4계층의 역할과 개요

※ 계층별 역할
1계층 : cable이 연결된 상태에서 신호 전달
2계층 : 신호 전달이 가능한 상태에서 segment 내 데이터 송수신
3계층 : segment끼리 데이터 송수신

3계층 이하에서는 수신처 존재 x, 데이터가 도중에 손실되는 등의 에러에 대해 reliable하지 않음
4계층 이상에서는 수신하거나 송신할 데이터에 대한 처리 담당
=> 확인 응답(ACK), 흐름 제어(buffer overflow 방지) 수행

port number를 이용해 송수신할 application 결정

확인 응답, 흐름 제어 => 신뢰성 높은 데이터 전송
즉, 포트 번호 => 송신할 application 판별

TCP/IP에서는 TCP / UDP 중 하나를 이용해 위의 작업을 수행

Connection과 Segment

TCP의 connection
- app끼리의 가상의 데이터 통로
  => connection establishment를 통해 확실한 전송을 보장

TCP header

- 최소 20 octet으로 이루어짐

Offsets	Octet	0								2								3								4							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port number(16)																Destination port number(16)															
4	32	Sequence number(16)																															
8	64	Acknowledgement number(if ACK set, 16)																															
12	96	Data offset(4)				Reserved 0 0 0			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window size(16)															
16	128	Checksum(16)																Urgent pointer(if URG set, 16)															
20	160	Options (if data offset > 5. Padded at the end with "0" bytes if necessary.)																															
...	...	...																															

- sequence number : 데이터를 분할해서 전송할 때 각 segment의 번호
- acknowledgement number
  - handshake 과정에서는 상대방이 보낸 sequence number + 1
  - 데이터 전송과정에서는 상대방이 보낸 sequence number + 자신이 받은 데이터의 bytes
  => 다음에 보내줘야 하는 데이터의 시작점을 의미
- data offset : 전체 segment 중에서 header가 아닌 data가 시작되는 위치(Word = 4B)로 표현
  - options의 크기가 고정되어있지 않기 때문에 필요
- reserved : 0으로 채워져야 함
- flags

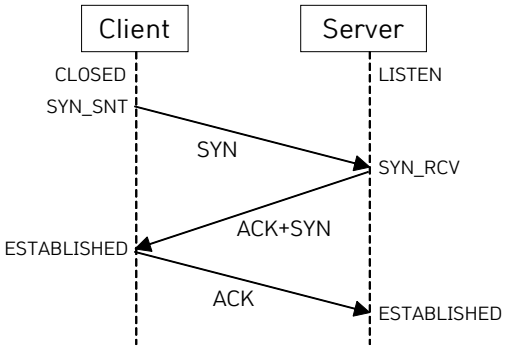
- 현재 segment의 속성을 나타냄
- deafult : 모두 0
- 6개에서 reserved 필드를 사용해 NS, CWR, ECE가 추가됨(flow control을 위해)

필드	Description
URG	Urgent pointer 필드에 값이 있음을 알림(urgent pointer 데이터는 먼저 처리됨)
ACK	Acknowledgement 필드에 값이 있음을 알림(0이면 ACK 필드 자체가 무시됨)
PSH	Push flag, 수신처에게 이 데이터를 빠르게 응용프로그램에게 전달해달라는 플래그 0이면 수신처는 버퍼가 다 채워질 때까지 기다림 1이면 이 segment 뒤에 연결된 segment가 없다는 것을 의미하기도 함
RST	Reset flag, 이미 연결이 확립되어 ESTABLISHED 상태인 상대방에게 연결을 강제로 리셋해달라는 플래그
SYN	Syncrhonize flag, 상대방과 연결을 생성할 때 sequence number의 동기화를 위한 segment임을 알림
FIN	Finish flag, 상대방과의 연결을 종료하고 싶다는 요청
NS	네트워크의 ECN(Explicit Congestion Notification)을 위한 플래그
CWR	
ECE	

- window size : 한 번에 전송할 수 있는 데이터의 양
  => 윈도우의 최대 크기는 64KB
  최근에는 options의 WSCALE을 이용해서 크기를 키워서 사용
- checksum : 송신 중 발생할 수 있는 오류 검출을 위한 값
  데이터를 16bits씩 나눠서 차례대로 더함, carry는 다시 더해줌, 결과에 1의 보수를 취한 값
  => 결과와 checksum을 더한 값은 모든 bit가 1임
- urgent pointer : URG플래그가 1이면 이 포인터가 가르키고 있는 데이터를 우선 처리
- options : TCP의 기능을 확장할 때 사용하는 필드들

3-way handshake

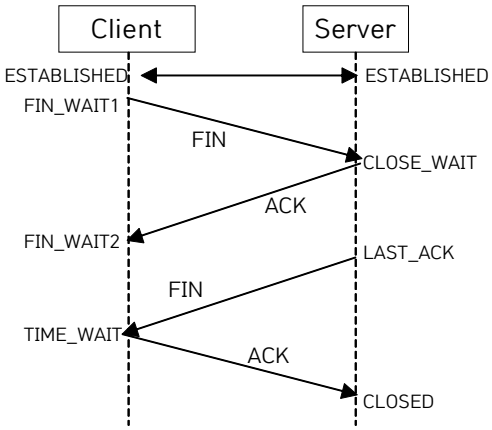
TCP에서 connection establishment를 할 때 사용하는 방법
3번의 데이터 송수신으로 쌍방향의 통신로를 확립
- 각 송수신은 해당하는 flag가 1인 상태로 전송됨
- 둘 다 ESTABLISHED 상태가 되면 connection 확립



4-way handshake

TCP에서 connection termination을 할 때 사용하는 방법

- CLOSED가 되면 connection 단절
- server의 ACK, FIN을 합치치 않는 이유 : 아직 전송을 마치지 못한 이전의 packet이 있을 수 있기 때문



TCP Segmentation

App으로부터 받은 data를 MSS(Max Segment Size)로 분할하여 캡슐화하는 것

- 보통 인터넷에서 1500 octet이 최대 데이터 크기이기 때문에 IP header(20 octet), TCP header(20 octet)를 뺀 1460 octet이 MSS가 됨
- 분할된 것을 각각 segment라 부름
- sequence number : 각각의 segment에 부여되는 번호
  - 1 octet마다 번호 할당 => MSS가 3 octet이면 sequence number는 1, 4, 7, ...

윈도우 제어

에러 복구

수신자는 Acknowledgement number를 보냄

=> 데이터를 어디까지 받았는지 송신처에게 알려줌

=> 송신측에서는 보낸 sequence num과 ack num을 비교해서 맞지 않거나 다른 에러(RTT를 넘는 등)가 발생하면 재전송

※ flow control에도 중요

RTT(Round Trip Time)

- 송신한 data에 대해 ACK가 돌아오기까지 걸린 시간

※ Error control

TCP는 ARQ(Automatic Repeat Request, 재전송 기반 에러 제어)를 사용

- Checksum
- Acknowledgement
- Retransmission
  - Retransmission after RTO

  => RTO 값이 크면 저장하고 있어야 할 segments가 많아짐
  - Retransmission after Three duplicate ACK segments

Retransmission after RTO
- RTO(Retransmission Time-Out) : retransmission timer
  - RTT를 기반으로 동적으로 계산됨
  - 첫 패킷의 경우 RTT 값을 알 수 없기 때문에 InitRTO를 사용(linux에서는 1초)
- 전송했지만 ACK를 받지 못한 모든 segment에 대해서 하나의 RTO timer가 돌아감
- 타이머가 끝날 경우 ACK를 받지 못한 첫 segment를 다시 전송

Retransmission after Three duplicate ACK segments
- 3번의 중복되는 ACK segment를 받을 경우 바로 missing segment 재송신
- RTO method는 RTO value가 크면 시간이 많이 소요됨 + 한 segment가 전송되지 않은 다음 너무 많은 segments를 받아야 할 수도 있음
  => RTO method에서는 의미 없는 전송을 많이 받아야 할 수 있지만, 이 method는 의미 없는 전송을 최대 3번만 받으면 됨

ARQ의 종류
- Stop-and-wait : segment 하나씩 전송, NAK 받으면 재전송
- Go-Back-N : segment 여러 개 전송, NAK 받으면 ACK num부터 모두 재전송
  - 수신 측에서는 재전송받으면 이전에 받았던 ACK 이후의 segment는 폐기
- Selective-Repeat : segment 여러 개 전송, NAK 받으면 ACK num만 재전송
  - 재전송받은 segment와 이전에 받았던 segment들을 재배열해서 원래 데이터 만듦
  - 프레임 재배열 등의 추가 구현, 버퍼가 필요 -> 구조가 복잡

Adaptive ARQ도 있지만, 비용이 많이 들어 사용하지 않음

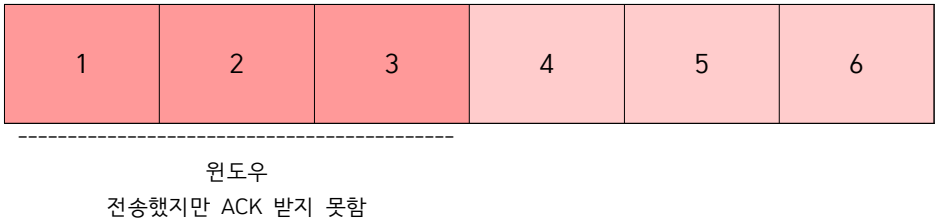
※ TCP는 GBN ARQ를 사용하나, SR ARQ를 사용하나?
- 두 방법을 혼합해서 사용!
- Go-Back-N과의 비교
  - 비슷한 점
  ACK를 받지 않은 segment의 개수에 제한이 있음
  - 다른 점
  GBN은 이후 모든 segment 재전송, TCP는 맨 처음 segment만 재전송
- Selective-Repeat과의 비교
  - 비슷한 점
  패킷이 유실되었을 때 모든 패킷을 재전송하지 않음
  - 다른 점
  수신할 때 SR은 패킷 모두에 대해서 ACK가 필요하지만, TCP는 다음에 받을 패킷 번호를 포함한 ACK를 한 번만 보냄

Sliding window

송신자에게 window size(자신의 현재 남은 buffer 양)를 ACK에 같이 보내서 수신 가능한 만큼을 한 번에 송수신하는 TCP의 송수신 기법
- 처음 window size는 3-way handshake를 할 때 정해짐
- 장점
  - 일일이 하나씩 ACK를 받는 stop-and-wait보다 전송속도가 빠름(WSCALE을 최대로 적용하면 최대 1GB를 ACK 응답 없이 연속적으로 전송 가능함)
  - 송/수신 측의 지속적인 통신을 통해 윈도우 크기 또한 조절 가능

e.g.

송신처에서 widnow가 [1, 3]일 때 1, 2, 3을 전송



=> 수신처는 자신의 처리속도에 맞게 데이터를 처리

=> ack num=4, window size=1을 송신처로 전송

=> 송신처는 window를 [2, 4]로 한 칸 slide하고 4를 전송함

포트 번호

---

application과 TCP/IP 통신을 연결하는 통로

⇔ 포트 번호를 이용해 data를 송/수신하는 app 특정

16bit => 65536개 존재

- 1 ~ 1023 : Well-known port
  - system에서 사용
- 1024 ~ 49151 : registered port
- 49152 ~ 65536 : dynamic port

Well-known port

20	FTP data	67	DHCP server	161	SNMP request
21	FTP control	68	DHCP client	162	SNMP trap
23	TELNET	69	TFTP	443	HTTPS
25	SMTP	80	HTTP	520	RIP
53	DNS	110	POP3		

e.g. 웹페이지를 받고 싶다 => server의 80번 port에 요청 => 웹페이지 데이터 수신

소켓

protocol, IP, port number로 정의됨

프로세스가 네트워크를 통해 데이터를 주고받기 위해서는 반드시 socket을 거쳐야 함

하나의 프로세스에서도 같은 protocol, IP, port로 여러 개의 socket을 가질 수 있음

UDP

UDP header

- 8 octet으로 이루어짐

Source port number(16)	Destination port number(16)	Total Length(16)	Checksum(16)

- Total Length : header와 data를 합친 datagram의 전체 길이
- Checksum : TCP header의 그것과 같음
- 뒤에 데이터를 더하면 UDP datagram이 완성됨

TCP와 UDP의 비교

TCP	UDP
정확한 전송	신뢰성 낮음
저속	고속

- TCP : 기능이 많지만 무거움
  UDP : 필요한 기능만 있어서 가벼움

UDP의 용도
- 고속, 실시간 송수신이 필요한 app
  e.g. VoIP, streaming
- broadcast가 필요한 app
  e.g. DHCP discover
  ∵ broadcast는 TCP로 불가능함(broadcast를 TCP로 하려면 모든 수신처와 3-way handshake를 해야함)
  DHCP의 경우 IP에 해당하는 수신처가 없을 수도 있음
  반면, UDP는 데이터가 1개로 끝나기 때문에 대역 소비 ↓, 버퍼 유지 X, 상대 주소 몰라도 수신 가능

NAT

Global IP address : Internet에 연결하기 위해 사용하는 IP
- ICANN이 관리
- public IP라고도 불림

Local-scope IP address : Internet이 아닌 local network에서 사용
- local IP, private IP라고도 불림

Private IPv4 addresses

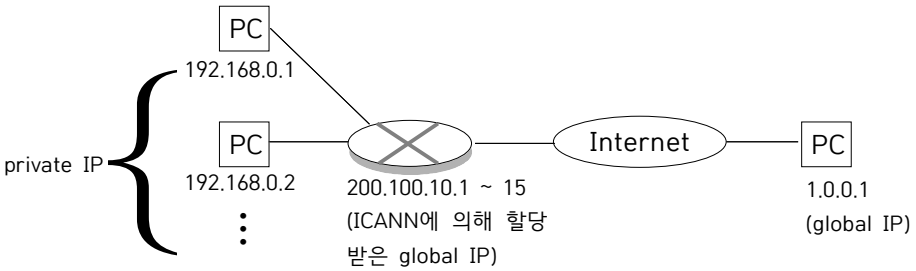
Class	IP	octet 1	octet 2
class A	10.0.0.0 ~ 10.255.255.255	00001010	xxxxxxxx
class B	172.16.0.0 ~ 172.31.0.0	10101100	0001xxxx
class C	192.168.0.0 ~ 192.168.255.0	11000000	10101000

global IP 고갈 문제의 해결 방안이 classless addressing, IPv6, NAT

NAT(Network Address Translation)

- 내부 network끼리는 private IP 사용, 외부와 통신할 때는 NAT를 이용해서 router에 할당된 global IP 중 하나를 사용해 통신
- static NAT : private IP, public IP가 1:1 mapping됨
  dynamic NAT : 아래에 설명하는 것처럼, N:N으로 접속할 때마다 연결되고 일정 시간 후 파기됨
- NAT table은 일정 시간 후 파기됨
- 동시 외부 network 접속자 수는 할당받은 global IP 개수로 제한됨
- 내부 network 보호 가능
  ∵ 외부 침입자가 공격하기 위해서는 private IP를 알아야 함

e.g.



192.168.0.1에서 1.0.0.1으로 송신하기 위해서는,

1. private IP 192.168.0.1을 할당받은 public IP 중 하나(e.g. 200.100.10.3)으로 바꿈
2. NAT table에 변환 기록 저장
3. 200.100.10.3으로 통신
4. 1.0.0.1로부터 수신받으면 NAT table을 참고해서 192.168.0.1로 보냄

NAPT

NAPT(Network Address Port Translation)

NAT처럼 private ↔ public으로 IP를 변환하면서 port를 사용해 하나의 public IP를 여러 PC가 사용가능하게 함

e.g.

192.168.0.1:1024 ⟷ NAPT ⟷ 200.100.10.5:6001
192.168.0.2:1025 ⟷ NAPT ⟷ 200.100.10.5:6002

- NAT처럼 변환한 것은 NAT table 이용(routing table이랑 다른 것!)
- 사용하지 않는 port로 packet을 수신한다면 파기 => 보안 강화
  => LAN 내부에서 외부로 공개하고 싶은 server가 있는 경우, 미리 NAT table에 변환을 저

장시켜둠(= 정적 NAPT, 수동으로 변환을 입력해둠)
- FTP의 경우, IP header에 송/수신처 IP, TCP header에 송/수신처 port, data에 송신처 IP, port가 들어감
  => NAT에서는 IP header만 변환되기 때문에 data 부분은 private IP가 그대로 남음
  => 전송 불가능
  data 부분에 송신처의 정보(IP, port)가 저장되는 것은 모두 NAT와 같이 사용할 수 없음
  (NAT를 수행하는 router가 개별적으로 대응해야 함)

IP Masquerade

Linux의 NAT 기능
내부 PC들이 리눅스 server를 통해서 외부 네트워크에 접속하게 해주는 것
포트 번호까지 port forwarding 시켜줌(수동으로 port forwarding을 해놓은 경우 static IP masquerade라고도 함)
- 내부 PC들의 요청은 server를 MASQ를 통해서 변환 후 연결되기 때문에 외부에서는 server의 IP만 알 수 있고 내부 PC의 존재를 알 수 없음
- 보안성 ↑↑
- 외부에서 먼저 내부와 통신을 시도할 수 없음 => port forwarding으로 해결
- IP masquerade를 이용하면 Load balancing이 가능한 듯(내부 네트워크를 이용한 병렬 처리)

Port forwarding
- 특정 port에 전송된 데이터를 미리 지정한 local IP에 전송하는 기능

5~7계층

5, 6, 7계층은 TCP/IP의 경우 통합해서 하나의 protocol이 된 경우가 많음

5계층: Session

Dialog control, Synchronization 수행
- Dialog control : data 송수신을 session으로 성립하도록 관리
- Synchronization : 송/수신 간의 전송 동기화를 위해 전송하는 데이터의 일정 부분마다 synchronization point를 제공(데이터 점검, 복구에 사용)

6계층: Presentation

Translation, Encryption, Compression 수행
- 컴퓨터 간의 data format 차이를 변환해서 없앰 => HW, OS에 따른 차이 없앰
  e.g. A(ASCII 사용) →(변환) 네트워크 표준 →(변환) B(EBCDIC 사용)
- 압축, 암호화 등 app과 관계없는, data의 전송을 위한 변환 수행

7계층: Application

응용 계층의 특정 서비스(파일 전송, 메일 등) 제공
- app의 목적에 따라 적절한 protocol 제공

e.g.

서비스	app	7계층
WWW	browser	HTTP
파일 전송	FTP client	FTP
메일 전송	Mail client	SMTP

OSI 참조 모델과 요약

'각 계층은 독립되어 있다'

Layer	상위 계층에 제공하는 기능	Description
layer 7	app별로 도달한 data 처리	Network service 제공
layer 6		Data format 변환
layer 5		Session 관리
layer 4	3계층 이하의 기능으로 PC에 도달한 data 처리	신뢰성 보증(TCP, UDP)
layer 3	2계층 이하의 기능을 이용한, 다른 network 간의 접속	Internetwork, IP addressing, Routing
layer 2	1계층의 기능을 이용한, 서로 다른 기기의 data 송수신	Network 내의 통신(Ethernet)
layer 1		전기, 신호