 데이터 조각의 맨 앞부분에 TCP 헤더를 부가한다.
- 그리고 IP 담당 부분에 건네주어 송신 동작을 실행한다(그림).



HTTP 응답 메시지

- 브라우저의 의뢰를 받아 프로토콜 스택이 HTTP 요청 메시지를 보내면 다음에는 웹 서버에서 응답 메시지가 돌아오기를 기다리고, 응답 메시지가 돌아오면 그것을 수신한다.
- 브라우저는 서버에서 돌아오는 응답 메시지를 받기 위해 read 함수를 호출한다.
- 그러면 read를 경유하여 프로토콜 스택에 제어가 넘어가고, 프로토콜 스택이 움직이기 시작한다.
- 데이터를 수신할 때도 데이터를 송신할 때와 마찬가지로 데이터를 일시 보관하는 수신 버퍼를 사용한다.
- 프로토콜 스택은 의뢰받은 작업 즉 수신 버퍼에서 수신 데이터를 추출하여 애플리케이션에 건네주는 작업을 잠시 보류했다가 서버에서 응답 메시지의 패킷이 도착했을 때 그것을 수신하여 애플리케이션에 건네주는 작업을 재개한다.
- 프로토콜 스택은 먼저 수신한 데이터 조각과 TCP 헤더의 내용을 조사하여 도중에 데이터가 누락되었는지 검사하고, 문제가 없으면 ACK 번호를 반송한다.
- 그리고 데이터 조각을 수신 버퍼에 일시 보관하고 조각을 연결하여 데이터를 원래 모습으로 복원한 후 애플리케이션에 건네준다.

5. 서버와의 연결 종료

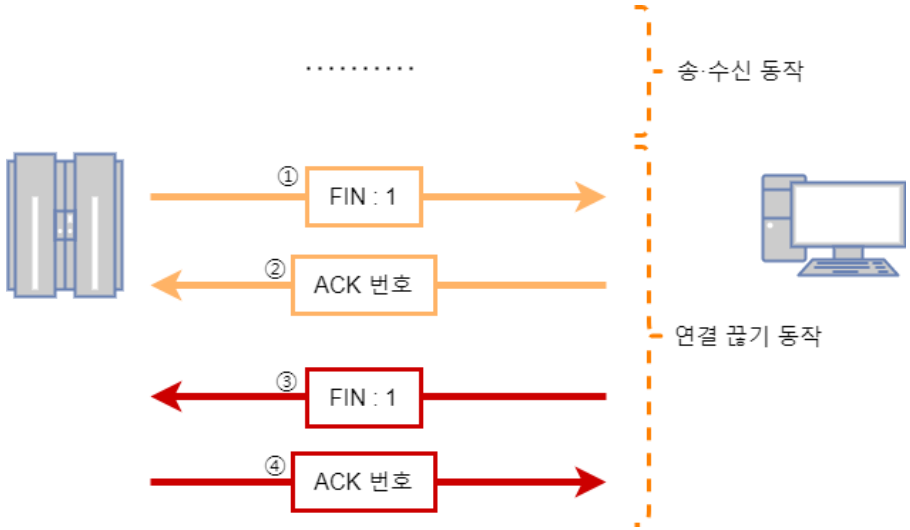
전송 완료 후 연결 종료

- 데이터 송 · 수신을 종료하는 것은 애플리케이션이 송신해야 하는 데이터를 전부 송신 완료했다고 판단했을 때이다.
- 그러면 송신을 완료한 측이 연결 끊기 단계로 들어가는데 어디에서 데이터 송 · 수신 동작이 끝나는지는 애플리케이션에 따라 다르다.
- 연결 종료는 애플리케이션에 따라 다르므로 프로토콜 스택은 어느 쪽에서 먼저 연결 끊기 단계에 들어가도 좋게 만들어져 있다.
- 데이터 보내기를 완료한 쪽에서 연결 끊기 단계에 들어가는데, 여기에서는 서버 측에서 연결 끊기 단계에 들어가는 것으로 가정하자.
- 이 경우 서버측의 애플리케이션이 먼저 Socket 라이브러리의 close 함수를 호출한다.
- 그러면 서버측의 프로토콜 스택이 TCP 헤더를 만들고, 여기에 연결 끊기를 나타내는 정보를 설정한다. (그림 ①).
- 이와 동시에 서버측의 소켓에 연결 끊기 동작에 들어갔다는 정보를 기록한다.

5. 서버와의 연결 종료

전송 완료 후 연결 종료

- 다음은 클라이언트측이다.
- 서버에서 FIN에 1을 설정한 TCP 헤더가 도착하면 클라이언트측의 프로토콜 스택은 자신의 소켓에 서버측이 연결 끊기 동작에 들어갔다는 것을 기록한다.
- 그리고 FIN을 1로 설정한 패킷을 받은 사실을 알리기 위해 ACK 번호를 서버측에 반송하고(그림 ②), 이것이 끝나면 애플리케이션이 데이터를 가지러 올 때까지 기다린다.



5. 서버와의 연결 종료

전송 완료 후 연결 종료

- 잠시 후 애플리케이션이 read 함수를 호출하여 데이터를 가지러 오면 데이터를 건네지 않고, 서버에서 보낸 데이터를 전부 수신 완료했다는 사실을 클라이언트측의 애플리케이션(브라우저)에게 알린다.
- 클라이언트측의 애플리케이션은 close를 호출하여 데이터 송 · 수신 동작을 끝낸다.
- 그러면 클라이언트측의 프로토콜 스택은 서버측과 마찬가지로 FIN 비트에 1을 설정한 TCP 헤더를 만들고 IP 담당 부분에 의뢰하여 서버에 송신한 후(그림 ③) 서버에서 ACK 번호가 돌아오면 (그림 ④) 서버와의 대화가 끝난다.

5. 서버와의 연결 종료

소켓 말소

- 서버와의 대화가 끝나면 소켓을 사용하여 서버와 대화할 수 없게 된다.
- 이때 소켓은 필요 없지만, 거기서 바로 소켓을 말소하지 않고 잠시 기다린 후 소켓을 말소한다.
- 앞의 그림의 설명과 반대로 클라이언트에서 연결 끊기 동작이 시작되어 다음과 같은 식으로 연결 끊기 동작을 계속하고 끝으로 클라이언트가 송신한 ACK 번호가 말소된다.
  - ① 클라이언트가 FIN 송신
  - ② 서버가 ACK 번호 송신
  - ③ 서버가 FIN 송신
  - ④ 클라이언트가 ACK 번호 송신
- 이 경우 서버는 ACK 번호가 돌아오지 않으므로 다시 한 번 FIN을 보낼 수도 있다.
- 이때 클라이언트의 소켓이 말소되어 있으면 어떻게 될까?
- 소켓을 말소하면 거기에 기록되어 있던 제어 정보가 없어지므로 소켓에 할당되어 있던 포트 번호도 몇 번인지 알 수 없게 된다.
- 명확한 규정은 없지만 일반적으로 보통 몇 분 정도 기다리고 나서 소켓을 말소한다.

Thank You