네트워크 컴퓨팅

네트워크 컴퓨팅

- **내용**
  - **네트워크 프로그래밍**
    - Unix C Version
    - C# Version
- **참고서적**
  - **컴퓨터 네트워크프로그래밍**-unix version
  - **비주얼C#프로그래밍**
- **평가**
  - **중간 , 기말 : 각** 30%
  - **리포트** : 15%
  - **출석, 평소** : 25%

강의 내용

- TCP/IP **프로토콜 개요**
- **네트워크 프로그램 설계 기초**
  - **네트워크 프로토콜** : OSI 7 **계층**, TCP/IP 4 **계층**
  - C/S **통신모델** : 2-tier, 3-tier, n-tier, P2P
  - **서버 구축 방법** : **연결/비연결**, stateful/stateless, iterative/concurrent
- **소켓 프로그래밍 기초**
  - TCP, UDP, **네트워크 프로그램 작성 방법**
- **다중처리 기술**
  - Multitasking : multi-process, multi-thread
  - Multiplexing : poling, selecting, interrupt
- **시그널, 프로세스간 통신**
- **데몬 서버 구축 방법, 멀티스레드 서버 구축**
- Raw **소켓 프로그래밍, 패킷 캡쳐 프로그래밍**
- C# **네트워크 프로그래밍 –기본, 응용**

0. TCP/IP 프로토콜

- 인터넷에서 사용하는 표준 프로토콜
  - 네트워크 액세스 계층
  - 인터넷 계층
  - 전송 계층
  - 응용 계층

0.1 네트워크 액세스 계층

- IP 데이터그램을 실제로 전달하는 서브네트워크
- 이더넷 : 가장 널리 사용되는 서브네트워크
- PPP : ADSL 등에서 사용

이더넷 (이더넷 프레임 구조)

- **이더넷 프레임 구조**

Preamble,SFD	DA	SA	Type	Info	FCS

  - Preamble(7**바이트**)
    - **수신측에서 하드웨어 비트 동기를 맞추기 위한 준비 신호**
  - SFD(Starting Frame Delimiter, 1**바이트**)
    - **다음 바이트 열이 프레임의 시작을 알림**
  - DA(Destination Address, 6**바이트**)
    - **목적지** MAC**주소**
    - **앞의** 3**바이트는 카드회사 식별코드이며 나머지** 3**바이트는 개별적인 목적지 식별용** NIC**카드임**
  - SA(Source Address, 6**바이트**)
    - **송신측** MAC**주소**
    - **자신의 주소를** ROM**에 기록, 초기화 시** ROM**에서 읽어 레지스터에 저장**
    - **프레임 송신 시 이 레지스터를 읽어 프레임의** SA**에 삽입**

이더넷(이더넷 프레임 구조)

- EtherType (2바이트)
  - MAC 프레임 다음의 데이터 부분에 상위 프로토콜의 종류를 표시
  - 0x0800 : Info에 있는 데이터의 프로토콜이 IP
  - 0x0806 : ARP 프로토콜
- Data(46~1500바이트)
  - 상위 프로토콜 데이터가 위치
  - 최대 허용길이(MTU)는 1500바이트
  - DA부터 FCS까지 전체길이가 64바이트 이상이어야 한다는 규정을 준수
- PAD
  - 최소길이 규정을 만족하지 못할 경우 '0'으로 채운다.
- FCS(Frame Check Sequence, 4바이트)
  - 프리앰블과 SFD를 제외한 MAC프레임의 비트열의 에러를 검사

이더넷

- MAC 주소 구조

0	0	기관별 ID	기관에서 배정하는 카드 ID

  - 처음 두 비트 : 항상 0
  - 다음 22비트 : LAN 카드 제조 회사별로 할당된 ID
  - 뒤의 24비트 : LAN 카드마다 유일하게 제조사가 부여하는 번호
- 이더넷 특징
  - 사용 프로토콜
  - 케이블
  - 스위칭 이더넷

이데넷(이더넷 특징)

- **사용 프로토콜**
  - CSMA/CD(Carrier Sense Multiple Access / Collision Detection)
    - **접근 제어**(MAC) **프로토콜로 사용함**
    - **프레임을 전송하려는 스테이션은 먼저 채널감지**(CS : Carrier Sense)**를 통해 다른 장비가 전송 중인지 확인함**
    - **채널이 사용 중이지 아닌 때에만 전송을 시도**
    - **프레임 전송 중에 다른 장비가 전송을 하여 충돌이 나는지 감시**(Collision Detection)
    - **충돌이 나면 즉시 전송을 중지**

이데넷(이더넷 특징)

- **케이블**
  - category 5 UTP(Unshielded Twisted Pair) **케이블을 널리 사용**
  - 4 pair(8**가닥**)**로 되어 있으며** 10Mbps **이더넷**(10BaseT)**에서는 두** pair**만 사용**
  - **동작속도** 100Mbps**인 고속 이더넷**(100Base4T)**에서는** 4 pair**를 모두 사용**
    - **한** pair**는 충돌감지용, 나머지 세** pair**는 각각** 33.3Mbps **속도로 데이터 전송**

이더넷 (이더넷 특징)

- **스위칭 이더넷**
  - **일반 허브를 스위칭 허브로 교체하여 허브에 연결된 장비가 각각 10Mbps 대역을 모두 사용**
  - **스위칭 이더넷의 속도를 증설하여 채널 속도 100Mbps인 것이 널리 사용**
  - **현재는 이를 개선한 1Gbps 또는 10Gbps의 확장된 기가비트 이더넷 사용**

PPP

- PPP **프레임 구조**
  - PPP(Point-to-Point Protocol)**는 전화회선**, xDSL(Digital Subscriber Line) **같은 일 대 일로 연결된 회선을 통해** IP **데이터 그램을 송수신하는 통신 프로토콜**

1	1	1	2	0~1500	2	1
Flag (7E)	**Addr (FF)**	**Control (03)**	**Protocol**	**info**	**FCS**	**Flag (7E)**

  - **프레임구조**
    - **프레임의 시작과 끝은 플래그로** 0x7E**를 사용**
    - Address**와** Control**은 각각** 0xFF**와** 0x03**으로 고정**
    - Protocol **필드는 상위 계층에 전달하는 정보**(Info)**의 종류 구분에 사용 예를 들어** 0x0021**는** IP **데이터그램**, 0x002B**는** IPX(Internet Packet Exchange) **패킷**
    - FCS(Frame Check Sequence)**는 프레임 전송 중 발생하는 비트 에러를 검출**

기타 서브네트워크

- X.25 패킷 교환망
  - OSI 표준 1-3계층을 따르는 패킷 교환 프로토콜
  - 주로 공중 데이터 통신망에서 사용
- ISDN
  - 전화망과 패킷 교환 방식의 데이터 통신망을 하나의 네트워크에서 서비스하는 네트워크
  - 내부동작은 X.25와 유사한 패킷 교환 방식을 사용
  - 외형적으로는 디지털 회선 교환 서비스를 통합하여 제공
- ATM망
  - 고속 패킷 교환 기술
  - X.25와 달리 OSI 계층 3이 아닌 물리 계층에서 직접 패킷 스위칭 수행
  - 일반 가입자 전송용이 아닌 교환기 사이의 고속 채널 교환 방식으로 사용
  - 모든 데이터를 53바이트의 고정된 길이의 셀 단위로 전송 및 교환
    - 교환기를 간단하고 빠르게 만들 수 있음
    - 여러 종류의 트래픽을 수용하기에 편리

기타 서브네트워크

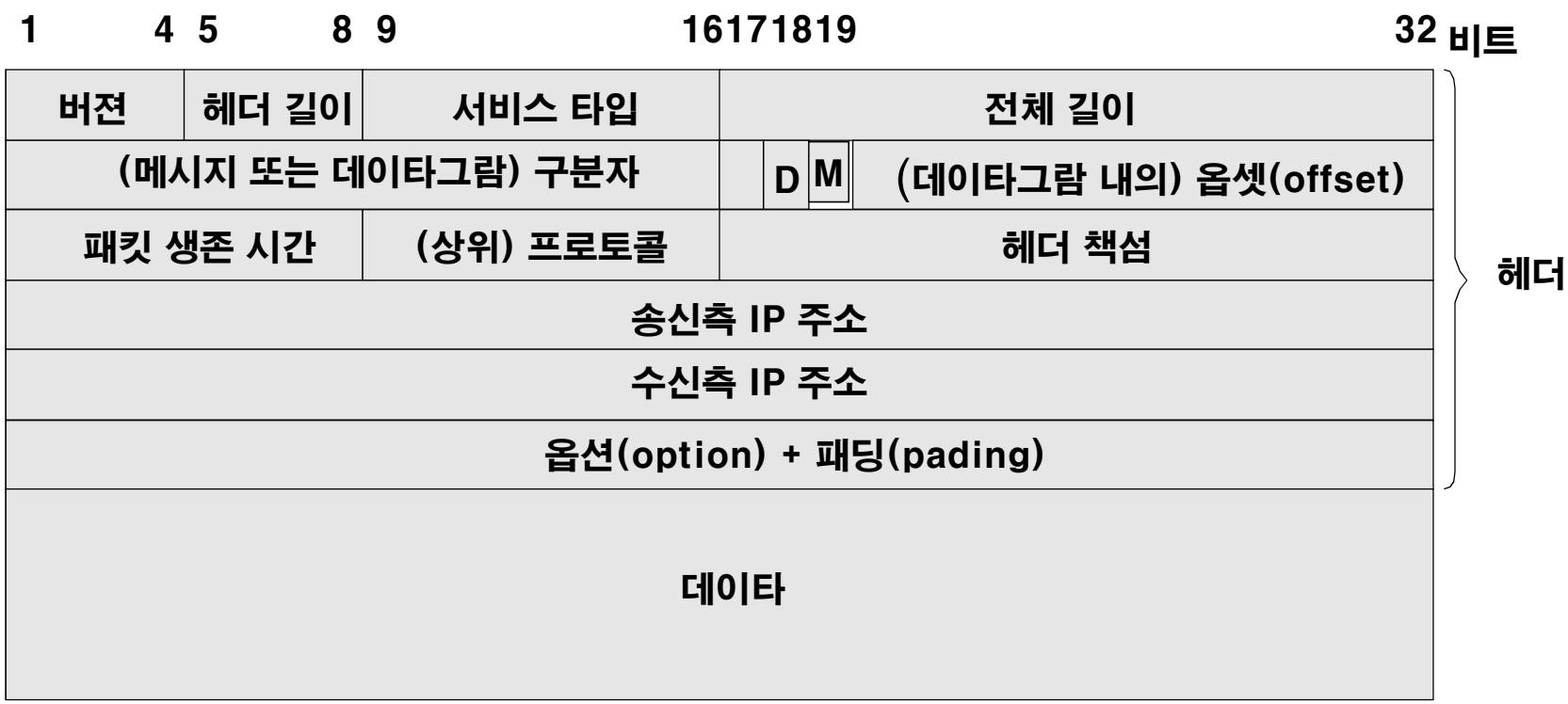
- 무선전화망
  - 무선 전화망을 통해 인터넷에 접속하는 것을 무선 인터넷이라 한다.
  - 유선 채널에 비해 대역폭이 좁고 채널 비용이 비싸다.
  - 통신량을 줄이는 것이 중요
  - 핸드폰, 소형 정보기, 노트북에서 사용

0.2 IP **프로토콜**

IP 프로토콜의 특징

- OSI 7계층의 관점에서 보면 IP계층은 주소와 라우팅을 담당하는 OSI 계층 3에 해당
- IP 계층은 서브네트워크를 이용해 IP 데이터그램을 임의의 호스트 사이에 전달하는 기능
- 임의의 호스트를 4바이트의 IP 주소만으로 찾고 데이터그램을 전달
- 비연결 방식으로 데이터그램을 전달
- 데이터그램의 분실, 중복, 전달 순서 바뀜, 비트 에러 등이 발생해도 IP 계층에서는 데이터그램 에러 확인이나 재전송 등을 하지 않는다.
- 데이터그램을 단순히 목적지로 전달만 하고 전송 결과 확인은 하지 않는다.

IP 데이터그램 구조


1 4 5 8 9 16 17 18 19 32 비트
버전
헤더 길이
서비스 타입
전체 길이
(메시지 또는 데이타그람) 구분자
D
M
(데이타그람 내의) 옵셋(offset)
패킷 생존 시간
(상위) 프로토콜
헤더 책섬
송신측 IP 주소
수신측 IP 주소
옵션(option) + 패딩(pading)
데이타
헤더

IP 데이터그램 구조(IP 데이터그램 헤더)

- Ver
  - IP 버전 값을 표시하며 현재는 4의 값을 가짐
- IHL(Internet Header Length)
  - 헤더 길이를 4바이트 단위로 표시
  - 최소 값은 5 (헤더의 최소 크기는 20 바이트)
- Service Type
  - 서비스 클래스 지정
  - 보통 0으로 지정

- Total Length
  - 헤더를 포함한 IP 데이터그램의 전체 크기를 바이트 단위로 나타냄
  - 16비트로 표현하므로 최대 값은 65535
- Identification
  - 상위 계층이 전송한 큰 메시지가 여러 데이터그램으로 단편화 되어 전송되었을 때 수신 측에서 원래 메시지를 재구성할 때 사용하는 번호
  - 한 메시지를 구성하는 데이터그램들은 모두 같은 Identification 값을 갖는다.

IP 데이터그램 구조(IP 데이터그램 헤더)

- Flag
  - **첫 번째 비트 : 사용하지 않음, 항상** 0
  - **두 번째 비트 : 단편화 금지 플래그** DF(don't fragment) **비트**
    - DF = 0 : **데이터그램의 단편화를 허용**
    - DF = 1 : **데이터그램의 단편화를 허용치 않음**
  - **세 번째 비트** : More **비트**
    - More = 1 : **이 데이터그램과 다음에 전송되는 데이터그램이 단편화되어 전송됨을 나타낸다.**
    - More = 0 : **한 메시지의 마지막을 구성하는 데이터그램 또는 메시지가 한 데이터그램**
- Protocol
  - **데이터그램에 실려 있는 데이터를 처리할 상위 계층 프로토콜을 지정**
    - TCP : 6
    - UDP : 17
    - ICMP : 1

IP 데이터그램 구조(IP 데이터그램 헤더)

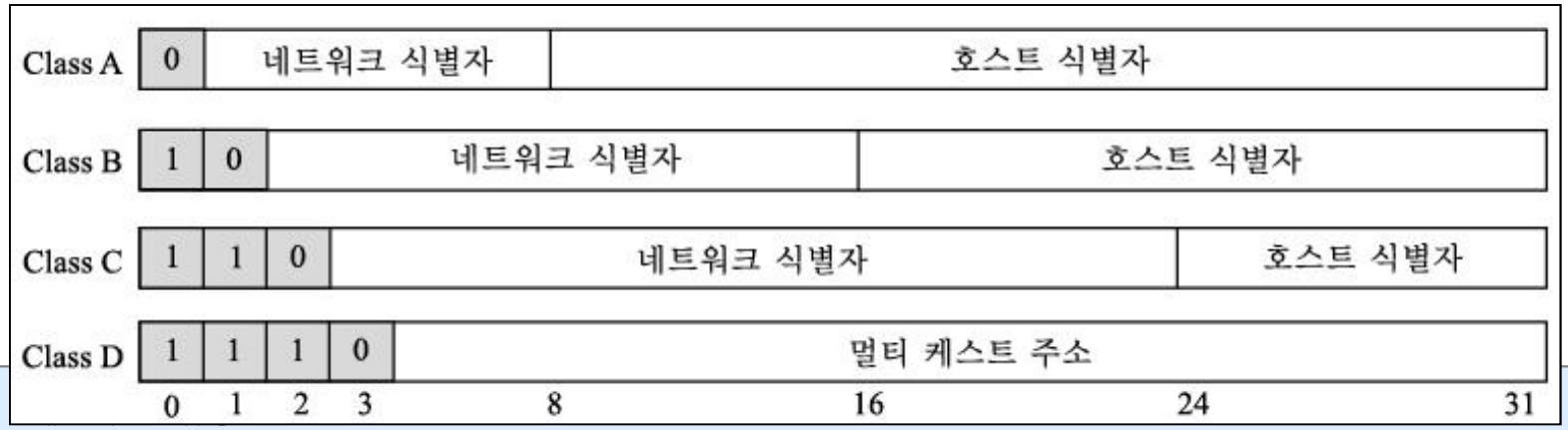
- TTL(Time To Live)
  - **데이터그램들이 인터넷 내에서 계속 돌아다니는 것을 방지하기 위하여 사용**
  - **보통** hop counter(**노드를 지나는 횟수**) **값을 사용**
  - **한 노드를 지날 때마다** 1**씩 감소하며** 0**이 되는 노드에서 이 데이터그램을 삭제**
  - TTL**값이 너무 클 때**
    - **목적지 주소에서 에러발생 시 데이터그램이 네트워크상에 오래 남아 트래픽 증가**
  - TTL**값이 너무 작을 때**
    - **데이터그램이 먼 거리를 돌아 갈 때 목적지에 도착 못함**
  - **디폴트로** 64**의 값을 가지며 멀티캐스트 같이 트래픽이 많이 발생할 때는** TTL**을 작게한다.**

IP 데이터그램 구조(IP 데이터그램 헤더)

- Header Checksum
  - IP **헤더의 비트 에러 검출을 위해 사용**
  - **헤더를** 16**비트 단위 크기로 나누고 각** 16**비트를 숫자로 취급하여 차례로 더한 결과에** 1**의 보수로 취해** Header Checksum**에 실어 송신**
  - **수신 측 결과와 다를 경우 에러 발생이므로 데이터그램 삭제**
- Source IP Address
  - **송신지** IP **주소**
- Destination IP Address
  - **수신지** IP **주소**
- IP Options
  - **옵션 선택**
  - **보통 사용되지 않음**
- Padding
  - 32**비트 단위로 헤더의 길이를 맞춤**
  - **보통 사용되지 않음**

IP 데이터그램 구조(IP 주소)

- IP 주소의 구성
  - IP 주소는 4바이트 크기를 가짐
  - dotted decimal IP 주소
    - IP 주소를 보기 쉽게 표시, IP 주소를 구성하는 4바이트를 10진수로 표시
    - 예) 11010010 01110011 00110001 11110100 -> 210.115.49.244
  - netid 필드 : 네트워크를 구분
  - hostid 필드 : 한 네트워크 내에서 호스트를 구분
  - 각 필드에서 사용하는 비트 수의 크기에 따라 네 가지 클래스로 분류
- IP 주소 체계의 단점
  - 호스트가 다른 네트워크로 이동하면 netid부분이 새로 접속된 네트워크의 netid로 변경
    - 컴퓨터의 IP 주소가 달라짐



IP 데이터그램 구조(IP 주소)

- **클래스 A 주소**
  - IP **주소 첫 번째 바이트의 첫 비트가** 0**이고 나머지** 7**비트가** netid
  - **뒤의 세 바이트가** hostid
  - **클래스 A 주소의** netid**는 약** 2^{24} **대의 호스트를 수용**
- **클래스 B 주소**
  - IP **주소 첫 번째 바이트의 처음 두 비트가** 10**이고 나머지** 6**비트와 두 번째 바이트가** netid
  - **뒤의 두 바이트가** hostid
  - **클래스 B 주소의** netid**는 약** 2^{16}**대의 호스트를 수용**
- **클래스 C 주소(방송용:모두** 1**, 자기자신: 모두** 0**)**
  - IP **주소 첫 번째 바이트의 처음 세 비트가** 110**이고 나머지** 5**비트와** 2**, 3번째 바이트가** netid
  - **마지막 한 바이트가** hostid
  - **클래스 C 주소의** netid**는 약** 2^{8}**대의 호스트를 수용**
- **클래스 D 주소**
  - IP **주소 첫 번째 바이트의 처음 네 비트가** 1110
  - **멀티 캐스트 주소로 사용**

IP 계층 기능

- MTU(Maximum Transmission Unit)
  - 서브네트워크가 한 번에 전달할 수 있는 데이터그램의 최대 크기
  - 이더넷의 경우 MTU는 1500바이트
  - PPP의 MTU는 전송매체에 따라 가변적이지만 전화선일 경우 주로 576 바이트를 사용
  - IP 계층이 상위 계층으로부터 받은 메시지의 크기가 path MTU보다 큰 경우
    - 이를 여러개의 IP 데이터그램으로 단편화하여 전송
    - 수신측에서는 단편화된 데이터그램을 재조립
  - 데이터그램의 identification 필드는 단편화 된 다수의 패킷을 원래대로 재구성하기 위해 존재
  - 전체 메시지를 구성하는 데이터그램 중 하나라도 손실 될 경우
    - 최종 수신측에서 전체 메시지를 버림
    - 메시지의 재조립은 라우터에서는 실행되지 않음

IP 계층 기능

- path MTU
  - 종점 호스트 사이에 존재하는 서브네트워크들의 MTU중 최소 MTU값
  - IP 헤더의 플래그 DF가 1이면서 IP 데이터그램의 크기가 path MTU보다 클 경우
    - 중간의 라우터에서 에러가 발생하며 송신측에게 이를 알려줌
  - 일반 데이터그램의 DF 비트는 항상 0으로하여 필요시 라우터에서 단편화함
  - DF 비트는 path MTU의 크기를 알아내는 path MTU discovery 프로토콜에서 사용됨
  - TCP 계층에서 일정 크기의 데이터그램을 DF = 1로 하여 전송 시
    - 목적지까지 전달되지 못하고 에러 발생하면 데이터그램의 길이를 조금씩 줄여서 재전송
    - 전송이 성공할 때까지 반복하여 path MTU를 찾는다.

IP 계층 기능

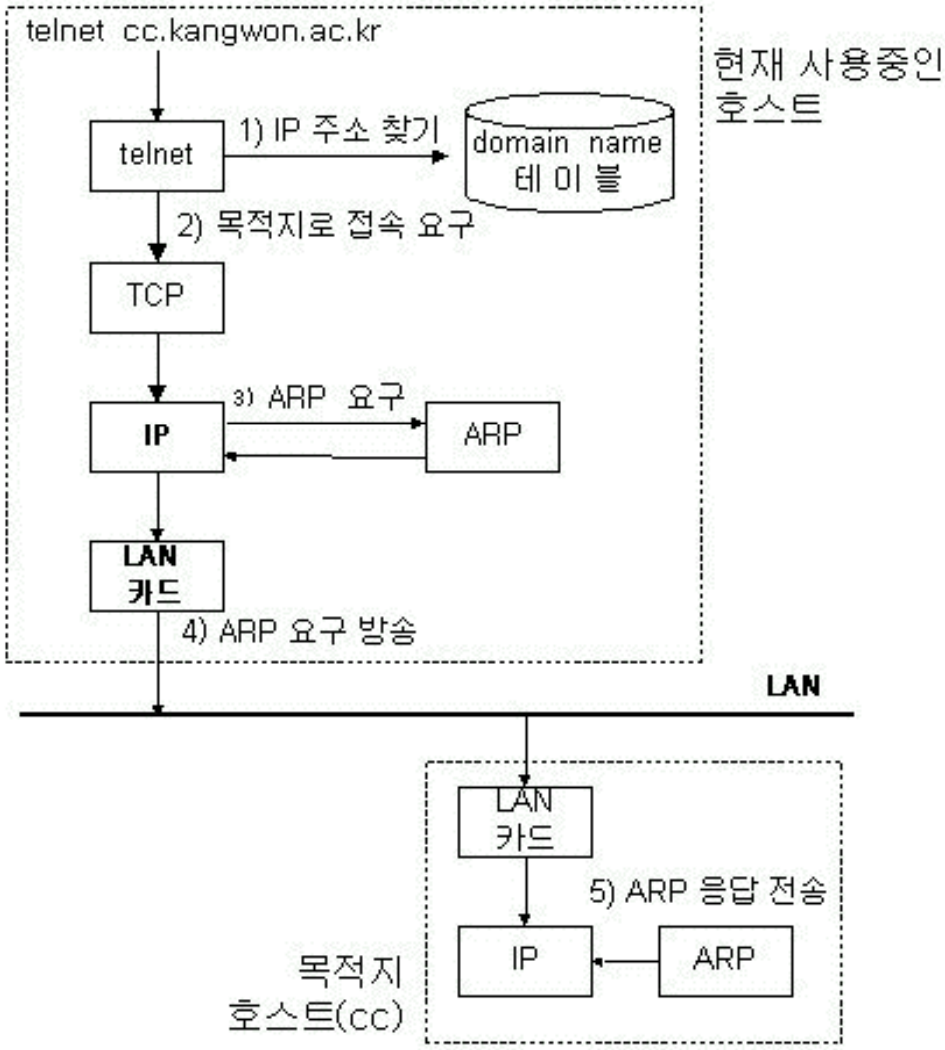
- 라우팅
  - IP 데이터그램이 목적지 호스트까지 전달되기 위해 거쳐야할 라우터를 정해주는 기능
  - 현재 인터넷이 널리 사용 될 수 있는 이유 중 하나로 안정적인 라우팅 동작이다.
  - 라우터
    - 라우팅을 처리하는 장비
  - 라우팅 알고리즘
    - 최적의 라우팅 경로를 알아내는 알고리즘
  - 라우팅 테이블
    - 라우팅 알고리즘으로 얻은 여러 라우팅 정보를 저장

IP 계층 기능

- ARP(Address Resolution Protocol)
  - LAN **내에서 특정** IP **주소를 가지고 있는 호스트의** MAC **주소를 알아내는 프로토콜**
  - LAN**에 접속된 호스트에게** IP **데이터그램을 전달하려면 그 장비의** MAC **주소를 알아야한다**.
    - LAN**에서는 모든 데이터가** MAC **프레임에 실려서 전달**
  - IP **주소로부터 해당 호스트의** MAC **주소를 알아내는 절차를** ARP**라고 한다**.
  - 4**바이트** IP **주소가 아닌** 6**바이트의** MAC **주소가 사용되기 때문에** ARP **프로토콜이 필요**

IP 계층 기능

- ARP 동작



- **응용 프로그램** telnet**은 먼저** cc.kangwon.ac.kr **호스트가** 210.115.49.242 **임을 찾음 ->** **네임 서비스**
- telnet**은 이** IP **주소를** TCP**에게 알려주고 이곳으로 접속 하도록 요청**
- IP **계층은 목적지 호스트의** netid**와 자신의** netid**가 같은지 비교**
  - **같은** LAN**에 있으면** LAN**을 통해 바로** telnet **서비스 요청을 전송**
  - **만약 같은** LAN**에 있지 않으면** telnet **서비스 요청을 디폴트 게이트웨이를 통해 외부로 요청**
- **목적지의** MAC **주소를 알아내기 위해** ARP**를 구동**
- ARP**에서 목적지 호스트의** MAC **주소를 찾기 위해** ARP request **패킷을** LAN **내의 모든 장비에 방송**
- **목적지 호스트**(vcn.kangwon.ac.kr)**만** ARP request**에 자신의** MAC**주소를 응답**

IP 계층 기능

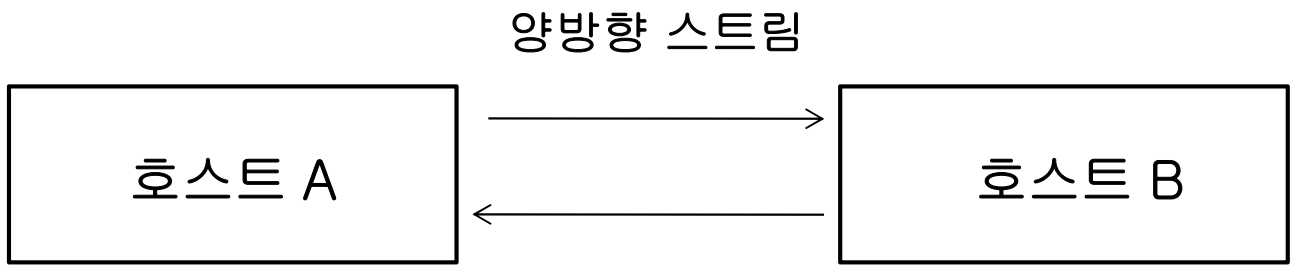
  - 동일한 목적지 호스트에 IP 데이터그램을 연속으로 보낼 때
    - 계속 ARP를 사용하면 트래픽이 증가
    - ARP로 얻은 최근 정보를 캐시에 기록하고 있어 이를 재사용하여 트래픽 문제 해결
- RARP(Reverse ARP)
  - ARP의 역과정
  - 48비트 MAC 주소로부터 그 장비의 IP 주소를 알아내는 프로토콜
  - 근거리 통신망 내에 물리적으로 존재하는 장치의 IP주소를 알아낸다.
  - 게이트웨이의 ARP목록이나 캐시로 부터 IP주소를 알아내기 위한 확인 요청을 하는데 사용
  - 동작방식
    - 네트워크 관리자가 근거리 통신망의 게이트웨이 라우터 내에 물리적인 장치가 그에 상응하는 IP 주소를 지칭하도록 표를 작성
    - 새로운 장치가 설정되었을 때 RARP 클라이언트 프로그램은 라우터 상의 RARP서버에게 그 장치를 위한 IP주소를 보내주도록 요청
    - RARP서버는 라우터 목록 내에 새로운 엔트리가 설정되었다고 가정하여 그 장치에게 IP 주소를 응답

0.3 TCP **프로토콜**

- **연결형 서비스**
- **양방향 바이트 스트림 제공**
- **신뢰성 있는 데이터 전달 보장**

TCP 특징

- **연결형 서비스 제공**
  - **종점 호스트 사이에 일 대 일 연결을 제공**
  - **연결설정, 데이터 송수신, 연결종료 과정이 필요**
  - **두 호스트 사이에 TCP 연결 설정이 이루어지면 두 개의 스트림이 생성되어 양방향 통신**
  - **호스트 A가 호스트 B로 연결요청을 한 경우에도 호스트 B가 호스트 A로 데이터 전송이 가능**



TCP 특징

- **신뢰성 있는 데이터 전달**
  - **신뢰성 있는 종점간 데이터 송수신을 보장**
  - **확인응답**(Ack), **체섬**(Checksum), **재전송**, **흐름제어 등을 사용**
  - **확인응답**(Ack)
    - **상대방** TCP**에게 잘 전달되었는지를 확인하기 위해 사용**
    - **데이터를 수신한 상대방은 데이터에 에러가 없고 순서에 문제가 없을 시 송신 측에** Ack**를 보냄**
    - **에러 확인은 체섬, 순서확인은 순서번호를 사용함**
    - Piggyback **기능**
      - **데이터를 보낼 때 마다** Ack**를 전송하면 불필요한 트래픽이 발생**
      - **수신 측은 즉시** Ack**를 보내지 않고 약간의 시간지연을 둔다.**
      - **지연시간 동안 보낼 데이터가 생기면 데이터에** Ack**를 포함시켜 전송**
  - **체섬**(Checksum)
    - **데이터 전송 중에 발생한 에러를 검출**
    - **체섬 에러 발견 시 데이터를 버리고** Ack**를 보내지 않는다.**
    - **별도로 에러 상황을 송신 측에 알려주지는 않는다.**

TCP 특징

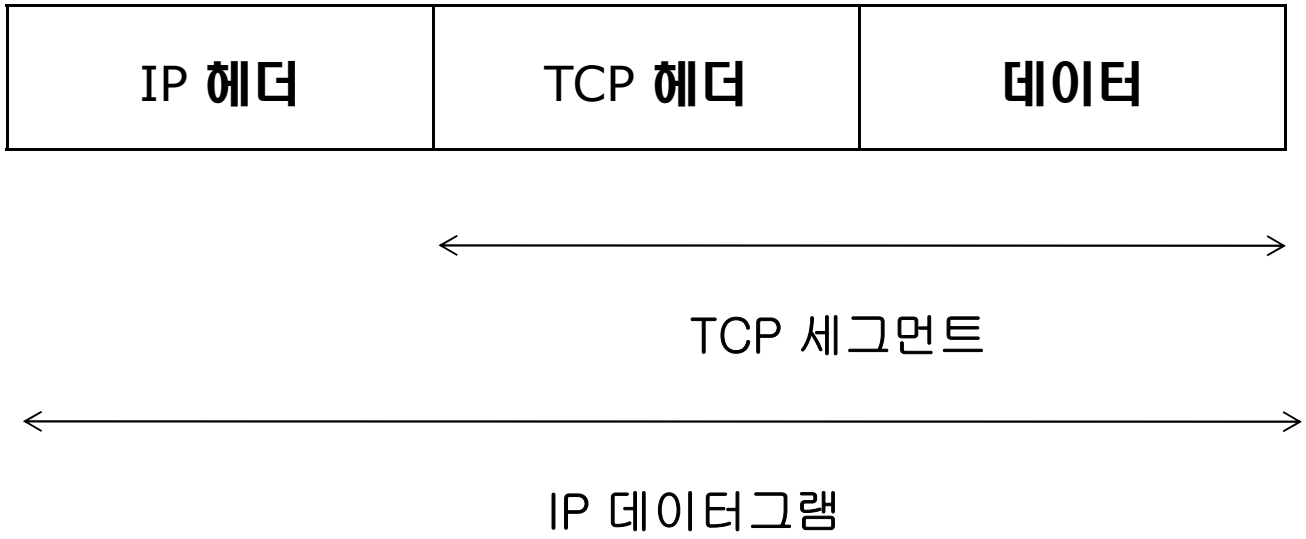
- **재전송**
  - **일정 시간동안 Ack가 오지 않을 경우 전송했던 데이터를 재전송**
  - **재전송을 위해 송신 측은 타이머를 사용**
    - **타이머 시간동안 Ack가 오지 않을 경우 재전송**
  - **전송 타이머 값은 RTT(Round Trip Time)에 비례하여 정해지며 횟수에도 제한**
    - **일정 회수 이상 Ack가 없을 시 종점간 연결을 종료**
- **흐름제어**
  - **수신측의 사정에 의한 흐름제어**
    - **상대방이 너무 많은 데이터를 보내지 못하게 하는 흐름제어**
    - **수신버퍼의 여유 크기를 고려하여 상대방이 전송할 수 있는 데이터 량을 바이트 단위로 알려준다.**
    - **수신 할 수 있는 데이터 량을 Window필드(16비트)를 통해 송신 측에 알려준다.**
  - **송신 측의 사정에 의한 흐름제어**
    - **송신 측에서는 송신버퍼가 부족하면 write(), send()같은 쓰기 함수가 블록된다.**

TCP 특징

- **스트림형 서비스 제공**
  - **송신 측에서 전송된 데이터는 바이트 단위로 차례대로 수신 측에 전달**
  - **스트림 서비스**
    - **송신된 데이터를 수신 측에서 차례대로 읽을 수 있게 제공되는 통신 채널 서비스**
    - **수신 측에서는 송신된 데이터를 한 번에 읽거나 여러 번으로 나누어 읽을 수 있다.**
    - **송신 측에서 write()등을 호출하여 전송한 데이터들의 경계를 수신 측에서는 알 수 없다.**
    - **송신 측이 전송한 데이터 경계를 수신 측이 알도록 전송 데이터 크기를 미리 보낸다.**

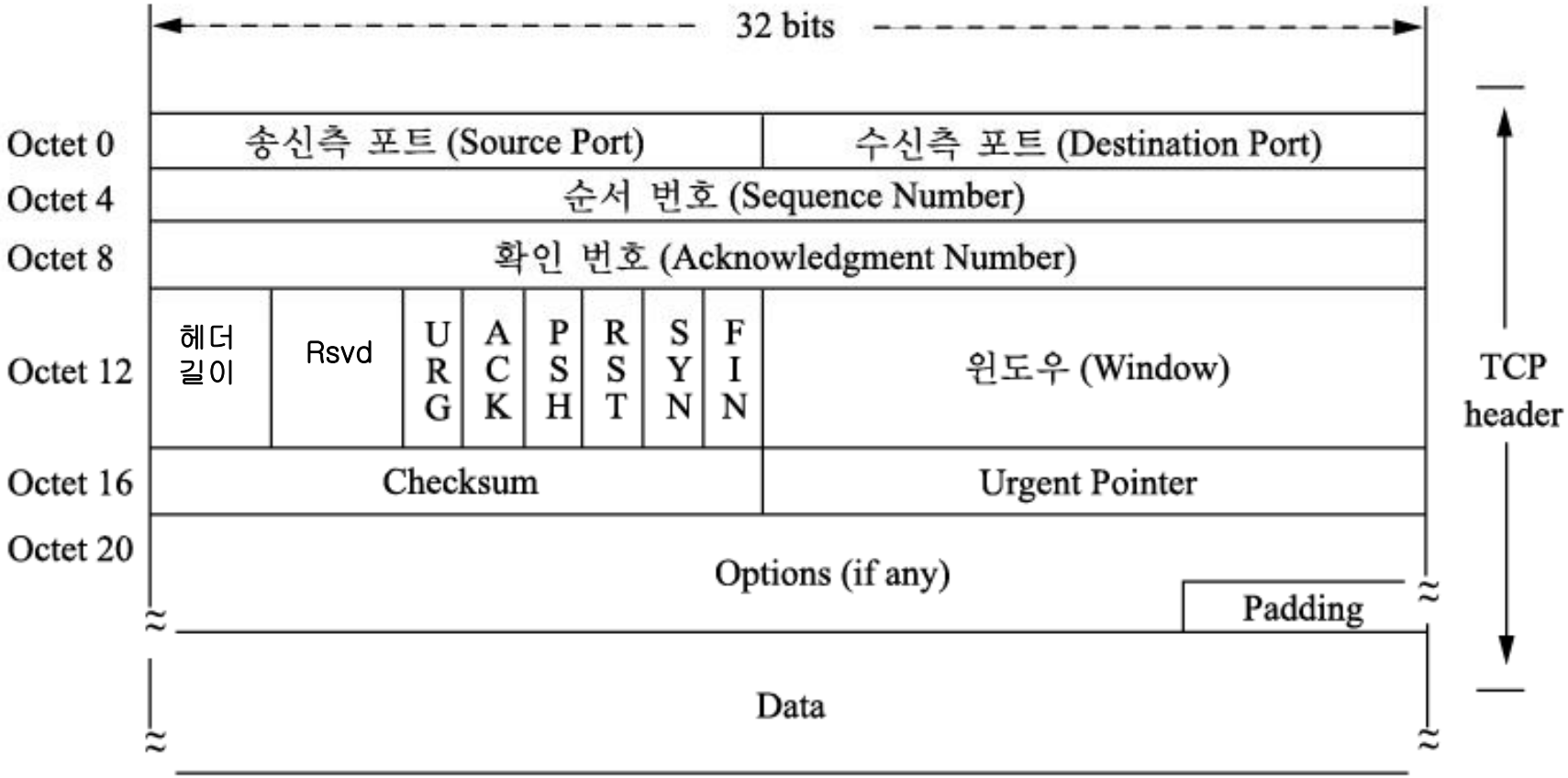
TCP 헤더

- **세그먼트**
  - TCP **계층이** IP **계층으로 내려 보내는 데이터 단위**
    - **종점간** TCP**들 사이에 송수신되는 데이터 단위**
  - **세그먼트의 구성**
    - TCP **계층에서 프로토콜 처리를 위해 필요로 하는 헤더와 응용 프로그램으로부터 받은 데이터로 구성**



TCP 헤더

- TCP **세그먼트와 헤더 구조**



TCP 헤더

- Source Port
  - 송신 측의 응용 프로세스를 구분하는 포트번호
- Destination Port
  - 수신 측의 응용 프로세스를 구분하는 포트번호
- Sequence Number 필드
  - 세그먼트에 실려 있는 데이터의 첫 번째 바이트의 순서번호 기록
  - 수신 측에서 데이터가 정상적으로 도착하는지 확인하는데 사용
- Ack(Acknowledgement) Number
  - ACK 플래그가 1일 경우에만 의미가 있다.

TCP 헤더

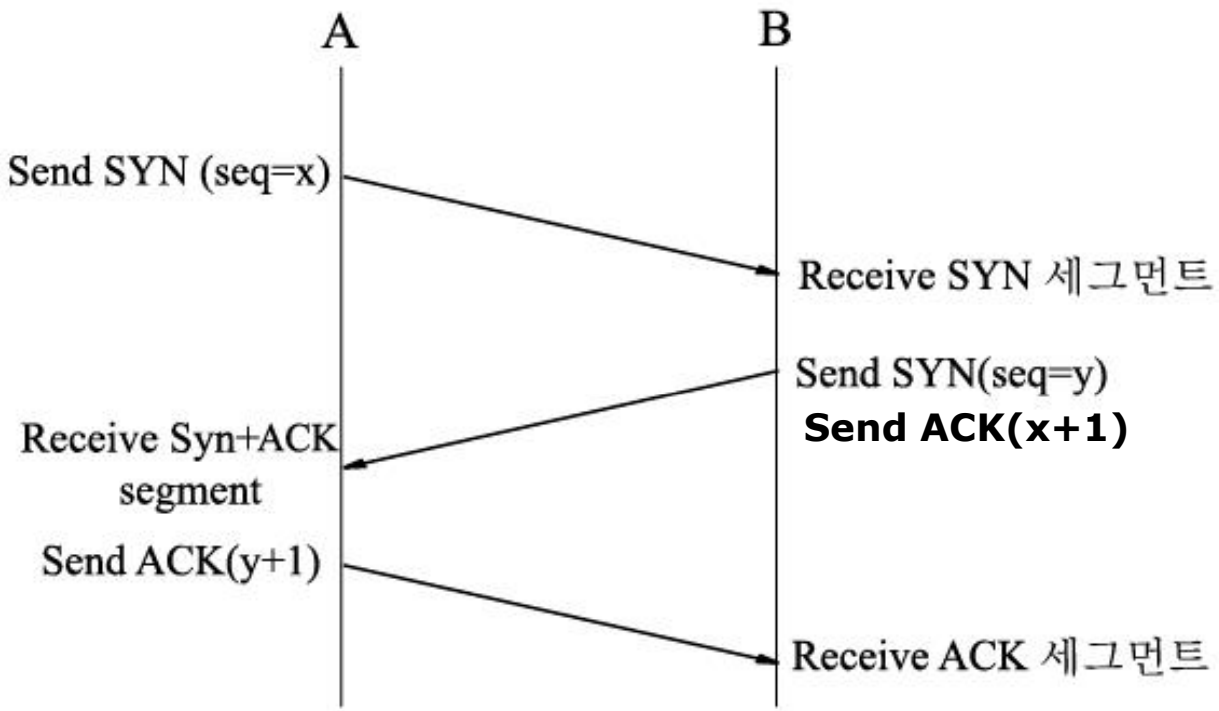
- Header Length
  - 헤더의 크기를 4바이트 단위로 나타낸다.
  - TCP에서 옵션을 사용하지 않는 경우 TCP 헤더의 크기가 20 바이트이므로 5로 set
- Rsvd
  - 현재 사용되지 않는 값으로 0으로 set
- Code Bits
  - URG : Urgent Pointer가 유효한 값임을 나타낸다.
  - ACK : Ack Number에 들어 있는 값이 의미 있는 값임을 나타낸다.
  - PSH : 이 데이터를 가능한 신속히 응용에 전달하도록 한다.
  - RST : 연결을 reset할 때 사용
  - SYN : TCP 연결을 시작할 때 사용
  - FIN : TCP 연결을 종료 할 때 사용

TCP 헤더

- Window
  - 수신 측이 현재 수신 가능한 데이터 버퍼 크기를 바이트 단위로 나타낸다.
- Checksum
  - pseudo 헤더
    - TCP 세그먼트 전체 + IP 헤더의 후반부 12바이트
  - pseudo 헤더에 대한 에러 검출 코드
- Urgent Pointer
  - URG 비트가 1일 때 의미있다.
  - 현재 세그먼트에 포함된 긴급 데이터의 마지막 위치를 가리키는 오프셋을 저장
  - 긴급 데이터는 Sequence Number 위치부터 시작하여 Urgent Pointer 바이트 만큼 범위에 속한다.

TCP 연결설정

- 3-way **핸드쉐이크**(handshake)



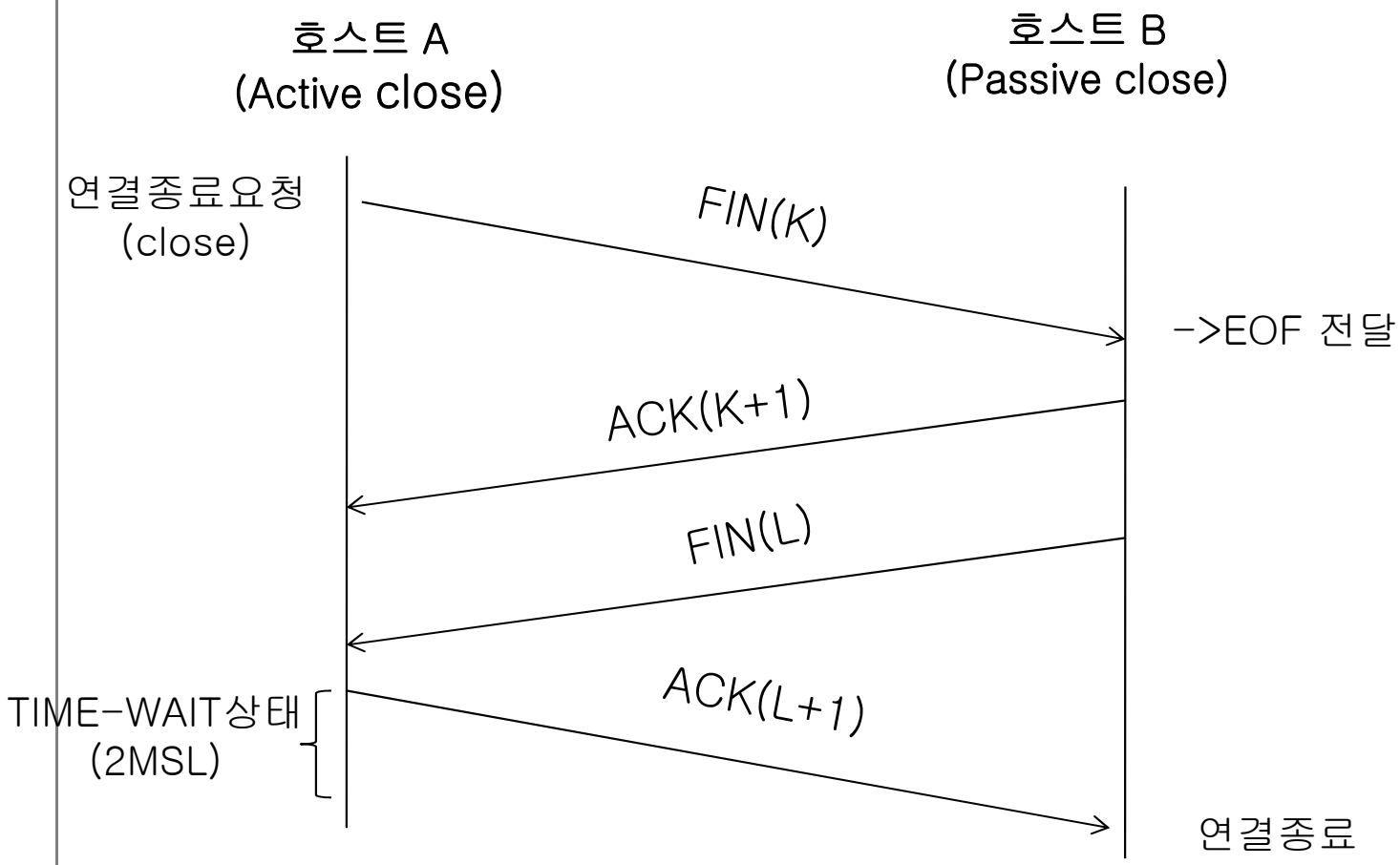
- 3-way
  - **최초 연결요청에 상대방이 확인하면서 다시 연결요청하고 이에 대해 처음의 연결요청 확인**
  - **3회의 통신이 이루어져야 한다.**

TCP 연결설정

- **연결설정 절차**
  - **연결요청 측에서 SYN 비트를 세트하고 순서번호를 랜덤 정수 X로 한 연결요청을 보낸다.**
  - **상대방은 응답을 위해 ACK 비트를 세트하고 Ack Number에 (X+1)을 기록하며 SYN 비트도 세트 한 후 다음 자신이 사용할 랜덤 순서번호 Y를 지정하여 보낸다.**
  - **최초의 연결요청자가 ACK(Y+1)로 응답하면 연결된다.**
  - **이때 SYN세그먼트의 크기가 1바이트이므로 X+1 또는 Y+1을 응답**
- **X와 Y값을 랜덤하게 정하는 이유**
  - **예측 가능한 일정한 값을 사용하면 연결설정 동안 해킹 공격을 받을 수 있다.**
- **TCP 연결요청에 ACK가 오지 않으면 잠시 기다린 후 재시도**

TCP 연결 종료

- 4-way **핸드쉐이크**

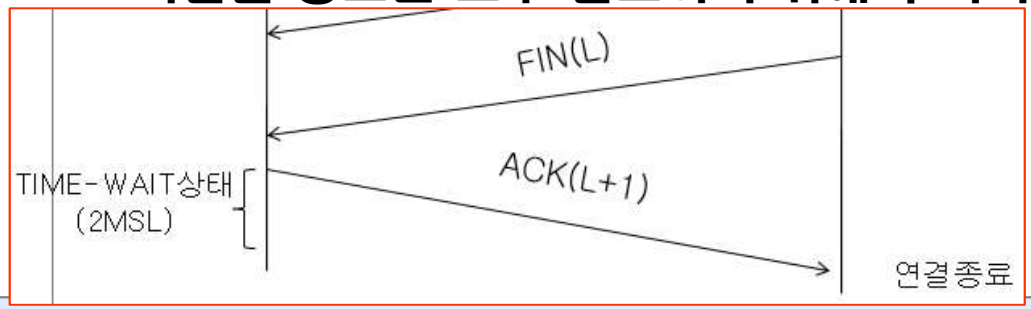


TCP 연결 종료

- 연결 종료의 원인
  - TCP 응용 프로그램에서 해당 소켓에 대해 close()를 호출한 경우
  - 응용 프로그램이 종료된 경우
- 연결 종료 절차
  - TCP는 상대방에게 FIN 플래그를 세트하여 전송
    - 더 이상 전송할 데이터가 없다는 뜻
  - 이에 대한 ACK를 받음으로써 송신 측 방향의 채널이 닫힘(half close)
  - FIN을 받은 측에서는 통상적으로 자신도 FIN을 보내어 양방향 채널을 모두 종료
- 처음 FIN을 보낸 측(그림 2.11의 호스트 A)
  - active close를 수행
- FIN을 받은 측(그림 2.11의 호스트 B)
  - passive close를 수행

TCP 연결 종료(TIME-WAIT)

- TIME-WAIT **상태**
  - Active close**를 한 호스트가** FIN**을 받고 이** FIN**에 대한** ACK**를 보내기 까지 잠시 머문 상태**
  - TIME-WAIT **상태에 있는 동안 지금까지 사용했던 포트번호를 재사용할 수 없도록 제한**
  - TIME-WAIT **상태를 두는 이유** 1
    - **연결종료를 신청한 측**(Active close)**에서** ACK(L+1)**까지 안전하게 전송해야 양방향 채널 종료**
    - **만약** ACK(L+1) **메시지가 사라지면 상대방**(Passive close)**은** FIN(L)**을 재전송**
    - **이때** TIME-WAIT **상태에 머물지 않았다면** FIN(L)**에 대한** ACK(L+1)**을 보낼 수 없다.**
    - TIME-WAIT **시간은 종료를 모두 완료하기 위해서 기다리는 시간**



TCP 연결 종료(TIME-WAIT)

- TIME-WAIT 상태를 두는 이유 2
  - 연결종료 직후에 같은 IP 주소와 포트번호로 새로운 연결요청을 허용할 경우
    - 과거의 연결에 사용되었던 세그먼트가 뒤늦게 도착하는 것을 잘못 수신할 수 있다.
  - 고의적인 해킹 공격 방지를 위해서 필요
- 2MSL
  - TIME-WAIT 상태의 다른 이름
  - TIME-WAIT 상태에서 기다리는 시간이 2*MSL(maximum segment lifetime)시간
  - MSL은 네트워크 내에서 세그먼트가 남아있을 수 있는 최대 예상 시간
  - 대부분의 시스템은 30초에서 2분사이의 MSL값을 갖는다.
  - TIME-WAIT 상태에 있는 동안 과거에 전송된 세그먼트가 도착 시
    - 이 세그먼트는 버려지고 2MSL 타이머는 재 시작
  - TIME-WAIT 시간 이후에 세그먼트 도착 시
    - 세그먼트는 무시되고 송신 측으로 리셋(RST) 비트를 set시켜 전송

TCP 연결 종료 (TIME-WAIT)

- Passive close 측은 TIME-WAIT 상태를 갖지 않는다.
  - FIN(L)을 보낸 후에 상대방으로부터 ACK(L+1)을 받으면 TCP 초기 상태로 간다.
- 클라이언트-서버 모델에서 서비스 사용이 종료될 때
  - 클라이언트가 active close를 신청하는 것이 바람직하다.
    - 서버 측이 TIME-WAIT 상태에 들어가지 않고 바로 같은 포트번호를 사용해서 다른 서비스 요청에 응답할 수 있다.

TCP 연결 종료(리셋)

- 리셋
  - 정상적인 연결종료(FIN)을 사용 하지 않고 RST를 사용해 즉시 연결 종료
    - TCP 연결종료 시에 FIN대신 RST을 전송 하려면 linger 소켓 옵션을 사용
    - SO_LINGER 옵션을 지정하면 연결 종료 시 즉시 또는 옵션에서 정한 일정한 시간 후에 RST 전송
  - 정상적인 TCP 연결설정이 되지 않은 상태에서 데이터 수신 시
    - TCP는 상대방으로 RST 비트를 set하여 전송
    - 통신 중에 상대방 컴퓨터가 재부팅된 경우에도 발생
  - 연결요청을 기다리고 있는 포트(서버 프로그램)가 없는 상태 시
    - 연결요청을 받아도 상대방으로 리셋을 전송

TCP 연결 종료(리셋)

- RST를 사용하여 즉시 종료(abortive release)를 할 경우
  - 아직 전송되지 않은 데이터에 대한 안전한 송신이 보장되지 않는다.
- RST를 수신한 측에서는 ACK를 보내지 않아도 된다.
  - 수신 측 TCP에서는 정상종료가 아닌 에러에 의한 연결 종료라고 응용 프로그램에 보고

데이터 송수신

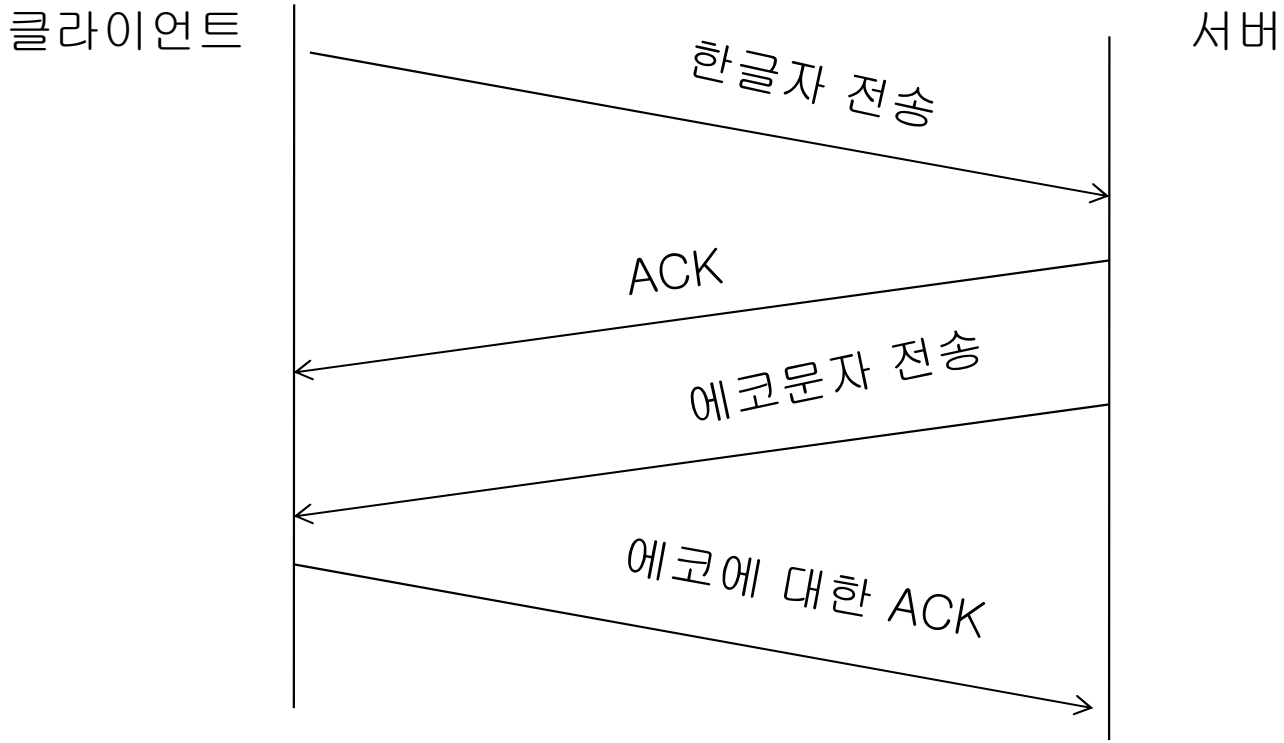
- MSS(maximum segment size)
  - IP 계층에서의 단편화를 피하기 위해 지정한 최대 세그먼트 크기
  - IP 계층 및 라우터에서 데이터그램의 단편화가 발생하지 않도록 TCP에서는 종점간의 path MTU를 미리 알아내고 세그먼트의 크기를 "path MTU - IP와 TCP의 헤더크기" 정하는 것이 최적의 선택
  - MSS 값은 TCP 연결설정 시에 상대방과 협약
    - SYN 전송 시 자신이 원하는 MSS 값을 상대방에게 알려줌
    - 상대방 호스트는 자신의 MSS 값과 받은 MSS 값을 비교해서 두 MSS 값 중 작은 값을 리턴
    - 디폴트 MSS 값은 536바이트
      - 이경우 IP 데이터그램의 크기는 536 + 40 = 576 바이트

데이터 송수신

- TCP 데이터 송수신
  - bulk 데이터 전송 : ftp, mail, http 등
  - interactive 데이터 전송 : telnet 등
- bulk 데이터 전송
  - 1000바이트 데이터 송신 시
    - 데이터 송신을 위해 write() 또는 send()를 호출
    - TCP는 송신버퍼에 1000바이트를 복사한 후 MSS 단위로 단편화
    - TCP 헤더 20바이트를 붙여 세그먼트로 만들어 IP 계층으로 넘긴다.
    - IP 계층에서는 각 세그먼트에 IP 헤더 20바이트를 붙여 링크 계층으로 보내어 전송
  - 전송한 데이터에 대해 상대방으로부터 Ack가 올 때까지 데이터는 송신버퍼에 보관
    - 상대가 Ack를 보내지 않으면 재 전송
  - 수신측
    - IP 계층에서 IP 헤더를 제거하고 TCP 계층으로 넘긴다.
    - TCP 계층에서 TCP 헤더를 제거하고 수신버퍼로 데이터를 복사

데이터 송수신

- Interactive **데이터 송수신**
  - TCP**는 크기가 작은 데이터 송수신 시에 전송 효율이 매우 낮다.**
    - **예로** rlogin**에서 키 입력 시마다 한 바이트씩 데이터를 전송하는데, 글자단위로 에코하므로 한 글자 입력에** 4**개의 세그먼트 전송이 발생**
  - **응용계층에서는 즉각적인 반응을 원하므로 이러한 비효율을 감수**



데이터 송수신

- **흐름제어**
  - **수신버퍼에 의한 흐름제어**
    - **수신버퍼의 여유 크기를 고려하여 상대방이 전송할 수 있는 데이터 량을 바이트 단위로 알려준다.**
    - **상대방이 너무 많은 데이터를 보내지 못하게 하는 흐름제어**
    - **수신할 수 있는 데이터 양을** Window**필드(**16**비트)를 통해 송신 측에 알려준다.**
  - **송신버퍼에 의한 흐름제어**
    - **송신 측에서는 송신버퍼가 부족하면** write(), send()**같은 쓰기 함수가 블록**
    - **송신 및 수신 버퍼의 크기 변경을 위해서는 소켓 옵션을 변경**
      - TCP **연결이 성립되기 전에 변경**
    - **처리 속도가 네트워크 속도에 비해 매우 빠르고, 충분한 메모리가 있기 때문에 버퍼 부족으로 인한 흐름제어가 거의 일어나지 않는다.**

데이터 송수신(혼잡제어)

- **혼잡제어**(congestion control)
  - 인터넷의 전송 용량 한계로 인해 혼잡이 발생될 시 송신 측의 전송 속도를 낮춰 혼잡 피한다.
  - TCP에서는 혼잡제어를 하기 위해 혼잡윈도우를 사용
  - 송신 측은 앞에서 설명한 흐름제어윈도우와 혼잡윈도우중 작은 값을 최종 윈도우 크기로 정한다.
    - 정해진 윈도우 크기 범위 내에서만 송신이 가능
  - 송신 측 TCP는 수신 측으로부터 정상적으로 Ack가 오면 혼잡윈도우를 1씩 증가
    - 네트워크에 여유가 있다고 판단
  - 네트워크에 혼잡이 발생하면 혼잡윈도우 값을 급격히 1로 줄여 전송량을 조정
  - 혼잡윈도우 값이 1로 감소했을 경우
    - 정상적으로 송신이 이루어질 때 Ack를 받을 때마다 다시 증가
    - 혼잡윈도우 값이 1로 되기 직전의 혼잡윈도우 값의 1/2이 될 때까지 2배씩 증가

0.4 UDP 프로토콜

- 데이터의 분실 확인, 전달 순서를 보장하지 않는다.
- 헤더크기가 작고 연결 지연이 없다.
- 멀티캐스트 서비스에 사용

UDP 헤더

- UDP 헤더
  - TCP 헤더보다 간단
  - 구성
    - 송신지와 수신지의 포트번호(16비트), 데이터그램 길이(16비트), 체섬(16비트)
    - UDP의 체섬의 적용 범위는 TCP 세그먼트의 경우와 동일
    - 송신지 포트번호는 송신측에서 수행되는 소켓을 구분하기 위한 번호
    - 목적지 포트번호는 목적지 소켓을 구분하기 위한 번호
    - 길이 필드는 헤더와 사용자 데이터를 합한 전체 길이로 16비트임(65535이내 표현)
    - 체섬은 헤더와 데이터를 모두 포함한 데이터그램 전체에 대해 에러 탐지에 사용

Source Port (16 bits)	Destination Port (16 bits)
Total Length (16 bits)	Checksum (16 bits)

UDP 헤더 정보

UDP 특징

- UDP의 특징
  - UDP는 비연결형 트랜스포트 계층 서비스를 제공
  - 분실 확인이나 전달 순서를 보장하지 않는다.
    - 연결설정 과정과 종료과정이 없다.
    - 스트림을 제공하지 않고 UDP 세그먼트 단위로 송수신이 이루어진다.
  - TCP에 비하여 헤더의 크기가 작고 연결 지연이 없어 간단한 데이터 전송에 TCP보다 유리
  - 흐름제어나 Ack 기능을 제공하지 않으므로 데이터를 전송한 후 버퍼에 남겨두지 않는다.
  - UDP에서도 수신된 세그먼트에서 체섬으로 에러 검사 후 에러 발생시 데이터 폐기
  - UDP를 사용해야 하는 경우
    - 방송 또는 멀티캐스트를 해야 하는 경우
    - 서버 프로그램이 UDP만을 사용하도록 작성되어 있는 경우
    - 오버헤드를 줄이기 위한 실시간 스트리밍 서비스 등

UDP 특징

- UDP 소켓의 수신 버퍼의 크기는 리눅스에서 65535로 잡혀있다.
  - UDP로 송신 또는 수신할 수 있는 데이터의 크기는 65507(=65535-8-20)
  - 큰 메시지는 MTU 크기 제한에 의해 단편화 되지만 UDP에서는 단편화를 피하는 것이 좋다.
    - 전달 순서 확인이나 재전송을 하지 않기 때문
- UDP에서 수신버퍼에 도착한 데이터를 응용 프로그램에서 읽을 때 작은 크기의 버퍼를 사용하면 데이터가 분실될 수 있다.
- UDP 구현에서는 최소한 576 바이트 이상의 수신버퍼를 사용하도록 약속되어 있다.
  - UDP 송신 시에 576 바이트 크기 이하의 메시지에 대해서는 수신버퍼 부족 현상이 없다.