Computer Network

- OSI Reference Model, TCP/IP Model 동작 원리 ~ 네트워크 프로토콜 작동원리에 대한 분석 -

INDEX

Client-Server network Model : 클라이언트들이 한 서버에서 네트워크 모델
Peer-to-peer network Model : 클라이언트가 서버의 역할을 하는 네트워크 모델

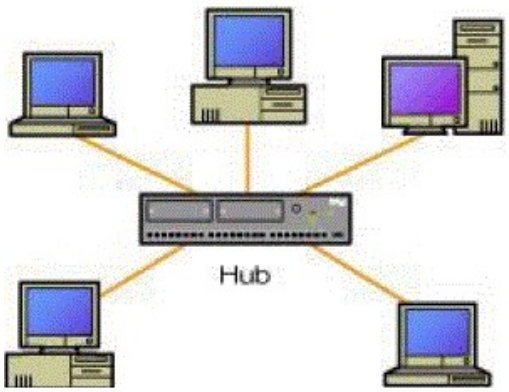
※ 개요
- 사물인터넷, 사물지능통신 (M2M: Machine to Machine)
 : 사물들이 센서를 통한 정보 교환으로 스스로 일을 처리 [<사물 - 사람> 자동차 후방센서]
 Iot(Internet of things)는 M2M의 발전된 형태 [<사물 - 사물> 자율주행 자동차][1)]

- 데이터
Bit -> Data -> Information -> Knowledge -> Wisdom
데이터는 비트(Bit)들의 의미있는 모임이고, 의미없는 정보(Information)이다.

- Protocol(프로토콜)★
프로토콜 : 컴퓨터들간의 원활한 통신을 위해 정해놓은 규약
Ex) Physical Layer에서 신호를 주고받을 때, 0과 1을 어떻게 구분할 것인지 정함.
 (송수신을 위해 사용할 부호화 방식을 미리 정해놓는 것 [차등 맨체스터])

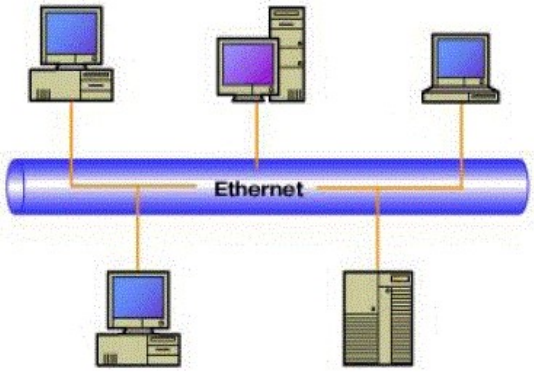
- 네트워크 구조
1. LAN(Local Area Network)
 비교적 근거리 통신 네트워크 구조 <범위수준 : 대학 캠퍼스 네트워크>

 1.1. 성형(Star Topology)
 : 중앙 서버 컴퓨터가 모든 클라이언트를 관리
 고장의 발견, 수리, 노드의 증설 및 이전의 편의성이 크다.
 한 노드의 Error가 전체 네트워크에 미치는 영향이 없다.
 중앙 컴퓨터 고장 -> 전체 네트워크 마비



 1.2. 버스형(Bus Topology)
 : 한 개의 통신회선에 여러 대의 단말장치가 연결되어 있는 형태
 양방향 데이터 전송, 전송된 데이터는 모든 노드에서 수신 가능
 구축이 쉽고, 비용이 적게 든다.
 케이블의 한 곳이 끊어지면 통신 불가능
 많은 컴퓨터를 연결하면 네트워크 성능이 저하된다.
 일자 이더넷 -> 전송거리에 제한이 있다.

1) M2M과 Iot의 차이에 대해 교수님은 큰 차이를 두지 않으심.



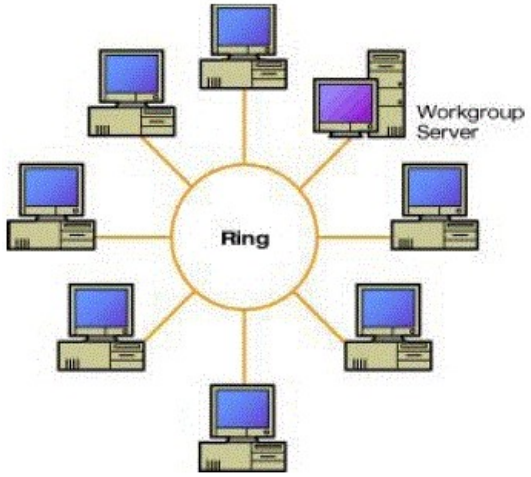
1.3. 링형(Ring Topology)

: 데이터의 전송이 단방향이고 병목현상이 드물다.

양방향으로 접근이 가능 -> 통신회선 장애에 대한 융통성이 있다.

컴퓨터 한 대를 추가 연결 -> 선을 끊었다가 다시 연결 -> 노드 증설의 불편성

회선이 끊어지면 전체 네트워크 마비



2. MAN(Metropolitan Area Network)

대도시 정도의 넓은 지역을 연결하기 위한 네트워크

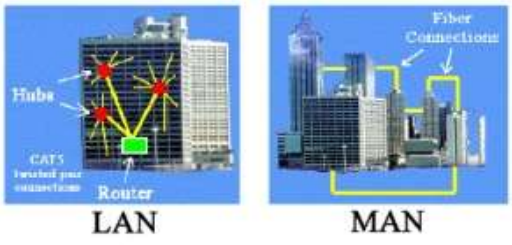
LAN들의 모임

3. WAN(Wide Area Network)

광범위한 지역을 수용, 수백~수천km 범위의 광역 네트워크

MAN들의 모임

* LAN, MAN, WAN 비교



4. PAN(Personal Area Network)
10m 이내의 단거리 네트워크
Ex) Bluetooth, zigbee, shiri

5. BAN(Body Area Network)
인체 중심 네트워크 (BSN : Body Sensor Network)
Ex) 팔뚝에 기계 달고 달리기하면서 여러가지 측정하는 것

- 표준화(Standardization)
* 표준화의 필요성?
표준화된 규약이 존재하지 않으면, 데이터를 보내는 형식과 데이터를 받는 형식이 서로 달라져서 제대로 된 송수신을 할 수 없음.
Ex) 한글2014 작성 -> Word2014 열람 (오류 : 한글에만 있는 기능을 사용한 경우)
<몇몇의 소프트웨어는 호환성이 떨어져도 괜찮지만 네트워크는 괜찮지 않다>

* De facto 표준화 vs De jure 표준화 ★
De facto Standardization : 많은 사람들이 사용 -> 암묵적 표준 (UNIX)
De jure Standardization : 권위있는 단체에서 제정한 표준

- 표준화 기구
* ISO -> OSI 7 Layer Model

* ANSI -> ASCII code (De facto) 지정
** ASCII code란?
현재 우리가 사용하는 컴퓨터에서 어떠한 글자를 입력했을 때, 그것들에 번호를 매겨놓은 것
Ex) Human : A -> ASCII : 65 -> Computer : 1000001(binomial number)

* IETF(Internet Engineering Task Force)
변화하는 네트워크 환경에 따라 새로운 기술들을 제시하는 인터넷 표준안 제정
Ex) wg1 -> RFC100 : 이러한 기능을 이렇게 사용하도록 규약을 맺는 것이 어떨까?
wg2 -> RFC100 좋은데? 이렇게 쓰자. -> 사용
wg3 -> RFC100의 이러한 규약을 이렇게 하는 더 낫지 않을까? -> 수정 -> RFC101
(wg : WorkingGroup)
위와 같은 과정을 거쳐서 최종적으로 안정화된 표준안을 인터넷 표준안으로 결정

** RFC 3단계 표준화 과정
제정단계 표준안 -> 6개월 경과, 2회 실험[2] -> 표준안 전단계 -> 4개월 경과, 2회 실험
-> 인터넷 표준안(안정적으로 동작되는 것이 확인된 문서)

'나머지 표준화 기구 생략'

- 인터넷의 발전
최초 : 미 국방성 산하의 DARPA의 ARPANET 등장
TCP/IP 사용 인터넷 등장 -> 널리 확산
WorldWideWeb(www)의 발전 -> 현재의 인터넷

2) 이 단계에서 다른 표준안으로 수정이 일어날 수 있고, 수정된 표준안은 다시 제정단계 표준안이 된다.
(Ex. RFC100 -> 3개월 후, 수정 -> RFC101 -> 6개월 경과, 2회 실험 -> 표준안 전단계)

- IPv6

-> 현재 우리가 흔히 사용하는 ip주소는 IPv4로 구성되어 있다.

IPv4 : xxx.xxx.xxx.xxx ($0 \leq xxx \leq 255$) -> 2^{32}개의 주소 할당 가능 (약 43억)

IPv6 : xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx (xxxx는 16진수[0000~FFFF])

-> 2^{128}개 주소 할당 가능

* 출현 배경

전세계 인구 60억 -> 할당 가능한 주소 43억 -> IPv4의 할당가능주소수의 한계

-> IPv6 : 2^{128}개 -> 향후 2000년은 문제 없음.

※ 데이터 전송기술과 전송미디어

- 정보 통신의 종류

단방향 통신(Simplex) : 방송, 무선 마우스, TV 등

양방향 통신(Full-duplex) : 전화기, 인터넷 등

Half-duplex : 한쪽이 보낼때는 다른쪽은 받기만 하고, 역도 성립 [무전기]

- 통신의 기본 동작

전송하기 위한 정보를 물리적 신호로 변조, 수신, 복조

[Data -> Modulator -> Sending -> Receiving -> Demodulator -> Data]

- 통신로 (물리적 매체)

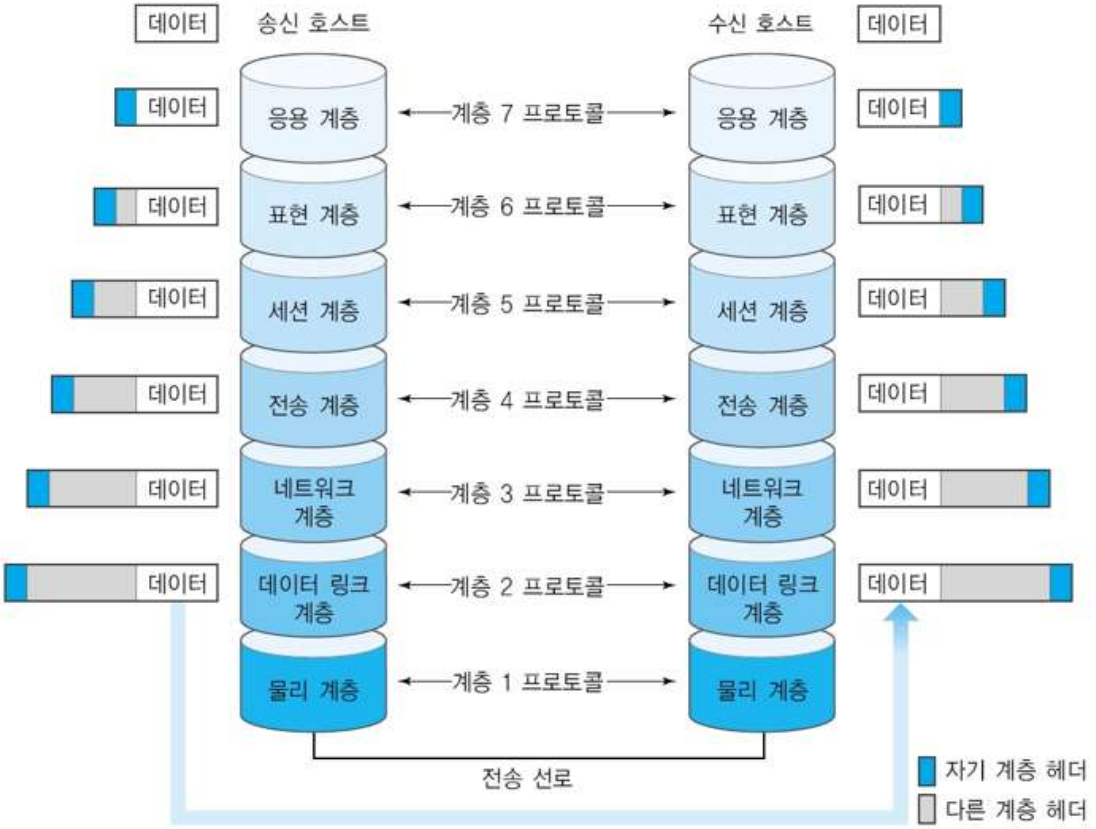
통신을 하기 위해서는 기본적으로 물리적인 매체가 있어야 한다. (전선, 공기, 진공 등)

유선통신 : 전선/동축케이블로 전기신호를 주고받는 통신

무선통신 : 공기 또는 진공으로 마이크로파 등을 주고받는 통신

광통신 : 광섬유로 빛을 주고받는 통신

- OSI 7 Layer Model ★★



- Description

OSI Reference Model은 ISO(국제표준화기구)에서 지정

4계층이상 : End-to-End Node Communication

3계층이하 : Neighberhood Node Communication

여기서 3계층을 IP-Layer, 4계층을 TCP-Layer로 바꾸고,

5, 6, 7계층의 일을 ApplicationLayer가 하는 것을 TCP/IP Model 이라고 한다.

- OSI 7계층 모델의 통신 원리 ★★

A가 B에게 데이터를 전송하려고 한다.

A의 7계층에서 전송하고자 하는 데이터를 만들어서, Header를 붙여 6계층으로 보낸다.

A의 6계층에서 7계층에서 보낸 Payload에 또 Header를 붙여 5계층으로 보낸다.

5, 4, 3, 2계층에서 상위 계층의 Payload를 받아 Header를 붙여 하위 계층으로 보낸다.

1계층(물리계층)에 도달한 Payload를 전송 선로를 따라 다른 컴퓨터의 1계층으로 보낸다.

다른 컴퓨터(A가 보낸 Payload를 받은)는 데이터를 2계층으로 올리고, 2계층에서 2계층 header를 제거하고, 3계층으로 보낸다.

3계층(NetworkLayer)는 이 데이터가 자신에게 온 데이터인지 주소를 확인한다.

case X) 자신의 주소가 맞으면 header를 제거하고 4계층으로 올려보낸다.

case Y) 자신의 주소가 아니면 다시 2계층으로 내려보내서 다른 컴퓨터로 데이터를 전송

X) 4계층 ~ 7계층까지 Demodulate를 하며 데이터를 올려보낸다. [전송 완료]

Y) 1-2-3-2-1계층을 반복하며 받는 주소가 일치하는 컴퓨터를 찾는다.

-> Python을 통해 본 OSI 7 Layer Model Network System

OSImodel.py

```
import random;

# OSI 7 Layer Model class Structure

class OSI:
# Generator
def   init  (self, userName):
# ipAddress is IPv4 version, generated randomly every time generated OSI class
self.ipAddress =
f"{random.randint(0,255)}.{random.randint(0,255)}.{random.randint(0,255)}.{random.
randint(0,255)}";
self.userName = userName;

# Modulating and Demodulate methods in DataLinkLayer(level2) to
PresentationLayer(level6)
# MoDem method Simply attach or detach some string excepting NetworkLayer
# Look Methods below
def DataLinkLayerModulate(self, data):
data = data + "datalink";
return data;
# NetworkLayer have to operate confirming receiver's IP address so when modulate,
put receiver's IP address
# then this Layer attach the address to data [This is NetworkLayer's header]
def NetworkLayerModulate(self, data, receiverAddress):
data = data + receiverAddress;
return data;

def TransportLayerModulate(self, data):
data = data + "transport";
return data;

def SessionLayerModulate(self, data):
data = data + "session";
return data;

def PresentationLayerModulate(self, data):
data = data + "presentation";
return data;

def ApplicationLayerModulate(self, data):
data += "application";
return data;

def DataLinkLayerDemodulate(self, data):
data = data.replace("datalink", "");
return data;

def NetworkLayerDemodulate(self, data, receiverAddress):
data = data.replace(receiverAddress, "");
return data;

def TransportLayerDemodulate(self, data):
data = data.replace("transport", "");
return data;

def SessionLayerDemodulate(self, data):
data = data.replace("session", "");
return data;

def PresentationLayerDemodulate(self, data):
data = data.replace("presentation", "");
return data;

# If receive message, printing the Sender & Receiver's information and message
content for Applicate method
def Applicate(self, sender, data):
# Demodulate data for ApplicationLayer header
data = data.replace("application", "");
print(f"ReceiverIP : {self.ipAddress}\nReceiverName : {self.userName}\nMessage :
{data}\n");
print(f"SenderIP : {sender.ipAddress}\nSenderName : {sender.userName}");

# Physical Layer Transform Datas to Binary Signals
# Transform character to ASCII Code int, (Not using)
def TransformToElectronicSignal(self, char):
#if char == " ": return 32;
#if char == "!": return 33;
#if char == "\"": return 34;
#if char == "#": return 35;
#if char == "$": return 36;
```

```
#if char == "%": return 37;
#if char == "&": return 38;
#if char == " return 39;
#if char == "(": return 40;
#if char == ")": return 41;
#if char == "*": return 42;
#if char == "+": return 43;
#if char == ",": return 44;
#if char == "-": return 45;
#if char == ".": return 46;
#if char == "/": return 37;
#if char.isdigit(): return int(char) + 48;
#if char == ":": return 58;
#if char == ";": return 59;
#if char == "<": return 60;
#if char == "=": return 61;
#if char == ">": return 62;
#if char == "?": return 63;
#if char == "@": return 64;
#if char == "A": return 65;
#.........skip..........
#if char == "~": return 126;
return ord(char); # ord() method transform char to ASCII int

# Int to Binary String method (Not using)
def binaryTransform(self, integer):
return "{0:b}".format(integer);

def PhysicalLayerDemodulate(self, signals):
data = "";
for signal in signals:
data += chr(int(signal, 2));
return data;

def PhysicalLayerModulate(self, data):
signals = [];
for char in data:
signals.append("{0:b}".format((ord(char))));
return signals;

# the sendMessage() method executed,
# sending content is modulated through PresentationLayer(level6) to
DataLinkLayer(level2)
def sendMessage(self, content, receiverAddress, Groups):
content = self.ApplicationLayerModulate(content);
content = self.PresentationLayerModulate(content);
content = self.SessionLayerModulate(content);
content = self.TransportLayerModulate(content);
content = self.NetworkLayerModulate(content, receiverAddress);
content = self.DataLinkLayerModulate(content);
# In Physical Layer, Data is changed to Binary Signal
signals = self.PhysicalLayerModulate(content);
# and PhysicalLayer(level1) send the modulated signals to Groups(OSIs
list[people 1, people 2,..., Kate, Bill])
self.move(signals, Groups);

# move() method have PhysicalLayer(level1)'s roll [move through OSI Group]
def move(self, signals, OSIs):
# Sended message move through OSIs in Group (PhysicalLayer[level1])
for OSI in OSIs:
# Sended signals are demodulated in Physical Layer
content = self.PhysicalLayerDemodulate(signals);
# contentShow is used to confirm finally
contentShow = self.PhysicalLayerDemodulate(signals);
# DataLinkLayer(level2) demodulate the message
content = OSI.DataLinkLayerDemodulate(content);
# In NetworkLayer(level3), confirm OSI's IP address whether that is equal to
receiver's IP address
if OSI.ipAddress in content[-15:]:
print(f"Sended Signals in Physical Layer : {''.join(signals)}");
print(f"BinarySignal to String(Demodulated Data only Physical Layer) :
{contentShow}\n");
receiverAddress = OSI.ipAddress;
# if find the receiver, the message move to TransportLayer(level4) and go through
HighLevel Demodulators
# In medium, print() is confirming medium demodulated data
print(f"Demodulated Data in DataLink Layer : {content}");
content = OSI.NetworkLayerDemodulate(content, receiverAddress);
print(f"Demodulated Data in Network Layer : {content}");
content = OSI.TransportLayerDemodulate(content);
print(f"Demodulated Data in Transport Layer : {content}");
content = OSI.SessionLayerDemodulate(content);
print(f"Demodulated Data in Session Layer : {content}");
```

```
content = OSI.PresentationLayerDemodulate(content);
print(f"Demodulated Data in Presentation Layer : {content}\n");
# Now, the content is completely demodulated except Application Layer Demodulating
# Receiver execute the application, can see the message
OSI.Applicate(self, content);
else:
# if IP address is not equal to receiver's IP address, return to
DataLinkLayer(level2)
# At returning, repeat modulate
content = OSI.DataLinkLayerModulate(content);
```

-> OSI model class File

SendAndReceiveNetworkExample.py

```
from OSImodel import OSI;

# Generate a thousand of OSI classes and appending OSIs list
# The OSIs' name is people 1 ~ people 1000
OSIs = [OSI(f"people {i}") for i in range(1, 1001)];

# Generate OSI classes named Kate, Bill (Sender & Receiver)
Kate = OSI("Kate");
Bill = OSI("Bill");

# Put Kate and Bill in OSIs list
OSIs.append(Kate);
OSIs.append(Bill);

# Confirm the process operate well
for OSI in OSIs:
print(OSI.userName, OSI.ipAddress);
print("\n");

# Kate send a message the content is "Hello Bill" to Bill
Kate.sendMessage("Hello Bill", Bill.ipAddress, OSIs);
```

※ 주의) 들여쓰기가 전혀 포함되지 않았음

- console window

```
people_1 244.64.88.102
people_2 203.118.254.81
people_3 171.87.200.42
people_4 19.200.85.85
people_5 175.214.82.190
people_6 71.247.14.62
people_7 8.167.160.76
```

========= skip people_8 ~ people_993 =========

```
people_994 52.189.47.8
people_995 84.11.234.215
people_996 42.87.44.106
people_997 95.120.164.231
people_998 123.78.169.137
people_999 183.28.215.61
people_1000 190.177.179.245
Kate 148.27.206.118
Bill 146.130.92.233


Sended Signals in Physical Layer : 1001000110010111011001101100110111110000010000101101001110110011010011000011110000111000011011001101
BinarySignal to String(Demodulated Data only Physical Layer) : Hello Billapplicationpresentationsessiontransport146.130.92.233datalink

Demodulated Data in DataLink Layer : Hello Billapplicationpresentationsessiontransport146.130.92.233
Demodulated Data in Network Layer : Hello Billapplicationpresentationsessiontransport
Demodulated Data in Transport Layer : Hello Billapplicationpresentationsession
Demodulated Data in Session Layer : Hello Billapplicationpresentation
Demodulated Data in Presentation Layer : Hello Billapplication

ReceiverIP : 146.130.92.233
ReceiverName : Bill
Message : Hello Bill

SenderIP : 148.27.206.118
SenderName : Kate

Process finished with exit code 0
```

Code-References : https://github.com/UnprettyCoder/OSI-References-model
-> OSI 7계층 모델의 작동원리 이해를 돕기 위한 파이썬 프로젝트

** 각각의 Layer에서 자신이 다루는 데이터 단위를 부르는 명칭

Layer	Name of DataUnit
Application Layer	Data
Presentation Layer	Data
Session Layer	Data
Transport Layer	Segment
Network Layer	Packet
DataLink Layer	Frame
Physical Layer	Bits

- Physical Layer [1계층]
역할 : 2계층에서 내려온 Payload를 전기신호로 바꾸어 전송
즉, 이 계층에서는 bit(0 or 1)를 어떻게 잘 주고받을지만을 고민

* 부호화 (Encoding)
전기적 신호로서 0과 1을 어떻게 판별할 것인지에 부호화한다.
통신을 주고 받는 부호화, 복호화 방식은 당연히 같아야 한다.
Ex) 맨체스터 부호화 -> 차등 맨체스터 복호화 -> 전혀 의미없는 데이터
맨체스터 방식 : 0 왼쪽, 1 오른쪽
차등 맨체스터 방식 : 0 유지, 1 반전 [현재 가장 많이 사용하는 방식]
** 많은 부호화 방식이 존재하는 이유 : 감쇄, 왜곡, 잡음으로 인한 데이터 전송의 어려움
-> 전기신호 부호화 방식을 좀 더 전송이 매끄러운 방식으로 업그레이드

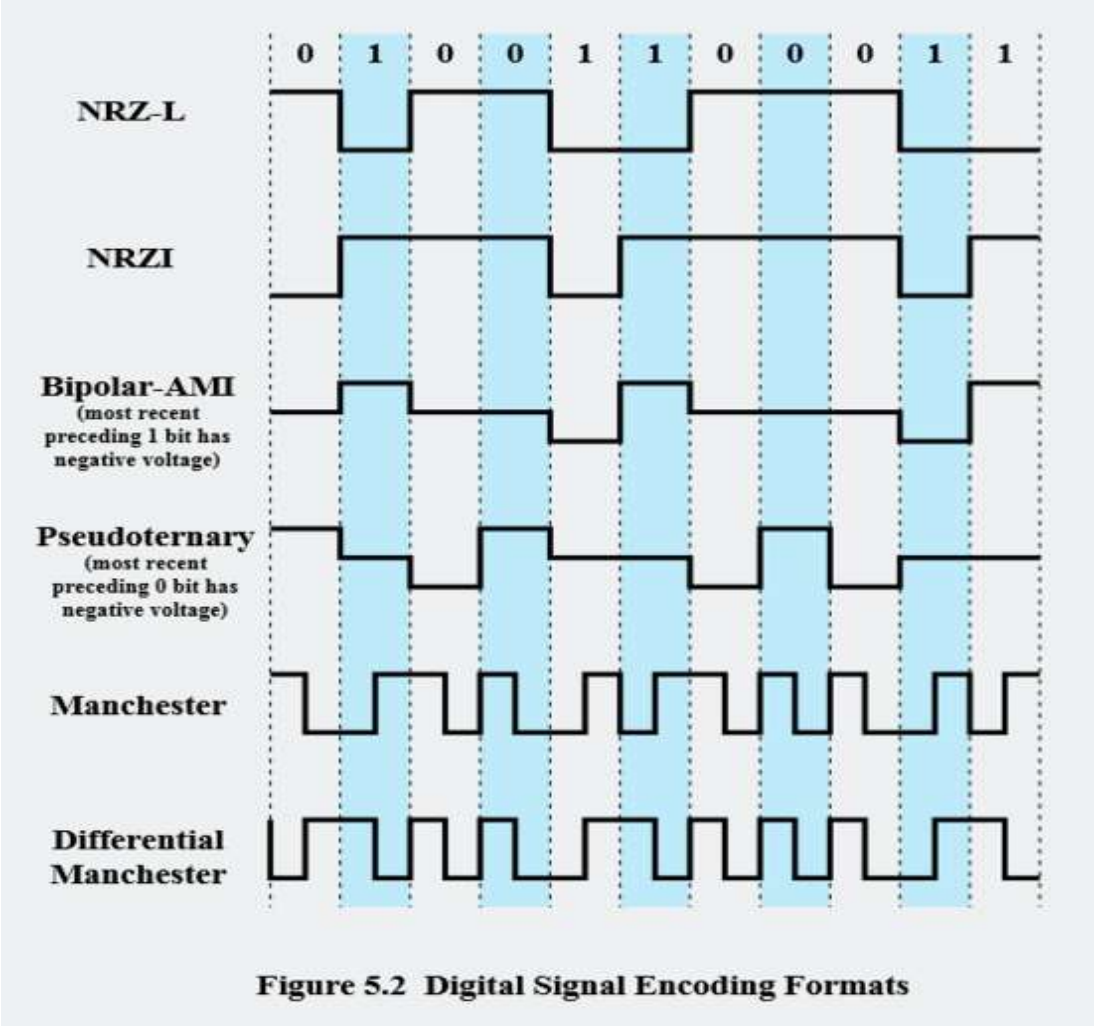
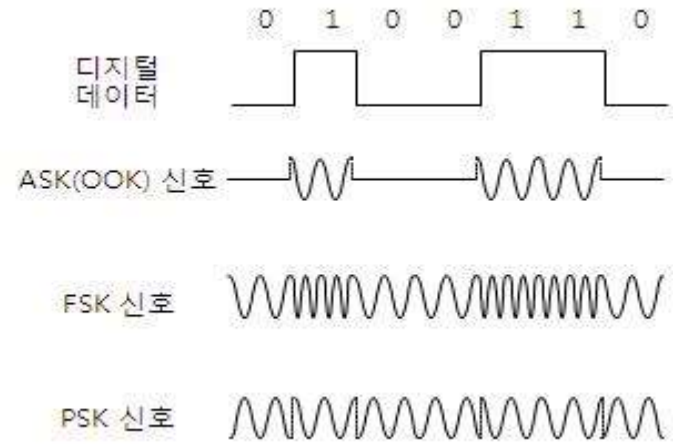


Figure 5.2 Digital Signal Encoding Formats

* Low Pass Filter : 저주파가 고주파보다 송수신에 유리하다.
-> 우리는 신호를 보낼 때 고주파 신호를 저주파 신호로 바꾸어주어야 한다.

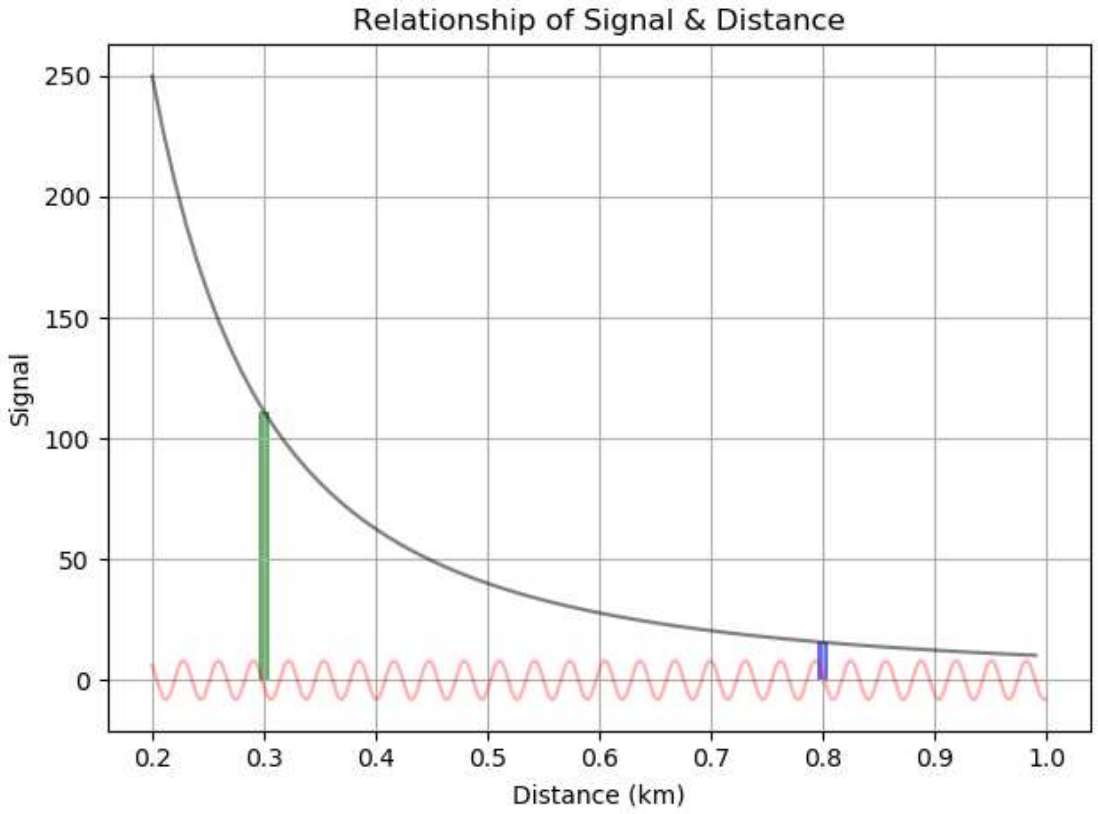
* 변조(Modulate) : 고주파 신호 -> 저주파 신호 변환
* 복조(Demodulate) : 저주파 신호 -> 고주파 신호 변환
-> Modem(모뎀) : Modulater + Demodulater

* 변조 방식



ASK(Amplitude Shift Keying) : 진폭 편의 변조
FSK(Frequency Shift Keying) : 주파수 편의 변조
PSK(Phase Shift Keying) : 위상 편의 변조 [가장 유리한 변조 방식]
-> 대용량 데이터 전송 가능

* 감쇄, 왜곡, 잡음



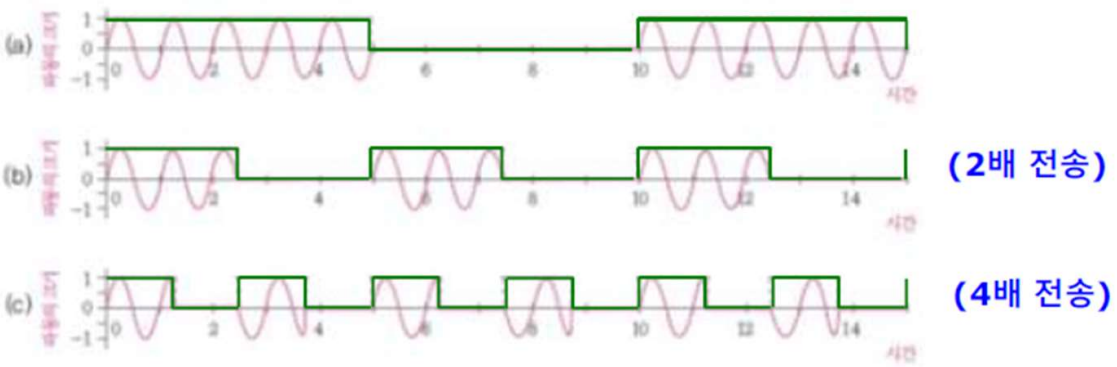
감쇄 : 전송 매체의 저항에 의해 신호의 감쇄현상이 발생한다. (Black line)
-> 신호의 세기는 거리의 제곱에 반비례하게 감소한다.

$$Signal \propto \frac{1}{Distance^2}$$

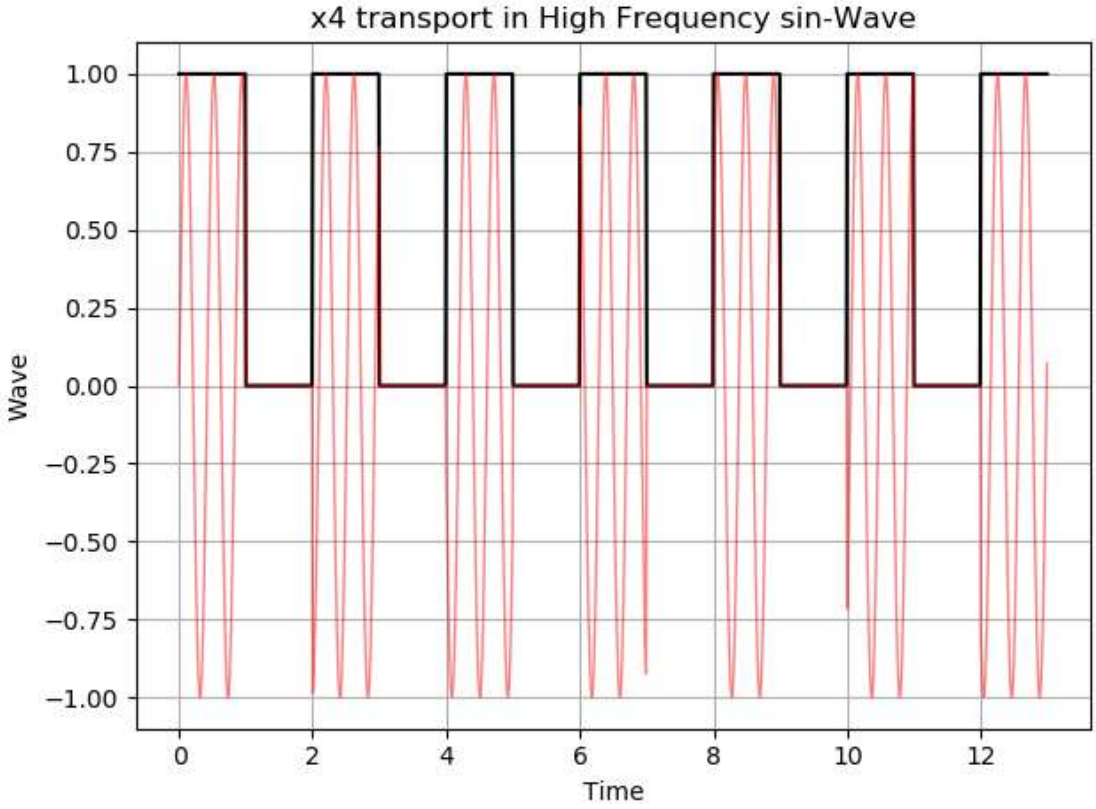
왜곡 : 전송 도중에 외부 영향에 의해 송신신호와 수신신호가 달라지는 현상
-> 멀리 떨어진 곳으로 데이터를 보낼 때는 잡음에 의해 왜곡이 발생한다.
-> 멀리 떨어진 곳으로 데이터를 보낼 때는 신호의 분산에 의해 왜곡이 발생한다.
잡음 : 도전체에서 전자의 열운동으로 인해 Random하게 발생[white noise] (Red wave)
Ex) 거리가 0.3일 때, 신호의 세기가 잡음에 큰 영향을 받지 않는다. (gap이 크기 때문)
거리가 0.8일 때, 신호의 세기가 잡음에 큰 영향을 받는다. (gap이 작기 때문)

- 통신속도 향상 방안

1. 높은 주파수 반송파 이용
 같은 시간동안 더 많은 이진수를 실어 보내면?



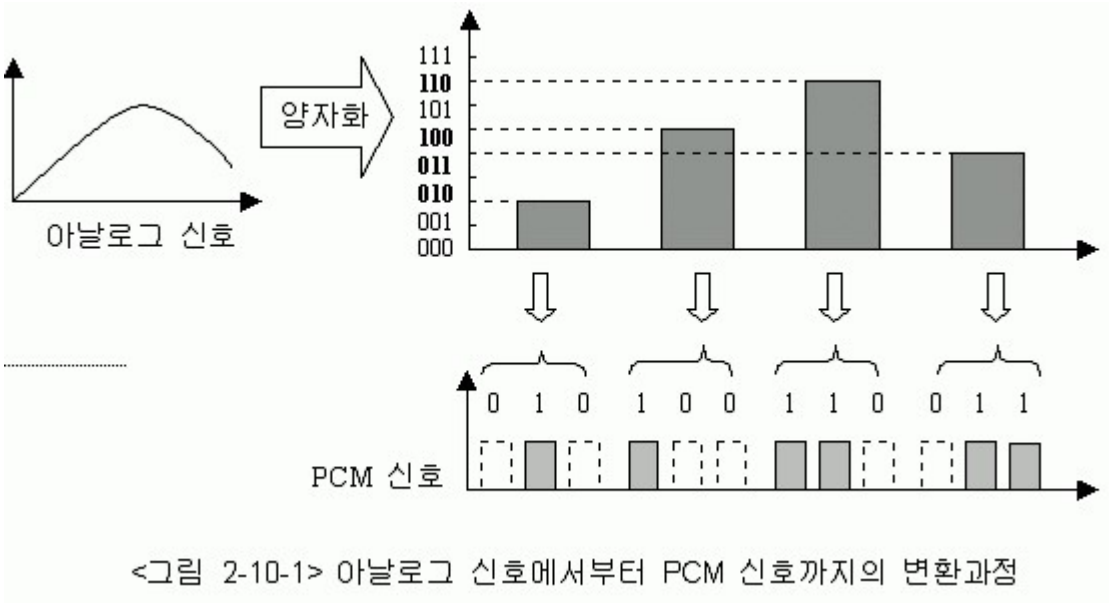
-> 그만큼 짧은 sin-wave(반송파)가 한 신호에 포함된다.
-> 너무 짧은 sin-wave는 신호를 구분하기가 힘들다.
** 해결방안 : sin-wave의 주파수를 크게한다.



-> 이런식으로 sin-wave의 주파수를 크게하면 전송의 속도를 높여도 한 비트에 포함되는 sin-wave가 반파장 정도밖에 안되는 경우를 방지할 수 있다. (신호의 신뢰성 향상)

2. M-ary 통신

M-ary 통신 : 2^m진 통신을 일컫는 말



<그림 2-10-1> 아날로그 신호에서부터 PCM 신호까지의 변환과정

-> 위 그림은 8진 통신의 예시이다.

2진 통신으로 1, 1, 0, 1, 0, 0을 보내기 위해서는 6번의 신호가 필요

4진 통신으로 11, 01, 00을 보내기 위해서는 3번의 신호가 필요 [전송 속도 2배]

8진 통신에서 110, 100을 보내기 위해서는 2번의 신호가 필요 [전송 속도 3배]

But, 2진 통신에서 0, 1은 신호가 있는지 없는지만 구분하면 됨.

8진 통신에서 000 ~ 111은 신호의 크기를 정밀하게 구분해야 함.

-> 너무 2^m진수를 높이면 신호를 구분하기가 힘드므로, 한계가 있다.

3. 다중화

3.1. 시 분할 다중화 (TDMA : Time Division Multiplexing Access)

: 같은 주파수 대역을 여러 사용자들이 시간에 따라 공유

Ex) A가 0~2시 이용, B가 2~4시 이용, C가 4~6시 이용

3.1.1. 통계적 시 분할 다중화 (STDMA : Statistical TDMA)

: 통계적으로 어떤 사람이 어떤 시간에 네트워크를 사용하는 것을 선호하는지를 분석하여 그 분석 결과에 따라 네트워크 사용시간을 할당하는 다중화 기법

-> 효율성은 올라가지만, 형평성은 떨어지게 된다.

3.2. 주파수 분할 다중화 (FDMA : Frequency Division Multiplexing Access)

: 여러 사용자에게 각각 사용할 수 있는 주파수를 할당

Ex) SK-Tel : 4GHz ± 0.5GHz, KT : 6Ghz ± 0.5Ghz, LG-U+ : 8Ghz ± 0.5Ghz

3.3. 파장 분할 다중화[광통신] (WDMA : WaveLength Division Multiplexing Access)

: 각 사용자들이 사용하는 빛의 파장을 다르게 하여 사용

Ex) A : RedLight(700nm), B : GreenLight(520nm), C : BlueLight(470nm)

3.4. 코드 분할 다중화 (CDMA : Code Division Multiplexing Access)

: 각 사용자들이 사용하는 소프트웨어의 코드를 다르게 하여 사용

Ex) Python파일을 Java interpreter로 번역하면 아무 의미가 없다.

-> 자신의 코드로 해석가능한 정보만을 송수신하는 다중화 기법

-> 같은 시간, 같은 주파수 대역을 사용해도 문제가 없다.

* 정보이론과 채널용량의 법칙

$C = 2B\log_2 N$ [하틀리 법칙]

C : 채널 용량, B : 채널 대역폭, N : 부호 레벨 수

-> 대충 대역폭이 크면 클수록 빠르고 정확한 전송을 할 수 있다는 법칙

- 교환기술(Switching)
 : 다수의 기기 상호 간에 최적의 연결성을 제공해주는 기술
 교환 네트워크 : 회선교환 네트워크, 메시지교환 네트워크, 패킷교환 네트워크
 방송 네트워크 : 패킷 라디오 네트워크, 위성통신 네트워크, 지역 네트워크

교환 방식의 비교

[표 2-4] 교환 방식의 특성

특징 \ 방식	회선교환	메세지교환	가상회선 패킷교환	데이터그램 패킷교환
전용 전송로	유	무	무	무
전송 단위	연속적인 데이터	메시지	패킷	패킷
메시지의 저장 여부	저장하지 않음	저장, 필요 시 검색	일시적 저장, 검색기능 없음	일시적 저장, 검색기능 없음
이용에 적합한 전송 형태	길이가 긴 연속적 전송	저속 메시지 전송	순간적인 대량 데이터의 고속전송	순간적인 대량 데이터의 고속전송
전송 경로의 형태	동일한 전송경로	메시지마다 경로설정	전체 패킷 전송을 위해 경로설정	각 패킷마다 경로설정
지연 시간 영향	연결호출 설정 지연, 전송 지연은 무시	메시지 전송 지연	연결호출 설정 지연, 패킷 전송 지연	패킷 전송 지연
과부하 시	연결호출 설정 중단	메시지 전송 지연 증가	연결호출 설정 중단 : 연결설정 후에는 패킷 전송 지연 증가	패킷 전송 지연 증가
코드 및 속도 변환	무	유	유	유
전송 데이터와 수신 데이터의 순서 일치 여부	일치	불일치	일치	불일치
대역폭	고정	동적사용 가능	동적사용 가능	동적사용 가능
회전 에러발생 시	다른 회선 재설정	여러 경로 중 선택	다른 회선 재설정	여러 경로 중 선택
오버헤드	연결설정 후 불필요	메시지마다 필요	각 패킷마다 필요	각 패킷마다 필요
응용 분야	실시간 대화형	실시간 대화형 부적합	실시간 대화형	실시간 대화형

Embedded Computing Laboratory
School of Information & Communication

* 회선교환 방식 (Circuit Switching)
미리 처음과 끝 연결해 놓음 (End to End Communication)
초기시간(circuit setup)이 오래걸린다.
circuit이 한번 setup 되고나면 대용량 데이터를 고속으로 전송하는 것이 가능하다.
한번 연결되어 사용중인 회선을 다른곳에서 사용할 수 없다.

* 메시지교환 방식 (Message Switching)
메시지를 옆 노드로 전달, 저장/전달하는 방식
초기시간이 거의 걸리지 않는다.
전달이 확실히 되었는지 확인할 수 없다. (Unstable)
한 회선을 계속해서 사용하지는 않기 때문에 융통성이 있다.

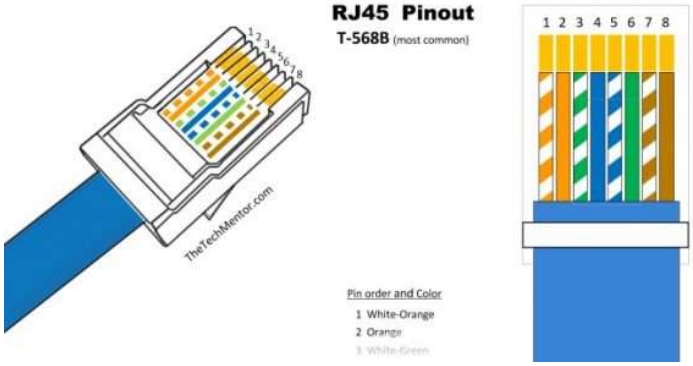
* 패킷교환 방식 (Packet Switching) ★
데이터를 패킷(Packet)단위로 분할하여 전송한다.
 데이터를 보내는 회선을 분할하여 보내는 방식 : datagram 방식
 데이터를 한 회선으로만 보내는 방식 : virtual circuit 방식

- 통신을 위한 물리적 매체
* 구리선(copper wire)
Twist pair : 구리선을 꼬아서 사용하는 것
 -> 꼬아서 사용해야 Noise에 강하다.
구리선은 장거리 통신이 힘들다.

* 동축케이블(굵은 구리선 1개를 사용)
속도와 안정성이 뛰어나다. But, 비싸다.

* 광섬유
잡음에 강하고, 빠르다.
장거리 통신에 최적이다.
But, 중간에 새로운 노드를 뚫는 것이 거의 불가능한 수준 [Spliter를 사용하면 가능하긴함]
Spliter, Repeater 등 부가장치의 비용이 비싸다.

** Physical Layer Protocol의 예시



우리가 사용하는 인터넷 랜선은 RJ45 라는 프로토콜(규약)에 의해 만들어진다.
1 ~ 8번까지의 선들이 각각 어떤 역할을 할 것인지 미리 정해놓는 것을 프로토콜이라고 하며, 이처럼 물리적인 요소가 포함된 규약은 Physical Layer에서 담당한다.

- DataLink Layer [2계층]

* Frame(프레임)

: DataLink Layer에서 다루는 데이터 단위

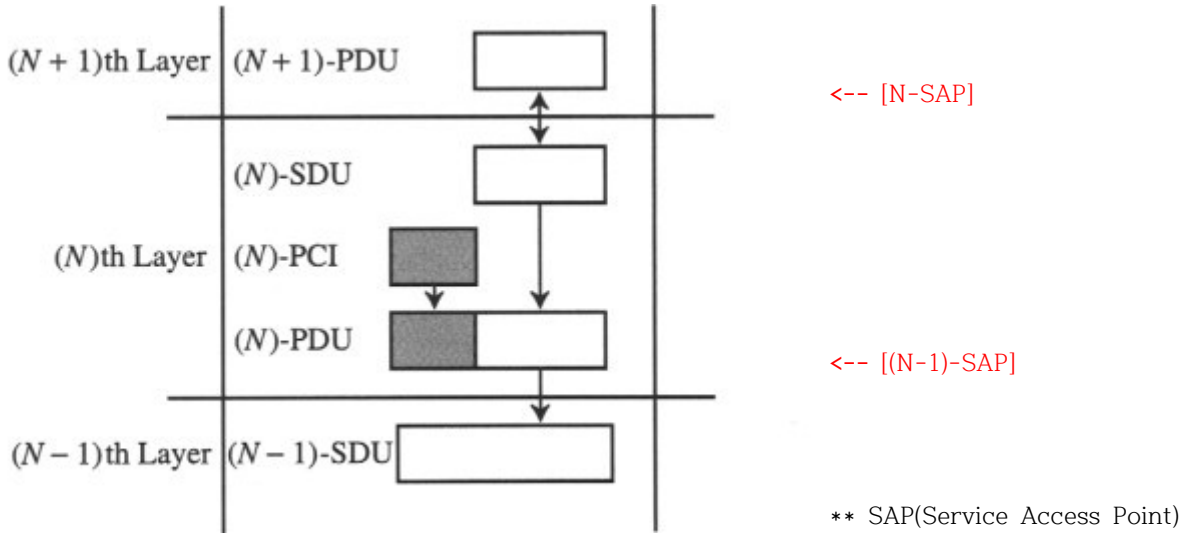
Destination Address (목적지 주소)	Source Address (전송지 주소)	Payload Data
----------Frame----------		

* Service Data Unit(SDU) & Protocol Data Unit(PDU)

: 동일한 개체 간에 데이터를 교환 하거나, 서비스 엑세스점(SAP)을 통해 인접계층의 개체 간에 데이터를 교환하기 위한 단위

** PCI(Protocol Control Information)

: 계층의 제어 장치 [Header(헤더)라고 해석해도 무방하다]



** SAP(Service Access Point)

: [간단히 말하면] 상하위 계층의 중간점에서 데이터 단위를 교환하기 위한 경로

순서 : N+1 level Layer | **N-SAP** | N level Layer | **(N-1)-SAP** | N-1 level Layer

*** 왜 하위계층-SAP인가?

: Service Access Point의 원래 정의는 "상위 계층이 **하위 계층에서 제공하는 서비스를 받기 위해** 하위 계층에 접근하는 영역"이다. SAP에서 서비스를 제공하는 것은 하위 계층이기 때문에 SAP을 표기할 때 하위계층-SAP으로 표기한다.

** 계산 방법 (그림 참조)

같은 계층에서, N-SDU + N-PCI = N-PDU

N계층 PDU가 N-1계층으로 가면서, N-PDU = (N-1)-SDU

N계층 SDU가 N+1계층으로 가면서, N-SDU = (N+1)-PDU = (N+1)-PCI + (N+1)-SDU

쉽게 생각해서, 올라갈 때는 SDU를 올려보내고 내려갈 때는 PDU를 내려보낸다.

올려보낸 SDU는 PDU가 되고, 내려보낸 PDU는 SDU가 된다.

그리고 N-계층에 존재하는 SDU, PDU, PCI의 앞은 모두 N-[XXX]이다.

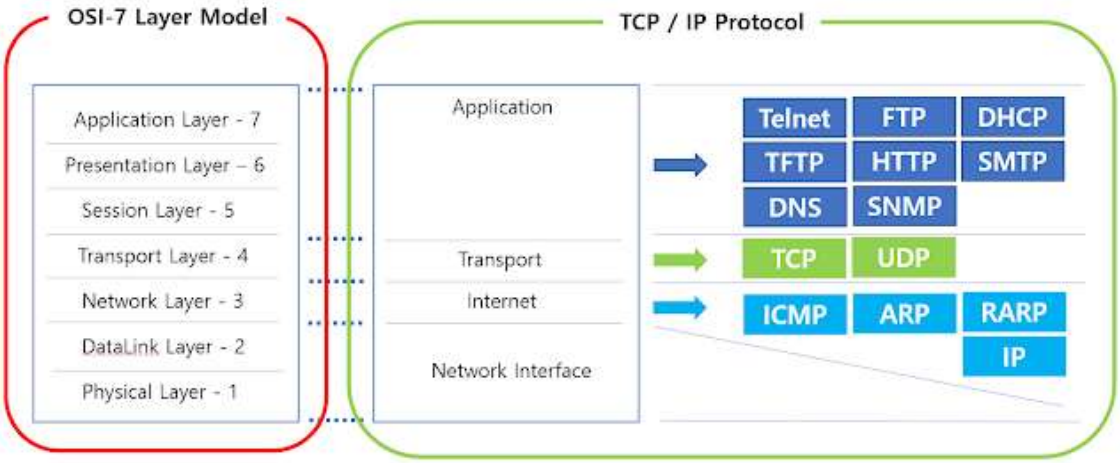
* DataLink Layer의 역할

: 연결된 Node(개체)에게 Frame을 "정확히" 전송 [오류 방지, 오류 복원, 충돌 방지]

에러 없는 데이터의 송수신을 가능하게 하는 것이 데이터 링크 계층의 역할이다.

데이터를 프레임화(Framing), 프레임의 전송 순서 제어

* TCP/IP Model



-> TCP/IP Model은 OSI 7 Layer Model을 좀 더 간단히 사용하는 것이다.
Network Layer : IP, Transport Layer : TCP, [5계층+6계층+7계층] -> Application
정도만 알고 있으면 될 것 같다.

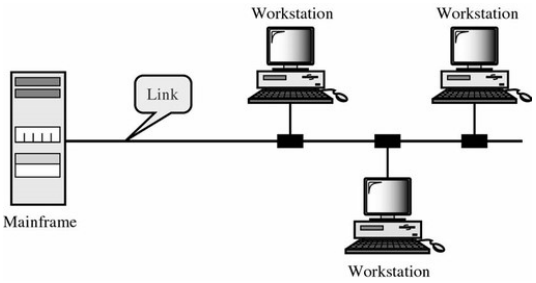
* MAC subLayer (Medium Access Control)
: DataLink Layer 내부의 subLayer (보조 계층)

DataLink Layer	LLC subLayer
	MAC subLayer

<- DataLink Layer 내부

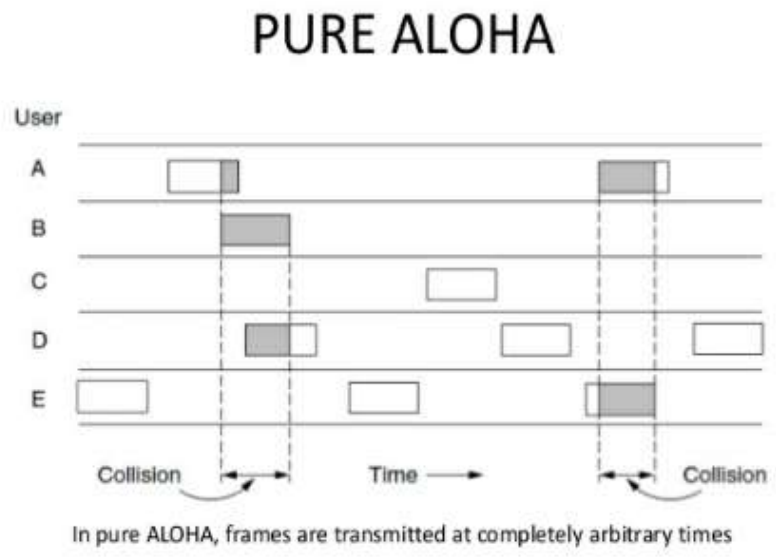
** MAC Layer의 역할
: Medium Access Control 말 그대로 중간에서 접근 조정의 역할을 한다.
(왜? Channel Allocation Problem 때문)
*** Channel Allocation Problem : 공유된 Media를 누가 사용할 것인지 정하는 문제
(공유된 미디어를 누가 사용할 것인지가 왜 문제? -> 모두 동시에 사용할 수 없기 때문)

※ P2P Network vs Shared Network
P2P(Point to Point) network : 모든 노드(객체)간에 각각의 전용선이 존재
-> 이 경우, Channel Allocation Problem이 발생하지 않는다.
-> 그림으로 보면 알겠지만, 컴퓨터의 개수가 많아지면 실제로 구현 거의 불가능
Shared Network : 많은 노드들이 하나의 라인(Media)를 공유
-> 대부분의 네트워크가 이 형태이기 때문에 Channel Allocation Problem이 발생하고,
MAC Layer의 역할이 요구된다.

P2P Network	Shared Network
 P2P-network	

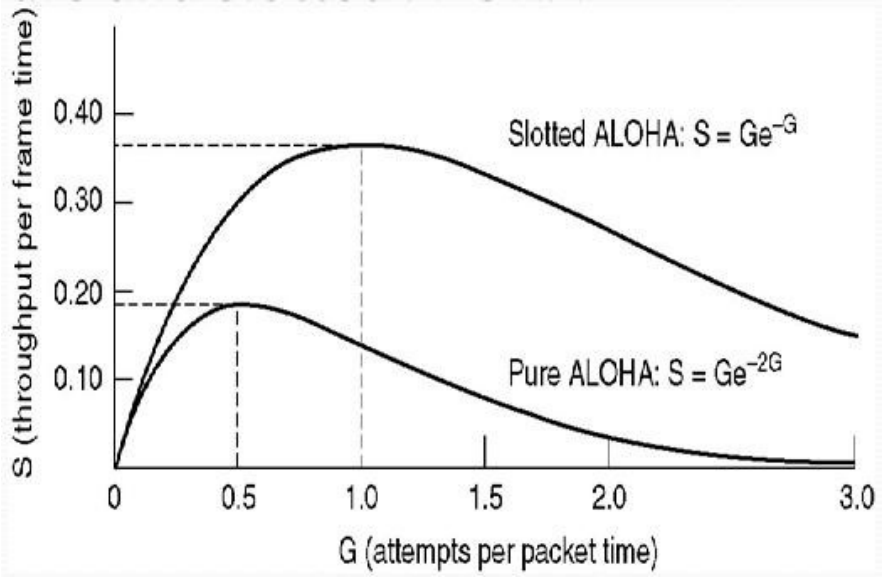
* ALOHA Protocol
이름 유래 : 하와이 4개 대학에서 통신을 주고받을 때 처음 사용했던 규약이기 때문
** Pure ALOHA
완전히 순수한 ALOHA 방식을 따르는 프로토콜이다.
자신이 데이터를 전송하고 싶을 때 보낸다. (Collision[3]이 발생하던말던)

Collision이 발생해서 데이터가 전송되지 않았다? -> 또 보낸다. (보내질 때까지)
-> 구축이 쉽고 단순한 방법이지만 무식한 방법
-> High arrival rate를 가지는 네트워크 서버에서는 좋지 않지만,
Low arrival rate를 가지는 네트워크 서버에서는 쓸만한 방법이다.



* Slotted ALOHA
: 정해진 시간에만 데이터(프레임)을 전송할 수 있도록 미리 정해놓음.
Ex) 모두 각 시간 정각에만 데이터 전송을 시작하도록 한다.
-> 동시에 데이터를 보내기 시작하면 Collision은 여전히 발생하지만,
일단 데이터를 보내기 시작하면, 이후 발생하는 Collision은 방지할 수 있다.
-> Collision의 발생 확률이 반으로 줄어든다.

* Pure ALOHA vs Slotted ALOHA [데이터의 도착률]



x축 : 얼마나 많은 노드들이 데이터를 전송하고 싶어하는지
y축 : 보낸 데이터가 정상적으로 도착할 확률(비율)
-> Pure ALOHA에서는 최대 18%의 도착률을 보이고, Slotted ALOHA에서는 최대 36%
-> 내가 100MB/sec 속도의 랜을 사용하는데, Pure ALOHA를 쓴다? -> 최대 18MB/sec

3) 2개 이상의 노드에서 하나의 media를 동시에 사용하는 현상 (충돌)

* ALOHA vs CSMA 데이터 도착 비율 비교

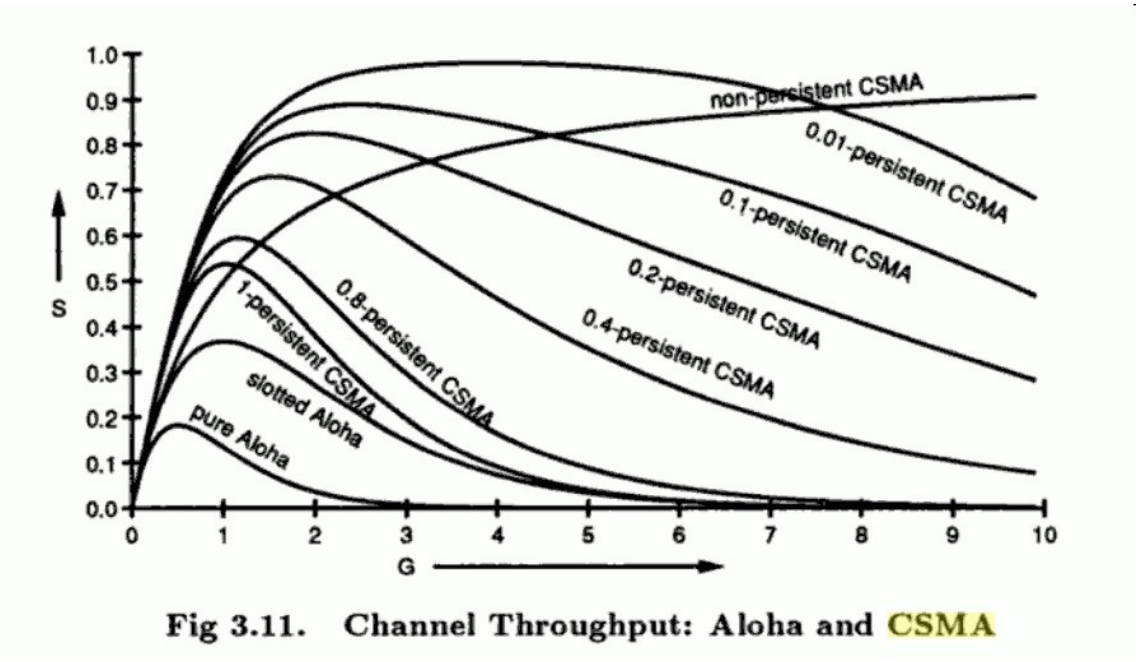


Fig 3.11. Channel Throughput: Aloha and CSMA

-> 우리가 현재 사용하는 프로토콜은 1-persistent CSMA Protocol이다.

-> 교수님은 Non-persistent CSMA를 1-persistent CSMA와 동일한 방식이라는 식으로 설명하셨지만, 실제로 non-persistent CSMA는 CS에서 다른 노드의 사용이 발견되면 임의의 시간(Random)이 지나고 다시 CS를 실행한다. 1-persistent CSMA는 CS에서 다른 노드의 사용이 확인되면 바로 다시 검사하므로 전혀 다른 방식의 protocol이다.

* CSMA(/CD) [Carrier Sense & Multiple Access(/Collision Detect)]

Multiple Access : 다중 접속으로, 보통 네트워크를 다수의 노드(컴퓨터)들이 사용함을 의미

** Carrier Sense?

: 프레임을 전송하기 전에 현재 데이터 전송망을 사용하고 있는 다른 노드가 없는지 확인하는 것 -> How : 다른 노드에서 전송망을 사용중이면 sin-Wave가 감지된다.

** Collision Detect?

: 충돌 감지를 의미하는 것으로, Carrier Sense를 하고 전송망을 사용중인 노드가 없어서 데이터를 전송했는데, 충돌이 일어날 수 있다.[4] 그때, 충돌이 일어났음을 감지하고 충돌이 일어나서 데이터가 전송되지 못했음을 알려주는 것

** CSMA의 논리적인 흐름 ★

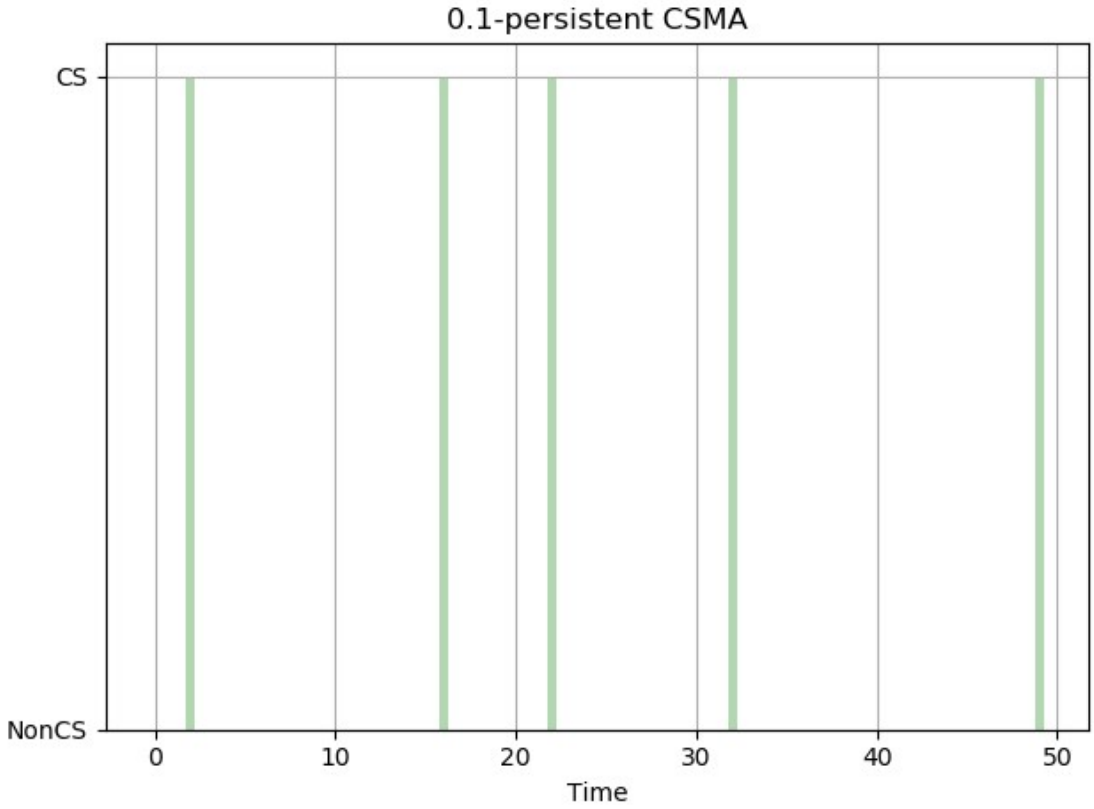
: A가 데이터를 전송망을 통해 C에게 보내려고 한다. A는 데이터를 보내기 전에 CS를 실시하여 전송망을 사용중인 다른 노드가 있는지 검사한다. 사용중인 다른 노드가 없다면 데이터를 전송한다. 사용중인 다른 노드가 있으면 **"어느정도의 딜레이"**를 두고 다시 CS를 한다.

** p-persistent CSMA (**"어느정도의 딜레이"**에 관한 내용)

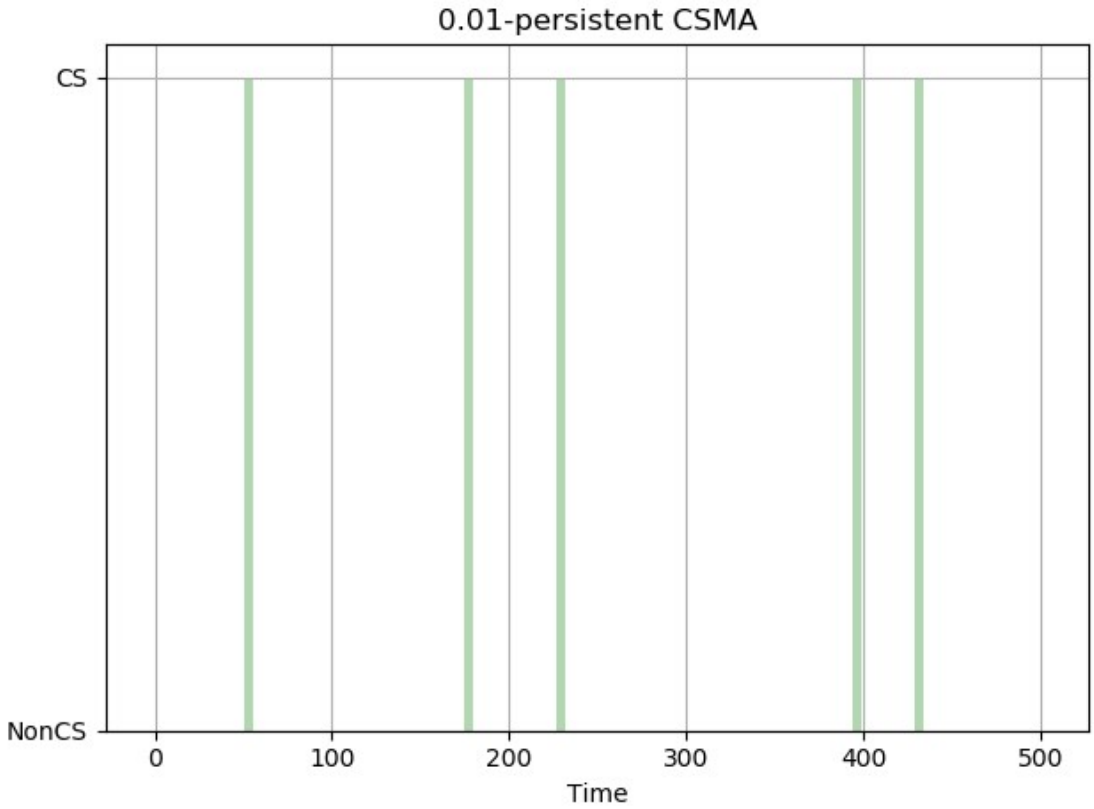
: Carrier Sense를 시행했는데 다른 노드가 전송망을 사용중일 때, 바로 CS를 다시 시행하지 않고, 다시 CS를 시행할 비율을 p로 두는 CSMA이다.

4) 실제로 다른 노드가 전송망을 사용중이지만 sin-Wave를 감지하지 못한 것은 데이터가 이동하는데에도 아주 짧지만 시간이 소요되기 때문이다. (이후에 다시 다룰 내용)

-> 0.1-persistent CSMA : CS를 10회 할 수 있는시간에 1번 시행한다.



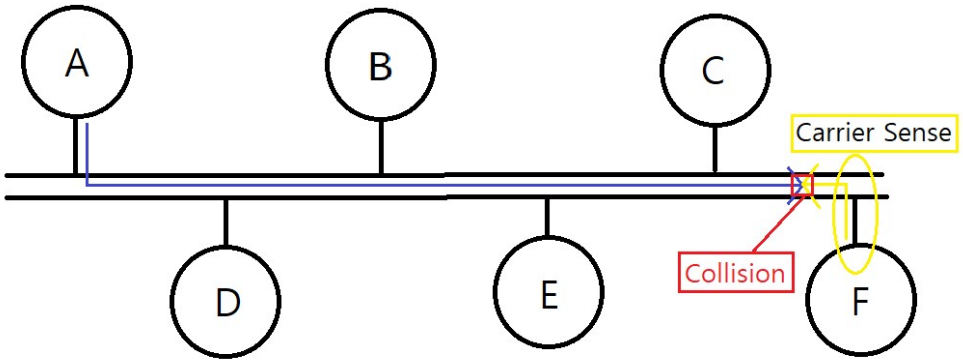
-> 0.01-persistent CSMA : CS를 100번 할 수 있는 시간에 1번 시행한다.



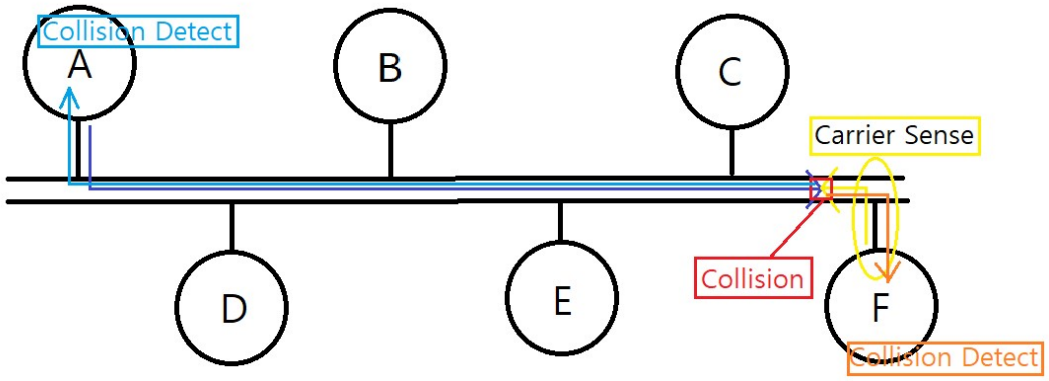
** p-persistent CSMA에서 p는 어떻게 결정?
-> 해당 전송망(네트워크)의 traffic[5]에 의해서 결정한다.

5) traffic : 전송망에서 얼마나 많은 노드들이 데이터를 전송하려고 하는지

** CD (Collision Detection)
: 위와 같이 CSMA방식을 사용하게 되면 이론적으로 Collision이 발생하지 않아야 한다.
그러나, 이전 각주 4)에서 설명했듯, 데이터의 이동에 시간이 걸린다는 것에 의해 실제로는 Collision이 발생하게 된다.



-> **A에서 보낸 데이터가 F의 CS범위가 미쳐 다다르지 못했을 때 F가 CS를 하고 사용중인 다른 노드가 없다고 판단하여 Data를 내보내면서 Collision이 발생한다. ★**

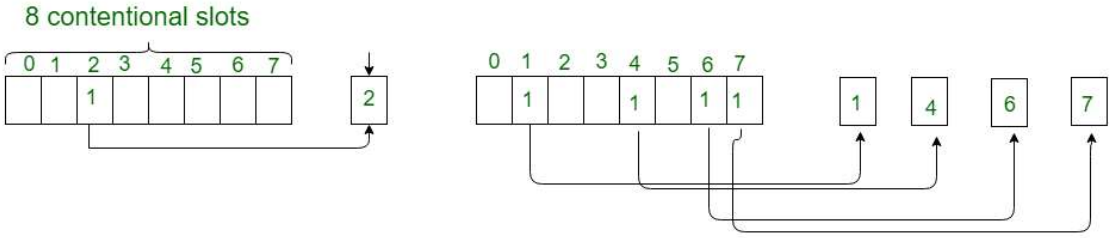


-> Collision이 발생하면, 각각의 전송지에 충돌이 일어났음을 보고한다.
※ 여기서 한 가지 문제가 발생하는데, A가 전송한 데이터(프레임)의 길이가 너무 짧으면 A는 프레임을 다 내보냈는데 이후에 Collision이 발생하는 경우가 생긴다. 이런 경우, A는 프레임을 모두 내보냈기 때문에 도중에 Collision이 발생했는지 정상전송 되었는지를 알 수가 없다.
-> 이러한 문제를 해결하기 위해 제정한 프로토콜
1. **랜선의 길이에 Maximum**을 둔다. [Physical Layer Protocol]
2. **케이블의 두께에 minimum**을 둔다. [Physical Layer Protocol]
3. **프레임(데이터)의 길이에 minimum**을 둔다. [DataLink Layer Protocol]
Ex) "A" 전송 -> "A00000000000000000000000000000"으로 프레임화 -> 전송

** Collision 보고를 받은 노드는 어떻게 행동하는가?
-> Collision 보고를 받음 (데이터를 다시 전송해야하는 상황) -> **delay & retry**
어느 정도의 delay를 가지고 기다렸다가 다시 전송한다.
delay : Binary Exponential back off를 따른다. [$Delay = 2^n \times \tau$ (τ : $Random\ time$)]
-> τ가 random time이기 때문에 언젠가는 데이터가 전송된다.

=> 이것으로 CSMA/CD Protocol에 대해서 모두 알아보았다. 이러한 방식을 사용한다면 큰 문제없이 네트워크 통신을 할 수 있다. 그러나 **ALOHA, CSMA/CD는 데이터를 보내고 그 데이터가 언제 상대방에게 도달할 것인지 예상할 수 없다는 단점이 있다. 이러한 단점 때문에 데이터의 확실한 도착시간이 보장되어야하는 네트워크(Real-Time-Network)에서는 사용할 수 없다.** 그렇다면 Real-Time-Network에서 사용하는 protocol은 무엇인가. 다음 페이지에 서술되어 있다.

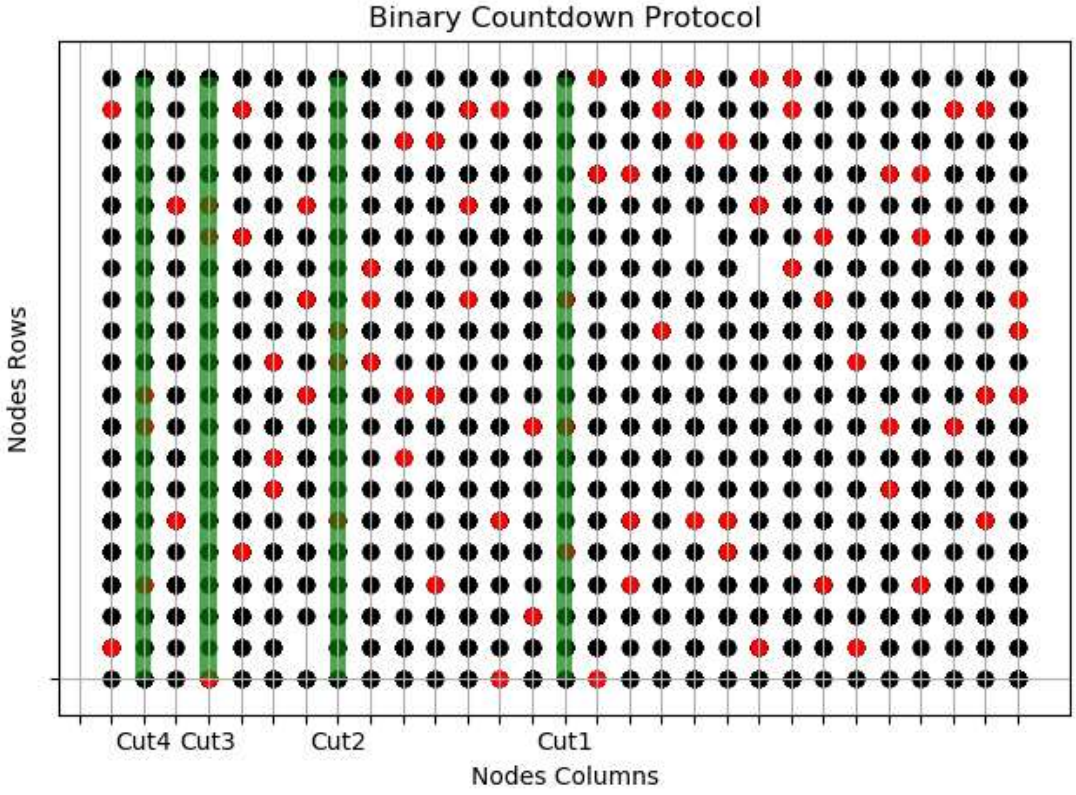
* Collision-Free Protocol
: 문자 그대로 충동에 자유로운 프로토콜이다.
이 프로토콜을 따르는 통신에는 Collision이 발생하지 않는다. (데이터 도착시간 예상 가능)
** Basic bit-map protocol



A Bit-map Protocol.

-> 데이터를 전송하기 전에 데이터를 전송하고 싶은 노드를 먼저 뽑아낸다. 그리고 각각의 노드들에게 우선순위를 부여해 데이터를 순차적으로 전송하도록 한다.

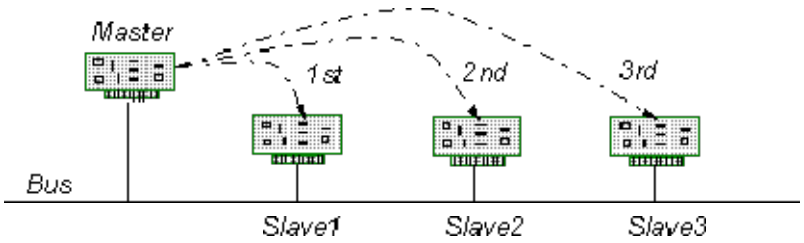
** Binary Countdown protocol



-> 노드들을 반으로 갈라서 좌측, 우측에 데이터를 보낼 노드가 있는지 질문한다. 만약, 좌측에 우선순위를 둔다면 좌측에서 또 반으로 갈라 데이터를 보낼 노드가 있는지 질문한다.
이러한 과정을 반복해 데이터를 보낼 노드의 우선순위를 정하고, 순차적으로 데이터를 전송하도록 한다. : bit-map protocol에서 N개의 노드에 N번의 질문을 해야하지만, Binary countdown protocol에서는 N개의 노드에 $\log_2 N$번의 질문을 하면 된다. (효율적이다)
-> 그림에서는 600개의 노드에 마치 4번의 질문만 하는 것으로 나타나 있지만, 사실 저기서 가장 왼쪽줄에 있는 20개를 대상으로 또 반으로 갈라 질문하고, 또 가르고, 또 가르고 해야한다. 그림 예제에서 필요한 질의 횟수 : 10회 ($\because \log_2 600 \approx 9$)

---------- [Not important] ----------

** M/S polling protocol (Master/Slaves polling protocol)

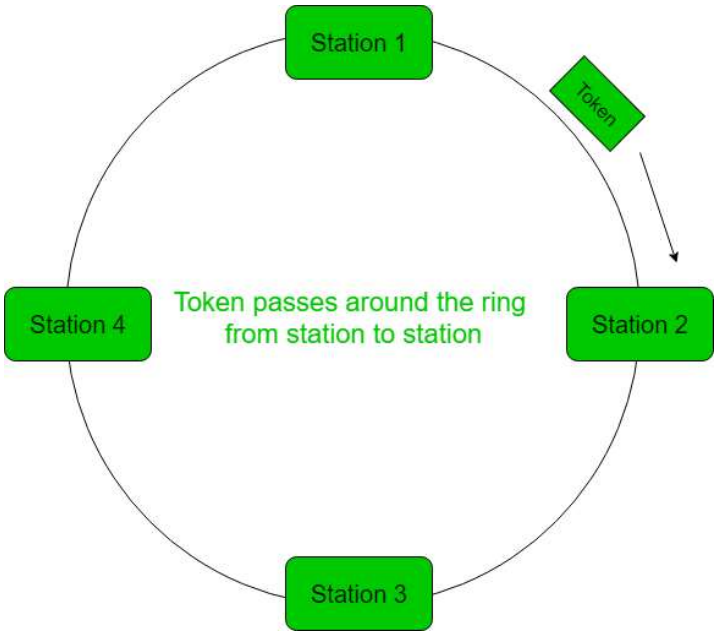


: Master가 Slaves에게 전송할 데이터가 있는지 순차적으로 질의하여 보낼 데이터가 있으면 보내고, 없으면 다음 slave에게 질의하는 방식

-> 장거리 네트워크 X

-> M/S polling protocol을 사용하는 대표적인 네트워크 : KTX 내부 네트워크

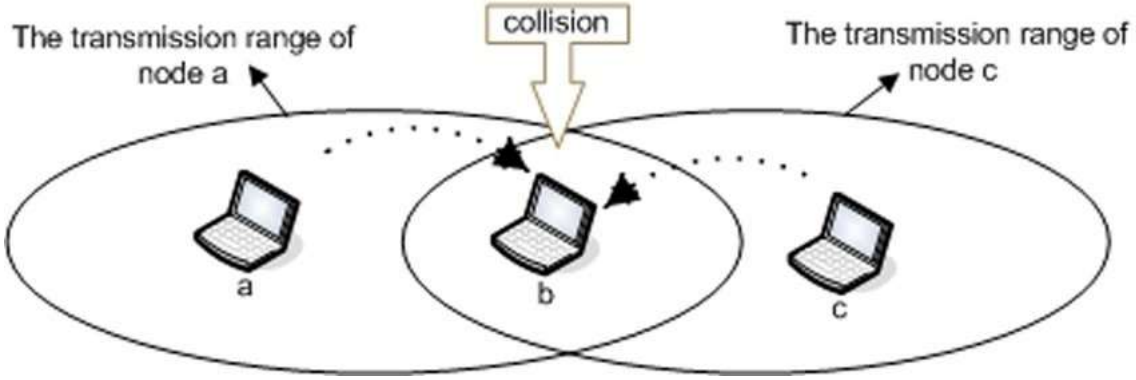
** Token ring protocol



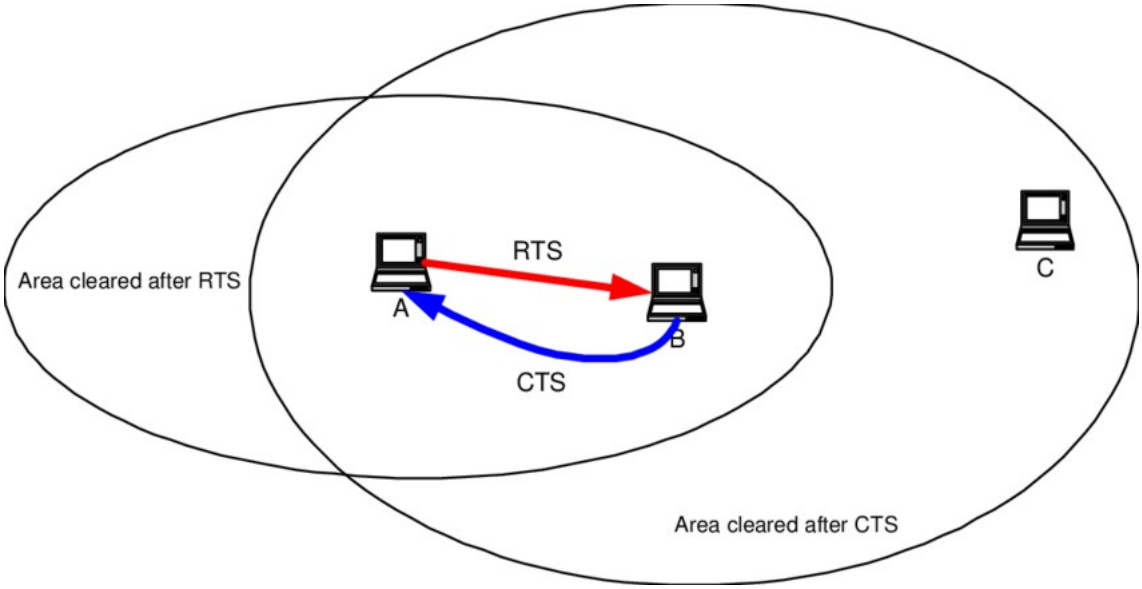
: Token(데이터를 전송할 수 있는 권리)를 돌려가며 자신의 차례에 데이터를 전송하는 방식

* Wireless LAN Protocol (무선 통신 프로토콜)
 : 무선 통신은 선으로 연결되어 있지 않기 때문에 통신 거리에 제약이 있다.
  (Wifi 제약 거리 : 법적으로 300m, 실제로 100m도 되지 않음)

** Hidden Node Problem ★
 무선 통신에 제약 거리가 있다보니 여기에서 문제가 발생한다. 그림에서 보듯, a와 c는 서로가 서로의 존재를 인지하지 못한다. 그래서 b에게 데이터를 전송하려고 Carrier Sense를 하면 전송이 가능한 상태로 인지한다. a와 c가 동시에 b에게 데이터를 전송하면 b에서는 collision이 발생한다. 이것을 "Hidden Node Problem(숨겨진 노드 문제)"라고 한다.



그럼, 이 문제를 어떻게 해결할 수 있을까? -> **RTS & CTS ★**



A가 B에게 데이터를 전송하기 전에 먼저 RTS(Request to Send)를 보낸다. RTS를 받은 B는 자신이 데이터를 수신할 수 있는 상태라면 CTS(Clear to Send)를 다시 보낸다. 이때, C에서 B의 CTS를 감지하고 지금 자신이 인지하지 못하는 노드(A)가 B에게 데이터를 전송하려고 하는구나 인지하고 데이터를 보내지 않는다. CTS를 돌려받은 A는 B에게 전송을 시작한다.
-> Collision을 방지할 수 있다.
=> 이러한 규약을 **MACA protocol(Multiple Access & Collision Avoidance)**[6]이라고 한다.

6) IEEE 802.11 : CSMA/CA(Carrier Sense Multiple Access with Collision Avoidance)라고 한다.

* Ethernet MAC subLayer

Protocol Frame의 구조 [DIX Ethernet] ★

8bytes	6bytes	6bytes	2bytes	0-1500bytes	0-46bytes	4bytes
Preamble	Destination Address	Source Address	Type	Data	Pad	Check-sum

Preamble : Carrier Sense를 위한 신호 + sender/receiver 시스템 시간 동기화

Destination Address : 도착지 주소

[최종 도착지 주소는 아니다. 최종 도착지 주소는 Network Layer에서 판별하는 것이고, 여기에서 쓰인 도착지 주소는 Neighborhood Node MAC Address[7]이다.]

Source Address : 발신지 주소 [이것 또한 Neighborhood MAC address이다.]

Data(Payload) : 이 프레임에서 전송하려고 하는 정보이다. MTU[8]는 1500bytes이다.

Pad : Collision Detect를 위한 최소길이를 유지시키는 역할을 하는 영역

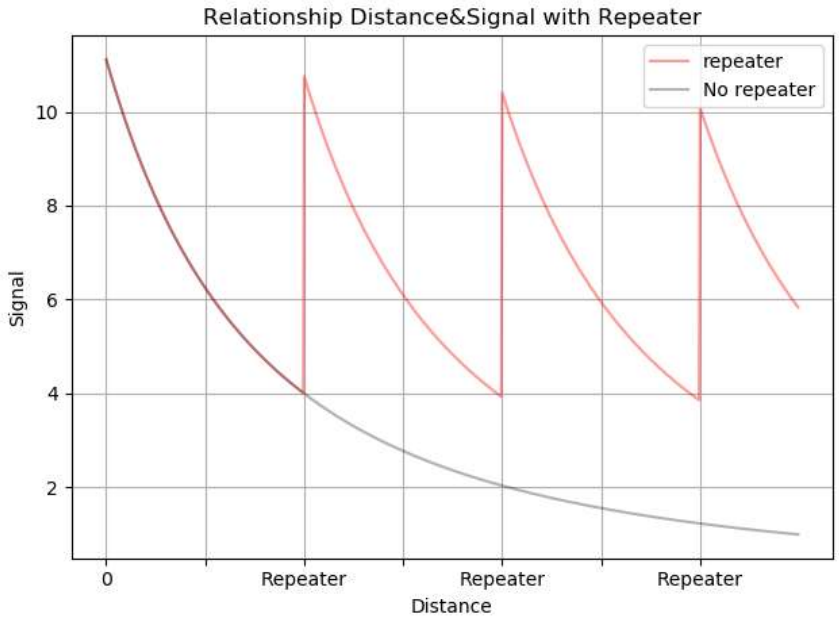
Check-sum : Error를 확인하는 영역 [어디서 에러가 발생했는지는 모름] (CRC[9] 사용)

=> 위에 보이는 그림(표)가 프레임의 구조이다. 이렇게 3계층에서 Data를 받아서 열거된 Header를 덧붙이는 것을 Framing(프레임화)라고 하며, 이것은 2계층에서 이루어진다.

* Repeater(리피터) ★

: 거리의 증가로 신호의 감쇄가 일어나면, 이 신호를 다시 증폭시켜주는 장치

이러한 역할을 하는 장치를 DataLink Layer에서는 Repeater, Network Layer에서는 Router(라우터)라고 한다.



7) MAC Address : 네트워크 기능을 가진 하드웨어에 고유하게 부여되는 주소이다. 6bytes는 48bits이므로 고유한 MAC Address의 개수는 256조 이다.

8) MTU : Maximum Transfer Unit [한 프레임이 가질 수 있는 최대 크기]

9) CRC : Cyclic Redundancy Check (순환 중복 검사), 데이터를 보낼 때 Check-sum과 받은 데이터의 Check-sum이 일치하는지 비교하는 방법 [서로 일치하지 않으면 중간에 오류가 발생했다고 인지]

- LLC subLayer (Logical Link Control subLayer)

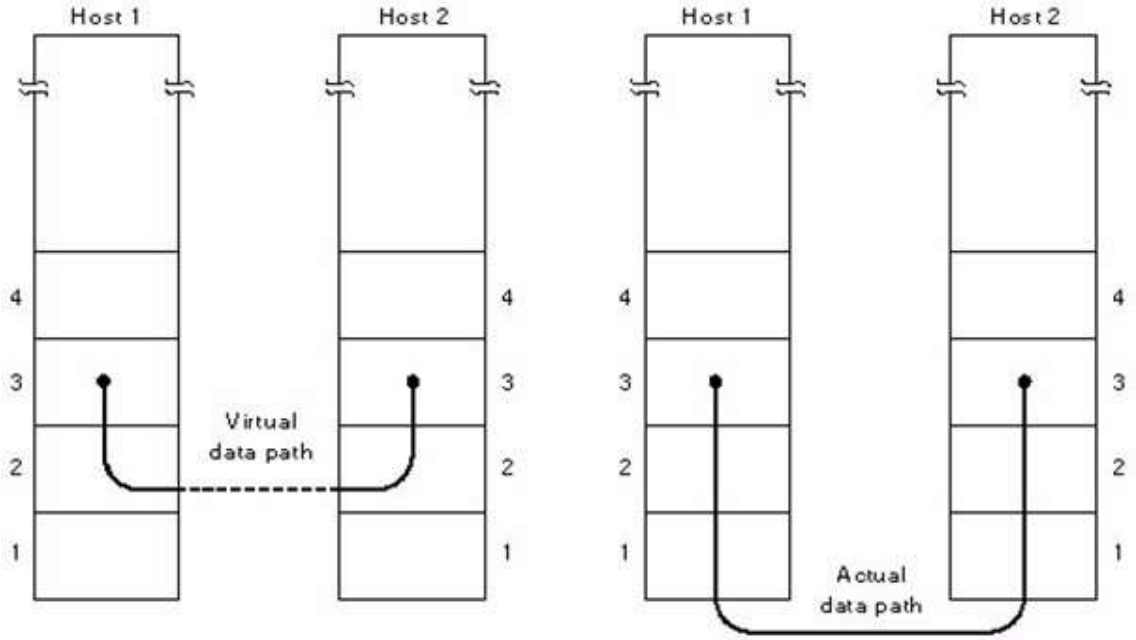
: DataLink Layer에는 2개의 subLayer[MAC, LLC]가 있는데, 그 중 하나이다.

MAC subLayer에서는 Medium Access Control, 말 그대로 공유된 전송망을 누가 사용할 것인지 조정하는 역할을 했다. (동시에 다수의 노드가 전송망을 사용할 수 없으므로) LLC subLayer에서는 Framing, Error Control, Flow Control 등의 역할을 한다.

이 중 Error Control은 MAC Layer에서도 이루어진다.

* Services Provided to Network Layer

: 이것 또한 LLC Layer의 역할인데, 해석해보면 "Network Layer에게 서비스를 제공하는 것"이다. 3계층에서 내려온 데이터는 실제로 2, 1계층으로 내려가서 1계층에서 데이터를 송수신하고 수신받은 데이터를 올려 3계층에 도달하는데, 이것을 3계층에서 마치 다른 호스트의 3계층과 바로 통신하는 것처럼 느끼게 해주는 서비스이다. [Virtual Communication]



* Framing (프레임화)

: 윗 계층에서 내려온 패킷을 MTU(Maximum Transfer Unit)에 맞도록 자르고, 헤더와 트레일러를 붙여 2계층에서 해석할 수 있는 프레임으로 만드는 과정

* Frame을 만드는 기법

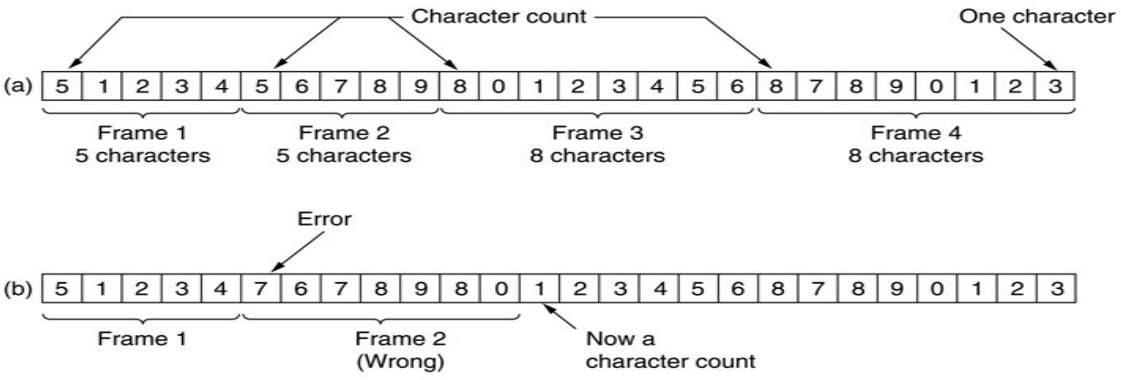
-- Character count Framing

-- Starting and Ending flags, with byte stuffing Framing

-- Starting and Ending flags, with bit stuffing Framing

1. Character Count Framing

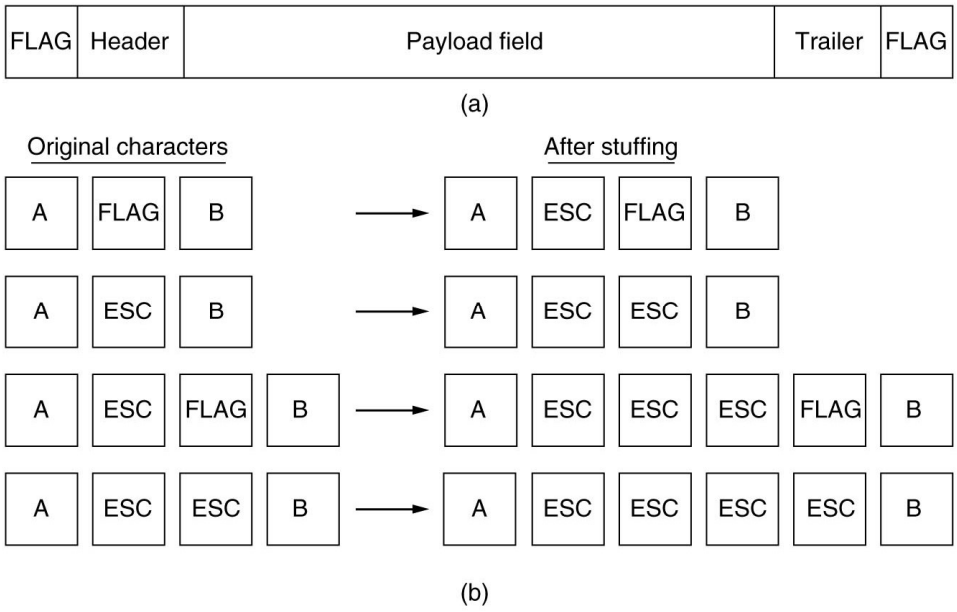
: 프레임의 시작을 그 프레임의 길이로 지정하여 해당 프레임의 길이를 알려주는 기법



(a)와 같은 방식으로 프레임의 길이를 알려주는 기법인데, (b)처럼 5가 와야 할 자리에 오류가 나서 7이 들어가면 그 뒤로 오는 데이터들은 전부 의미없는 데이터가 되고 만다. [단점]

이러한 오류에 대한 취약성 때문에 사실상 거의 쓰이지 않는 기법이다.

2. Starting and Ending flags, with byte stuffing Framing
: 프레임의 시작과 끝에 FLAG를 넣어서 프레임 단위를 구분할 수 있게 하는 기법



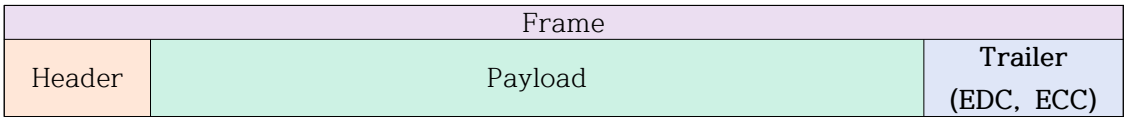
그림에서 (a)는 프레임을 FLAG로 둘러싸는 이 기법에 대한 설명이다. (b)는 만약 전송하려는 데이터 내부에 FLAG 혹은 Escape byte가 들어가면 어떻게 처리하는지에 대한 설명이다. FLAG를 "!"로 지정했다고 가정했을 때 프레임의 내용이 "Hello! Everyone~"이라면, Flag framing을 할 때 이것이 "!Hello! Everyone~!"으로 변환될 것이다. 수신측에서는 이 프레임의 시작과 끝을 "!"로 구분하므로 실제로 전달되는 데이터는 "Hello"뿐이고 나머지("! Everyone~")는 손실되게 된다. 이러한 예외를 해결하기 위해 등장한 것이 Escape byte이다. 데이터 내부에 들어있는 FLAG앞에는 Escape byte를 붙여 전송하는 것이다. 예를 들어 Escape byte를 "\"라고 가정하면 "Hello! Everyone~"은 "!Hello\! Everyone~!"으로 변환되고, 이것을 수신측에서는 "Hello! Everyone~"으로 전부 받을 수 있다.
그렇다면 보낼 데이터가 "\ is Backslash"라면 어떻게 해야할까? 이럴 경우, 탈출문자(Escape byte)앞에 또 탈출문자를 붙여주는 것으로 해결한다.
즉, 이 데이터는 "!\\ is Backslash!"로 변환되고, 수신측에서 이것을 오류없이 받을 수 있다.
이렇게 Escape byte를 붙여주는 것을 stuffing(채워넣기)이라고 한다.
물론 이 기법에서도 오류는 발생할 수 있는데, flag가 오류로 인해 다른 문자로 변형되는 것이다. "Hello"는 수신측에서 "!Hello!"로 수신되어야 하는데, 오류로 인해 "@Hello!"로 수신되면 이 프레임은 쓸 수 없어진다. 그러나 Character Count 기법에서 오류 이후 모든 데이터를 손실하는 것에 비해 이 기법에서는 다음 프레임을 정상적으로 사용할 수 있다는 점에서 오류에 조금 덜 취약하다고 할 수 있다.

3. Starting and Ending flags, with bit stuffing Framing
: 2번 기법에서 바뀐 것은 byte->bit이다. bit전송이 byte전송보다 빠르기 때문에 사용한다.
기본적으로 작동하는 메커니즘은 2번 기법과 동일하다.
Ex) FLAG = 01110, Escape bit = 011 이후에 0

Original data	**01110**10101011**011**010**101110**
Stuffed data	**01110**10101011**0**1**1**0010**101110**

FLAG가 01110이기 때문에 Payload에 01110이 나오는 것을 방지해 주어야 한다. 여기에서는 011이 앞에 나오면 바로 다음 escape bit 0을 넣어주는 것으로 01110이 나오는 것을 방지했다. Escape bit를 결정하는 방법은 FLAG를 어떤 것을 사용하는가에 따라 당연히 항상 다르다.

- EDC, ECC [에러 발견 코드, 에러 수정 코드]



: Error Detecting Codes(EDC), Error Correcting Codes(ECC)는 Frame 내부에 Trailer부분에 들어간다.

* Error-Detecting Codes [Checksum]
Hamming Distance를 이용한 에러 검출 기법
k개의 에러를 발견하고 싶으면 Hamming Distance를 k+1이 되도록 만들어줌.
Ex) 2bit 데이터 전송
[00, 01, 10, 11] -> 2bit로 만들 수 있는 데이터들
[00**0**, 01**1**, 10**1**, 11**0**] -> 뒤에 1bit를 더 붙여서 모든 데이터들 사이의 H.D.를 2로 만듬.
여기에서 덧붙여진 0이나 1을 parity bit[10]라고 한다.
모든 데이터들 사이의 H.D.가 2이면 3개의 bit중 1개가 오류가 나면 오류 발생 확인 가능
왜? 000이 오류로 인해 001이 된다. -> 001은 위의 4가지 경우에 없다. -> 오류 발생 확인

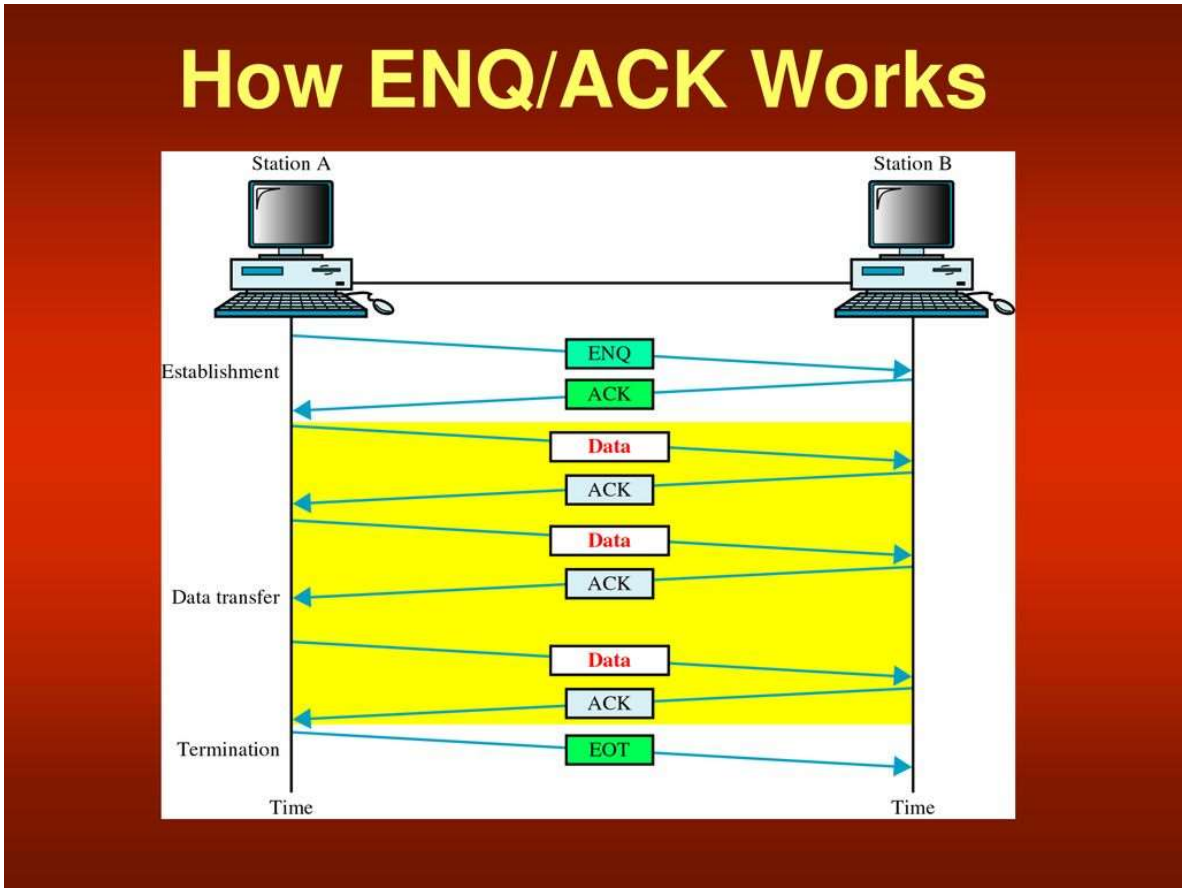
** 단점 : 1byte에 1bit의 parity bit를 넣어야 하는데, byte가 커지면 커질수록 오류확인용 bit 또한 커지기 때문에, 용량적으로 비효율적이다.

** Polynomial code checksum
하나의 프레임 전체를 특정 Generator로 나누어서 나머지 R을 EDC자리에 붙여 전송한다.
수신측에서는 덧붙여져 온 나머지 R을 같은 Generator로 나누어서 나머지가 0이 되면 정상, 나머지가 0이 아니면 오류가 있다고 판단한다. [99.9% 신뢰도][11]

* Error-Correcting Codes
: 에러를 수정하는 코드이다.
방법은 k bit의 데이터를 전송할 때 Hamming Distance를 2k+1로 만들어주는 것이다.
세부적인 방법 -> Googling
데이터 전송에 매우 오랜 시간이 걸리는 데이터, 정확성이 확실히 보장되어야 하는 데이터를 전송할 때만 이 방법을 사용한다. (용량적으로 매우 불리하기 때문)

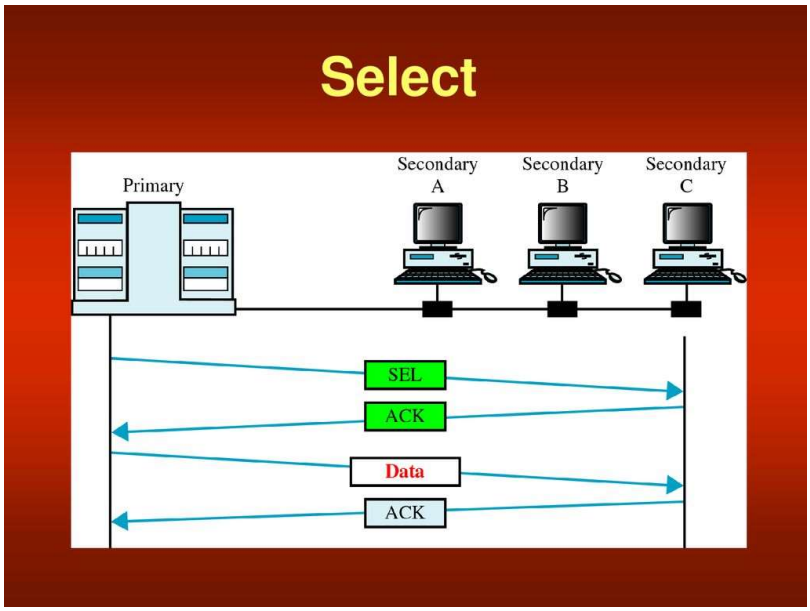
10) parity bit : 덧붙여져 Hamming Distance를 2로 만들어주는 bit
11) 이 방법으로 오류를 100% 확인하지는 못한다는 것을 의미한다. 그 대신에 한 프레임에 나머지 R 하나만 붙여서 전송하면 되므로 용량적으로 매우 유리하다.

- Flow Control [회선 제어]
 : 쪼개진 패킷(프레임)들의 순차적인 전송
* Stop-and-Wait Protocol
** ENQ/ACK 기법 [Enquiry / Acknowledgement]



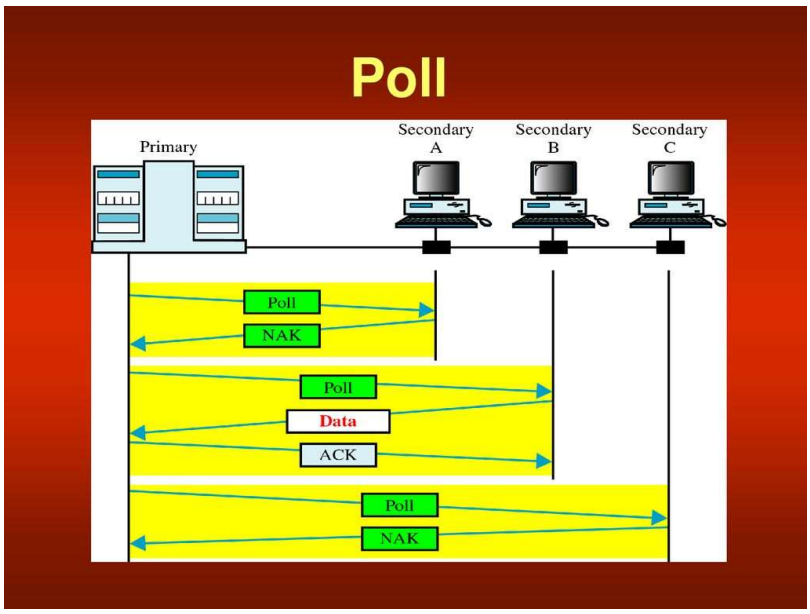
송신측에서 수신측에 ENQ(조회)를 보내고, 수신측에서 ACK(응답)을 보냄으로써 데이터의 송수신이 시작된다. 송신측은 프레임 하나 보내고 응답받고, 보내고 응답받고를 반복한다. 데이터를 전부 송신했으면 EOT(End of Text)를 수신측에 보내는 것으로 마무리한다.

** 폴링 기법
 : 하나의 주 스테이션에 여러 개의 종속 스테이션으로 구성된 네트워크에서 사용
1. 선택 모드



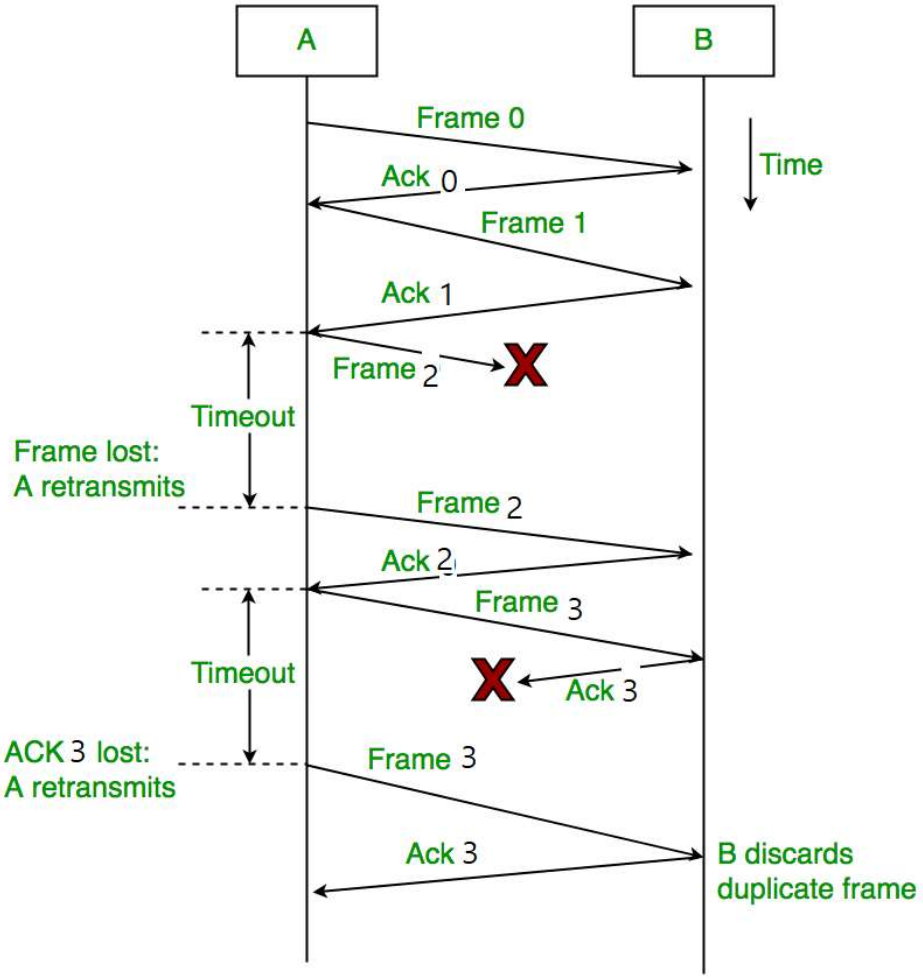
-> 주 스테이션이 종속 스테이션에게 데이터를 송신할 때 사용하는 모드로, 주 스테이션은 ENQ 대신 SEL을 해당 스테이션에게 전송하고, ACK가 오면, 데이터 송수신을 시작한다.

2. 폴 모드



-> 주 스테이션이 다수의 종속 스테이션들에게 데이터 전송 의사를 물어보는 모드이다.
주 스테이션이 순차적으로 Poll을 보내면, Poll을 받은 종속 스테이션은 전송할 데이터가 있는 스테이션은 Data를 주 스테이션으로 전송[12], 없는 스테이션은 NAK(Negative-Acknow..)를 8전송한다.

* Stop-and-Wait ARQ [Autometic Repeat & reQuest]
: 위에서 설명한 ENQ/ACK 기법, 폴링 기법이 모두 여기에 속한다.



위 그림이 Stop-Wait ARQ를 전부 설명한다. 송신측에서 프레임을 전송할 때 각 프레임에 TAG(태그)를 붙여 전송한다. 수신측은

12) 교재와 다른 부분이다. 교재에서는 보낼 데이터가 있으면 ACK를 전송하는 것으로 되어있다. 만약 시험에 나오면 ACK를 전송한다고 하는 것이 낫다.

받은 프레임에 대한 TAG를 붙여 ACK를 전송한다. 이러다가 수신측에서 받은 프레임에 오류가 있다고 판단하면 NAK를 보내서 해당 프레임을 다시 전송하도록 한다. 또한 송신중에 프레임이 손실되면 응답이 오지 않을테니 일정시간(Timeout)이 지나면 송신측이 다시 프레임을 전송한다. 수신확인신호 ACK가 분실되는 경우도 있는데, 이 경우 또한 일정시간이 지나면 다시 프레임을 전송한다. 수신측은 같은 프레임을 2번 받은 것이 되니 같은 TAG의 프레임은 나중에 받은 것을 삭제한다. 이러한 알고리즘을 가지고 송수신하는 것이 Stop-Wait ARQ protocol이다.

단점 : 0.001% 비율로 발생하는 오류를 검출하기 위해 모든 데이터를 보낼때마다 응답을 기다리는 것은 시간적으로 매우 비효율적이다.

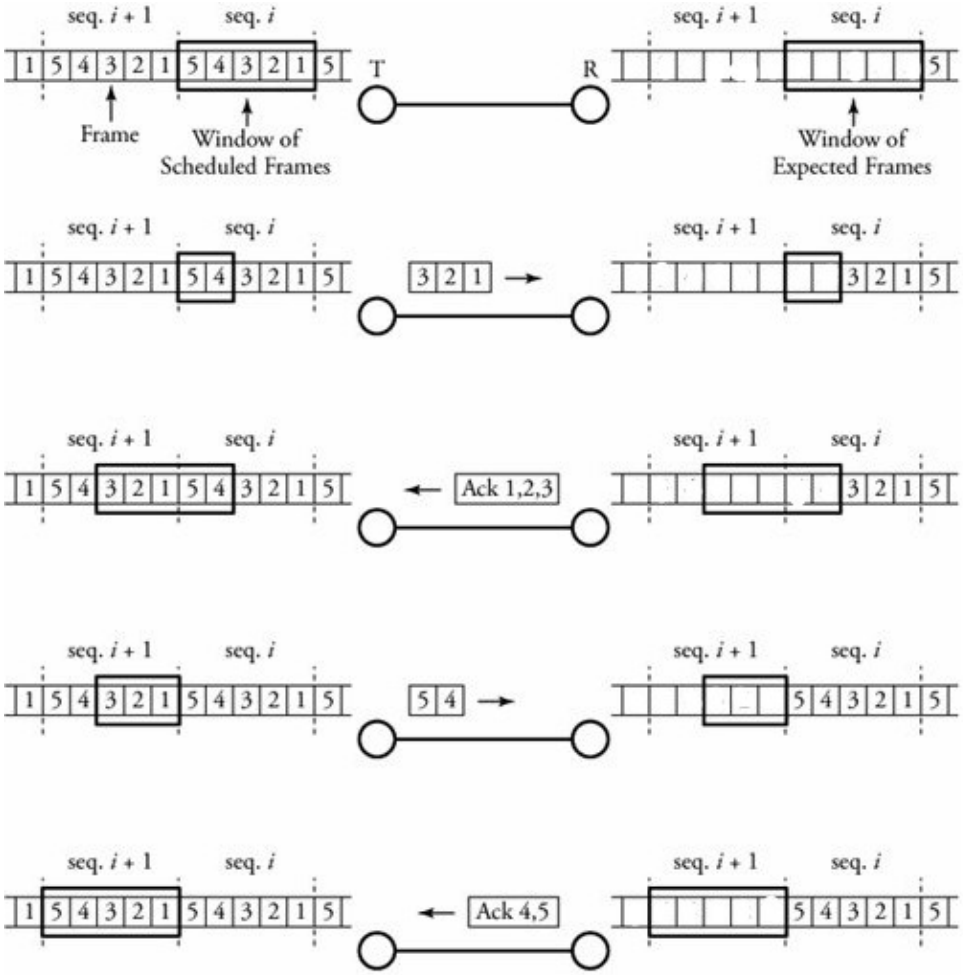
-> 단점을 보완하기 위해 나온 기법 : Sliding Window Flow Control

* Sliding Window Flow Control ★★

: 일정한 크기의 Window Size를 정해 그 크기만큼의 프레임은 ACK가 돌아오지 않더라도 일단 전부 전송하는 기법

이 기법에서는 자동으로 window가 밀리듯 이동하며 오류가 없는 경우 대기시간을 가지지 않고 연속적으로 데이터를 전송할 수 있다. [아래 예시 참조]

if Window size : 5, T : 송신측, R : 수신측

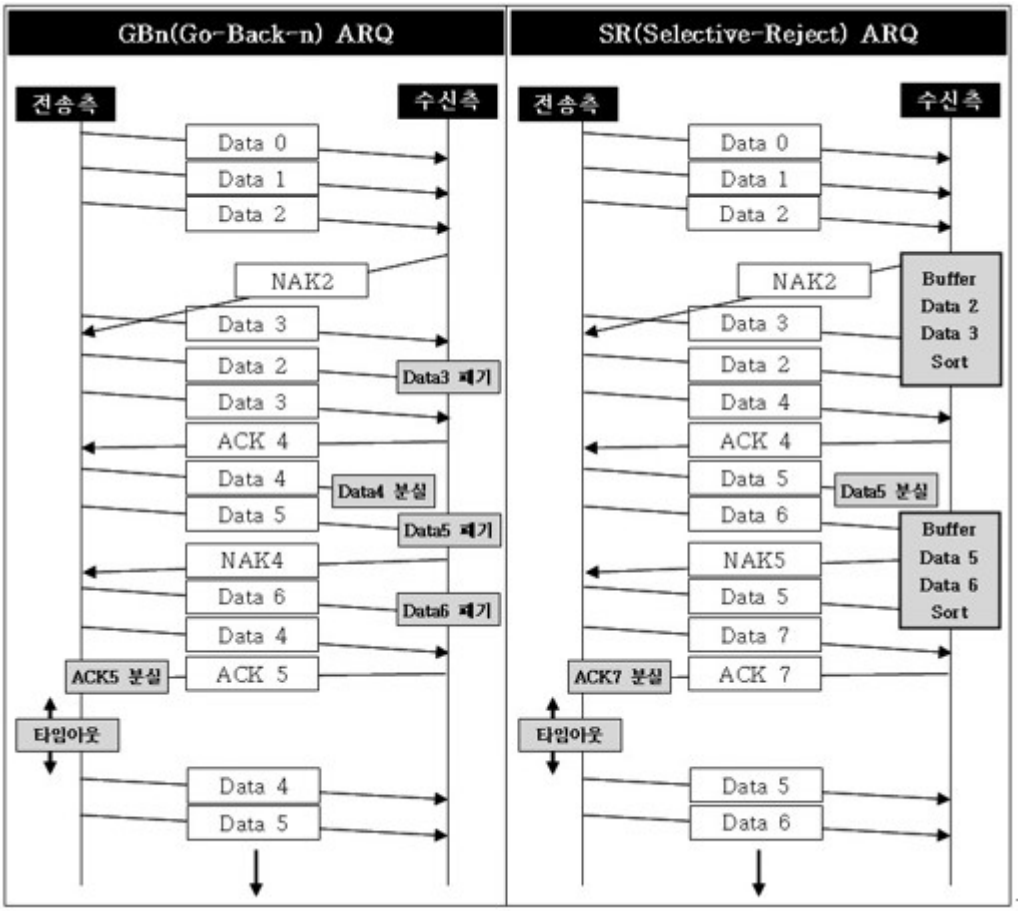


** 전송측 : 데이터를 전송하면서 window size가 줄어들고, ACK를 받으면서 window size가 다시 원상태로 돌아온다.

** 수신측 : 데이터를 수신하면서 window size가 줄어들고, ACK를 보내면서 window size가 다시 원상태로 돌아온다.

-> 이런식으로 window size가 줄어들었다 커졌다 반복하면서 데이터를 응답대기시간 없이 연속적으로 전송하는 것이 슬라이딩 윈도우 플로우 컨트롤의 기본 메커니즘이다.

** Go-Back-N ARQ & SR[Selective Repeat] ARQ ★



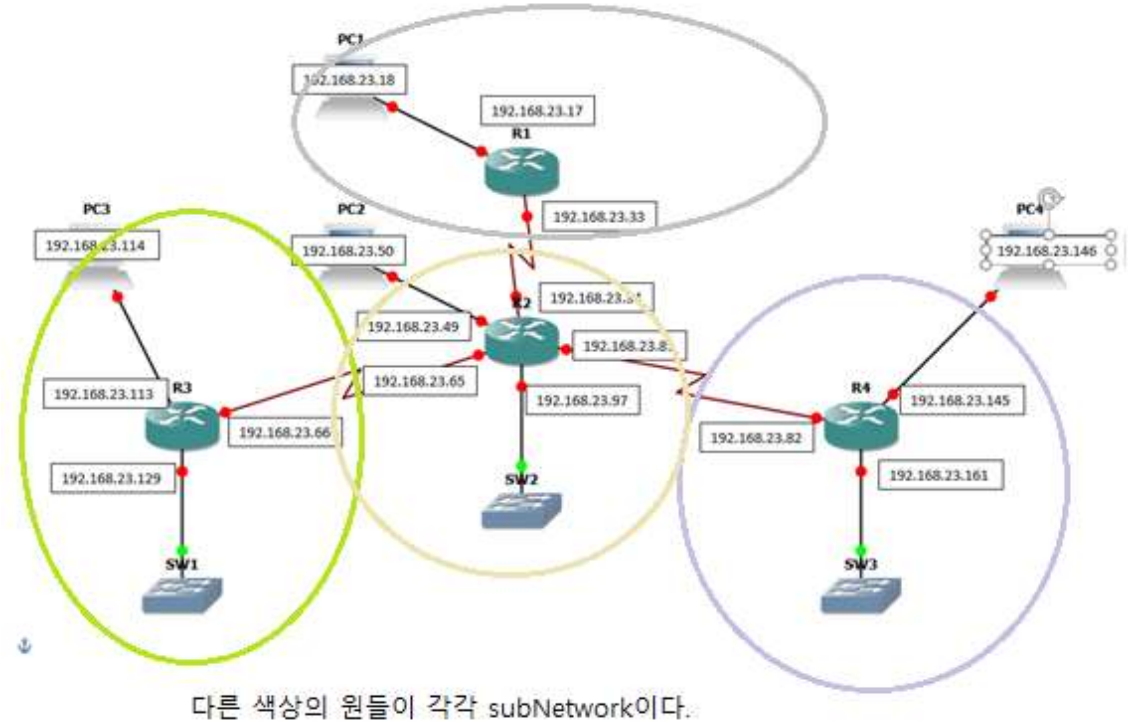
*** GBn ARQ : 수신측에서 손실되거나 분실된 프레임에 대해 NAK 혹은 무응답을 보내면서 **그 프레임 이후로 수신된 프레임을 전부 폐기**한다. 그리고 전송측에서는 **오류가 있었던 프레임부터 다시** 연속적으로 프레임을 전송한다.

*** SR ARQ : 수신측에서 손실되거나 분실된 프레임에 대해 NAK혹은 무응답을 보내면서 **그 프레임 이후로 수신된 프레임은 저장**해 둔다. 그리고 전송측에서는 **오류가 있었던 프레임만**을 다시 전송해준다.

-> 얼핏 보기에도 SR ARQ가 훨씬 효율적인 기법이라는 것을 알 수 있다. 그러나 SR ARQ는 구조가 복잡해서 유지관리비용이 많이 든다. 그래서 필요한 경우에만 사용하고, 많은 경우에 GBn ARQ를 사용한다. (Stop-Wait ARQ는 거의 사용되지 않는다.)

- Network Layer [3계층]
 : 기본적이고 가장 핵심적인 역할은 데이터를 Final Destination까지 보내는 것이다.
  이것을 위한 기능을 “Routing”[경로 생성]이라고 한다.
  (초반부 설명은 라우팅 기법이 아니라 패킷 전송 기법)

* Router
 : 데이터를 전송측에서 최종 수신측까지 전송할 경로를 만드는 기능을 하며, 그에 따른 규약은 Network Layer Protocol을 따른다. 모든 Router는 자신의 Router Table을 가지고 있으며, 그 Table을 보고 패킷을 이웃 노드 중 어떤 노드로 전송할 것인지 결정한다. 그러나 실제로 Router에 모든 이웃 노드들에 대한 정보가 모두 담을 수는 없다. (컴퓨터가 많기 때문)
그래서 subNetwork라는 개념이 탄생하는데, 이것은 특정 범위의 노드들을 모아 하나의 그룹을 만들어서 형성된다. 라우터의 성능이 좋으면 최적의 경로를 찾아 패킷을 전송할 것이고, 나쁘면 조금 덜 좋은 경로를 찾을 것이다.

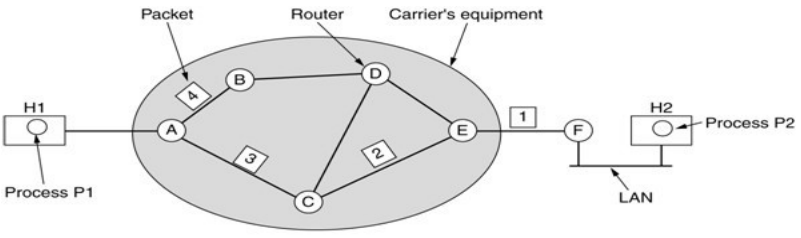


다른 색상의 원들이 각각 subNetwork이다.

Ex) PC3에서 PC4로 통신하기 위해서는 데이터를 일단 R3로 보내고, R3에서는 subNet 단위 라우팅을 실시한다. [노드 단위 라우팅보다 효율적]

* Store-and-Forward Packet Switching
 : 16페이지에서 다루었던 Switching Technology 중에서 패킷 스위칭 기법이다.
  기본적으로 패킷 스위칭은 패킷을 저장하고 다음 노드로 내보내는 방식이다.
  패킷 스위칭 기법에는 Datagram방식과 Virtual circuit방식이 있었다.

** Connectionless Service [Datagram]



A's table

initially		later	
A	–	A	–
B	B	B	B
C	C	C	C
D	B	D	B
E	C	E	B
F	C	F	B

Dest. Line

C's table

A	A
B	A
C	–
D	D
E	E
F	E

E's table

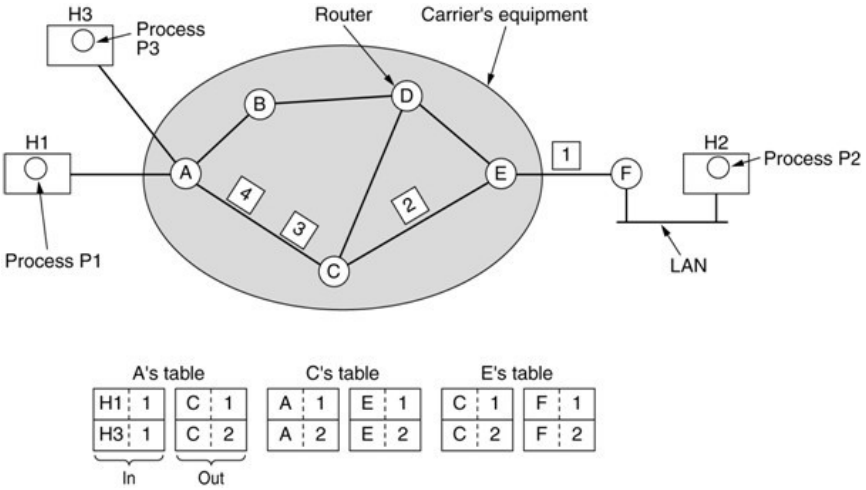
A	C
B	D
C	C
D	D
E	–
F	F

메커니즘 : H1에서 H2로 전송하려는 패킷을 A로 전송 -> A에서는 자신의 라우팅 테이블을 보고 B or C로 전송 -> 또 받은 노드의 라우터에서 자신의 라우팅 테이블을 보고 적당히 전송 -> 반복 -> H2 패킷 수신

*** 특징

1. Packet을 보낼때마다 Routing Table을 검사한다. Routing Table은 적당한 주기를 가지고 Refresh되므로 전송한 패킷들이 항상 같은 경로로 이동하지는 않는다.
2. **통신 경로 중에 하나의 노드가 고장이 나도 큰 영향을 받지 않는다. [Robust]**
   (패킷을 보낼때마다 라우팅 테이블을 검사하므로)
3. 통신 품질관리나 부하관리가 어렵다. [실시간 스트리밍에는 이용하기 힘들다.]

** Connection-Oriented Service [Virtual Circuit]



메커니즘 : H1에서 H2로 패킷을 전송하기 위해 A라우터에서 F까지의 가상의 경로를 건설 -> H1에서 전송할 패킷을 건설된 경로를 통해 전부 전송 -> H2 패킷 수신

*** 특징

1. 보내는 패킷에 건설된 경로의 이름을 붙여 전송하면 된다. (TAG의 경량성)
2. 건설된 경로 중에 하나의 노드가 고장나면 패킷을 전송할 수 없다. [오류에 취약]
3. 통신 품질관리나 부하관리가 Connectionless Service에 비해서 쉽다.

* Routing Algorithm [길을 만드는 기법]

** 요구사항

최적성, 단순성, 안정성, 유연성

** 정적 라우팅 기법 [수동]

패킷들이 이동하는 경로[Routing Table]를 관리자가 수동으로 작성

주기적으로 업데이트 해줘야 하기 때문에 대규모 네트워크에서는 사용할 수 없다.

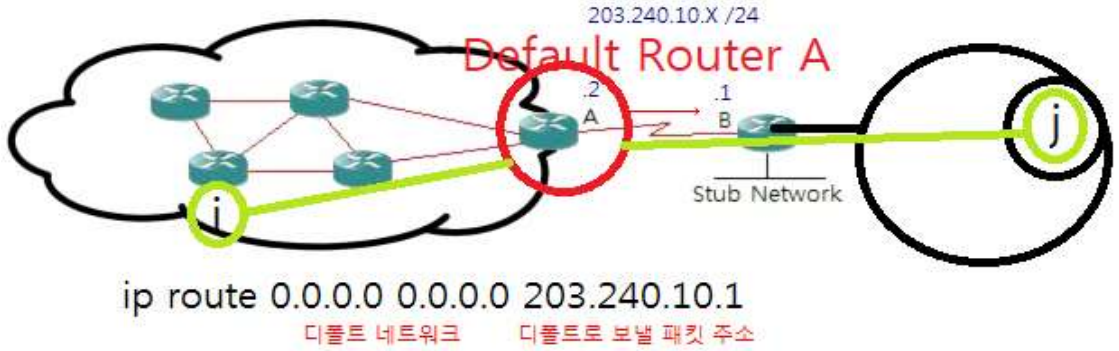
** 동적 라우팅 기법 [자동] ★

라우터가 주위 라우터들과 정보를 교환하며 그 정보를 기반으로 스스로 Routing Table 작성

** 디폴트 라우팅 기법

하나의 subNetwork안에 하나의 Default Router를 정하여 바로 인접 노드로 보내는 패킷이 아니면 전부 Default Router로 보내서 거기에서 최적의 루트를 찾아서 전송하는 기법

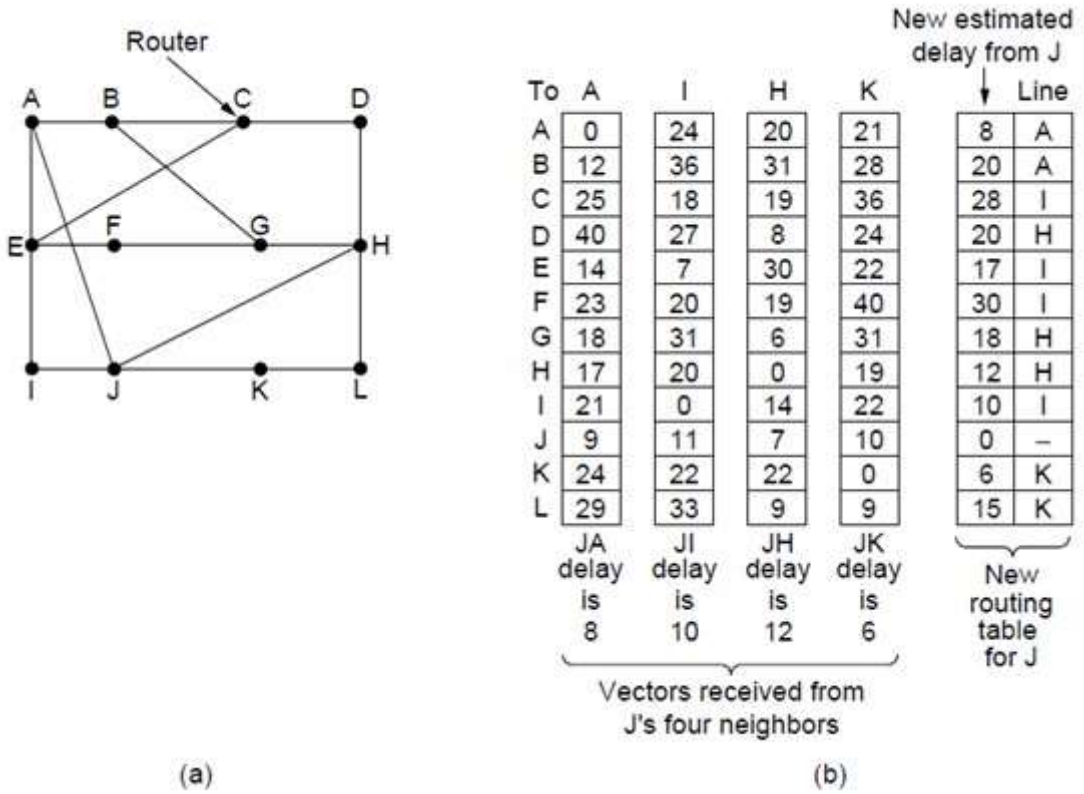
[다른기법보다 Default Router의 성능이 좋아야 한다.]



위 그림에서 i는 j에게 패킷을 전송하고 싶지만, 자신의 라우팅 테이블에 j의 주소가 없다. 그럴때는 자신이 속한 subnet의 Default Router인 A에게 그 패킷을 전달한다. (자신이 루트를모르면 디폴트 라우터로 전송) 그러면 A가 다른 subnet에 있는 j의 주소를 찾아 패킷을 전달한다. [밑에 ip route 0.0.blabla는 신경쓰지 않아도 됨]

** 자치 시스템(AS : Autonomous System) [하나의 subNetwork]
: **동일한 라우팅 프로토콜**을 사용하는 네트워크
** IGP & EGP
IGP [Interior Gateway Protocol]
: 같은 서브넷 상에서 라우팅 정보를 교환하는 프로토콜이다.
EGP [Exterior Gateway Protocol]
: 다른 서브넷과 라우팅 정보를 교환하는 프로토콜이다.
** AS, IGP, EGP는 왜 필요할까?
앞에서 살펴보았듯이 인터넷을 구성하는 프로토콜은 매우 많다. 그런데 완전히 획일화되어 모든 컴퓨터에서 같은 프로토콜을 사용하지는 않는다. 다른 프로토콜을 사용하더라도 당연히 통신이 가능해야한다. 그렇기 때문에 특정 범위내(AS)에서는 같은 프로토콜(IGP)을 사용하고, 다른 프로토콜을 사용하는 자치시스템과 통신할 때는 그것을 통역해주는 매개체가 필요하다. 이 역할을 하는 것이 EGP이다.[13)]

** Routing Algorithm's species ★★
1. Flooding Routing [홍수]
전송하고자 하는 패킷을 자신과 연결된 모든 노드로 배포하는 기법
-> 매우 비효율적, 고비용
-> 그러나 Error에는 굉장히 유리하다. (어떻게든 목적지에 도달)
2. Distance Vector Routing [거리 벡터]



J에서 인접 라우터 A, I, H, K의 Routing Table의 정보를 받으면서 그 정보가 도달하는 delay를 측정한다. 이것을 기반으로 자신이 네트워크 상의 다른 노드로 패킷을 전송하려면 어디로 전송하는 것이 가장 유리한지를 분석하여 자신의 Routing Table을 만든다. 네트워크는 항상 변화하므로 적절한 주기마다 정보를 교환하며 자신의 라우팅 테이블을 갱신한다.

Ex) J가 F로 패킷을 전송하고 싶다. -> A, I, H, K의 라우팅 테이블과 딜레이를 비교한다. -> A를 거쳐가면 23(A-F) + 8(delay) = 31, I를 거쳐가면 20(I-F) + 10(delay) = 30,

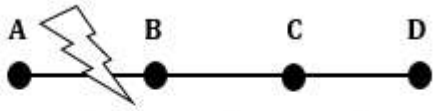
13) 마치 모든 프로토콜에서 이 용어를 사용하는 것처럼 써 놓았지만 실제로 이 용어들은 라우팅 정보 교환 프로토콜에서 사용되는 용어이다. [AS, IGP, EGP]

H를 거쳐가면 19(H-F) + 12(delay) = 31, K를 거쳐가면 40(K-F) + 6(delay) = 46 이다. -> I를 거쳐가는 것이 30으로 가장 적은 Routing metric[14]이 든다. -> 자신의 테이블 F칸에 "30 | I"를 기입한다.

2.1. Distance Vector Routing's fault

대표적인 단점 : **The count-to-infinity problem**

Link Between A & B is Broken



	A	B	C	D
A	0, -	1, A	2, B	3, C
B	1, B	0, -	2, C	3, D
C	2, B	1, C	0, -	1, C
D	3, B	2, C	1, D	0, -

	B	C	D
Sum of Weight to A after link cut	∞, A	2, B	3, C
Sum of Weight to A after 1st updating	3, C	2, B	3, C
Sum of Weight to A after 2nd updating	3, C	4, B	3, C
Sum of Weight to A after 3rd updating	5, C	4, B	5, C
Sum of Weight to A after 4th updating	5, C	6, B	5, C
Sum of Weight to A after 5th updating	7, C	6, B	7, C
Sum of Weight to A after nth updating	.....	....	....
∞	∞	∞	∞

이처럼 일자로 연결된 네트워크에서 Distance Vector Routing protocol을 사용할 때 하나의 노드가 끊어지면, 라우팅 정보를 교환할때마다 Routing metric의 값이 올라가면서 그것이 무한대로 발산하는 문제를 "The count-to-infinity problem"이라고 한다.

Ex) A-B 연결이 끊어진다. -> B는 A로 데이터를 보내기위해 라우팅 테이블을 본다. -> A로 직접 보내면 routing metric이 ∞이고, C를 통해 보내면 2이다. -> A로 데이터를 보내기 위해 C로 데이터를 보낸다. [자신과 A의 연결이 끊긴 것을 인지하지 못함] -> 그것을 다시 C는 자신의 라우팅 테이블을 보고 B로 보낸다. -> 무한 Loop -> 데이터는 전송되지 않는다. -> 정보교환 주기가 되어 정보를 교환한다. -> 정보를 교환할 때마다 A로 데이터를 전송하기 위한 Routing metric이 점점 증가한다.

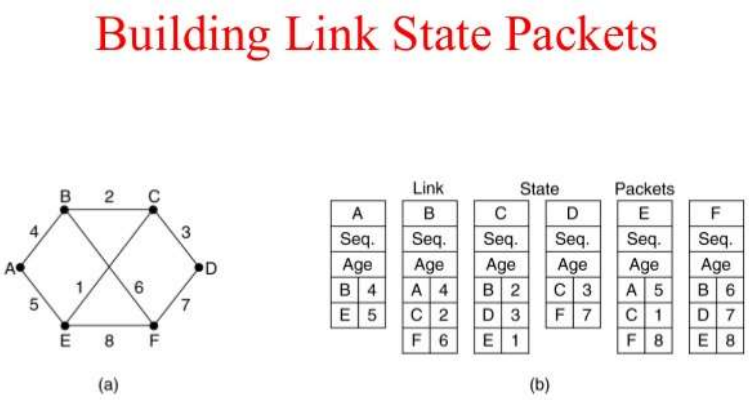
=> 이러한 문제는 각 Router들이 전체 Network 구조에 대한 정보를 가지고 있지 않기 때문에 발생한다.

14) Routing metric : 비용, 시간, 신뢰성 등을 고려하여 패킷을 전송하는데 들어가는 비용을 실수값으로 환산한 것이다. COST이므로 적을수록 좋은 루트이다.

3. Link State Routing (링크상태 라우팅)

: 자신과 직접 연결된 노드과의 Routing metric 정보를 자신이 속한 subnet 전체에 배포한다. 자신**[A]**이 속한 subnet**[a]**의 라우터들**[B, C, D, E, F]**의 직접 연결 정보를 기반으로 그 subnet의 전체적인 구조를 유추하여 자신의 Routing Table을 작성한다.

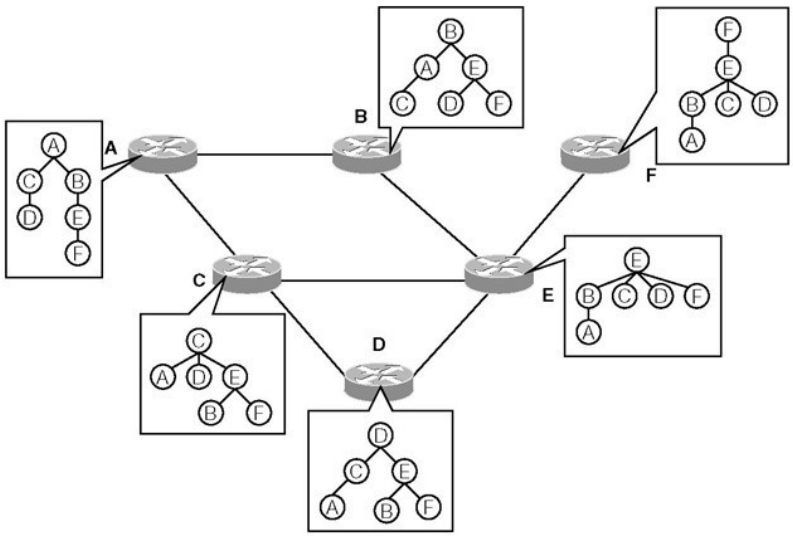
(**[]**는 아래 그림의 A 입장에서의 예시)



(a) A subnet. (b) The link state packets for this subnet.

-> 전체 subnet에 자신의 정보를 배포하는 대신, 정보는 인접 노드와의 Routing metric만을 담고 있으면 되므로 배포하는 정보는 경량화된다.

(D.V.[15]에서는 subnet전체에 대한 Routing metric 정보를 인접 노드에 배포)



-> subnet에 배포된 라우터들의 정보들로 각 라우터들이 자신의 라우팅 테이블을 작성

-> 라우터들이 전체 Network 구조를 인지할 수 있다. [D.V.의 단점 보완]

-> Routing Algorithm이 어려워서 아직 D.V.가 더 보편적이다.

** Distance Vector vs Link State

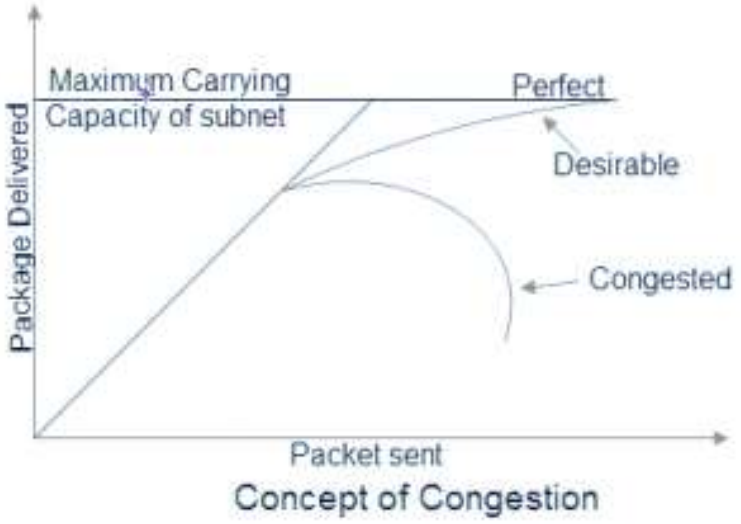
[표 5-1] 거리벡터와 링크상태 방식의 장단점 비교

거리벡터 알고리즘	링크상태 알고리즘
이웃한 라우터의 시각에서 네트워크를 인식	네트워크 전체를 인식
라우터와 라우터 간의 거리를 더하여 계산	다른 라우터까지의 최단경로 계산
주기적으로 갱신 데이터를 교환(컨버전스 시간의 증가)	이벤트 기반의 갱신 신호 교환(빠른 컨버전스 시간)
이웃한 라우터와 라우팅 테이블 교환	링크상태 정보만을 교환

15) Distance Vector

** IGP(Interior Gateway Protocol)에서 사용되는 기법
OSPF(Open Shortest Path First) : Link State Algorithm
RIP(Routing Information Protocol) : Distance Vector
IGRP(Interior Gateway Routing Protocol) : Distance Vector [시스코]

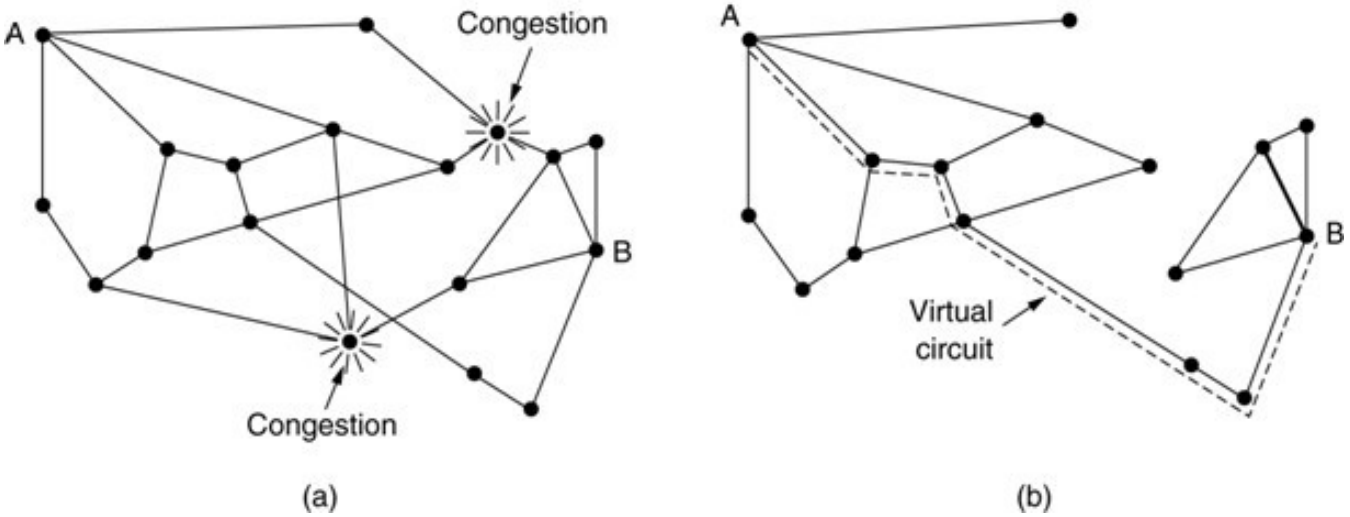
* Congestion Control [부하관리]
: 갑자기 특정 루트의 사용자가 많아지면 Congestion이 일어나고, 네트워크의 속도가 현저히 느려지는데 이것을 관리하는 것이 Congestion Control이다. C.C.은 Network Layer 이외의 다른 Layer에서도 하는데, 근본적으로 Route를 바꿀 수 있는 네트워크 계층의 C.C.가 가장 효과적으로 나타난다.
** Congestion



Concept of Congestion

이상적인 경우가 Perfect, 지향하는 것이 Desirable, 실제는 Congested의 그래프를 따른다. 사용자가 많아지면 Maximum Capacity를 넘지 않았는데도 과부하가 걸리는 이유는 Packet sent가 많아지면 Collision이 발생하고, 두 노드 모두 Back-Off하고 일정 Delay 후에 데이터를 재전송한다. 또, 여기서 자신이 보낸 데이터에 대한 응답(ACK)이 오지 않아서 전송측은 데이터가 중간에 손실된 것으로 생각하고 Re-transmission(재송신)한다. 혼잡한 서버에 이렇게 또 패킷이 점점 늘어나면서 실제로 600명이 사용해도 패킷은 1000~2000개 이상이 전송되는 현상이 발생한다. Maximum Capacity가 1000이라고 가정하면 600~700명만 사용해도 패킷은 그 한계를 넘어서서 Congestion이 발생하는 것이다.

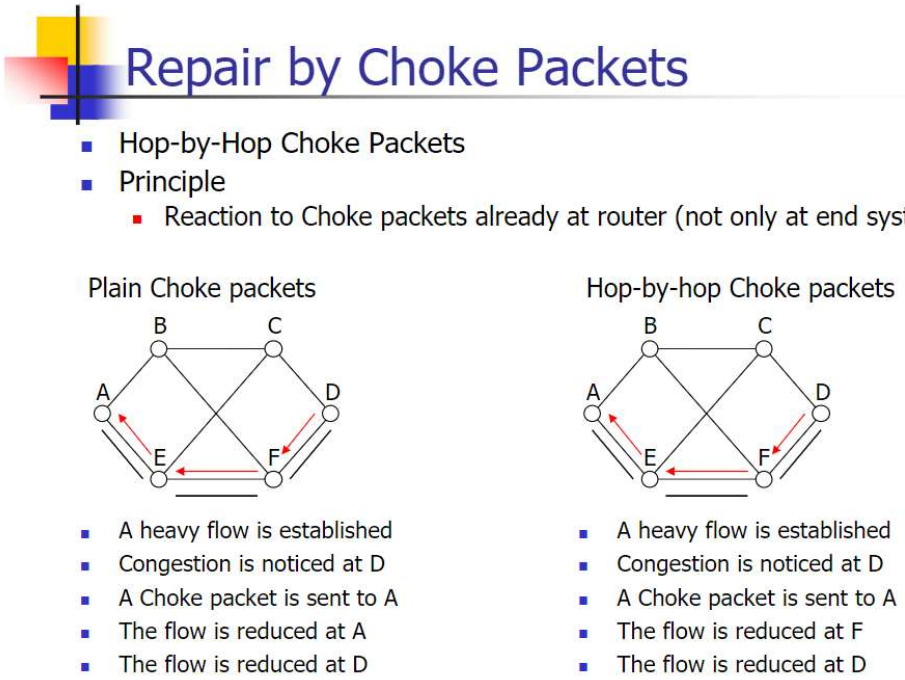
** Congestion Control Methods
기본적인 방법 : Monitoring -> Congestion 발생 -> Congestion 발생을 타 노드에게 알림
1. Congestion Control in Virtual-Circuit subnets
: Congestion 발생 소식을 접한 전송측은 수신측으로 패킷을 이동시킬 때 Congestion이 발생한 루트를 피해 임의의 Virtual-Circuit을 설정하여 그 경로로만 패킷을 전송한다.
[Datagram 방식으로는 각 패킷들이 이동할 경로를 예측할 수 없기 때문에 Congestion 발생루트로 패킷이 이동할 가능성이 있다.]



2. Plain Choke Packets vs Hop-by-Hop Choke Packets

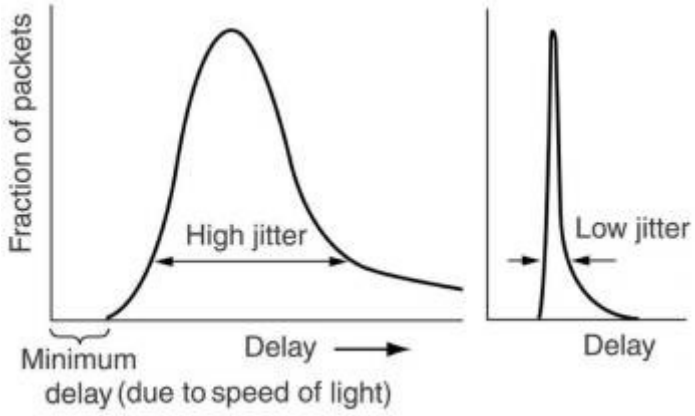
: 데이터를 전송하는 경로 중 어떤 경로에 Congestion이 발생하면 발생지에서 전송지까지 Choke Packet을 전송하여 자신의 루트가 현재 막혀있음을 전달한다. 이를 전달받은 전송측은 Congestion을 해소하기위해 데이터의 전송량을 줄인다. 아래 그림의 Plain은 Choke Packet이 전송측까지 가면 전송측에서부터 데이터 전송량을 줄이는 방식이고, Hop-by-Hop은 Choke Packet을 받은 노드에서부터 점차적으로 데이터 전송량을 줄이는 방식이다.

Hop-by-Hop이 Plain보다 피드백이 빠르다고 볼 수 있다.



* Jitter Control [Jitter ≈ 오차]

** Jitter?



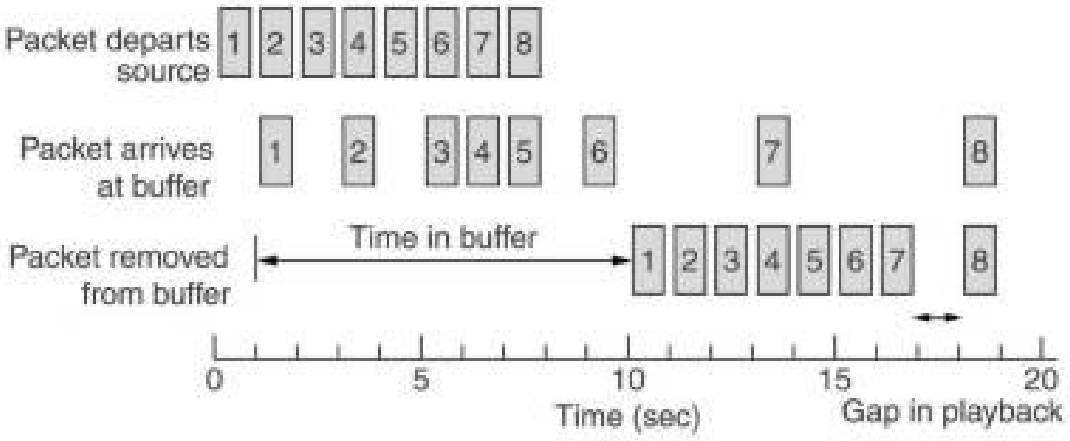
: Jitter는 위에 보이듯 Delay 분산값의 크기라고 할 수 있다. 예를 들어 High Jitter의 경우 데이터를 5개 보내면 1초, 3초, 2초, 6초, 2초의 Delay가 소요될 수 있다고 하면, Low Jitter의 경우 데이터를 5개 보내면 3초, 3.1초, 2.9초, 2.96초, 3.02초의 Delay가 소요된다. 즉, 데이터의 도착시간의 오차가 Jitter의 크기에 따라 결정된다고 볼 수 있다. 실시간 스트리밍같이 일정량의 데이터가 일정한 주기에 도착해야하는 경우, Low Jitter는 필수적이다. QOS[16]는 데이터 전송의 신뢰성, 딜레이, Jitter, Bandwidth등의 품질을 관리하는 것을 의미한다.

1. Buffering (버퍼링)

: Jitter Control을 하기에 가장 손쉬운 방법이다. 데이터가 High Jitter로 수신되어 패킷이 수신되는 Delay time이 들쭉날쭉한 경우, **"수신측에서"** 일정량의 데이터가 수신될 때까지 앞서받은 패킷을 잠시 저장해두었다가 한꺼번에 사용자에게 데이터를 보여주는 방식이다.

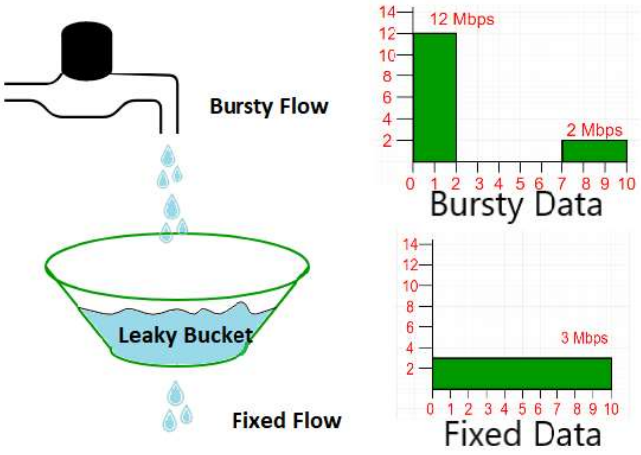
Ex) 유튜브 동영상이 실행되기전까지 Delay time이 들지만, 실행되면 어느정도의 분량은 끊김없이 볼 수 있다.

16) QOS : Quality of Service [서비스의 품질 관리]



2. The Leaky Bucket Algorithm [물 새는 양동이 알고리즘]

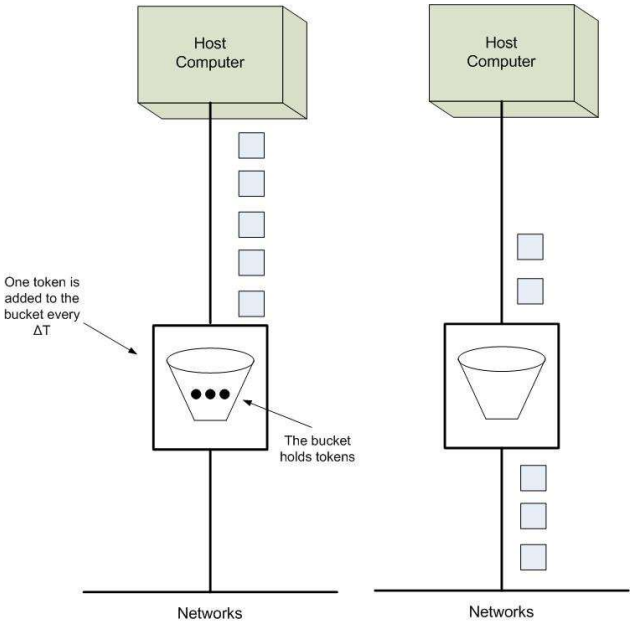
: 전송측에서 데이터를 보낼 때 막 보내는 것이 아니라 임의의 공간에 보낼 데이터를 담아두고, 데이터 전송속도를 정해 그 공간에서 일정한 속도로 데이터를 내보내는 알고리즘이다. 이렇게하면 데이터의 전송속도를 일정하게 조절 가능하기 때문에 Jitter를 관리할 수 있다. 이 알고리즘으로는 Congestion이 없으면 양동이에 큰 구멍을 뚫어 고속으로 데이터를 전송하고, Congestion이 발생하면 구멍을 줄여서 저속으로 데이터를 전송함으로 Congestion도 어느정도 관리할 수 있다.



---

3. The Token Bucket Algorithm

: Leaky 방식으로 Jitter Control은 했지만, 사용자들의 **데이터 전송속도 형평성 문제**가 발생한다. (누군가 계속 데이터를 전송해서 막힌 루트 때문에 이제 막 데이터를 전송하려는 사용자도 저속으로 네트워크를 사용해야한다는 문제) 이것을 해소하기위해 제시된 것이 Token Bucket 알고리즘이다. 이 방식은 사용자마다 Δt시간당 1개의 Token을 부여하고, 사용자가 데이터를 전송하기 위해서는 Token을 소모해야하도록 정한다. 이렇게하면 계속 네트워크를 이용하는 사용자는 Token을 받음과 동시에 소모하고, 네트워크를 이제 막 이용하려는 사용자는 그동안 쌓인 Token으로 고속으로 네트워크를 이용할 수 있게 된다.

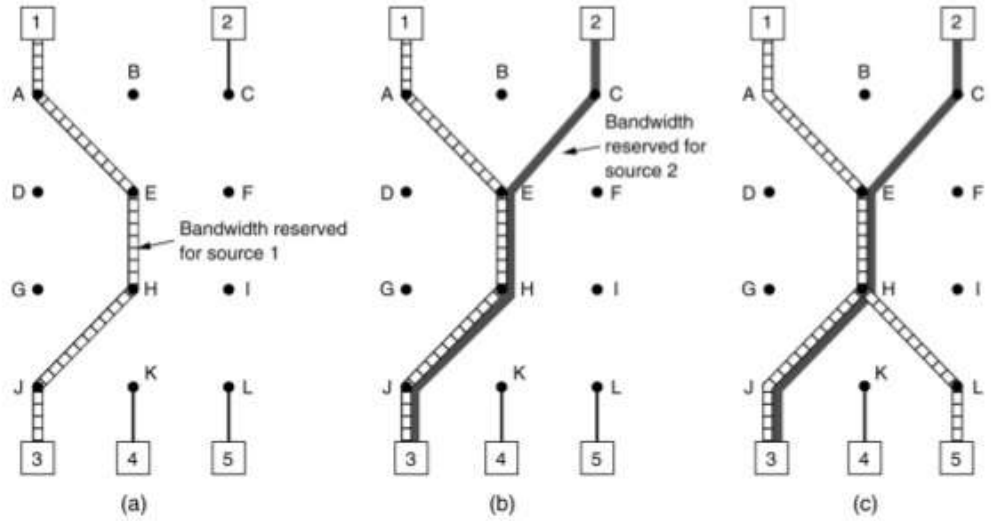


** RSVP-The ReSerVation Protocol [예약 프로토콜]

: 이 방식은 Bandwidth를 예약자에게 할당하는 것으로 Congestion을 관리하는 방식이다.

Ex) SKT에서 0~1000만큼의 Bandwidth를 정부로부터 할당받았다. 그러면 0~1000에 해당하는 Bandwidth는 SKT에서 예약한 것이므로 KT, LGU+에서는 사용하지 못한다. 그러므로 SKT는 KT,LGU+에서 엄청난 Congestion을 겪고 있다하더라도 자신이 예약한 Bandwidth를 사용하여 편안한 네트워킹을 할 수 있다.

RSVP-The ReSerVation Protocol (2)



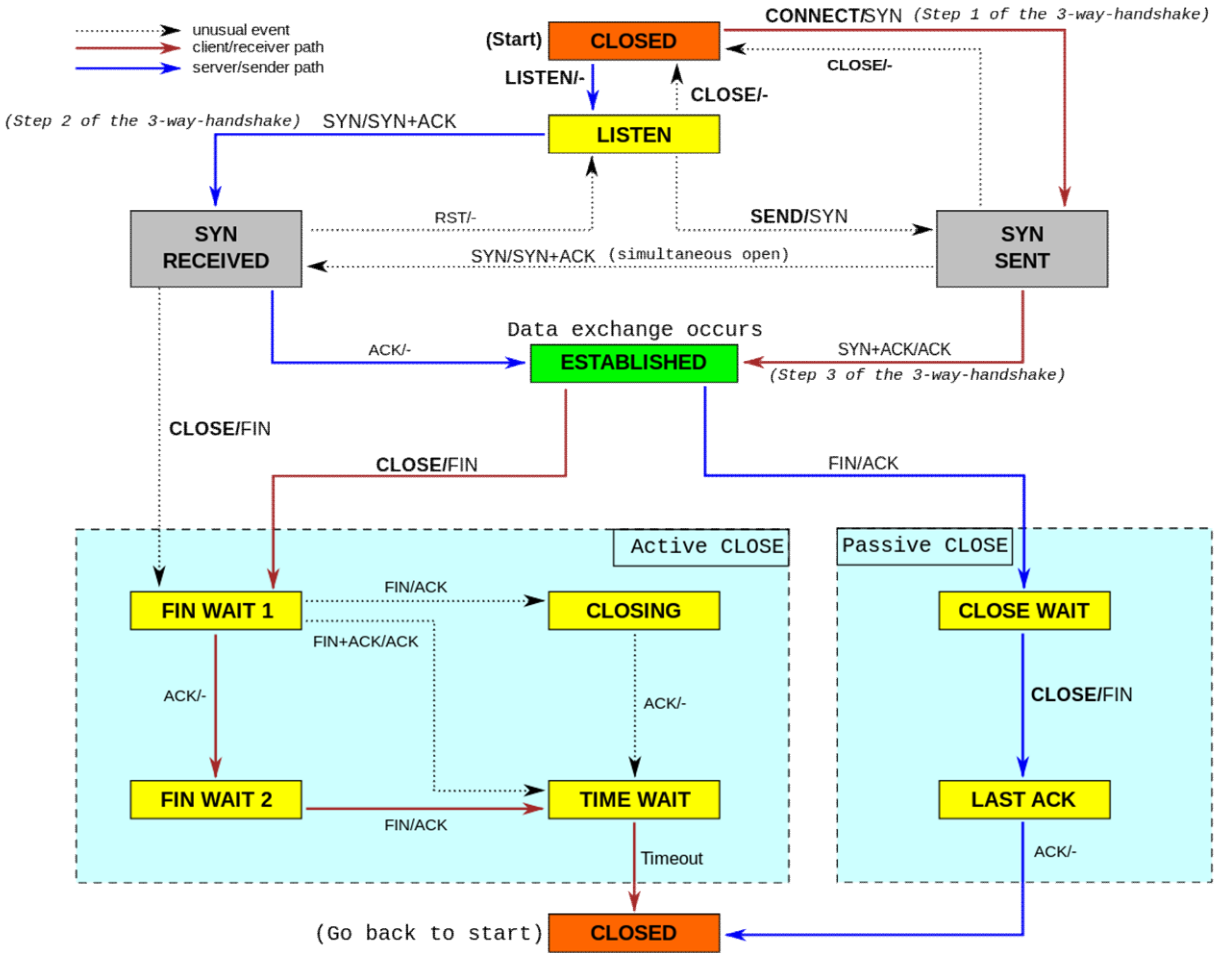
(a) Host 3 requests a channel to host 1. (b) Host 3 then requests a second channel, to host 2. (c) Host 5 requests a channel to host 1.

- Transport Layer [4계층]

* Transport Layer의 역할

Transport Layer은 1, 2, 3계층과 달리 처음으로 End-to-End Communication을 하는 계층이다. 이를 위해 Source 와 Destination의 연결 요청, 연결, 연결 해지 등의 작업을 수행하고**(Connecgtion Establishment)**, 데이터의 port 번호를 구분하여 Session Layer의 어떤 서비스에 해당 데이터를 할당할 것인지를 결정한다.**(Addressing)** 또한 데이터가 전송되는 것에 대한 **Flow Control**도 한다.

* Connection Establishment & Release Model



SYN : Synchronization [연결 요청]

RST : Reset [즉시 연결 종료]

ACK : Acknowledgement [긍정 응답]

FIN : Finish [연결 종료 요청]

CLOSED : 초기 상태 (비연결 상태)

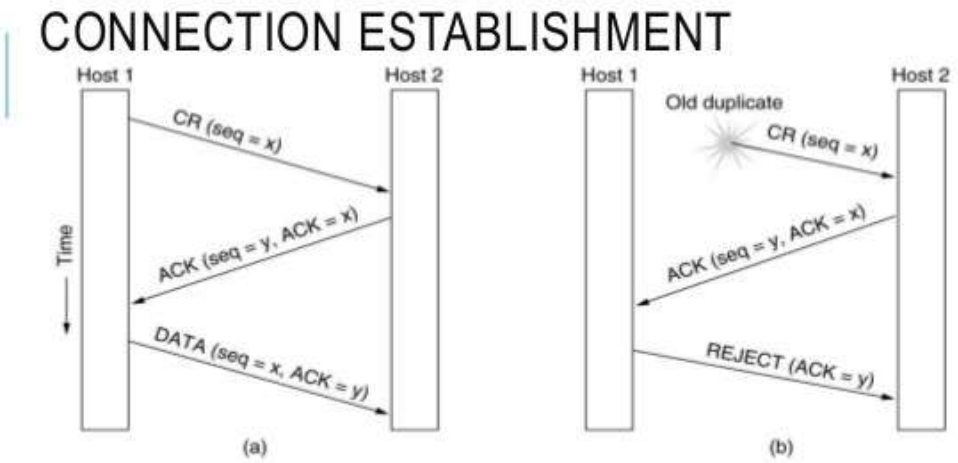
ESTABLISHED : 연결 성립 상태

(그림에서)

빨간선이 Active, 파란선이 Passive

연결 종료할 때 빨간선이 먼저 연결 종료 요청

* Connection Establishment [Not using Three-way Handshake case]

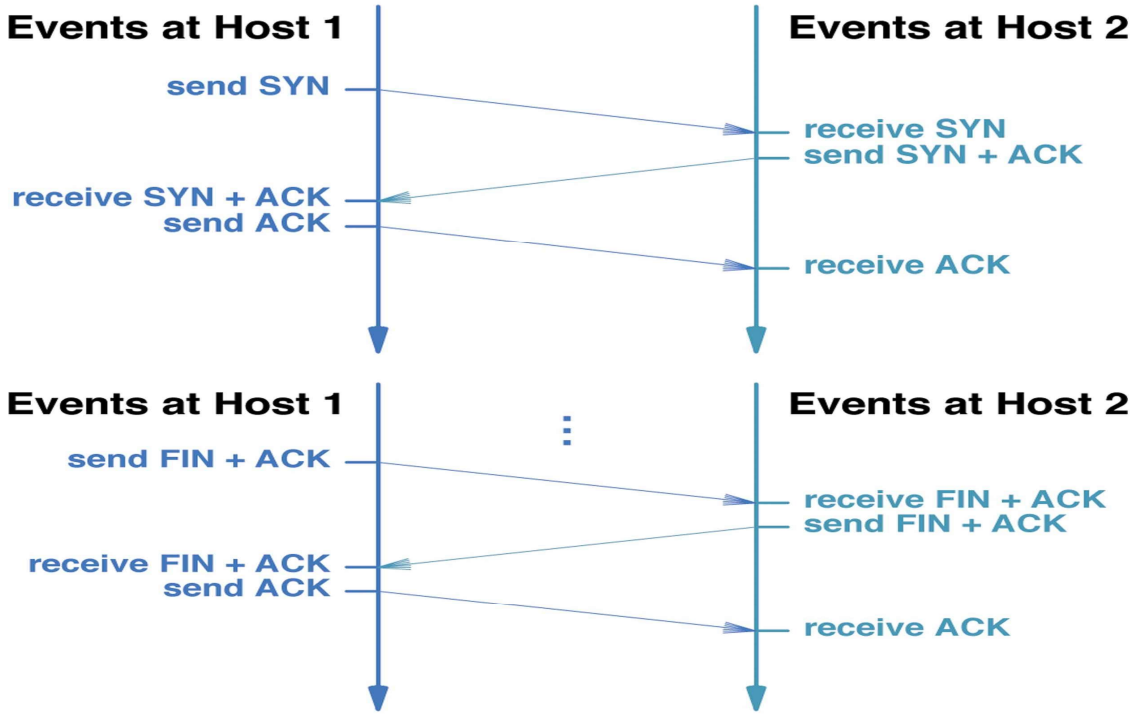


Scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST.
(a) Normal operation,
(b) Old CONNECTION REQUEST appearing out of nowhere.

일반적인 경우 (a)와 같이 Host1에서 CR(Connection Request)를 보내고, Host2가 응답하고, Host1과 Host2가 연결된다. 그러나 (b)와 같이 **Old duplicate**가 발생하여 이미 Host1과 Host2가 연결된 상태인데 Host1에서 보낸 CR이 Host2에 도착하는 경우가 발생한다.[17]

이 경우 2가지 해결방법이 있는데, 첫 번째 방법은 Host2에서 이미 연결된 Host1으로부터 도착한 CR1을 무시하는 것이다. 두 번째 방법은 Host2에서는 그냥 CR1에 대한 ACK1을 Host1으로 전송하고, Host1에서 2개의 연결 중 하나를 Reject해서 끊어내는 것이다. 이 모든 것이 가능한 이유는 CR과 ACK, REJECT, DATA 모두가 **각각의 seq(Sequence number)를 가지고 있어서 구분 가능하기 때문**이다.

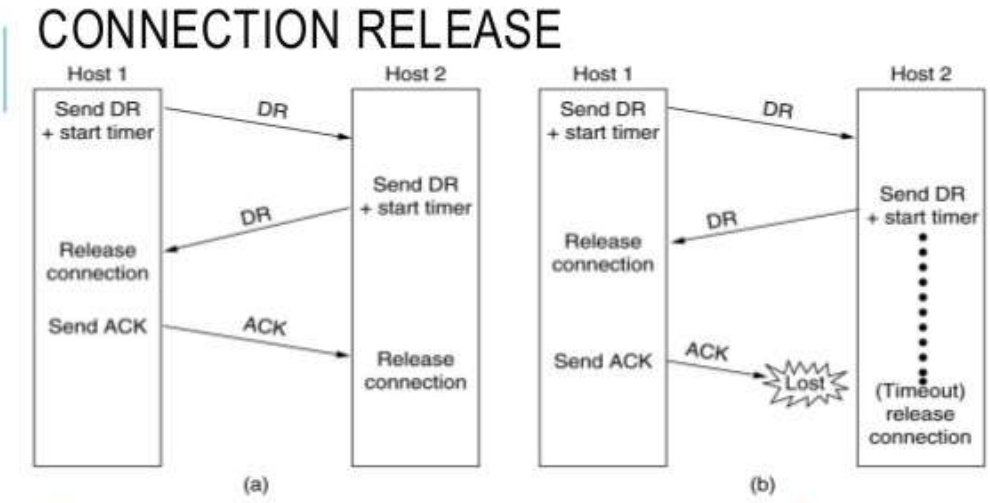
* Three-way Handshake [3회 인사 기법]



17) Old duplication 발생 원인 : Congestion과 Re-transmission 때문 [Host1에서 CR1을 보냈는데 CR1이 가는 도중 Congestion이 발생하여 전송이 지연될 경우, Host2로부터 ACK를 받지 못한 Host1은 CR1에 오류가 있다고 판단하고 CR2를 Re-transmission한다. 그에 대한 ACK2가 도착하여 Host1과 Host2는 연결이 맺어진 상태인데, Congestion 때문에 늦어진 CR1이 Host2에 도착한다. 이것이 old duplication이다.

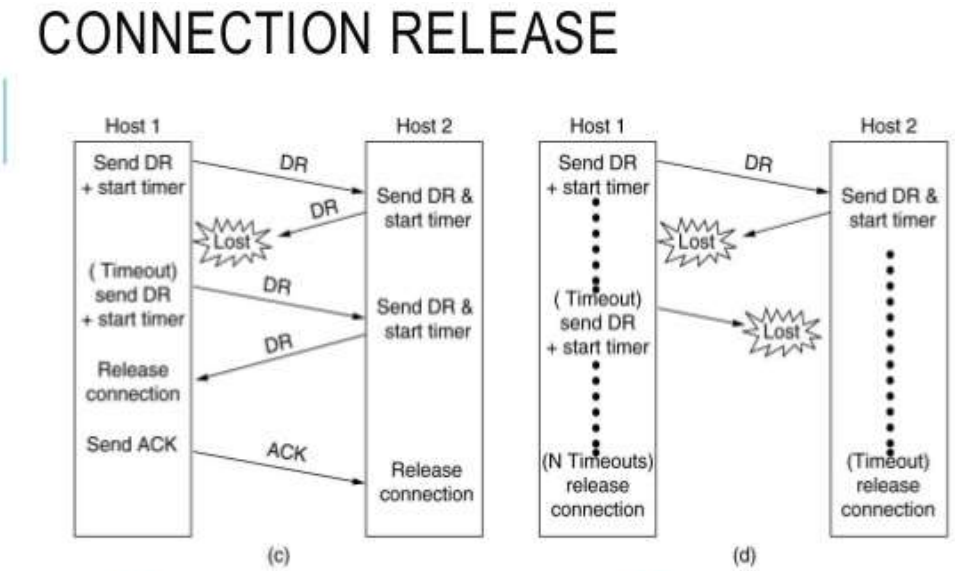
이처럼, 3번에 걸쳐 확인하는 기법을 Three-way Handshake Algorithm 이라고 한다. 위에 보이듯 Three-way Handshake는 연결시, 연결 종료시에 사용한다. 그러나 이 기법은 연결 종료시 문제가 있다. 위 그림에서 예를 들면, Host1이 Host2가 자신이 보낸 마지막 ACK를 제대로 받았는지 확인할 수 없다는 것이다. 이 문제를 "**The two-army problem**"이라고 한다. 이 문제를 해결하기 위해 도입하는 것이 **Timeout** 이다. 일정시간을 기다리다가 응답이 없으면 ACK가 오지 않아도 연결을 종료하는 것이다.

** Connection Release **with TimeOut**



(a) Normal case of a three-way handshake.
ACK lost.

(b) final



(c) Response DR lost subsequent DRs lost.

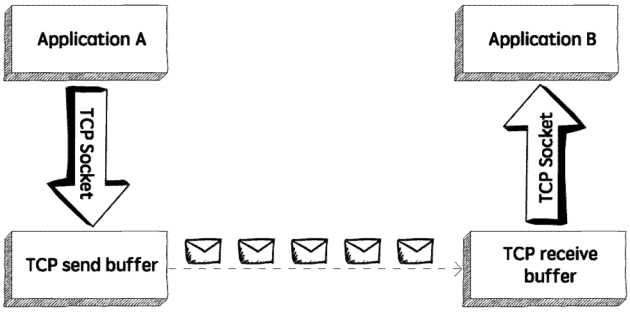
(d) Response lost and

(c)와 같이 2번째 Handshake에서 오류가 난 경우에는 일정시간이 지난 후, Active한 쪽에서 다시 DR(Disconnection Request)를 보내고, 다시 3회의 인사를 한 후, 연결을 종료한다.

(d)와 같이 극단적으로 data loss가 많이 발생하는 경우, Active한 쪽에서 N번의 DR 재전송을 실시하고, 아무응답이 없으면 Active도 연결을 종료하는 것이다. (Passive의 입장에서는 자신이 DR을 수신하고 ACK를 보냈으므로 일정시간이 지나면 연결을 종료한다.) ((d)보다 더 극단적으로 첫 번째 DR마저 도착하지 않고, 몇 번의 DR재전송도 모두 lost된다면 그것은 이미 연결된 상태라고 할 수 없다.)

* Flow Control and Buffering in Transport Layer

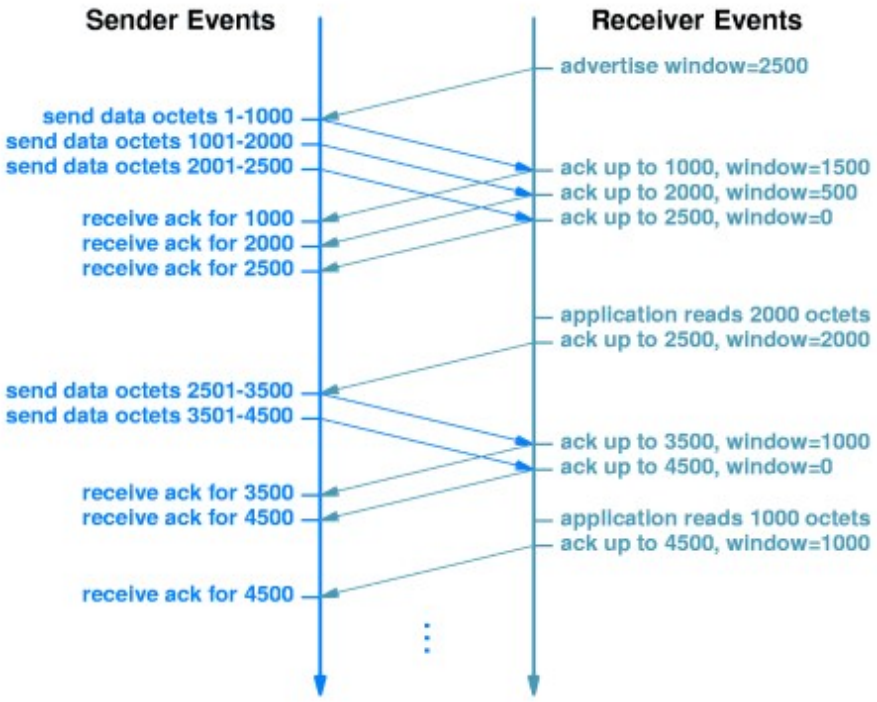
** TCP Layer 데이터 전송 모델



-> TCP Layer의 모델이지만 Transport에서도 동일한 구조로 동작한다.

-> 여기서 말하는 buffer는 “데이터를 임시로 담아둘 공간”이라고 인식하면 된다.

-> buffer에 일정량의 데이터가 모이면, 윗 계층으로 데이터를 보낸다.



위 그림에서 advertise window의 크기는 자신이 수신을 위해 준비한 buffer의 크기를 의미한다. 기본적으로 데이터 송수신에서 Sliding Window Algorithm을 사용한다. Sender에게 자신의 buffer 여유량을 알려주는 이유는 buffer가 가득 찼는데 데이터가 들어오면 데이터를 받을 수 없기 때문이다. 교제에서는 가장 먼저 Sender가 Receiver에게 미리 어느정도 크기의 buffer를 준비하라고 하는 신호를 보내는 것으로 나와있다.

(application reads xxx octets는 윗 계층으로 xxx만큼의 데이터를 올린 것을 의미한다.)

* Addressing [서비스 주소 찾기]

: Transport Layer에서는 위의 중요한 연결, 흐름제어의 역할도 하지만 서비스 주소를 찾아주는 기능도 한다. Addressing이란 Transport Layer에서 Upper Layer로 데이터를 올릴 때 데이터의 포트번호(port number)[18]를 확인하여 이 데이터가 어떤 서비스를 위한 데이터인지 구분하여 해당 서비스 포인트로 올리는 기능이다.

** 대표적인 국제표준 port number

The TCP Service Model

Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial File Transfer Protocol
79	Finger	Lookup info about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

Some assigned ports.

-> 전송된 데이터의 포트번호가 21번이다. -> Upper Layer의 FTP서버로 전송

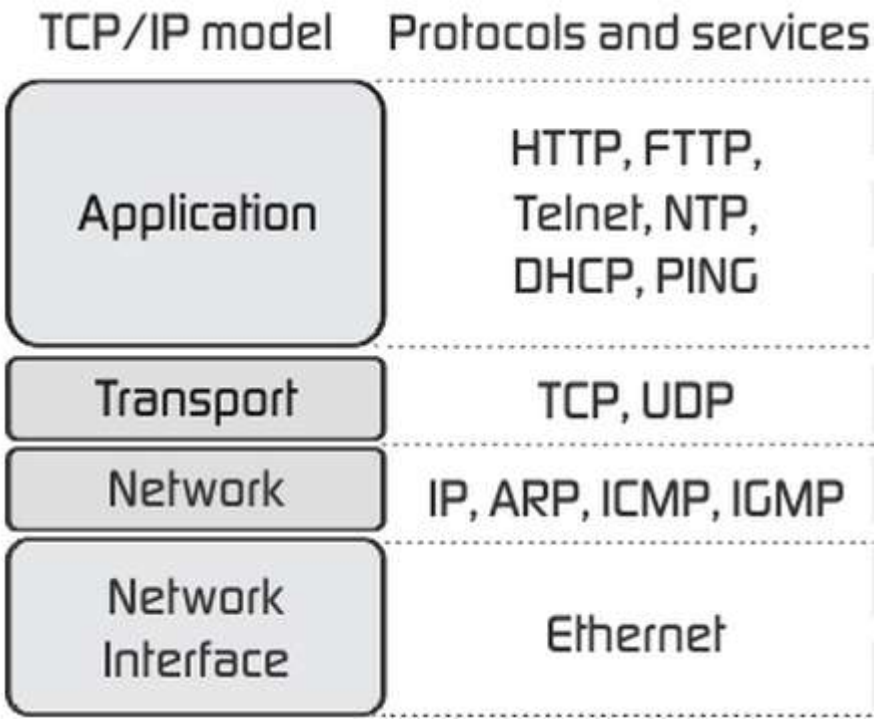
-> 전송된 데이터의 포트번호가 80번이다. -> Upper Layer의 HTTP서버로 전송

[사실상 TCP/IP 모델에서는 TCP 위가 바로 Application Layer라서 이러한 부분을 간단하게 분류하지만, OSI 7 Layer Model에서는 바로 Application Layer가 아니라서 조금은 더 복잡한 단계를 거치게 된다.]

18) port number는 16bits 정수형태로 구분되어 0~65535번까지 사용할 수 있다.

- TCP/IP Network Model

: OSI Reference Model의 Network Layer는 IP Layer, Transport Layer는 TCP Layer로 대체하고 Session Layer, Presentation Layer, Application Layer는 하나의 Application Layer로 통합하여 네트워크 통신을 단순화한 네트워크 모델 [오늘날 많은 비중을 차지]



- IP Layer(Internet Protocol Layer) [Network Layer's role]

: 기본적으로 하는 역할은 OSI Model의 Network Layer의 그것과 동일하다.

Internet Protocol에서는 비연결형 비신뢰성 데이터그램 프로토콜을 사용 (속도⟰, 효율⟰)

[Best Effort Delivery Service 제공 (비신뢰성)]

* ICMP(Internet Control Message Protocol)

: Routing에 관한 정보를 관할하는 패킷

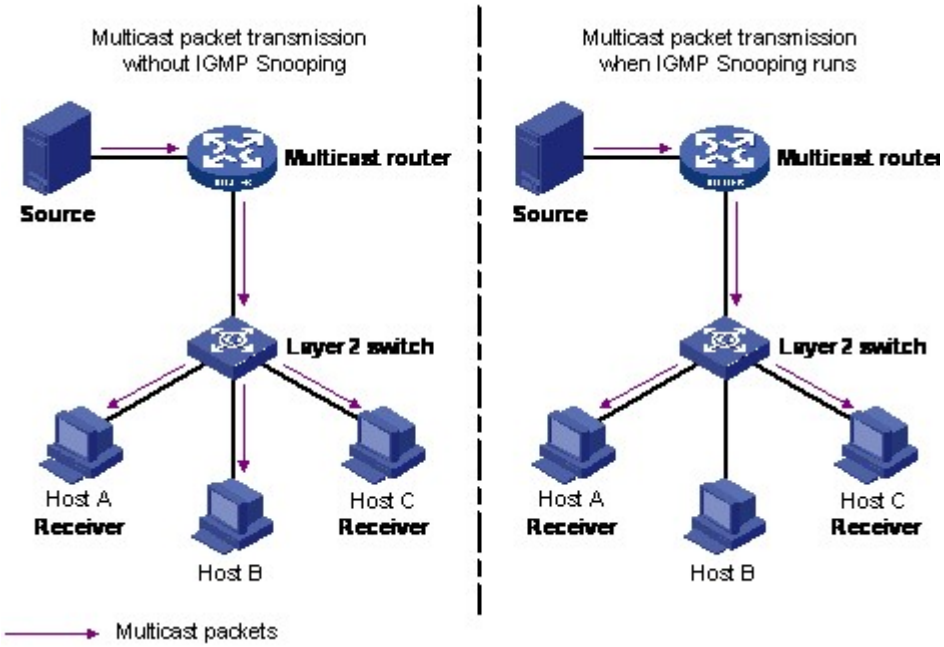
예를 들어 컴퓨터의 cmd에 "ping www.google.co.kr"을 입력하면 google의 서버 컴퓨터 ip를 DNS[19]로부터 받아와서 해당 IP에 해당하는 서버로 ICMP[Echo Request] Packet을 보낸다. 이것을 받은 구글 서버는 전송지로 ICMP[Echo Reply] Packet을 전송하여 응답한다. 이것이 정상적으로 작동된다면, 전송Host와 수신서버간의 통신에는 문제가 없다는 의미다. 그래도 구글 홈페이지가 열리지 않는다면 통신의 문제가 아니라 Application의 문제임을 확인 할 수 있다. 아래의 예시는 Ping의 예시와 ICMP Packet의 Type과 용도이다.

```
C:₩Users₩KSC>ping www.google.co.kr

Ping www.google.co.kr [172.217.24.131] 32바이트 데이터 사용:
172.217.24.131의 응답: 바이트=32 시간=35ms TTL=53
172.217.24.131의 응답: 바이트=32 시간=33ms TTL=53
172.217.24.131의 응답: 바이트=32 시간=34ms TTL=53
172.217.24.131의 응답: 바이트=32 시간=33ms TTL=53

172.217.24.131에 대한 Ping 통계:
    패킷: 보냄 = 4, 받음 = 4, 손실 = 0 (0% 손실),
왕복 시간(밀리초):
    최소 = 33ms, 최대 = 35ms, 평균 = 33ms
```

19) Domain Name Server : 도메인 이름과 서버 ip를 매핑하여 가지고 있는 서버, 이것을 우리가 탐색창에 도메인 이름(www.~~)로 입력한 것을 해당 서버의 ip주소로 변환해준다. 자신의 컴퓨터가 어떤 도메인 이름 서버를 사용하는지를 확인하려면 cmd에서 nslookup을 입력하면 된다.

Type	Code	Description
0 – Echo Reply	0	Echo reply
3 – Destination Unreachable	0	Destination network unreachable
	1	Destination host unreachable
	2	Destination protocol unreachable
	3	Destination port unreachable
	4	Fragmentation needed and DF flag set
	5	Source route failed
5 – Redirect Message	0	Redirect datagram for the Network
	1	Redirect datagram for the host
	2	Redirect datagram for the Type of Service and Network
	3	Redirect datagram for the Service and Host
8 – Echo Request	0	Echo request
9 – Router Advertisement	0	Use to discover the addresses of operational routers
10 – Router Solicitation	0	
11 – Time Exceeded	0	Time to live exceeded in transit
	1	Fragment reassembly time exceeded
12 – Parameter Problem	0	Pointer indicates error
	1	Missing required option
	2	Bad length
13 – Timestamp	0	Used for time synchronization
14 – Timestamp Reply	0	Reply to Timestamp message

* IGMP(Internet Group Management Protocol)
: Multi-Casting을 위한 그룹 정보를 관리하는 패킷



위 그림에서 알 수 있듯 IGMP Snooping[20]을 사용하고 하지 않고는 저 정도의 차이가 있다. 데이터를 원하는 그룹에서 원하는 호스트로만 전송할 수 있도록 한다.

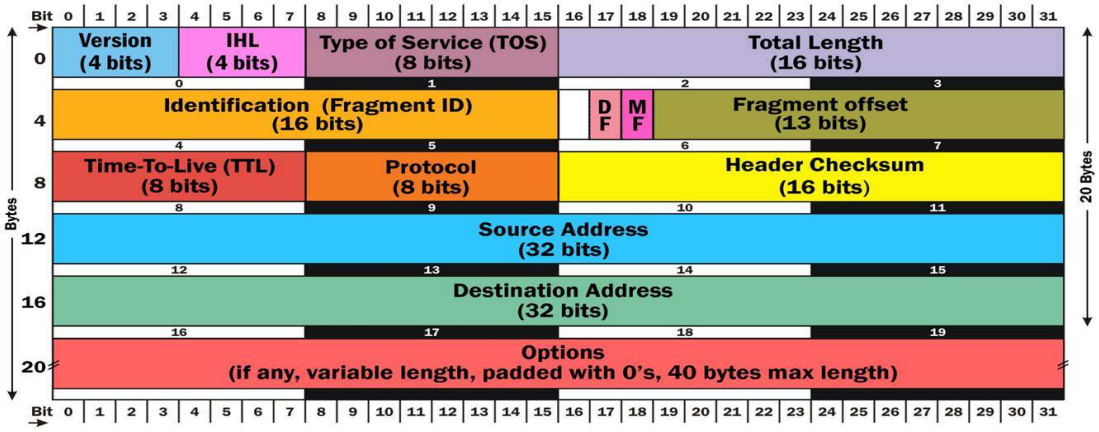
20) IGMP Snooping : Switch가 Host와 Router간의 대화내용을 훔쳐보는 것이다. Switch가 이것을 중간에 확인함으로써 Group내의 어떠한 Host가 데이터를 요구하는지 확인하고, 요구하는 호스트에게만 데이터를 전송하는 기법이다. 당연히 UniCasting이나 BroadCasting에서는 사용되지 않는다.

* ARP(Address Resolution Protocol)
 : 가지고 있는 IP Address 정보를 기반으로 그에 대응되는 MAC Address를 알려주는 Packet이다.
 Ex) ARP Request Broad-Cating -> ARP Reply [MAC Address 정보]

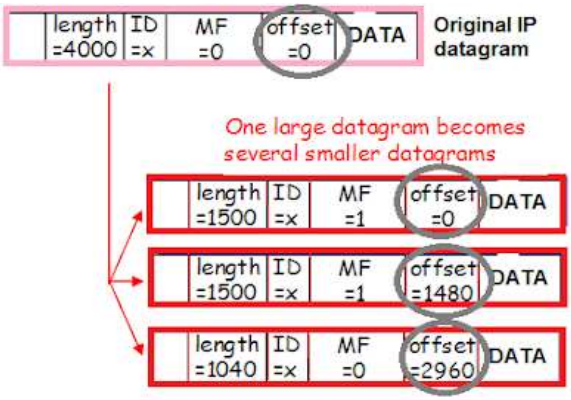
* RARP(Reverse ARP)
 : 가지고 있는 MAC Address 정보를 기반으로 그에 대응되는 IP Address를 알려주는 Packet이다.
 Ex) RARP Request Broad-Cating -> RARP Reply [IP Address 정보]

=> ARP, RARP는 간단히 말하면 지금 보내야하는 정보의 도착지의 MAC Address를 모를 경우, ARP Packet의 Destination Address를 255.255.255.255(BroadCasting)로 설정하여 모든 노드에 뿌린다. 그러면 그 ARP Packet을 받은 많은 노드들 중 호스트가 찾는 IP Address를 가진 노드가 자신의 MAC Address 정보를 ARP Reply에 담아 전송지로 보낸다. 이렇게 함으로써 전송측에서는 IP정보를 가지고 그 IP에 해당하는 MAC정보를 얻는 것이다.
RARP는 위의 예에서 IP Address와 MAC Address의 위치만 바꾼 것이다.

* IPv4 Header Packet ★



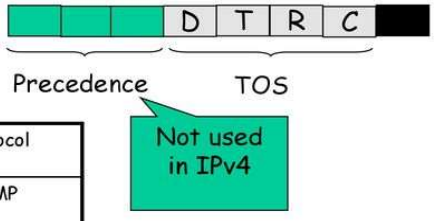
-- Version : IP Header Packet의 version 정보가 담겨있다. (IPv4에서는 4)
-- IHL(IP Header Length) : IP Header의 길이 정보 저장 (20 ~ 60byte)
-- Total Length : Header + Payload 전체 길이 정보 저장 (20 ~ 65535byte)
 -> Payload가 너무 크면 Fragmentation(분할)하여 전송
-- Identification : Fragmentation된 패킷의 고유 식별 번호 저장
-- DF(Don't Fragment) : 중간 노드에서 패킷을 더 이상 분할하지 못하도록 설정 [DF가 1이면]
-- MF(More Fragmentation) : 뒤에 분할된 패킷이 더 남아있음을 알림 [MF가 1이면]
-- Fragment Offset : 분할된 패킷의 순서를 알리기 위해 패킷을 시작점 정보를 저장



-- ToS(Type of Service) : 서비스의 종류 구분
D(Delay) : 지연시간 최소화
T(Throughput) : 처리율 최대화
R(Reliability) : 신뢰성 최대화
C(Cost) : 비용 최소화
Tos값에 의해 Routing Algorithm을 설정 (라우터에 따라 무시될 수도 있음)

Type of Service

D: Minimize delay R: Maximize reliability
T: Maximize throughput C: Minimize cost



TOS bits	Description	Protocol
0000	Normal	ICMP
0001	Minimize cost	NNTP
0010	Maximize reliability	SNMP
0100	Maximize throughput	FTP (data)
1000	Minimize delay	FTP(control) DNS

8. Network layer 2 CS470 8-19

-- TTL(Time To Live) : 해당 패킷의 수명 정보를 저장 (패킷이 영원히 떠돌지 않도록)
-> 라우터를 하나 지날 때마다 TTL이 1씩 감소 -> TTL이 0이 되면 패킷을 처분
-- Protocol : 해당 패킷을 사용하는 상위 프로토콜 정보 저장 (SCTP, TCP, UDP, ...)
ex) 1 : ICMP, 2 : IGMP, 6 : TCP 등등
-- Header Checksum : Header 내부의 오류를 확인하기 위한 정보 저장
-- Source Address : 송신측의 IP Address 정보 저장
-- Destination Address : 수신측의 IP Address 정보 저장

* IPv4 Address Classes

IP address Classes

Class	# Network Bits	# Hosts Bits	Decimal Address Range	Subnet mask
Class A	8 bits	24 bits	1-126	255.0.0.0
Class B	16 bits	16 bits	128-191	255.255.0.0
Class C	24 bits	8 bits	192-223	255.255.255.0
Class D	Reserved for Multicasting		224-239	N/A
Class E	Reserved for R & D		240-255	N/A

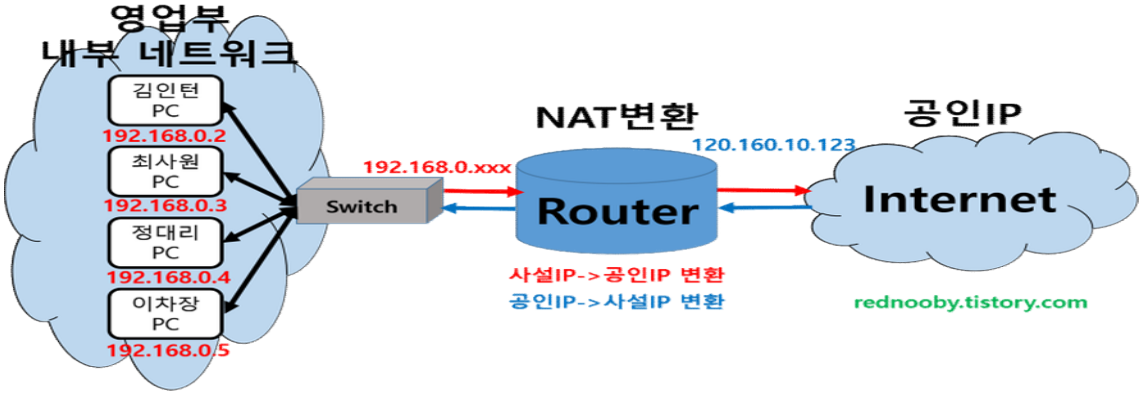
CLASS	LEADING BITS	NET ID BITS	HOST ID BITS	NO. OF NETWORKS	ADDRESSES PER NETWORK	START ADDRESS	END ADDRESS
CLASS A	0	8	24	2^7 (128)	2^{24} (16,777,216)	0.0.0.0	127.255.255.255
CLASS B	10	16	16	2^{14} (16,384)	2^{16} (65,536)	128.0.0.0	191.255.255.255
CLASS C	110	24	8	2^{21} (2,097,152)	2^8 (256)	192.0.0.0	223.255.255.255
CLASS D	1110	NOT DEFINED	NOT DEFINED	NOT DEFINED	NOT DEFINED	224.0.0.0	239.255.255.255
CLASS E	1111	NOT DEFINED	NOT DEFINED	NOT DEFINED	NOT DEFINED	240.0.0.0	255.255.255.255

위처럼 IPv4 Address에는 등급이 나누어져 있다. 각 등급에서는 Host IP Address를 사용할 수 있는 개수가 정해져있다. (결국 43억개의 주소도 다 사용하지 못함 -> 비효율성)
D class의 Address는 Multi-Cast용 주소이고, E class의 Address는 여유분이다.

* Subnet
: Network Traffic을 분산시키기 위해 전체 네트워크를 분할하여 사용하는 것
Ex) 165.246.30.xxx -> 인하대 정석 1층 네트워크
165.246.31.xxx -> 인하대 정석 2층 네트워크

* Subnet Mask
: Network ID와 Host ID를 분할시키는 IP Address
Ex) 내가 정석 1층에서 165.246.30.1의 ip주소를 할당받아 사용하고 있으면 이것을 Subnet Mask (255.255.255.0)과 &시키면 165.246.30.0이 나온다. 친구가 동일하게 165.246.30.3 ip주소를 할당받아 사용하고 있으면 이 또한 Subnet Mask에 &기키면 165.246.30.0이 나온다. 이 둘의 정보가 같으므로 둘은 같은 Subnet안에 있다는 것을 알 수 있다. 만약 Subnet Mask를 255.255.0.0으로 설정하면 서브넷 정보가 165.246.0.0이 되므로 인하대 내부 ip주소를 사용하는 모든 사람들과 같은 서브넷을 사용함을 알 수 있다.

* NAT(Network Address Translation)



: 주어진 하나의 공인 IP Address를 여러 개의 사설 IP Address로 쪼개서 네트워크를 이용하는 방법 (이 방법으로 인해 IPv4로도 IP Address가 부족함을 해소)
가장 대표적인 예는 집에서 사용하는 공유기이다. 집으로 들어오는 공인 ip address는 하나인데 이것을 컴퓨터, 노트북, 휴대폰 등 많은 기기에서 사용하는 것은 공유기에서 NAT기법을 사용하여 각 노드에 사설 IP를 할당해 주기 때문이다. 이 노드들(컴퓨터, 노트북 등)의 구분은 각각의 노드에 **포트번호**를 할당함으로써 이루어진다.

```
무선 LAN 어댑터 Wi-Fi:

   연결별 DNS 접미사. . . . . :
   링크-로컬 IPv6 주소 . . . . . : fe80::4452:4cc8:87c4:d590%17
   IPv4 주소 . . . . . . . . . . : 192.168.1.82
   서브넷 마스크 . . . . . . . . : 255.255.255.0
   기본 게이트웨이 . . . . . . . : 192.168.1.1
```

이것으로 보아, 현재 사용하는 노트북은 카페의 공유기로부터 사설ip를 할당받았음을 확인할 수 있다. (192.168.1.82) 이것을 해당 서브넷 마스크(255.255.255.0)과 &시키면 현재 이 카페에서 나와 같은 공유기를 사용하는 서브넷 정보를 알 수 있다.

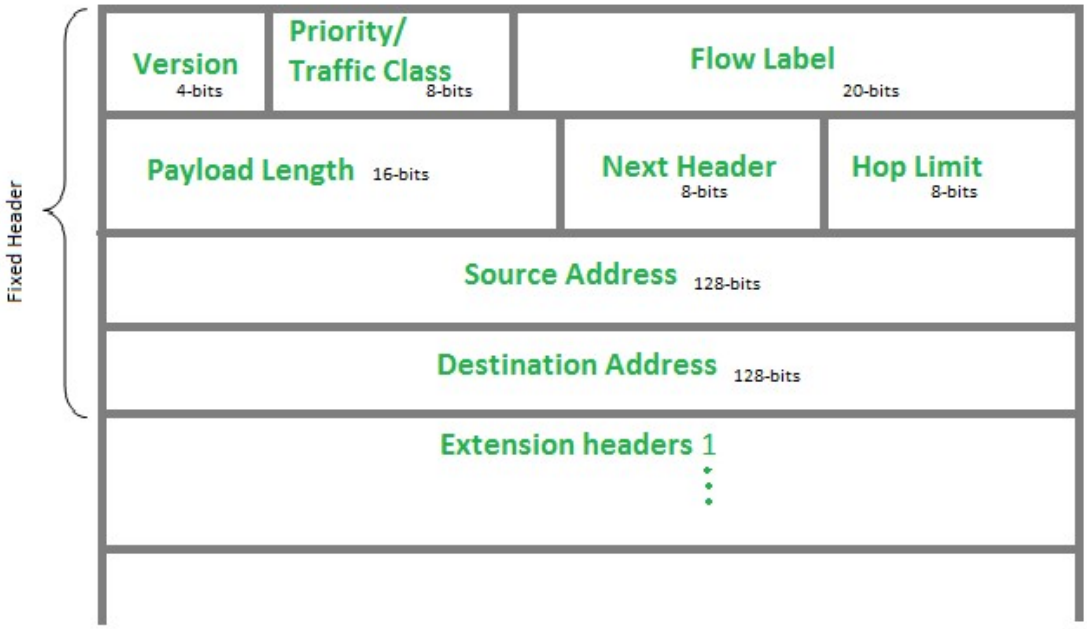
* DHCP(Dynamic Host Configuration Protocol)
: 네트워크 장치를 켤 때마다 DHCP Server로부터 자동으로 IP주소를 할당받는 기법
DHCP는 일종의 컴퓨터의 기능으로 활성화/비활성화가 가능하다. 비활성화 시키면 수동으로 자신이 사용할 IP Address를 지정해두어야 한다.

위 그림은 DHCP Server로부터 IP Address를 받아오는 과정을 설명한 것이다. 클라이언트가 IP주소를 서버에 요구하고, 서버는 가용한 주소를 클라이언트에게 제공한다. 그 주소가 마음에 들면 서버에 알겠다고 하고, 서버는 그 주소를 클라이언트에게 할당한다.

* IPv4와 IPv6의 차이점

구분	IPv4	IPv6
주소길이	32비트	128비트
표시방법	8비트씩 4부분으로 10진수로 표시 예) 202.30.64.22	16비트씩 8부분으로 16진수로 표시 예) 2001:0230:abcd:ffff:0000:0000:ffff:1111
주소개수	약 43억개	약 43억×43억×43억×43억개
주소할당	A, B, C 등 클래스 단위의 비순차적 할당	네트워크 규모 및 단말기 수에 따른 순차적 할당
품질제어	지원 수단 없음	등급별, 서비스별로 패킷을 구분할 수 있어 품질보장이 용이
보안기능	IPsec 프로토콜 별도 설치	확장기능에서 기본으로 제공
플러그 앤드 플레이	지원 수단 없음	지원 수단 있음
모바일IP	상단히 곤란	용이
웹개스팅	곤란	용이

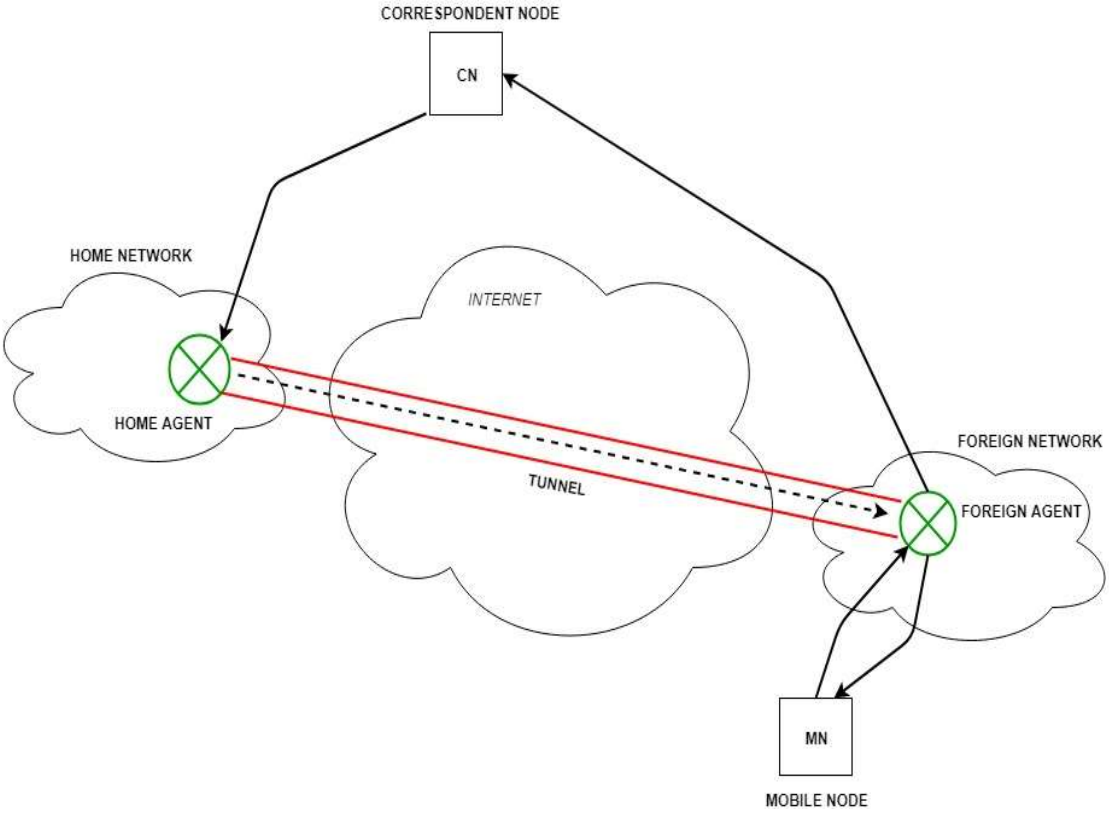
* IPv6 Header Packet Frame



Version : IP Address version 정보 저장
Traffic class : IPv4의 ToS(Type of Service) 기능 [DTRC]
Flow label : virtual circuit 경로 번호 저장 [아직 미구현 단계]
Payload length : Payload의 길이를 저장
Hop limit : IPv4의 TTL(Time To Live) 기능
Next header : 다음에 오는 확장헤더가 어떤 유형인지에 대한 정보 저장(확장헤더 사용여부)
해당 헤더가 마지막 헤더인 경우, IPv4의 Protocol 기능
Source Address : version 6의 송신지 정보
Destination Address : version 6의 수신지 정보

* 확장 헤더의 종류
홉 간 옵션 헤더, 소스 라우팅 헤더, 단편화 헤더, 인증 헤더 등등
=> 확장 헤더로 인해 IPv4에 비해 헤더가 단순화 되었고 인증 헤더 같은 경우, IPv4에서는 이것을 4계층의 IPsec을 사용하여야 하지만 IPv6는 3계층 확장 헤더에 넣을 수 있다.

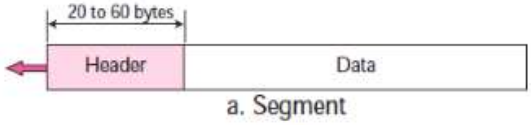
* 홈 주소와 관심 주소
: 호스트가 원래 주소와 함께 임시 주소를 하나 더 갖게 함으로써 이동간의 IP Address 유지에 기여한다.



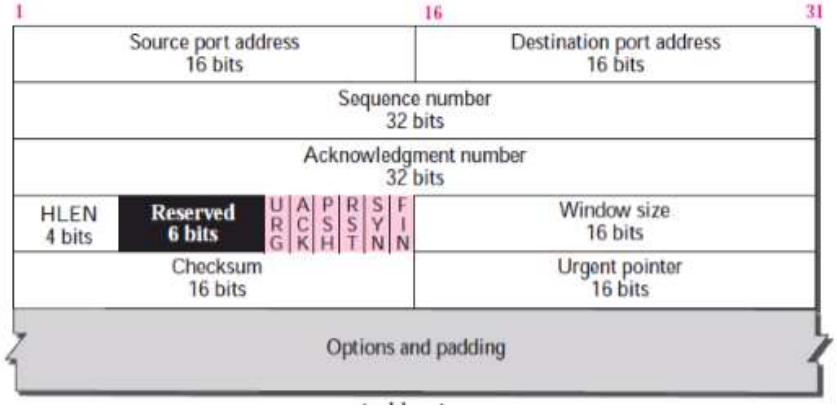
** Home Address : Mobile Node의 영구적인 주소, MN이 Home network에 있을 때 사용
** Care-of-Address : MN이 Home network를 벗어나 Foreign network에 들어갔을 때 FN의 DHCP Server로부터 부여받은 주소, MN이 Home network를 벗어났을 때 사용
** Routing method outside of Home network[21)]
1. MN이 데이터를 보내는 경우
: MN이 Foreign agent(router)를 통한 라우팅을 실시하여 목적지로 데이터를 전송, 이때 데이터 패킷의 Source Address는 Home Address가 된다.
2. MN이 데이터를 받는 경우
: Home Address를 Destination Address로 한 데이터가 Home Agent로 전송되면, Home Agent가 MN이 현재 접속중인 Foreign agent와 미리 연결해둔 Tunnel을 통해 받은 데이터를 전송해준다. 이를 받은 Foreign agent는 MN으로 그 데이터를 전송한다.

21) CN(Correspondent Node) : MN과 데이터를 주고 받는 임의의 상대방 노드

- TCP Layer (Transmission Control Protocol)
* 역할 : 신뢰성 있는 byte 단위(스트림) 전송 서비스
* TCP Segment Header Frame



a. Segment



-- Source port : 송신측에서 사용한 프로세스 정보 저장
-- Destination port : 수신측에서 사용할 프로세스 정보 저장
-- Sequence number : 보내는 측 seq(일련번호)
-- Acknowledgment number : 받은 데이터에 대한 ACK 일련번호[22]
-- HLEN(Header LENgth) : 해당 Header의 길이 정보 저장
-- Reserved : 미래 추가 기능을 위해 남겨놓은 예비 필드

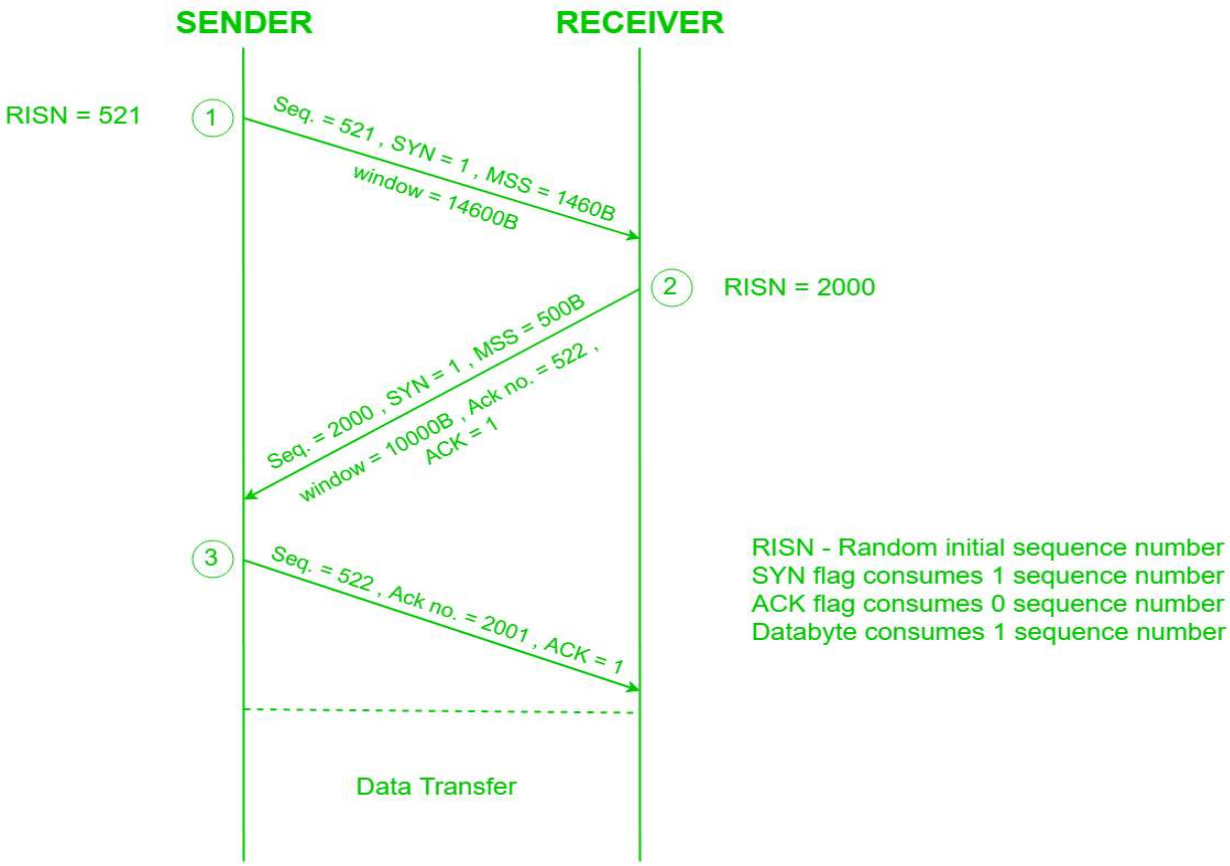
** Flags
-- URG(Urgent) : Urgent pointer의 값이 유효한 값인지를 설정
-- ACK : Acknowledgement number의 값이 유효한 값인지를 설정
-- PSH(Push) : 데이터를 윗 계층으로 보낼만큼 buffer가 덜 찼음에도 윗 계층으로 전송할지 여부를 설정 [1로 설정되면 적은 데이터도 바로 전송]
-- RST(Reset connection) : 비정상적으로 Connection을 release한다는 flag
-- SYN(Synchronization) : 초기 Connection을 construct한다는 flag
-- FIN(Finish connection) : 정상적으로 Connection을 release한다는 flag
-> 현재는 NS, CWR, ECE 세 개의 FLAG가 더 지정되어 있다.[23]

-- Window size : Sliding window algorithm에서 수신 window의 크기
-- Checksum : Header & Payload의 오류 검출 용도 데이터
-- Urgent pointer : URG flag가 설정된 경우, 그 긴급 메시지의 위치 정보를 저장

*** Segment
: TCP Layer에서 데이터 단위 유닛 지칭 단어 [≒ Frame(Layer2), Packet(Layer3)]

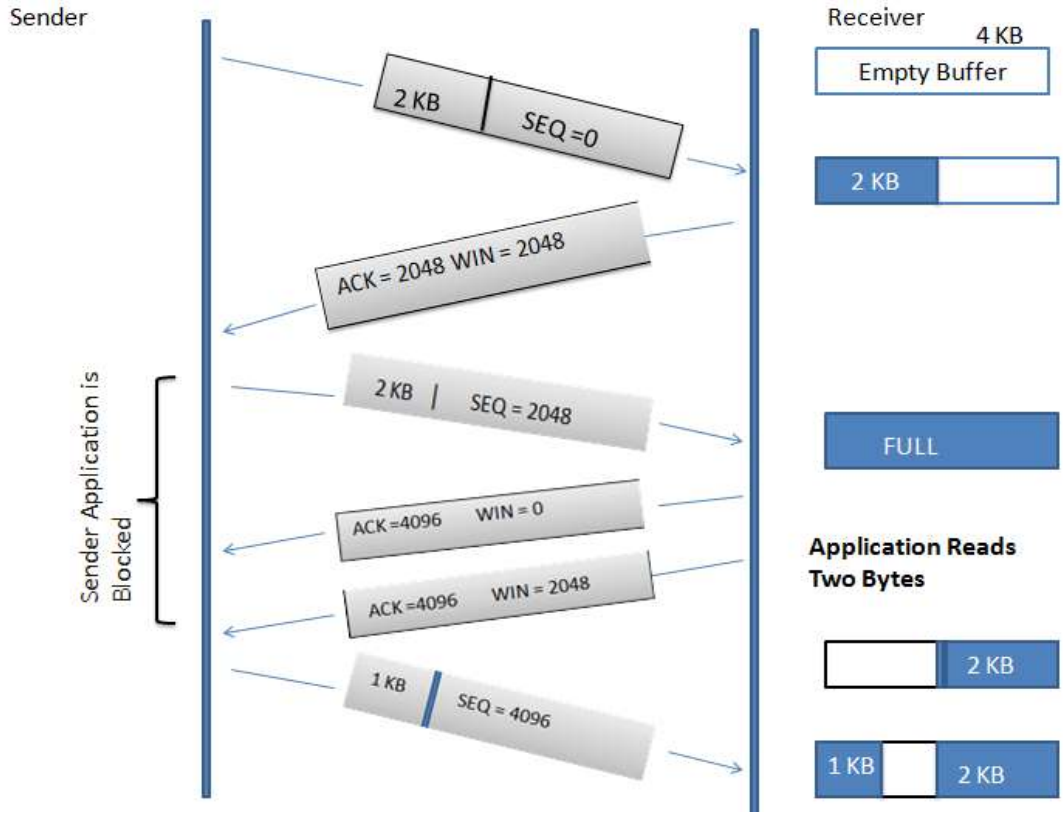
22) SEQ&ACK number에 대해서는 page 37, 53의 그림을 참조하면 이해하기 쉽다. [Flow Control]
23) [https://ko.wikipedia.org/wiki/전송_제어_프로토콜] 참조

* TCP Connection Establishment



TCP Connection Establishment에서는 Three-way Handshake 기법을 사용하는데, 이것의 동작 원리는 page 51에 기술되어있다. 그림에 나오는 MSS는 Maximum Segment Size의 약어로 현재 연결에서 송수신할 세그먼트의 최대 크기를 의미한다. MSS는 Sender와 Receiver의 Capacity 중 작은 값으로 규정한다. Connection release의 경우 또한 page 51에 기술되어있다. TCP Connection Structure에 대한 정보는 page 49를 참고 [State Diagram]

* TCP Flow Control with Buffer



page 53에 기술되어있는 Flow Control 원리는 이 그림의 이해에 용이할 것이다.

* Difference of TCP & UDP[24]

프로토콜 종류	TCP	UDP
연결 방식	연결형 서비스	비연결형 서비스
패킷 교환 방식	가상 회선 방식	데이터그램 방식
전송 순서	전송 순서 보장	전송 순서가 바뀔 수 있음
수신 여부 확인	수신 여부 확인	수신 여부 확인하지 않음
통신 방식	1:1 통신	1:1 or 1:N or N:N 통신
신뢰성	높다	낮다
속도	느리다	빠르다

* Determine of Window Size

: TCP Connection에서 window size는 수신측 window와 혼잡window[25]에 의해 결정한다.

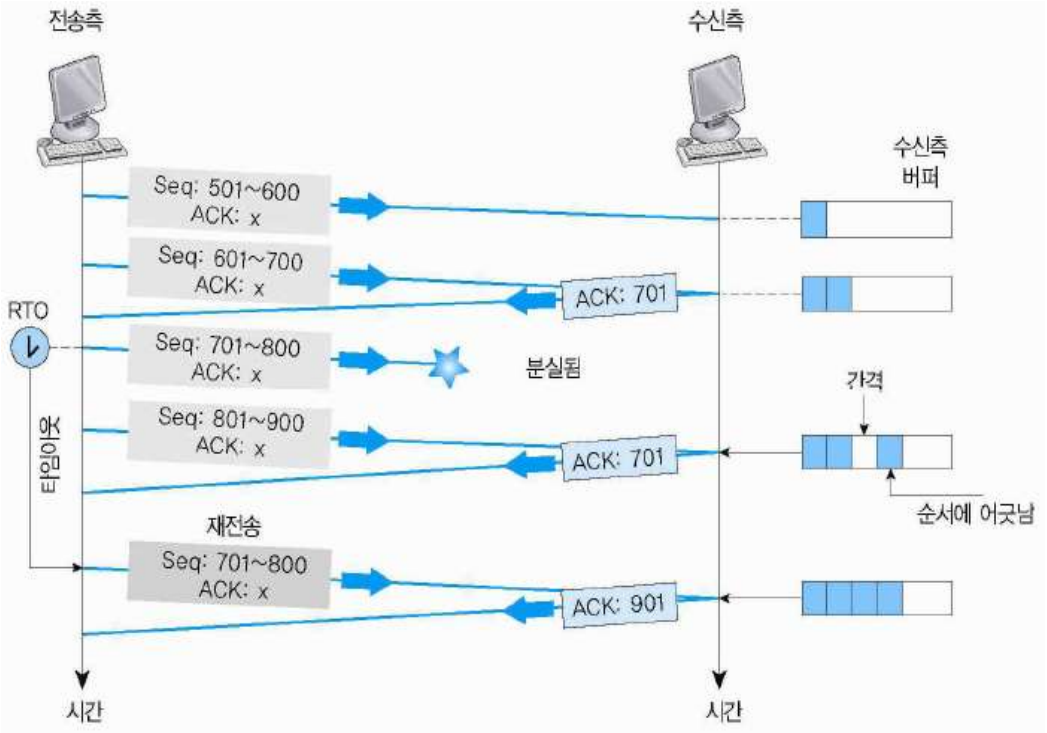
=> 연결에서 사용하는 $Window\,Size = \min(rwnd, cwnd)$ $[rwnd = receiver\,window]$

항상 $Last\,Byte\,Sent - Last\,Byte\,ACKed \leq \min(rwnd, cwnd)$를 만족해야 한다.

이때, Sender의 전송률은 $T.Rate \leq \frac{cwnd}{RTT}$[26]이다. [Sender만을 고려하기 때문에 cwnd]

* TCP Error Control

기본적으로 TimeOut-Retransmission 기법을 사용한다.



위 그림에 대해 간략히 설명하면, 중간에 701-800 데이터가 분실되었다. ACK:801[800까지 받았으니 801을 전송하라는 신호]이 도달하지 않은 상태에서 Sender는 801-900을 전송하고, 이것을 받은 Receiver는 Seq를 보고 701-800이 정상 전달되지 않음을 인지하고 바로 ACK:701[701을 보내라는 신호]를 전송한다. 이 신호를 받은 Sender는 즉시 701-800을 Retransmission하는 것이 아니라 처음 701-800을 전송했을 때부터 계산된 RTO[27]가 지나면, 그때 701-800을 재전송한다.

---

24) UDP(User Datagram Protocol) : Datagram방식을 사용하기 때문에 신뢰성이 낮고, 속도가 빠름.
25) cwnd(congestion window) : Congestion이 발생하지 않도록 네트워크에서 결정하는 window size
26) RTT(Round Trip Time) : Data unit의 왕복 시간
27) RTO(Retransmission TimeOut) : ACK가 돌아오지 않으면 재전송하기까지 대기시간

* RTO's Formula
RTT(Round Trip Time) : 데이터 유닛의 왕복 시간
SRTT(Smoothed RTT) : RTT들의 가중 평균치 [최근 데이터 가중치⇑]
RTTVAR(RTT variation) : RTT들의 가중 분산
RTO(Retransmission TimeOut) : ACK를 받지 못하면 재전송까지의 대기시간

여기에서 RTT를 확률변수 X로 생각하면, SRTT는 E(X), RTTVAR은 V(X)라고 생각하면 이해하기 편하다. 그러나 실제로 SRTT나 RTTVAR은 가중치이기 때문에 E(X)와 V(X)와는 다르다. 통상적으로 $RTO_0 = 1(\sec)$이다. (처음 RTO는 계산할 데이터가 없기 때문)

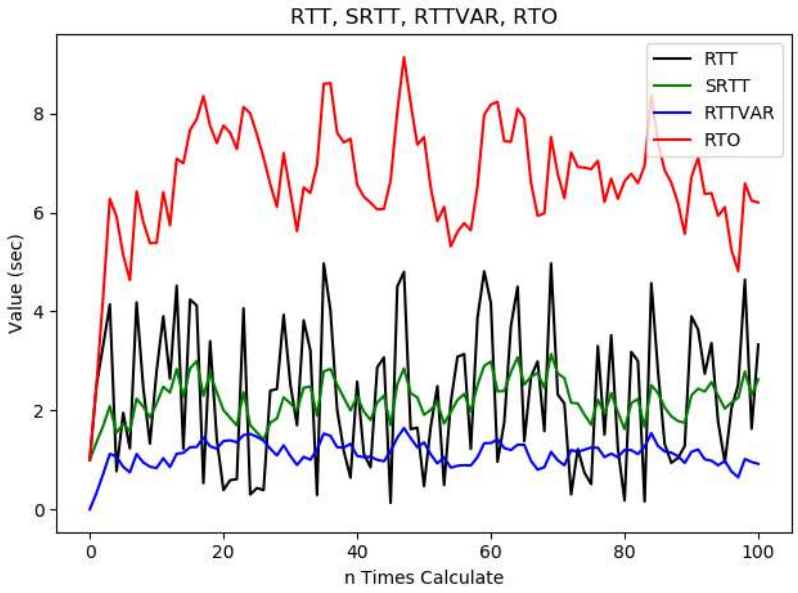
$$SRTT_n = (1-\alpha)SRTT_{n-1} + \alpha\, RTT_n$$

$$RTTVAR_n = (1-\beta)RTTVAR_{n-1} + \beta|RTT_n - SRTT_n|$$

$$RTO_n = SRTT_n + 4\, RTTVAR_n$$

-> 여기에서 사용되는 α, β는 관례적으로 각각 0.125, 0.25로 사용한다.

※ Random RTT를 통해 본 RTT, SRTT, RTTVAR, RTO Graph

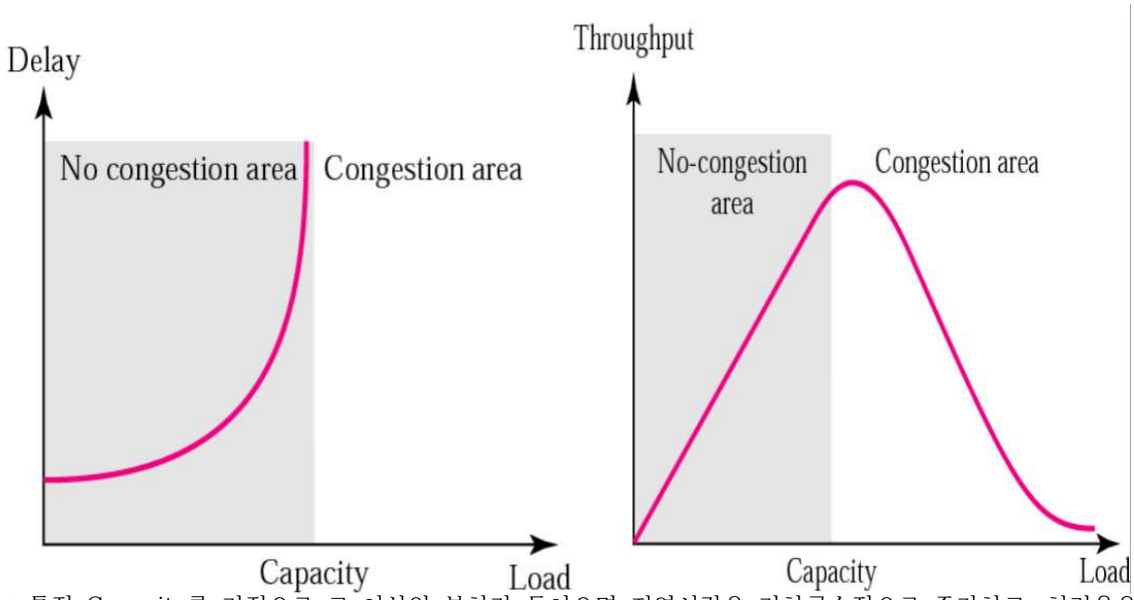



```
import random;
import matplotlib.pyplot as plt;

RTTs = [1]; SRTTs = [1];
RTOs = [1]; RTTVARs = [0];
SRTT = 1; RTO = 1; RTTVAR = 0;

def SRTT_calc(rtt, alpha):
global SRTT;
SRTT = (1-alpha) * SRTT + alpha * rtt;
SRTTs.append((1-alpha) * SRTT + alpha * rtt);
return SRTT;

def RTTVAR_calc(rtt, srtt, beta):
global RTTVAR;
RTTVAR = (1-beta) * RTTVAR + beta * abs(rtt-srtt);
RTTVARs.append(RTTVAR);
return RTTVAR;

def RTO_calc(srtt, rttvar):
global RTO;
RTO = srtt + 4 * rttvar;
if RTO < 1:
RTO = 1;
RTOs.append(RTO);
return RTO;

for i in range(100):
RandomRTT = random.randint(0, 500) / 100;
RTTs.append(RandomRTT);
srtt = SRTT_calc(RandomRTT, 0.125)
RTO_calc(srtt, RTTVAR_calc(RandomRTT, srtt, 0.25));

print(RTTs);
print(SRTTs);
print(RTTVARs);
print(RTOs);

x_range = range(101);
plt.plot(x_range, RTTs, color="black", label="RTT");
plt.plot(x_range, SRTTs, color="green", label="SRTT");
plt.plot(x_range, RTTVARs, color="blue", label="RTTVAR");
plt.plot(x_range, RTOs, color="red", label="RTO");
plt.title("RTT, SRTT, RTTVAR, RTO");
plt.xlabel("n Times Calculate");
plt.ylabel("Value (sec)");
plt.legend();
plt.show();
```

- > python에서 작동시키면 계속 달라지는 그래프 개형을 확인할 수 있다.

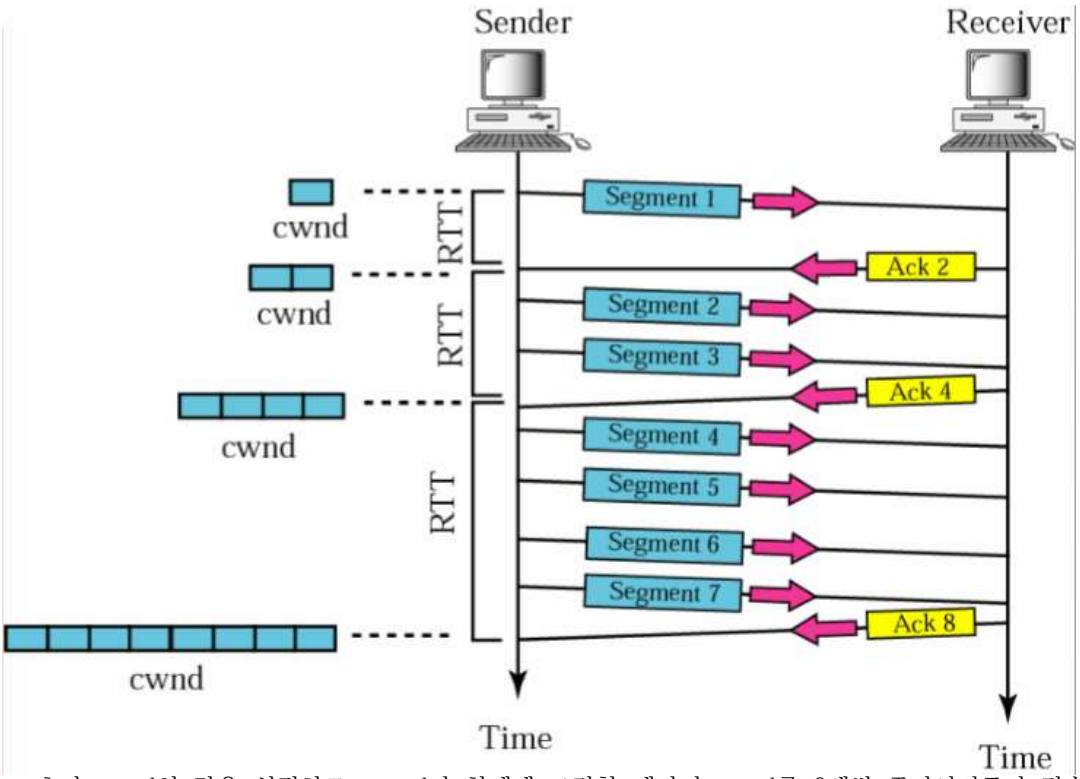
* Congestion Control on TCP Layer

** Congestion Area(혼잡 영역)



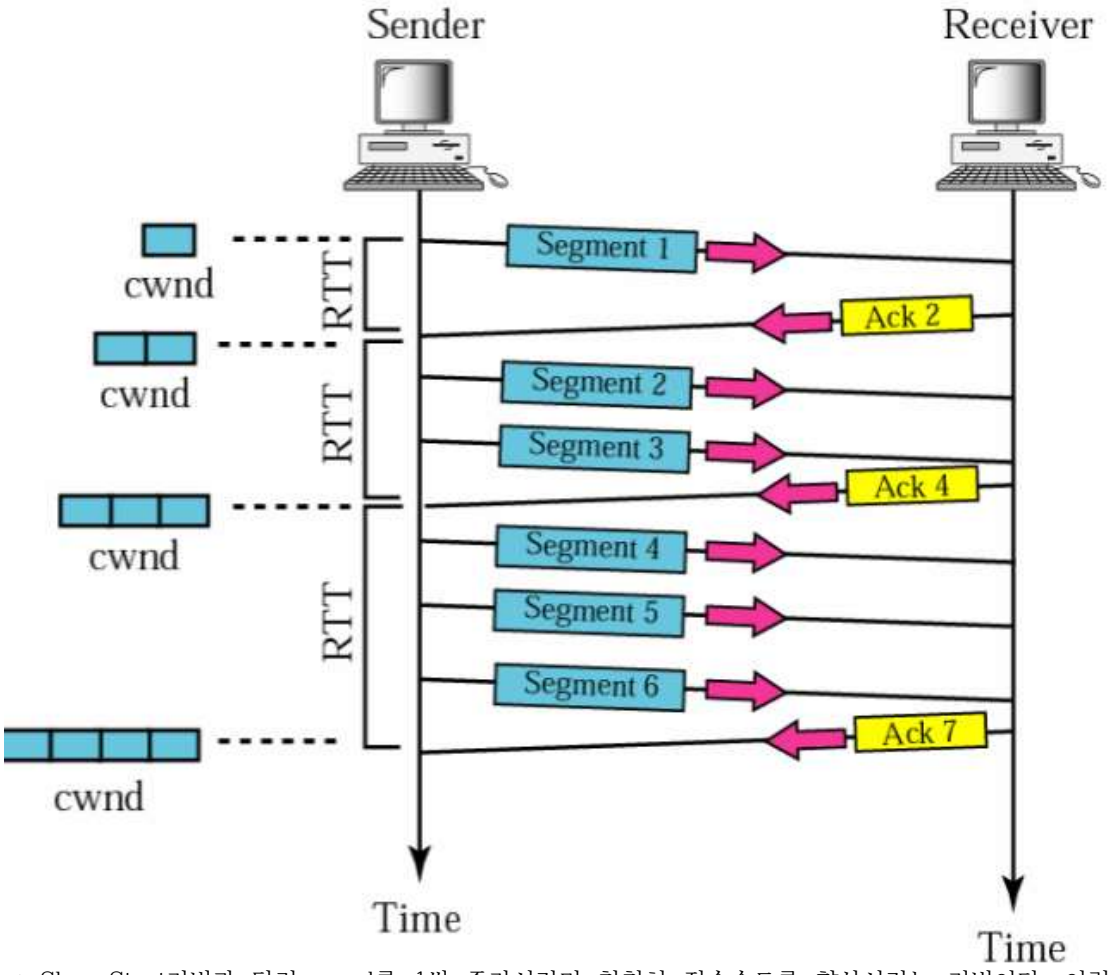
: 특정 Capacity를 기점으로 그 이상의 부하가 들어오면 지연시간은 기하급수적으로 증가하고, 처리율은 급격히 떨어진다.

** Slow Start Algorithm



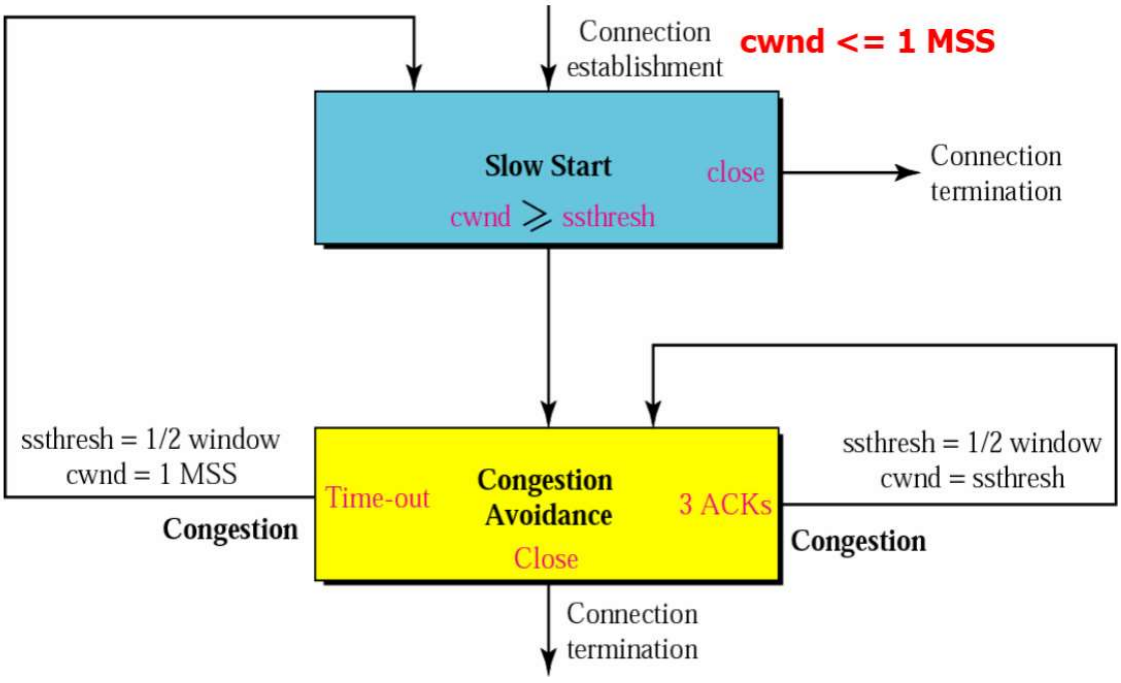
: 초기 cwnd의 값을 설정하고, cwnd가 한계에 도달할 때까지 cwnd를 2배씩 증가시켜주며 전송 속도를 향상시켜주는 기법이다. cwnd가 ssthrsh에 도달하면 Congestion avoidance기법으로 전환, TimeOut이 발생하면 다시 cwnd를 초기값으로 내리고 다시 Slow Start기법을 실시 (이때, ssthrsh의 값을 재설정)

** Congestion Avoidance Algorithm

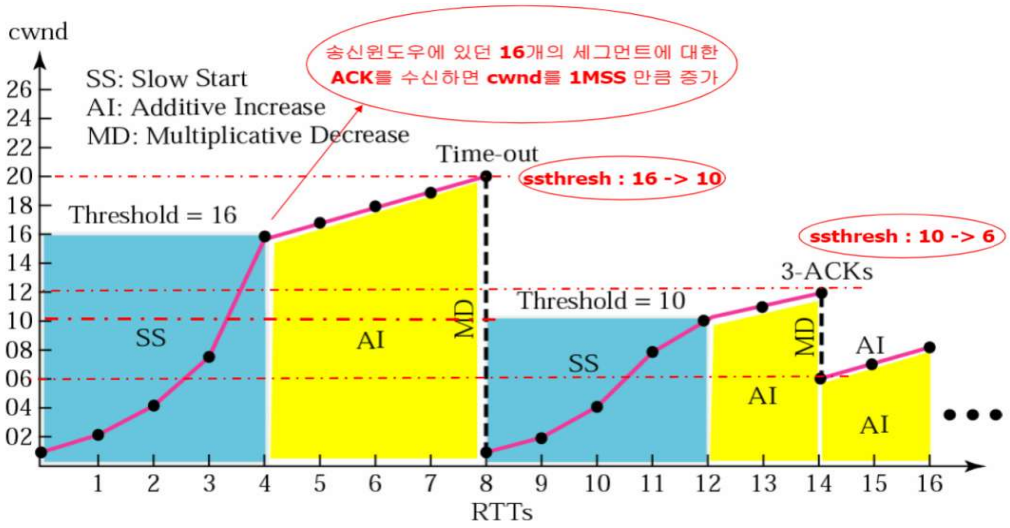


: Slow Start기법과 달리 cwnd를 1씩 증가시키며 천천히 전송속도를 향상시키는 기법이다. 이렇게 cwnd를 1씩 증가시키다가 TimeOut이 발생하거나 3번의 ACK가 중복수신되는 경우, Congestion이 발생한 것으로 판단한다. TimeOut이 발생하면 Slow Start기법으로, ACK Duplicate가 발생하면 Congestion Avoidance기법으로 들어간다.

** Congestion Control State Diagram



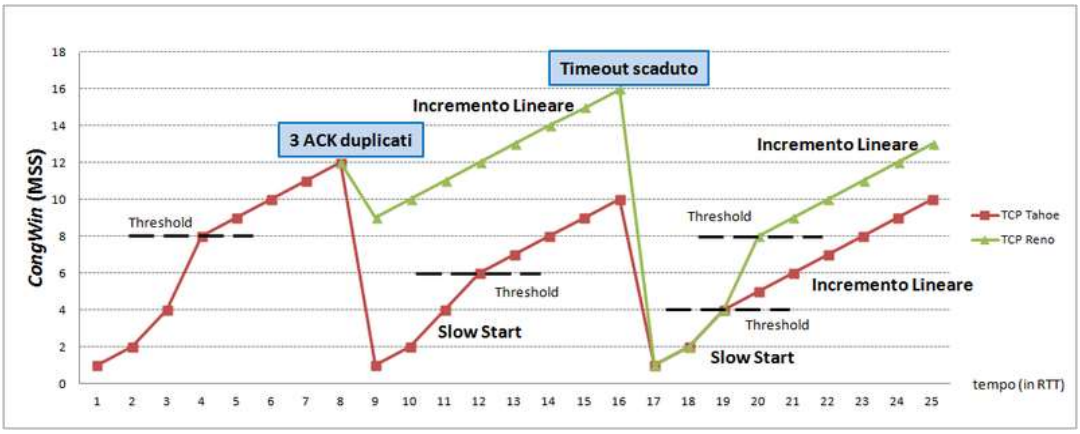
** Congestion Control Example Plot



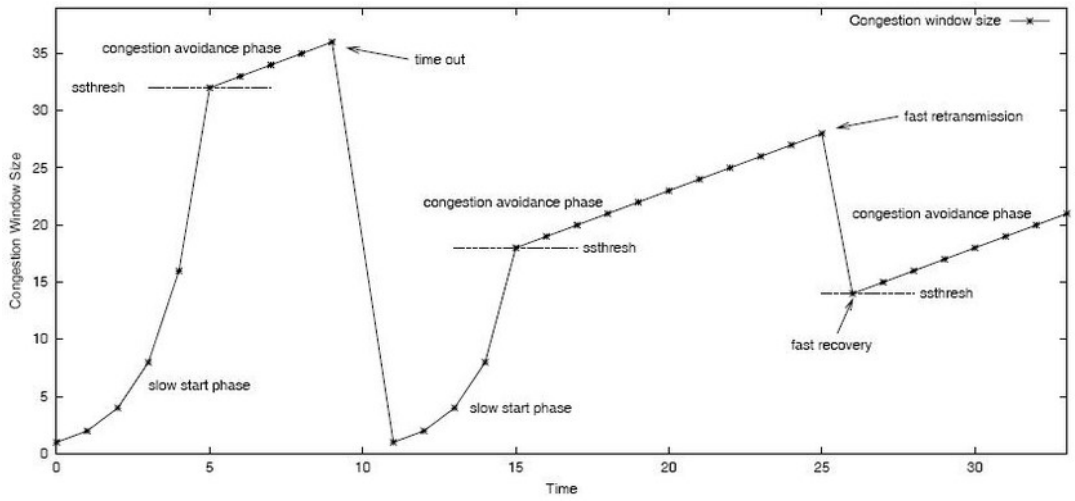
: 위 예시에 대해 간단히 설명하면, init threshold=16, 처음에는 Slow Start기법을 사용하여 cwnd를 증가시킨다. 이 cwnd가 threshold에 도달하면 Congestion Avoidance기법으로 전환, cwnd를 1씩 증가시킨다. 그러다 Timeout이 발생하면 Threshold를 Timeout발생 시점의 cwnd의 반으로 재설정하고, Slow Start기법을 다시 실시한다. 또 cwnd가 threshold에 도달하면 Congestion Avoidance기법으로 전환하고, **3-ACKs가 발생하면 threshold를 3-ACKs발생 시점의 cwnd의 반으로 재설정하고 Congestion Avoidance를 실시하는데, 이때 시작 cwnd는 재설정된 threshold값이다.** 위에 쓰여진 MD는 threshold값을 재설정하는 것을 의미한다. (그림에서 Additive Increase는 Congestion Avoidance기법을 의미한다.)

** Fast Retransmit & Recovery (FR)

: 위 예시에서 빨간 글씨로 표기된 것은 실제로 Fast Retransmit&Recovery가 적용된 것이다. 이것이 적용되지 않으면 3-ACKs가 발생하여도 다시 처음 Slow Start기법으로 넘어가는데, 이렇게 되면 시간이 낭비되기 때문에 이러한 기법을 적용시킨 것이다.



-> TCP Tahoe는 FR 미적용, TCP Reno는 FR 적용



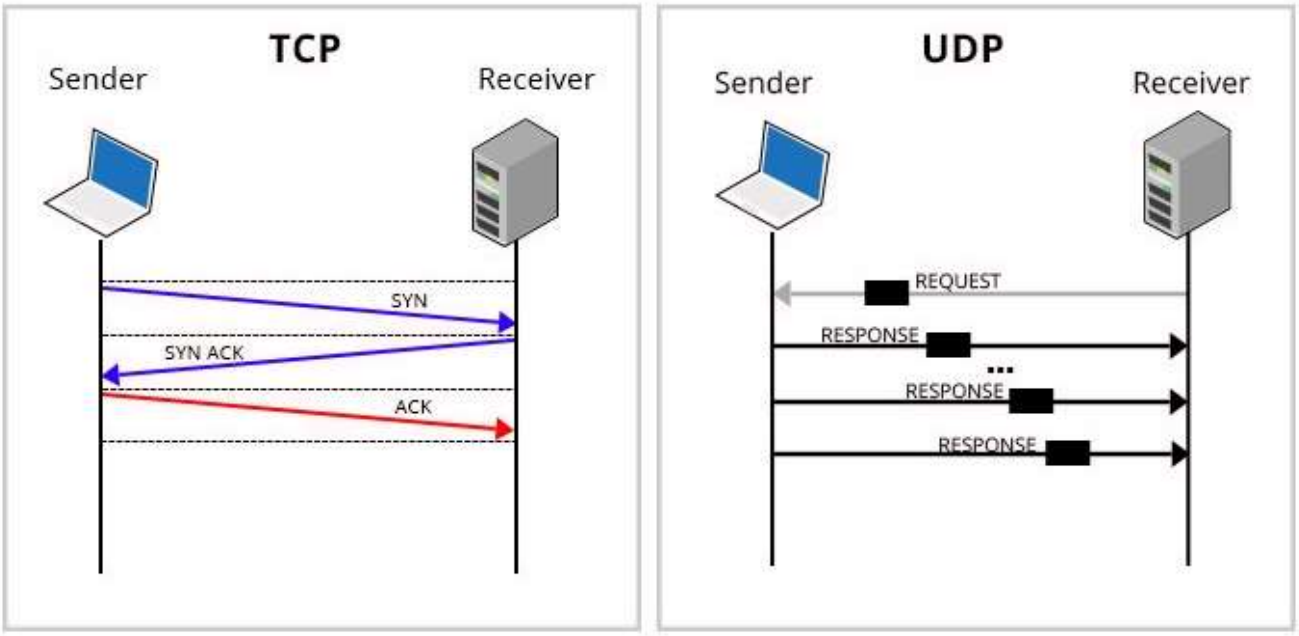
=> 위 예시가 완전히 이해되면 TCP Congestion Control이 어느정도 이해된 것

- UDP(User Datagram Protocol)

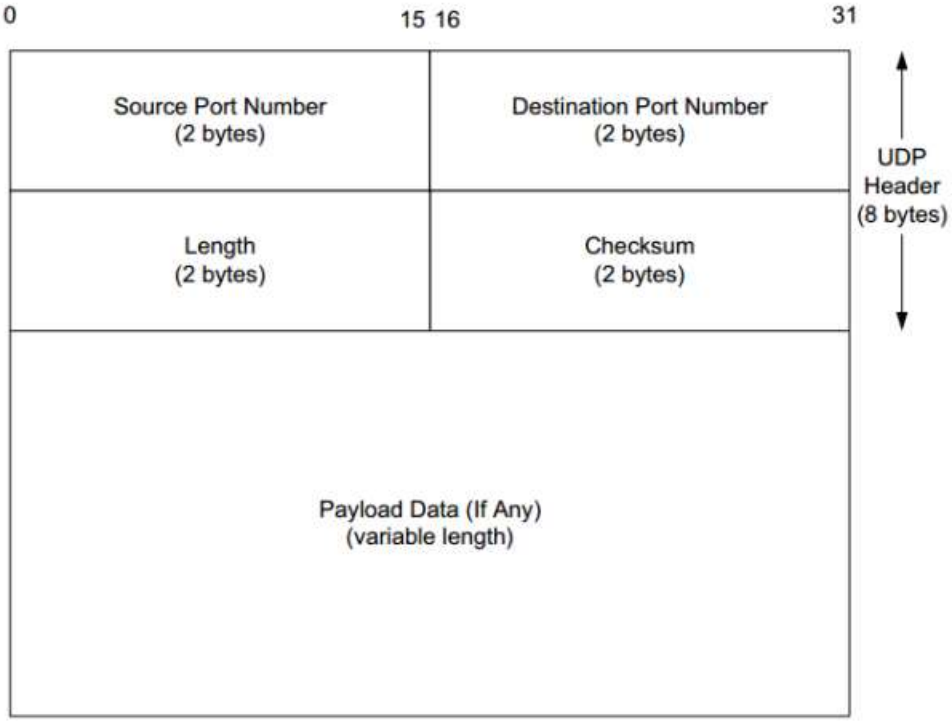
* UDP의 특성

1. 비연결형, 비신뢰성 전송 프로토콜
2. Receiver가 데이터를 제대로 받았는지 확인하지 않고 Sender가 단방향으로 전송
3. 확인 절차가 생략되기 때문에 TCP에 비해 데이터 전송속도가 빠르다.
4. 신뢰성보다는 연속성이 중요한 실시간 스트리밍에 주로 이용된다.(Youtube 등)

* Difference of TCP & UDP



* UDP Header



-- Source Port Number : 전송측에서 사용한 프로세스의 포트번호 정보 저장

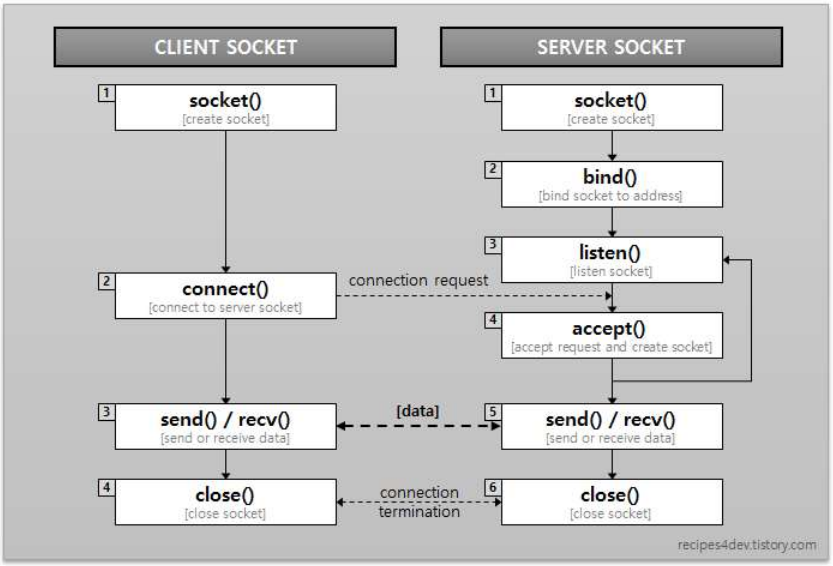
-- Destination Port Number : 수신측에서 사용한 프로세스의 포트번호 정보 저장

-- Length : Header + Payload의 길이 정보 저장 (minimum 8byte)

-- Checksum : 오류 검출을 위한 정보 저장

- Socket Programming

: Network Socket이란 Host Computer와 Internet 사이를 연결해주는 연결부와 같은 느낌이다. Socket Programming이란 이러한 소켓(연결부)를 다양한 프로그래밍 언어로 구현하는 것이다.




```
public class SocketExample {

    public static void main(String[] args){

        ClientSocket cs = new ClientSocket();
        ServerSocket ss = new ServerSocket();

        ss.socket("172.42.58.1", 0);
        ss.bind(80);
        ss.listen();
        cs.socket("192.168.50.15", 0);
        cs.connect(ss.getAF_INET());
        ss.accept();
        cs.send("Hello Server");
        ss.receive();
        ss.send();
        cs.receive();
        ss.close();
        cs.close();
    }
}
```

-> Java로 구현한 simple Socket Programming [작동부분과 실행결과만 첨부][28]

내부 동작 클래스와 연결을 위한 정적 클래스를 포함한 자료는 아래를 참조 ↴

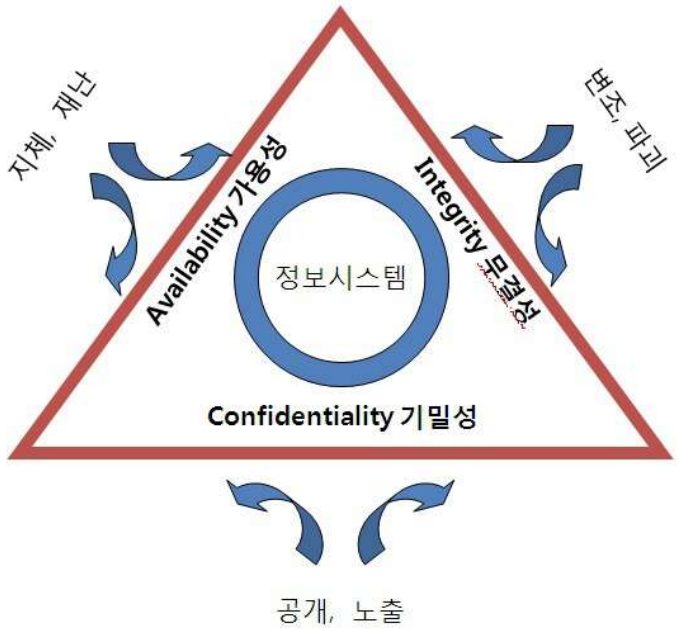
https://github.com/UnprettyCoder/SimpleSocketProgramming

28) 실제로 Socket Programming은 이런 조잡한 코드로 이루어지지 않는다. 구글링을 해보면 실제 소켓 프로그래밍이 어떤 로직으로 작동하는지 상세히 기술되어있다. 위 코드는 소켓이 어떻게 작동되는지 과정만 참고하면 될 것이다.

```
<Console window>

SERVER socket() function operate
Server's IP : 172.42.58.1
------------------------------

SERVER bing() function operate
Server's AF_INET : 172.42.58.1 : 80
------------------------------

SERVER listen() function operate
------------------------------

CLIENT socket() function operate
Client's IP : 192.168.50.15
Socket protocol : SOCKET_STREAM
------------------------------

CLIENT connect() function operate
Connect SERVER AF_INET : 172.42.58.1 : 80
Protocol type : SOCKET_STREAM
------------------------------

SERVER accept() function operate
Connected Client's IP : 192.168.50.15
------------------------------

CLIENT send() function operate
------------------------------

SERVER receive() function operate
Accepting Message : Hello Server
------------------------------

SERVER send() function operate
ACK to 192.168.50.15
------------------------------

CLIENT receive() function operate
Transmission operated normally
Connection is not released.[TCP]
------------------------------

SERVER close() function operate
Connection released.(Close)
------------------------------

CLIENT close() function operate
Connection released.(Close)
------------------------------
```

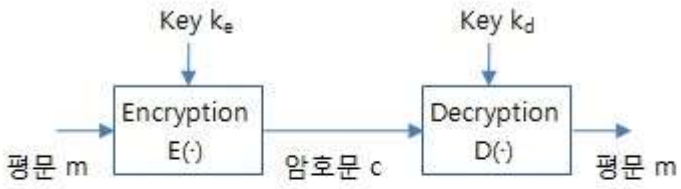
- Network Security (네트워크 보안)

* Network Security의 3대 요소



1. 기밀성(Confidentiality)
 정보가 비인가자에게 노출되지 않도록 보호하는 것
 [해커가 누군가의 E-mail을 중간에 훔쳐본다. -> 기밀성 위반]
2. 무결성(Integrity)
 정보가 중간에 위/변조 되지 않도록 보호하는 것
 [해커가 그 E-mail의 내용을 중간에 바꿔서 전송한다. -> 무결성 위반]
3. 가용성(Availability)
 인가된 사용자가 정보를 제대로 이용할 수 있도록 하는 것
 [해커가 특정 사이트에 엄청난 양의 패킷을 전송하여 사이트를 마비시킨다.(DDos)
  -> 가용성 위반(사이트가 제대로 기능하지 못하므로)]

* Cryptography (암호학)
 : 정보를 인가자 외 사람이 볼 수 없도록 암호화시키는 것



암호화 : $E_{k_e}(m) = c$

복호화 : $D_{k_d}(c) = m$

(m : 평문, c : 암호문, E : 암호알고리즘,
 k_e: 암호키, k_d: 복호키)

** 암호화 방식 (대칭키 vs 비대칭키)

대칭키 VS 비대칭키

구분	대칭키 암호 방식	공개키 암호 방식
역사	BC 500년 경	1976년
키	대칭키(비밀키)	비대칭키(공개키,사설키)
키의 상호 관계	암호화키=복호화키	암호화키≠복호화키
암호화키/복호화키	비밀/비밀	공개/비밀
암호 알고리즘	공개	공개
키의 개수	n*(n-1)/2	2*n
장점	*계산속도 빠름(1000배:DES/RSA) 알고리즘이 다양*	*암호화키 사전 공유 불필요 통신 대상의 추가가 용이 인증 기능 제공*
단점	*키 분배 및 관리의 어려움 기밀성만 보장*	*계산속도 느림*
대표적인 예	***DES, AES, IDEA***	***DH, RSA, ECC***

: 암호화 방식은 크게 대칭, 비대칭 방식으로 2가지로 분류할 수 있다. 대칭 방식과 비대칭 방식은 위와 같은 특성을 가지고 있다.

** 대체 암호화
: 특정 규칙에 따라 임의의 문자를 다른 문자로 대체하는 암호화 기법

*** 시저 암호화(Ceasar's Cipher) [대체 암호화]

```
Plain:
    a b c d e f g h i j k l m n o p q r s t u v w x y z
Cipher:
    D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
```

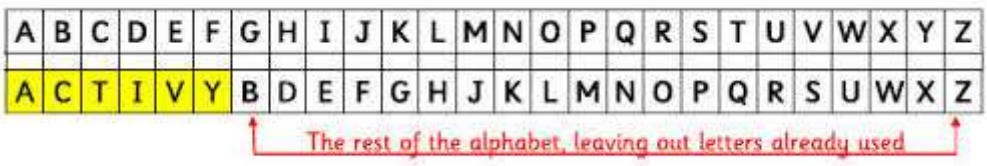
: 시저 암호화에서는 단순히 plain char를 Key만큼 뒷칸으로 밀어서 서로 매핑[29]시킨다. 그리고 원문의 char들을 암호문의 char로 바꾸어서 전송한다. 복호화 과정에서는 Key만큼 앞으로 밀어서 매핑시킨 복호문의 char들을 읽으면 된다. 장점은 단순하다는 것이고, 단점은 단순한 만큼 쉽게 해독 가능하다는 것이다.

```
String message = "Hello Ceaser";
CeaserCipher cc = new CeaserCipher();
String EncryptedString = cc.encryption(3, message);
String DecryptedString = cc.decryption(3, EncryptedString);

EncryptedString : "Khoor#Fhdvhu"
DecryptedString : "Hello Ceaser"
```

-> 시저 암호화를 이용한 암호화 복호화 시뮬레이션 (Key = 3)

*** 키워드 암호화 [대체 암호화]

29) Mapping : 매핑이란, 1:1 대응으로 key:value를 묶는 것을 의미한다. [예) Python의 Dict구조]



: 위의 예시는 키워드가 "ACTIVY"인 경우이다. 키워드 암호화는 시저 암호화에 비해 해독이 조금은 어렵다는 장점이 있다. 그러나 여기에도 여전히 1개의 plain char는 1개의 cipher char에 대응된다는 취약점이 있다. (즉, 위의 예에서 D는 항상 I이다.)

```
String newMessage = "helloceaser";
KeywordCipher kc = new KeywordCipher();
kc.makeMap("network");
String KCES = kc.encryption(newMessage);
String KCDS = kc.decryption(KCES);

EncryptedString : "aoffitonpom"
DecryptedString : "helloceaser"
```

-> 키워드 암호화를 이용한 암호화 복호화 시뮬레이션 (Key = "network")

*** 복수 문자 변환 암호화 [대체 암호화]

: 키워드 암호화에서도 취약했던 1:1 대응의 문제를 해결하기 위해 고안된 방법이다. 말 그대로 2개 이상의 문자 변환표를 사용해 n번째 문자를 변환할 때 $\mathrm{mod}_m(n)$[30] = 0, 1, 2, 3, ...일 때마다 다른 변환표의 값을 이용한다.


```
String thdMessage = "hellodoublemap";
DoubleMapCipher dmc = new DoubleMapCipher();
dmc.makeFirstMap("network");
dmc.makeSecondMap(3);
String DMCES = dmc.encryption(thdMessage);
String DMCDS = dmc.decryption(DMCES);

EncryptedString : "ahfoigixeoopns"
DecryptedString : "hellodoublemap"
```

-> DoubleMap 암호화를 이용한 암호화 복호화 시뮬레이션 (Key=3, Keyword="network")

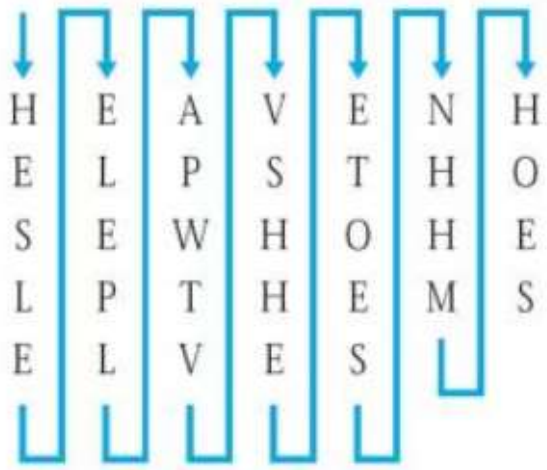
30) $\mathrm{mod}_m(n)$: Modulo 연산으로, n을 m으로 나눈 나머지를 의미한다.

** 위치 암호화
: 위에서 본 대체 암호화와 달리 문자를 다른 문자로 대체하지 않고, 문장의 나열 순서를 변화시켜 전송하는 암호화 기법

*** Column Cipher (컬럼 암호화)
: 전체 문장을 지정한 열의 길이만큼씩 잘라서 matrix 형태로 나열하고, 그 matrix를 열 순서대로 전송하는 암호화 기법이다. 쉽게 말해서 column의 길이를 fix시키고 row-방향으로 문장을 나열한다. 그렇게 만들어진 행렬을 column-방향으로 읽어서 보낸다.

예: 컬럼의 길이가 7 인 경우

- 원문서: HEAVEN HELPS THOSE WHO HELP THEMSELVES
- 암호문1: hesle elepl apwtv vshhe etoes nhhm hoes



※ 강의 노트에 적혀있는 암호문2는 matrix의 공백을 “z”로 채워준 케이스이다.

```
String frtMessage = "heaven helps those who help themselves";
ColumnCipher coc = new ColumnCipher(7);
String COCES = coc.encryption(frtMessage);
String COCDS = coc.decryption(COCES);

EncryptedString : “hhhotveeo healshesvpeemzes lszn wpez th lz”
DecryptedString : “heaven helps those who help themselves”
```

-> Column Cipher를 이용한 암호화 복호화 시뮬레이션 (Key=7)

*** Keyword & Column Cipher
: Column Cipher는 비교적 해법이 단순하기 때문에 이것을 조금 더 복잡하게 만들 필요가 있었다. 그래서 이번에는 컬럼 암호화에서 만든 matrix에서 단순히 열 순서대로 전송하지 않고 Keyword를 지정하여 그 Keyword의 알파벳 순서대로 전송한다. 이것을 위치 암호화에서의 키워드 암호화이다.

- 예: NETWORK

(a) 원문서: HEAVEN HELPS THOSE WHO HELP THEMSELVES

키워드	N	E	T	W	O	R	K
순서	3	1	6	7	4	5	2

(b) 암호화 과정

H	E	A	V	E	N	H
E	L	P	S	T	H	O
S	E	W	H	O	H	E
L	P	T	H	E	M	S
E	L	V	E	S	Z	Z

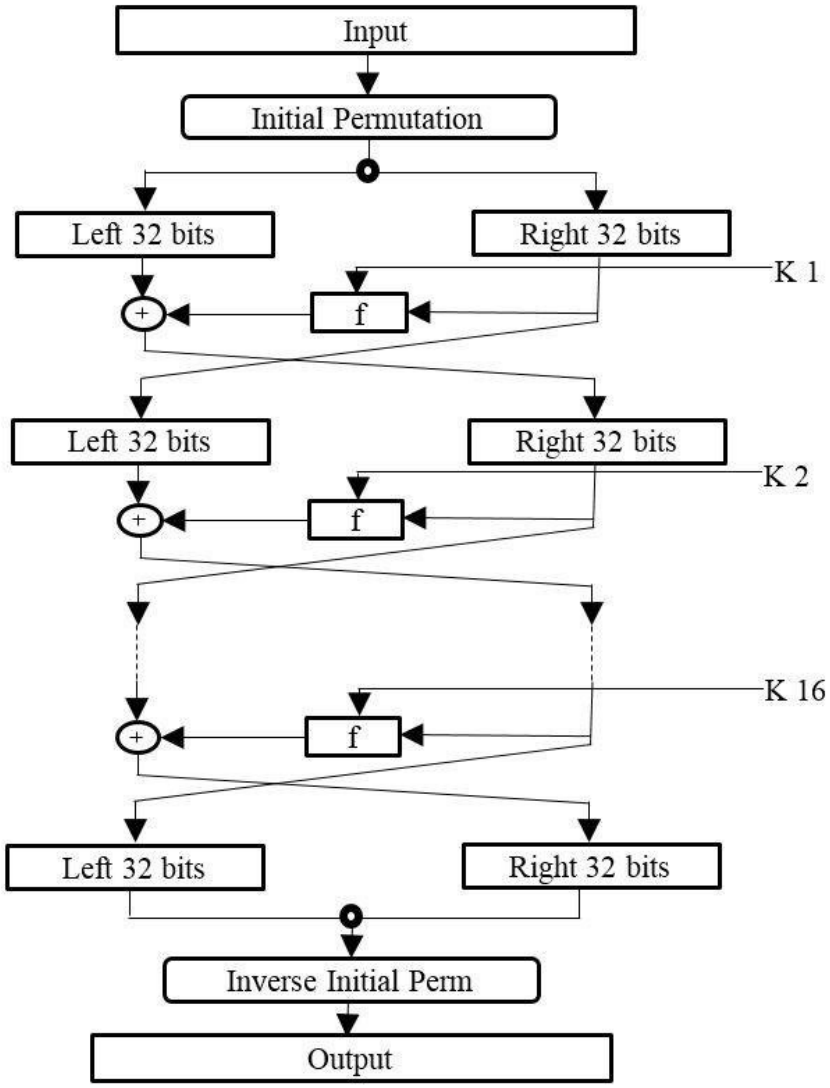
(c) 암호문: elepl hoesz hesle etoes nhhmz apwtv vshhe

```
String fthMessage = "heaven helps those who help themselves";
ColumnKeywordCipher ckc = new ColumnKeywordCipher();
ckc.setKeywordAndColumnLength("network", 7);
String CKCES = ckc.encryption(fthMessage);
String CKCDS = ckc.decryption(CKCES);

EncryptedString : “eeo he th lzhhhotves lszn wpezalshesvpeemz”
DecryptedString : “heaven helps those who help themselves”
```

-> Keyword&Column Cipher를 이용한 암호화 복호화 시뮬레이션 (Keyword=“network”, Key=7)

※ Details of Simple Cipher Algorithm Implementation in Java ↴
https://github.com/UnprettyCoder/Cipher-Algorithm

* DES Algorithm [Data Encryption Standard]
: DES 알고리즘은 2001년(AES로 대체되기 전)까지 높은 보안성을 인정받았던 암호화 기법이다. 이 암호화 기법은 **대칭키**를 사용하며, 암호키의 크기는 56bits[31] 이다.

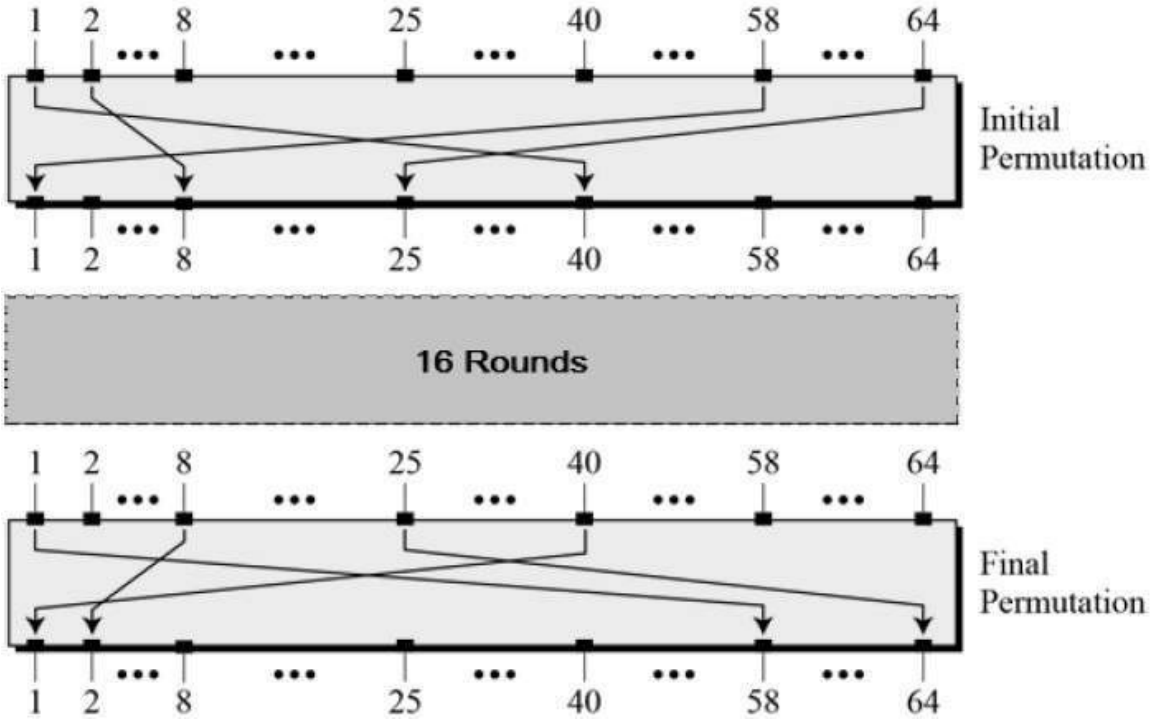


위 그림이 DES algorithm의 기본 로직이다. +로 표기된 곳에서는 두 32bits들의 XOR연산이 시행된다. 64bits의 Plain Text를 DES에 넣으면, Initial permutation[32]으로 데이터를 재배열한다. 또 이것을 반으로 나누어 Left1, Right1을 구분, Right1을 Key1으로 암호화한 후 Left1과 XOR연산[33]을 시행한다. 그것을 Right2로 넣고, Left2에는 Right1을 그대로 넣는다. 이렇게 Key2, 3, 4,

31) main 암호키의 크기는 56bits이고 여기서 48bits 크기의 subKey들을 16개 생성한다.
32) Permutation : 원래는 순열이라는 의미를 가지지만 여기서는 특정한 규칙을 가지고 데이터를 섞는 것을 의미

..., 16에 대해서 암호화를 반복한다. Key16까지 순환되면 64bits 데이터로 다시 합치고 Initial permutation의 역으로 다시 재배열 한다. 이렇게 해서 나온 64bits 데이터가 DES Cipher Text이다.

** DES Permutation 구조



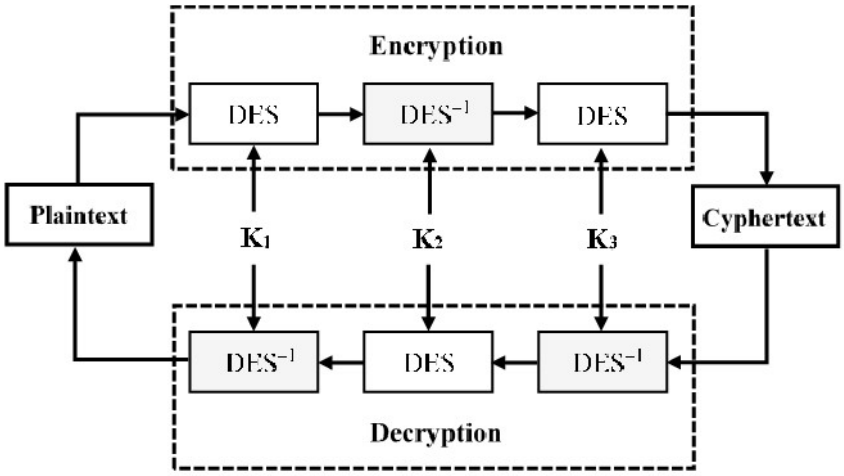
그림에서 보이는 Final Permutation은 IP^{-1}(Inverse Initial permutation)과 동일하다. Permutation의 순서는 키 정보에 있어야한다. 즉 DES 암호화에서는 처음 64bits Data를 이러한 방식으로 재배열 한다는 것이다. 아래 그림은 위 구조에 대한 Permutation Table이다.

Initial Permutation	*Final Permutation*
58 50 42 34 26 18 10 02	40 08 48 16 56 24 64 32
60 52 44 36 28 20 12 04	39 07 47 15 55 23 63 31
62 54 46 38 30 22 14 06	38 06 46 14 54 22 62 30
64 56 48 40 32 24 16 08	37 05 45 13 53 21 61 29
57 49 41 33 25 17 09 01	36 04 44 12 52 20 60 28
59 51 43 35 27 19 11 03	35 03 43 11 51 19 59 27
61 53 45 37 29 21 13 05	34 02 42 10 50 18 58 26
63 55 47 39 31 23 15 07	33 01 41 09 49 17 57 25

33) XOR 연산 : 둘 중 하나만 true, 하나는 false인 경우 true를 return하는 연산
ex) XOR(0011, 0101) = 0110 [통상적으로 0 : false, 1 : true]

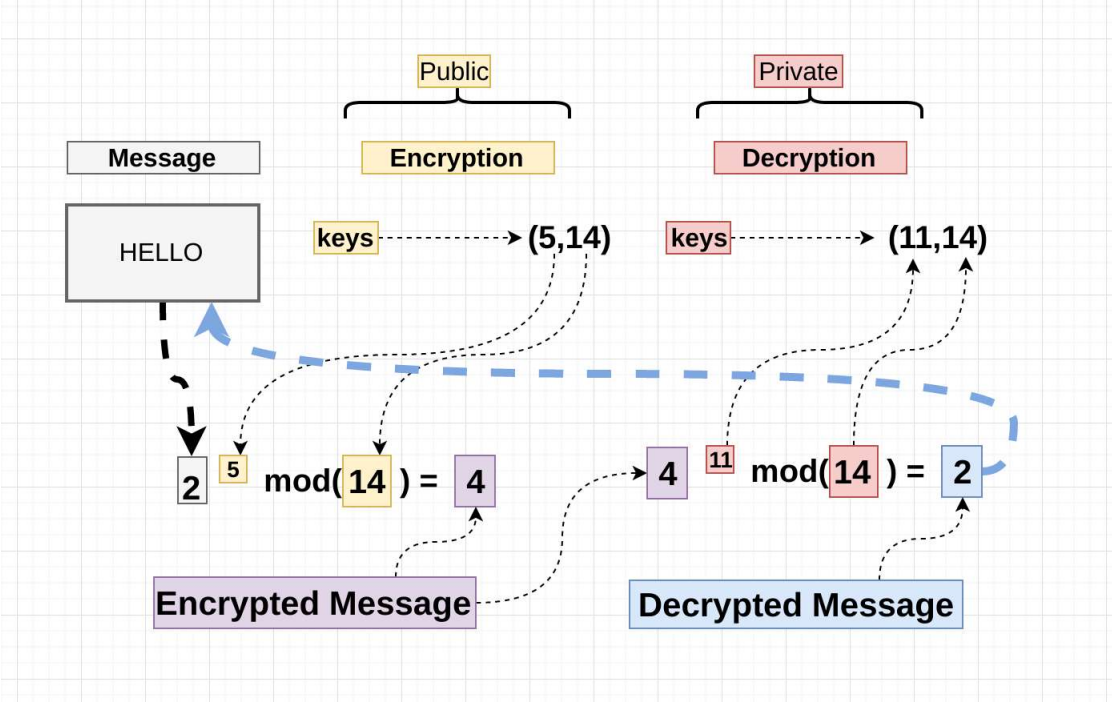
* 3-DES Algorithm

: 말 그대로 DES 암호화를 3번에 반복하는 암호화 기법이다. 암호화를 3번 반복한다고 하기보다 실제로는 Plain Text를 DES-1 암호화 -> DES-2 복호화 -> DES-3 암호화해서 Cipher Text를 만들어낸다. 내부에서 시행되는 DES Algorithm은 동일하다. 각 DES 과정에서 사용되는 Key는 옵션에 따라 모두 독립적일수도, K1 = K3일수도 있다. 당연히 모두 독립적인 키를 사용하는 옵션이 가장 보안이 뛰어나다. K1 = K2 = K3 옵션은 시간과 메모리만 버리는 행위이기 때문에 사용되지 않는다. 아래 그림은 3-DES의 로직이 시각화된 것이다.



* RSA Algorithm [RSA는 이 암호화 기법을 만든 세 명의 이름에서 따온 것 : 의미X]

: RSA는 DES, AES와 달리 **비대칭키**를 이용하는 암호화 기법이다. 공개키를 이용하여 암호화하고 그것을 비밀키를 이용하여 복호화하는 방식이다. 대칭키와 달리 암호화의 키가 공개되어도 비밀키를 쉽게 알아내지 못할 대수적인 알고리즘[소인수분해 등]을 사용해야 하며, 비교적 짧은 데이터를 전달하는데 용이하다. 아래 그림은 RSA 암호화의 기본적인 예시이다.



** RSA의 기본적인 로직

Encryption:

$$Plaintext \qquad M < n$$
$$Ciphertext \qquad C = M^e (mod\ n)$$

Decryption:

$$Ciphertext \qquad C$$
$$Plaintext \qquad M = C^d (mod\ n)$$

Public Key = (e, n) | Private Key = (d, n)

=> (e, n)을 사용하여 M을 암호화시킨 C를 (d, n)을 사용하여 다시 M으로 복호화한다.

공개키 생성, 개인키 생성에 관한 자세한 자료는 아래 사이트에 친절하게 설명되어 있다.

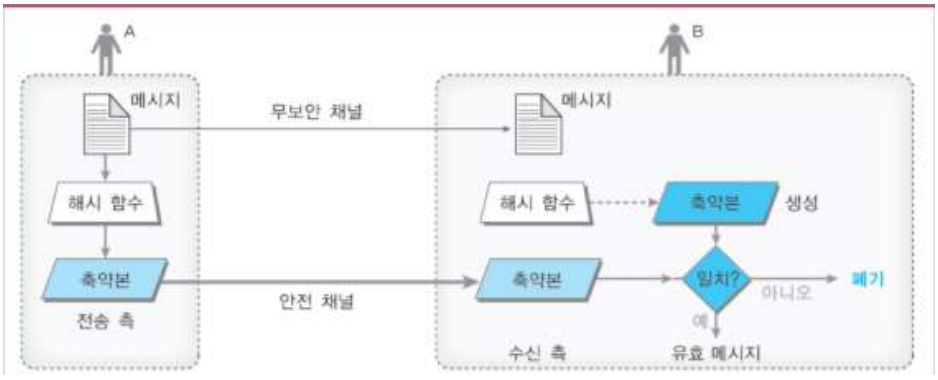
-> https://www.crocus.co.kr/1203

* 메시지의 무결성 인증 기법 [진본 여부 판별]

** 메시지 인증 기법

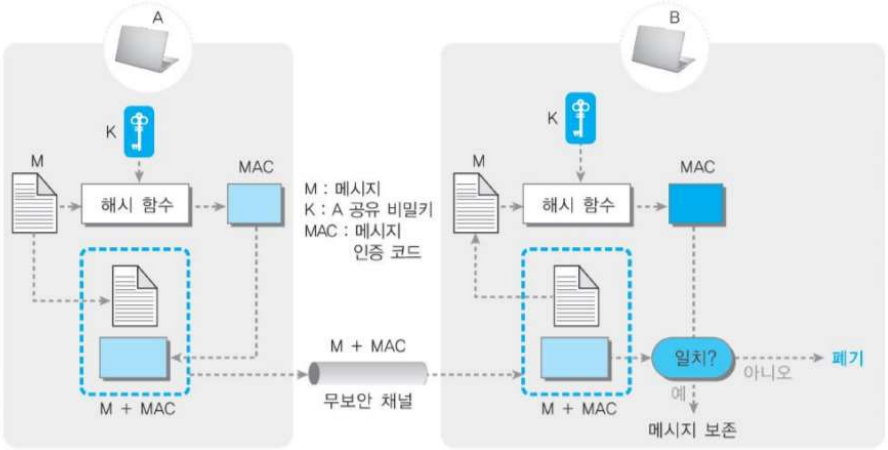
: 문서의 무결성을 검증하기 위해 사용되는 기법, 암호 해시 함수[34]를 사용한다. 먼저 메시지를 암호 해시 함수를 사용하여 메시지 축약(digest)를 생성한다. (Compressed image[35] 생성)

이후 메시지는 무보안 채널로, 축약본은 보안 채널로 전송한다. 수신자는 메시지를 같은 암호 해시 함수를 사용하여 축약하고, 그것이 송신자가 보낸 축약본과 동일하면 메시지가 손상되지 않았음을 확인할 수 있다.



** MAC(메시지 인증 코드) 인증 기법 [대칭키]

: 송신자는 해시 함수를 사용하여 MAC를 생성하고, 메시지와 MAC을 전송한다. 수신자는 이것을 메시지와 MAC으로 분리하고, 메시지에 해시 함수를 사용하여 새로운 MAC을 생성한다. 이 두 개의 MAC을 비교하여 메시지의 손상여부를 확인한다.

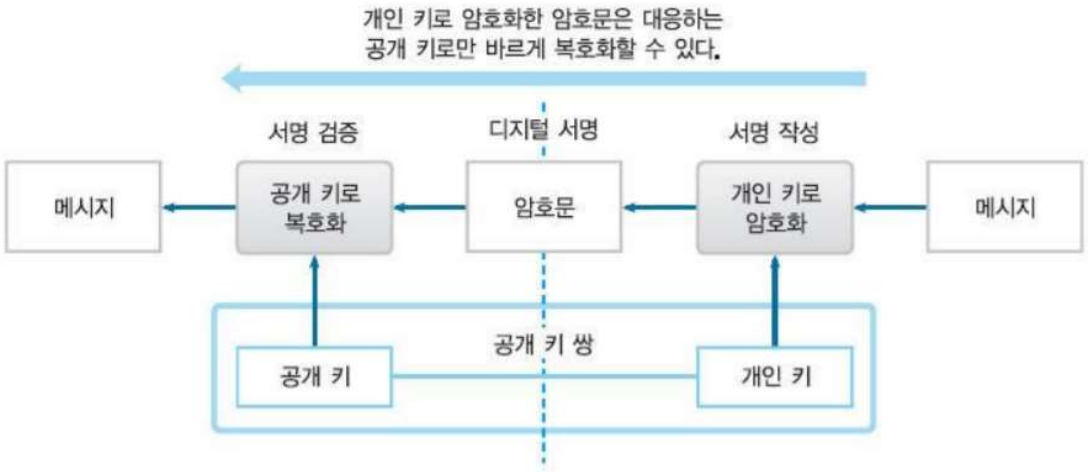


34) 해시 함수(hash function) : 가변적 길이의 메시지를 고정된 길이로 축약하는 함수
해시 알고리즘의 예) MD2, MD4, MD5, SHA-1, SHA-2 등

35) Compressed image : 메시지에서 지문과도 같은 것 (진본 여부 판별을 위해)

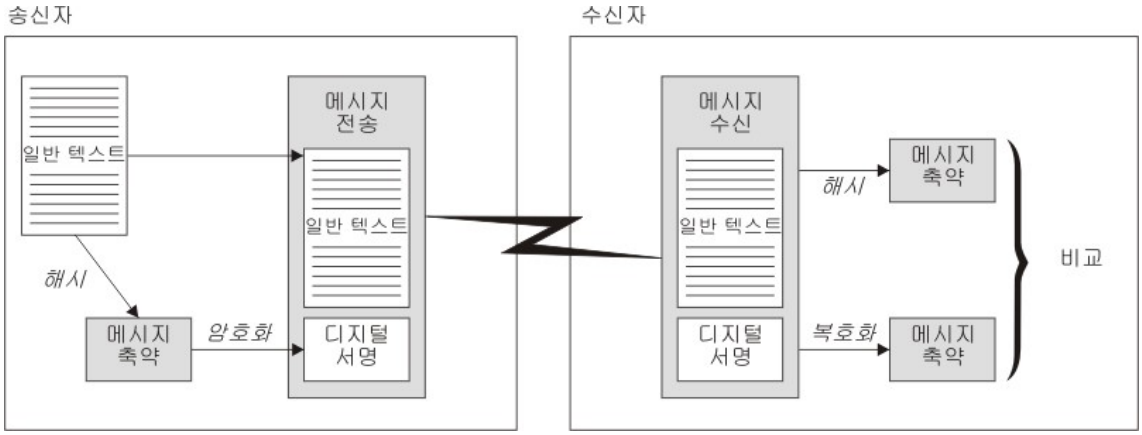
** 전자 서명 (Digital Signature)

: 전자 서명은 RSA의 과정을 반대로 수행하는 로직을 이용한다. 서명을 작성하여 그것을 메시지와 함께 개인키로 암호화해서 전송하고, 수신자는 그것을 공개키로 복호화한다.



** 메시지 축약 서명

: 위에서 언급한 전자 서명은 비대칭키 암호화를 이용한다. 그러나 비대칭키 암호화는 긴 메시지를 다루는데 용이하지 않다. 이에 대한 해결책으로 메시지의 축약본에 서명을 하고, 메시지와 서명된 축약본을 전송하는 "메시지 축약 서명 기법"이 제시되었다.



위 그림은 메시지 축약 서명에 대해 매우 간략하게 표현하고 있다. 왜 메시지 축약 서명 기법을 이용하면 비대칭키로 암호화된 데이터의 용량이 줄어드는지 이해할 수 있을 것이다.

* 보안 프로토콜

** 방화벽 [Firewall]

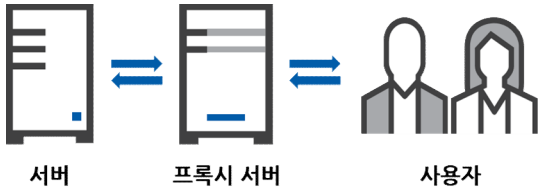
*** Firewall with Router

: 한 subNet의 Default router에서 악의적인 패킷 등을 subNet 내부로 들어오지 못하도록 차단하고, 특정 포트를 막아두는 보안 프로토콜

ex) 특정 IP가 접근하지 못하도록 차단, 특정 포트로 접근하지 못하도록 차단 등

*** Firewall with Proxy

: 가상의 응용프로그램으로 해당 행위를 미리 시뮬레이션 해보도록 하는 보안 프로토콜
시뮬레이션 결과가 양호하면 수신하고, 불량하면 차단한다.



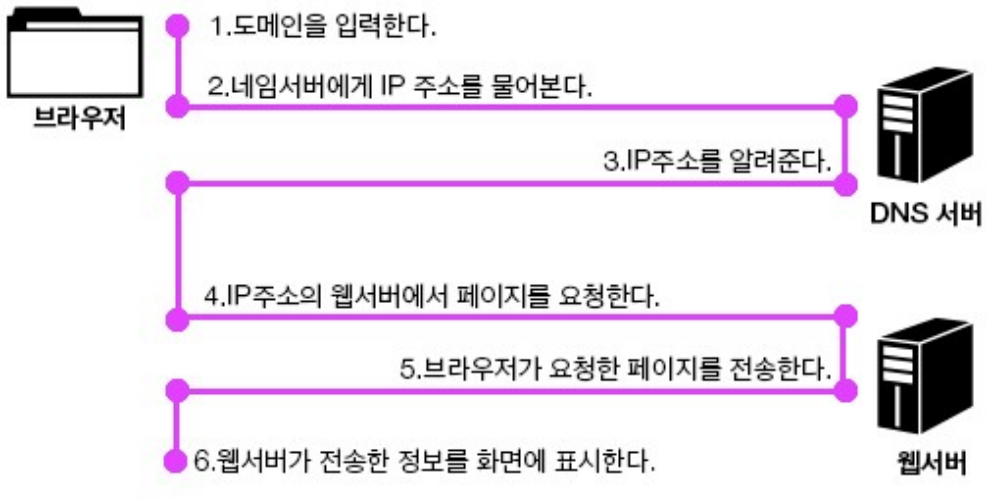
- HTTP (Hyper Text Transfer Protocol)
: 우리가 웹 브라우징을 할 때 항상 이용하는 프로토콜이다. 기본적으로 80번 포트를 이용한다. 최근에는 보안의 문제로 HTTPS(HTTP Secure)로 많이 대체되는 추세이다.

* Web Service
: Client-Server Model을 이용하여 사용자가 서버에 접근하고 정보를 입력하고 받아오고 한다.

** URL (Uniform Resource Locater)
: 프로토콜, 서버의 호스트 네임, 서버 내부의 파일 경로로 표현되어 찾고 있는 정보를 찾는 것을 도와준다. 아래 그림은 URL의 기본 구조이다.

URL : **protocol://ServerHostName:port/resourcePath** (port번호는 생략되면 Default 값)

** Web Service 동작 원리 (URL 입력 ~ 서버 접속)

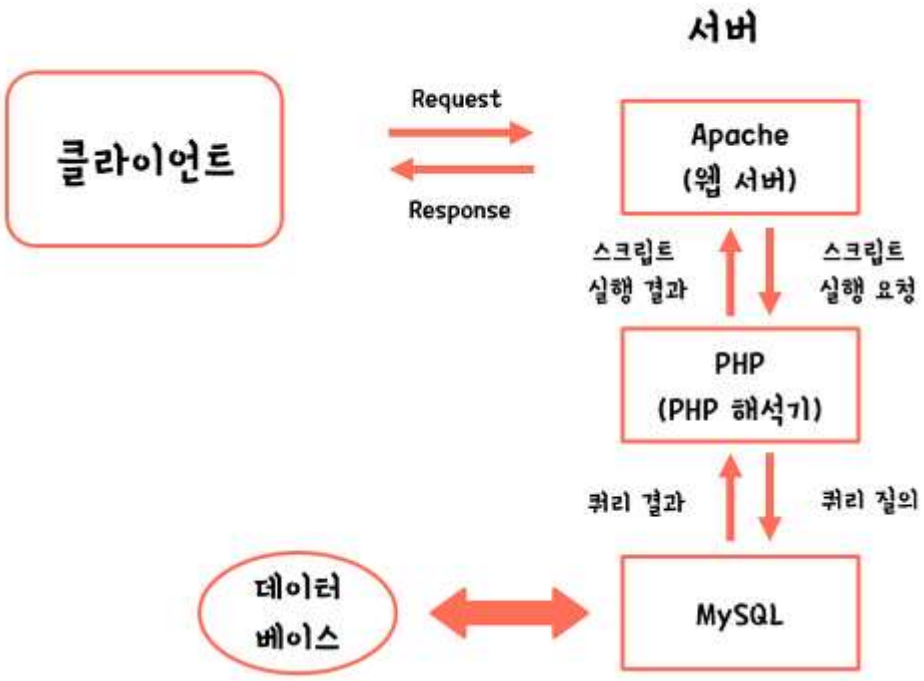


DNS서버[36]에 IP 주소를 질의하는 과정에서 조금 더 복잡한 과정을 거치지만 가볍게 이해하기에는 위 그림이 도움이 된다. 6번 과정에서 웹서버로부터 받은 정보(html, php 등)는 웹 브라우저(Chrome, Explorer 등)에서 Rendering(이미지화)해서 사용자에게 보여준다.
지금은 거의 발생하지 않지만 과거 크롬 브라우저에서 여러 사이트에 대한 호환성이 떨어졌던 것은 Rendering 기술이 부족했기 때문이다.

36) DNS Server : {Domain : IPaddress}정보를 매핑하여 가지고 있는 서버,
55 page에 자세히 설명되어있다.

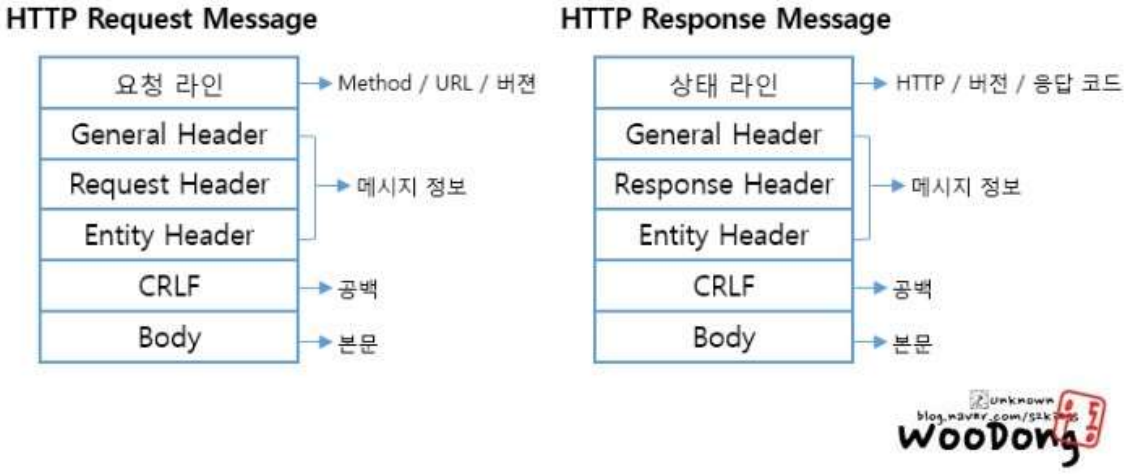
* APM (Apache + PHP + MySQL)
Apache : Web Server를 열어주는 프로그램
PHP : 브라우저 창에 정보를 띄워주는 웹 프로그래밍 언어
MySQL : 웹에서 정보를 가져올 수 있도록 연동되는 데이터베이스

** APM의 동작 원리



: 클라이언트 웹 브라우저(like Chrome)가 Apache에 정보를 요청한다. Apache가 다시 PHP를 통해 창에 입력된 정보[HTML code[37]]를 돌려주고, 필요에 따라 MySQL에 정보를 요구하면 MySQL이 데이터베이스에서 요청받은 정보를 꺼내서 돌려준다. 이렇게 웹이 동적으로 움직일 수 있는 이유는 PHP가 CGI(Common Gateway Interface)이기 때문이다.

* HTTP Request Message & Response Message



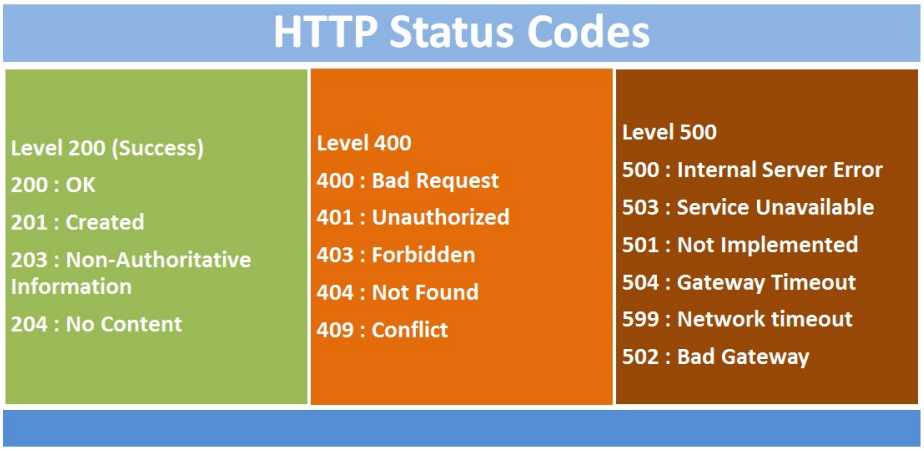
: HTTP에서는 웹 클라이언트가 요청 메시지를 보내면 이것을 서버로 전송하고, 서버에서 응답 메시지를 받아와서 클라이언트에게 제공한다.

37) HTML(Hyper Text Markup Language) : 가장 기본적인 웹 프로그래밍 언어, 정적인 창을 표현

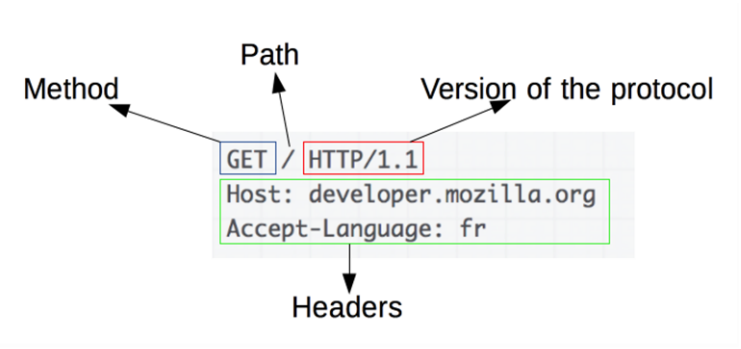
** Method code of Request Message [요청]

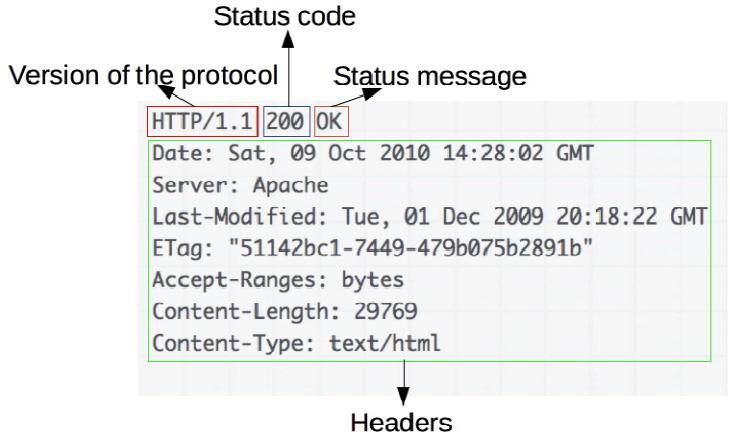
Method	Description
GET	Request to read a Web page
HEAD	Request to read a Web page's header
PUT	Request to store a Web page
POST	Append to a named resource (e.g., a Web page)
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Reserved for future use
OPTIONS	Query certain options

** Status code of Response Message [응답]



** HTTP 요청&응답 메시지 예



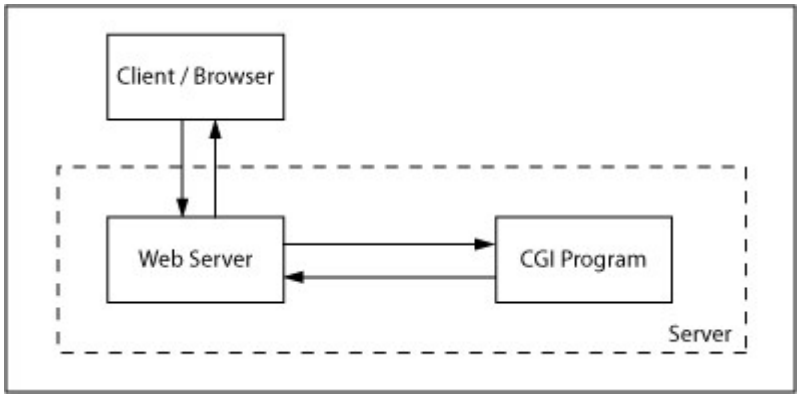


```
--본문 영역-- [index.html]
<html>
<head>
<title> Title </title>
</head>
<body>
<h1> This is HTML Language </h1>
</body>
</html>
```

* HyperText [연계 텍스트]
 : 문서의 일부를 다른 문서와 연결시켜놓은 텍스트 (hyperlink)

* HTML (Hyper Text Markup Language)
 : 웹 문서를 작성하는 데 이용되는 웹 프로그래밍 언어

* CGI (Common Gateway Interface)
 : 사용자가 웹 문서를 동적으로 다룰 수 있도록 도와주는 태그
 [HTML 기본 태그만으로는 하나의 멈춰있는 창을 보여줄 뿐이다.]



이전에 HTML을 이용한 웹 프로그래밍을 조금이라도 접해봤다면 <form>, <input> 등의 태그들을 본 적이 있을 것이다. <input>은 사용자로부터 입력값을 받는 것이고, <form>은 사용자로부터 입력받은 값을 <submit>해서 서버로 전송하는 것이다. 이러한 CGI들이 없다면[38] 구글에 로그인하는 것은 불가능할 것이다.

* HTML5
 : HTML version의 일종이다. 기존의 HTML4 등의 version에서 웹 게임, 실시간 스트리밍 등의 빠른 속도를 요구하는 작업을 감당할 수 없어서 제시되었다. 주요 기술 중 하나로는 캔버스 API가 있는데, 이것은 옛 버전에서 값을 입력하면 그 값을 서버로 전송하고 서버에서 그림을 그려서 사용자에게 static한 창을 보여주었던 것을 입력한 값을 서버로 전송하지 않고 웹 브라우저가 실시간으로 그릴 수 있게 해주었다. [처리 속도 ⟰] 이외에도 이전 버전과 비교할 수 없을만큼 좋은 기술들이 소개되었다.

* HTTP 외의 주요 Protocols
 FTP : File Transfer Protocol [연결에 ID, Password가 필요]
 TFTP : Trivial(보안이 중요하지 않은) FTP [ID, Password가 필요하지 않다.]
 SMTP : Simple Mail Transfer Protocol [Email 송수신에 사용되는 프로토콜]
 Telnet : 원격지 조작에 사용되는 프로토콜
  => window의 “원격 데스크톱 연결”에 이용되는 것이 바로 TELNET이다.

38) 물론 로그인은 데이터베이스도 함께 연동되어야 한다.

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8" />
    <title>Hello : CoForward</title>
    <script src="jsFiles/jsExample.js"></script>
    <link rel="CSS" href="cssFiles/cssExample.css" />
</head>
<body>
    <p>Printing "Hello Program" Code in Python & Java</p>
    <!--This is used to input Programming code-->
    <pre id="Code">
        In Python,
        print("Hello Python")

        In Java,
        public class Main{
            public static void main(String[] args){
                System.out.print("Hello Java");
            }
        }
    </pre>

    <table border="1"> <caption>This is A's Routing Table</caption>
        <th colspan="2">A's Routing Table</th>
        <tr><td>A</td><td>-</td></tr>
        <tr><td>B</td><td>A</td></tr>
        <tr><td>C</td><td>A</td></tr>
        <tr><td>D</td><td>C</td></tr>
        <tr><td>E</td><td>B</td></tr>
        <tr><td>F</td><td>E</td></tr>
    </table>
    <br>
    <a href="hrefFile.html">click to Move Page</a>
    <img src="subNetwork.png" />
    <hr>
</body>
</html>
```

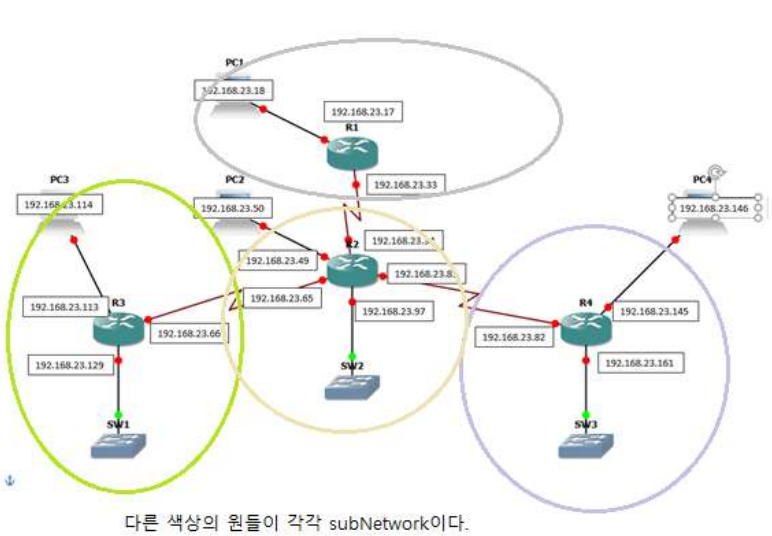
-> javascript, css를 적용한 html파일 예시 (index.html)

Printing "Hello Program" Code in Python & Java

```
In Python,
print("Hello Python")

In Java,
public class Main{
    public static void main(String[] args){
        System.out.print("Hello Java");
    }
}
```

This is A's Routing Table

A's Routing Table	
A	-
B	A
C	A
D	C
E	B
F	E



다른 색상의 원들이 각각 subNetwork이다.

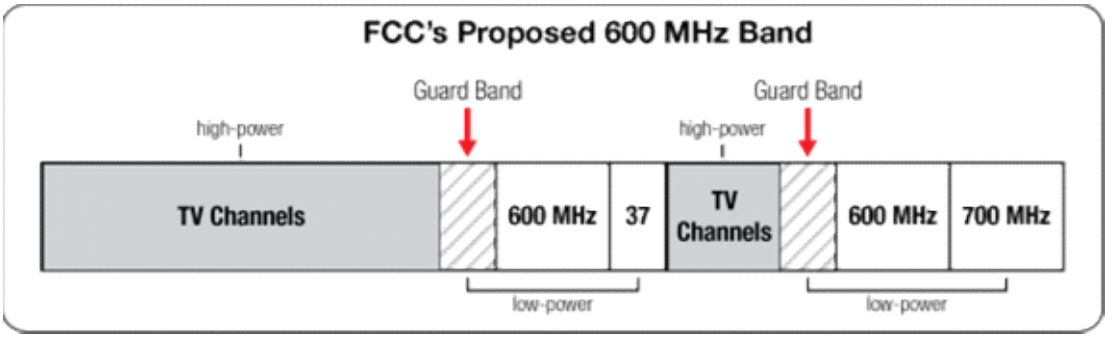
-> 위 예시의 결과창

- Wireless Network

* 무선 LAN

: 일정 범위 내의 사용자가 물리적인 연결 없이 네트워크 접속 가능하게 한 LAN 기술
(기본적으로 주파수를 기반으로 통신)

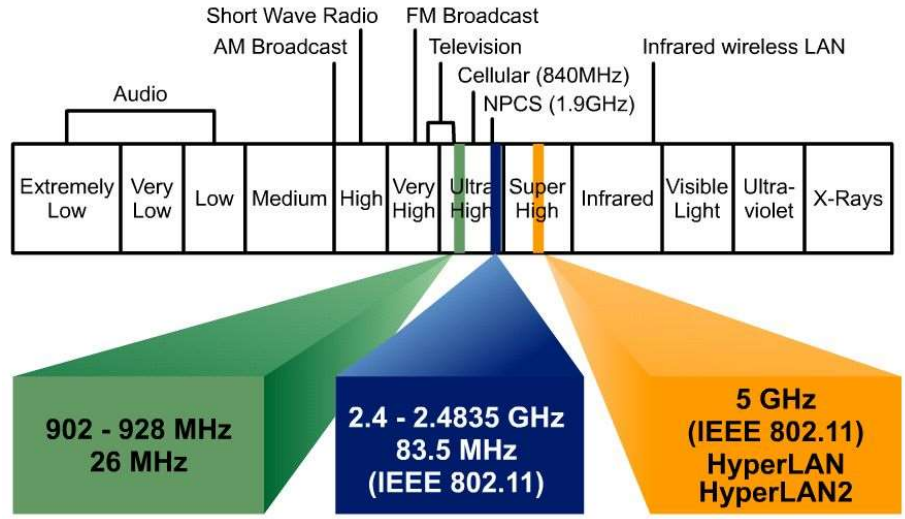
* BAND PLAN (1Hz ~ 100GHz)



: 무선 LAN 기술은 주파수를 기반으로 통신한다고 했는데 ALOHA Protocol에서 소개되었듯 같은 주파수를 사용하며 동시에 전송하게되면 Collision(충돌)이 발생하여 통신을 제대로 할 수가 없다. 그렇기 때문에 미국의 FCC(통신단체)에서는 BAND PLAN을 수립하였다. Band Plan은 Bandwidth(주파수 대역)을 어떻게 사용할 것인지에 대한 계획이다. 위 그림은 그것에서 극히 일부분을 발췌하여 가져온 것이다.[39] 이러한 주파수 대역을 정부에서 일정 금액을 받고 사용자에게 빌려주는 것이다. 가장 통신에 좋은 주파수 대역은 대부분 군사용으로 쓰인다. 그러나 모든 주파수 대역을 유료로 설정해두면 산업, 과학, 의료기술의 발전이 더뎌지게 된다. 이것을 방지하기 위해 FCC는 ISM Band를 설정하여 무료로 사용할 수 있도록 하였다.

** ISM Band

: Industrial, Scientific, Medical의 앞글자를 따서 ISM Band이다. 아래에 보이는 3개의 영역이 FCC에서 지정한 ISM radio Band이다. 산업, 과학, 의료기술의 발전을 위해 무료로 사용할 수 있도록 지정해놓은 영역이다.



39) 원본 : United_States_Frequency_Allocations_Chart_2016_-_The_Radio_Spectrum.pdf

* IEEE 802
: 1980.2에 설립된 LAN/MAN에 대한 표준을 정하는 IEEE 내부 단체

* IEEE 802.11
: Wireless Local Area Network에 대해 IEEE가 지정하는 일련의 Protocol

* IEEE 802.11α

종류	속도	거리	주파수대역	장점	단점
802.11a	54 Mbps	300m	5 GHz	주파수 대역이 구별되어 전파 간섭이 적은 편	주파수 대역이 달라 호환성 없음
802.11b	11 Mbps	450m	2.4GHz	가장 많이 사용되고 있음	전송 속도가 느림
802.11g	54 Mbps	450m	2.4GHz	802.11b와 호환 가능	2.4GHz 기기들로부터 간섭이 있을 수 있음
802.11n	300 Mbps	450m	2.4GHz/ 5GHz	다중 안테나 기술과 채널 본딩을 통한 성능 증가	2.4GHz기기들로부터 간섭이 있을 수 있으며, 제대로 활용하기 위해 무선랜카드 선택이 중요함.

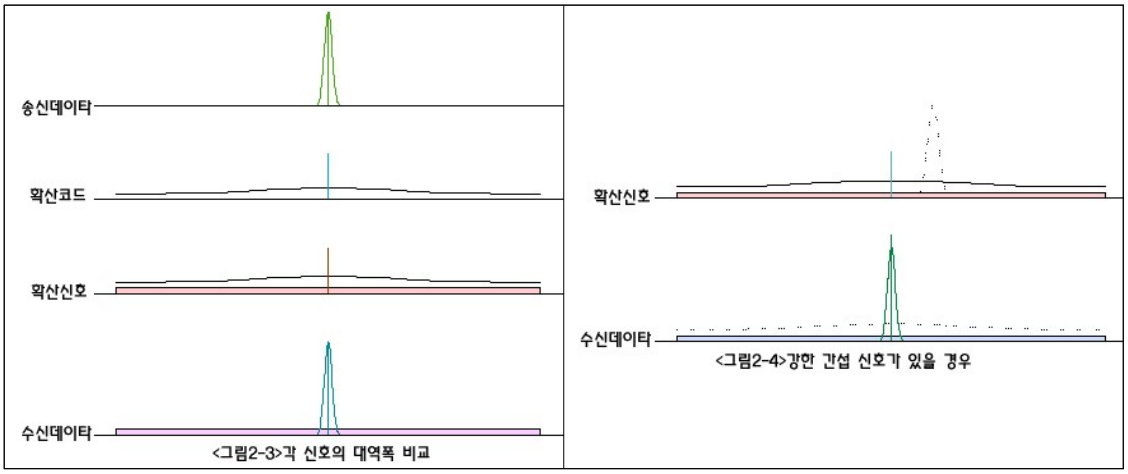
<자료 = 와이파이 연합>

: 위에 보이는 것처럼 IEEE 802.11 중에서도 a, b, … 등 다양한 규격이 존재한다. 이는 사용하는 주파수 대역이 다르기도, 대역 변조의 방식[40]이 다르기도 하다. 그에 따라 사용거리, 데이터의 최대 전송속도 등이 달라진다.

규격 표준	사용 주파수 대역	대역 변조 방식	전송률
IEEE 802.11a	5GHz	OFDM	6 ~ 54 Mbps
IEEE 802.11b	2.4GHz	FHSS, DSSS	5.5 ~ 11 Mbps
IEEE 802.11g	2.4GHz	OFDM	Max 54 Mbps
IEEE 802.11n	2.4GHz, 5GHz	OFDM + MIMO	Max 600 Mbps
IEEE 802.11ac	5GHz	256-QAM + MIMO	Max 2.6 Gbps

* 확산 대역 변조 (Spread Spectrum)
: jammer에서 발생하는 jamming[41]으로부터 간섭을 최소화하기 위해 데이터를 전송할 때 Narrow Bandwave로 전송하지 않고 그것을 Spread하여 Spread Waveform으로 전송하는 변조 기법

** 확산 대역 변조를 사용하는 이유

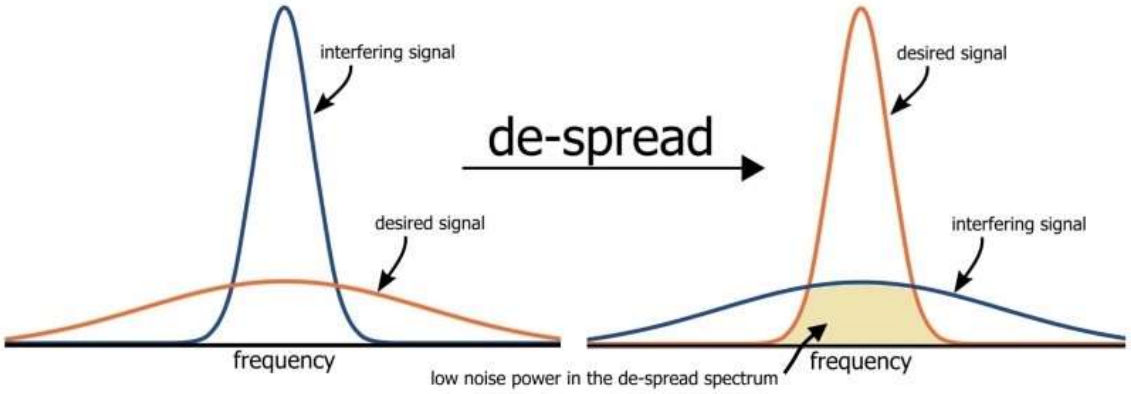


<그림2-3>각 신호의 대역폭 비교
<그림2-4>강한 간섭 신호가 있을 경우

위의 두 그림에서 좌측 상단에 있는 것이 전송하고자 하는 원본 데이터이다. 이것을 확산시켜 보내면 그것을 받은 수신측은 그것을 다시 De-spread시켜 원본 데이터를 얻는다. 좌측 그림은 Jamming이 없는 경우이다. 우측 상단의 점선으로 표현된 것이 jamming

40) 이후 다루게 될 FHSS, DSSS, OFDM 등을 일컫는다.
41) jamming : 방해, 간섭 [주로 고의적]

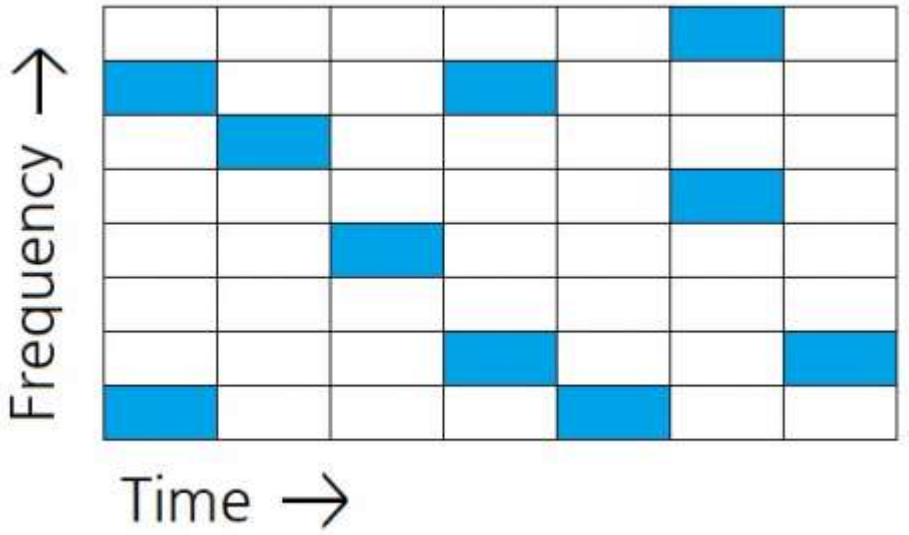
이다. 우측은 jamming이 발생한 경우지만, Spread를 사용하면 De-spread시켰을 때 거의 대부분을 회복하여 데이터를 받을 수 있다. 아래 그림은 jamming이 들어왔을 때 확산시켜진 데이터를 역확산시키면 jamming에 영향을 받은 영역을 줄일 수 있다는 것에 대한 그림이다.



** 확산 대역 변조 기법

1. FHSS (Frequency Hopping Spread Spectrum)

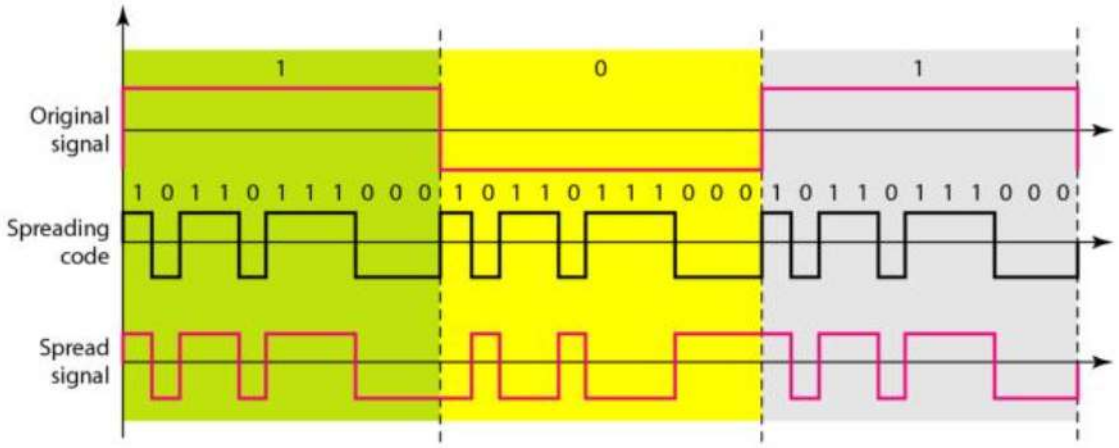
: 직역하면 주파수를 도약하는 확산 기법이다. 먼저 전송하고자 하는 대역폭이 있으면 그 주변의 여러 개의 대역폭을 선택한다. 그리고 보내고자 하는 데이터를 대역폭의 개수에 맞게 쪼개서 각각의 대역폭에 뿌리는 확산 기법이다. [대역폭의 순서를 미리 정해 순차적 전송]



Ex) 800MHz 대역으로 데이터를 전송하려고 한다. 신호를 확산시키기 위해 임의로 주변 주파수들을 선택한다. [790, 795, 800, 805, 810]MHz를 선택했다고 하자. 이것들의 전송 순서를 미리 정한다. {0 : 800, 1 : 810, 2 : 795, 3 : 790, 4 : 805}로 순서를 정했다고 하자. 전송하고자 했던 신호를 5개로 분할한다. 분할된 신호를 미리 정해둔 순서대로 800 > 810 > 795 > 790 > 805 MHz를 통해 전송한다. 만약 800MHz가 jamming에 의해 간섭을 받았다고 가정해보자. 나머지 4곳으로 분할되어 원본의 80%는 통신이 된 것이다. (분할하지 않고 800 MHz로 모든 신호를 보냈다면 전송률은 0%가 되었을 것)

2. DSSS (Direct Sequence Spread Spectrum) < Fast & Non-Noise >

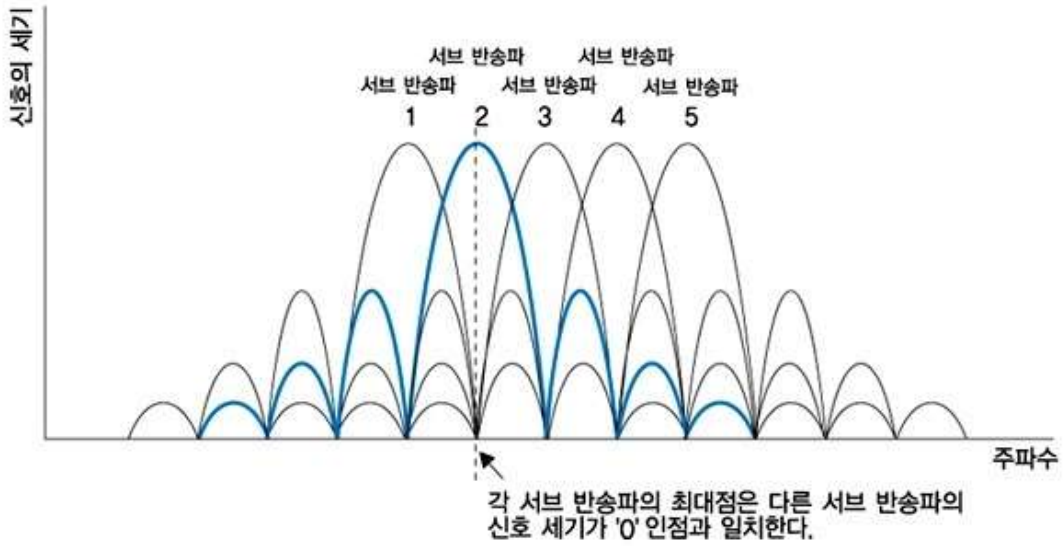
: 송/수신자 간의 특정 Chip-code를 이용하여 신호를 확산시키는 기법이다. 아래의 그림을 보면 Chip-code가 "10110111000"인 것을 확인할 수 있다. Original Signal을 이 칩-코드에 어떠한 연산을 입혀서 신호를 분할한다. 이렇게 분할된 신호를 전송하고자 하는 주파수와 그 주변 주파수들에 나누어 전송한다. [동시 전송] 여기에서는 Chip-code로 11bits를 사용하므로 Chipping-Sequence가 11이 된다. 이렇게 전송된 신호는 특정 Chip-code를 가지고 재번역하지 않으면 의미 없는 데이터(Dummy)가 된다. 그러므로 jamming 방지뿐만 아니라 도청과 같은 해킹으로부터 보안성도 높은 변조방식이다.



3. OFDM (Orthogonal Frequency Division Multiplexing)

: 하나의 Main-Signal을 여러 개의 Sub-Signal로 나누어 전송하는데, 각각의 sub-signal들은 자신의 신호 세기가 최대일 때 다른 sub-signal의 신호 세기가 0인 것들로 한다. 이렇게 (Orthogonal하게) 설정하는 이유는 sub-signal들 사이의 독립성을 유지하기 위함이다.

[IEEE 802.11a에서는 52개의 sub-signal을 사용한다. (48개 : 데이터, 4개 : 제어기능)]

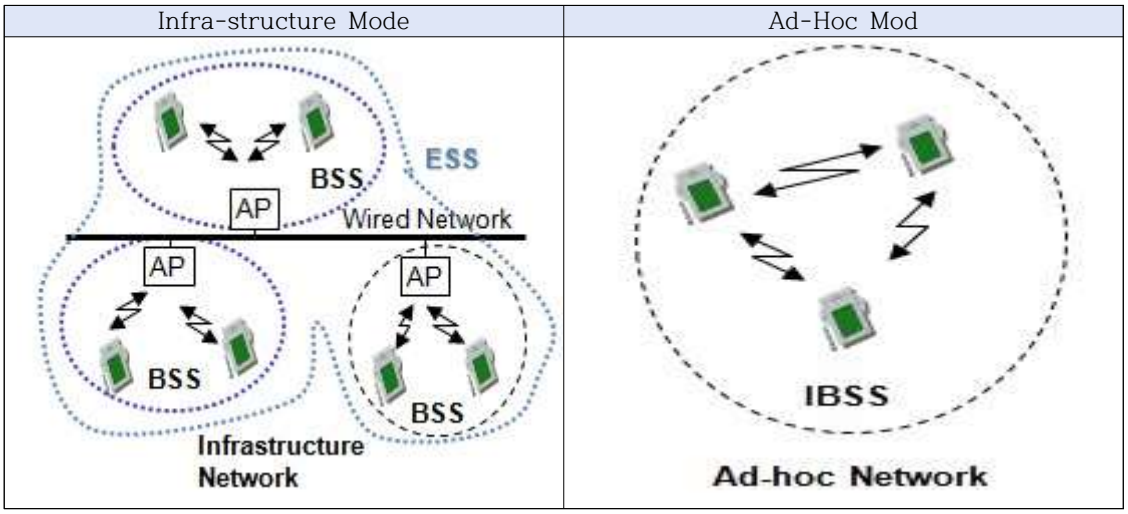


※ MIMO (Multiple-Input & Multiple-Output)

: 여러 개의 안테나를 사용하여 전송속도를 높이는 기법

※ 256-QAM(Quadrature Amplitude Modulation) : 높은 차수 변조 방식

* 무선 LAN의 구조



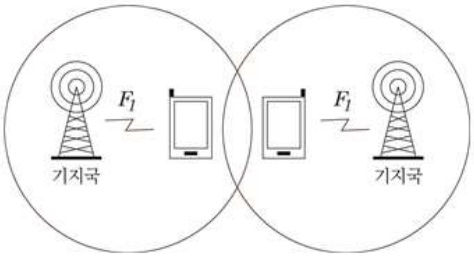
1. Infra-structure Mode [하부 구조 방식]

: Infra-structure mode는 무선 단말기들이 Network에 접근할 때 AP(Access Point)를 통하여 통신하도록 하는 방식이다. 여기서 AP는 기지국 같은 것들이다. 아래 그림에 BSS, ESS가 표시되어 있는데, BSS는 Basic Service Set으로 하나의 AP로 제어되는 논리적 집합이고, ESS는 Extended Service Set으로 다수의 BSS가 이루는 논리적 집합이다.

2. Ad-Hoc Mode [애드혹 방식]

: Ad-hoc mode는 단말기가 다른 단말기와 무선으로 통신하려고 할 때 AP를 거치지 않고 Peer-to-peer 방식으로 직접 통신을 주고받는 것이다. WPAN[블루투스 등]에서 대부분 이용하는 방식이다. (블루투스 마우스를 사용하는데 AP를 거칠 필요는 없기 때문)

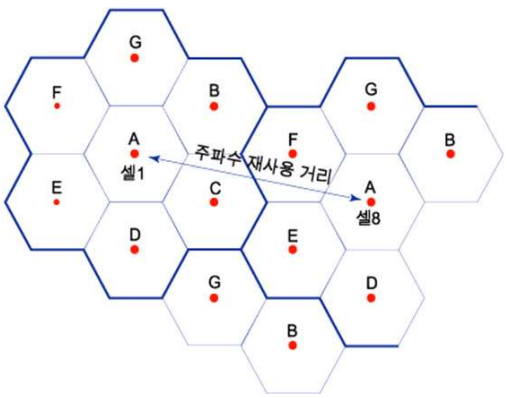
* Access Point 위치 지정 고려 사항



: 위에 보이는 것처럼 하나의 AP와 다른 AP가 서로 인접해 있고 같은 주파수를 사용하여 통신할 경우, 두 AP의 영향권의 교집합 영역에서 Collision이 발생한다. 이것을 방지하기 위해 인접한 AP는 서로 다른 주파수를 사용하여 통신하도록 한다. 또한 Multipath Fading 현상에 의한 신호의 변화 또한 고려해야하기 때문에 셀을 배치하는 것은 쉬운 일이 아니다.

** Cellular System

: 아래 그림에서 각 알파벳은 주파수를 의미한다. 이처럼 인접 AP들은 다른 주파수를 사용하여 통신하고, 주파수 재사용 거리보다 멀리 떨어진 위치의 AP들은 같은 주파수를 사용해도 되도록 하여 인접 채널 간섭 현상을 피하도록 하는 시스템이다.



** MultiPath Fading [다중 경로 페이딩]

: 위처럼 신호가 발신지에서 수신지까지 일직선으로도 가지만, 장애물 등에 의해 반사, 굴절하여 시간이 지연되어 가기도 한다. 이러한 신호는 서로 간섭을 일으켜 원래 신호와 수신된 신호의 차이가 생기게 된다. 이러한 간섭 현상을 Fading이라고 한다.

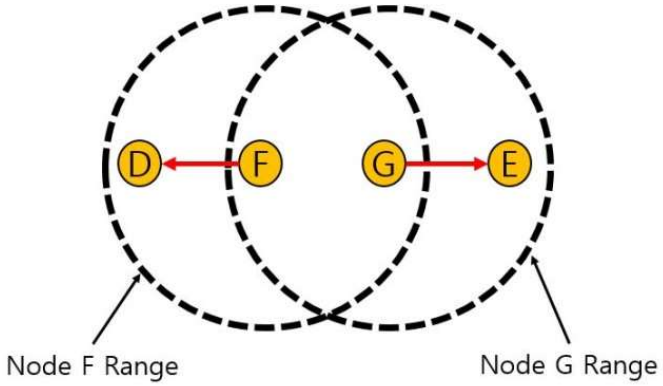
* IEEE 802.11 MAC subLayer [2계층]

: 위에서 기술된 IEEE 802.11, Band Plan, 확산 대역 방식 등은 모두 Physical Layer에 관련된 것들이다. 여기서 기술할 것은 IEEE 802.11에 지정된 MAC Layer에 관한 프로토콜이다.

* CSMA/CA [Carrier Sense Multiple Access with Collision Avoidance]

: OSI Reference Model을 공부하며 보았듯 MAC subLayer에서는 Medium Access Control의 역할을 한다. 이러한 역할이 필요한 이유는 대부분의 네트워크가 공유 네트워크를 사용하기 때문이라고 했다. 무선 LAN에서는 단말기끼리 서로의 신호 영역 내에서 동시에 데이터 전송을 하려고 하면 Collision이 발생한다. CSMA protocol을 사용하면 대부분의 문제는 해소되었지만 Hidden Node problem, Exposed Node problem은 해소되지 않는다. 이것을 해결하기 위해 RTS, CTS를 이용하는 CSMA/CA 기법이 제시되었다.[42)]

** Exposed Node Problem [노출된 노드 문제]

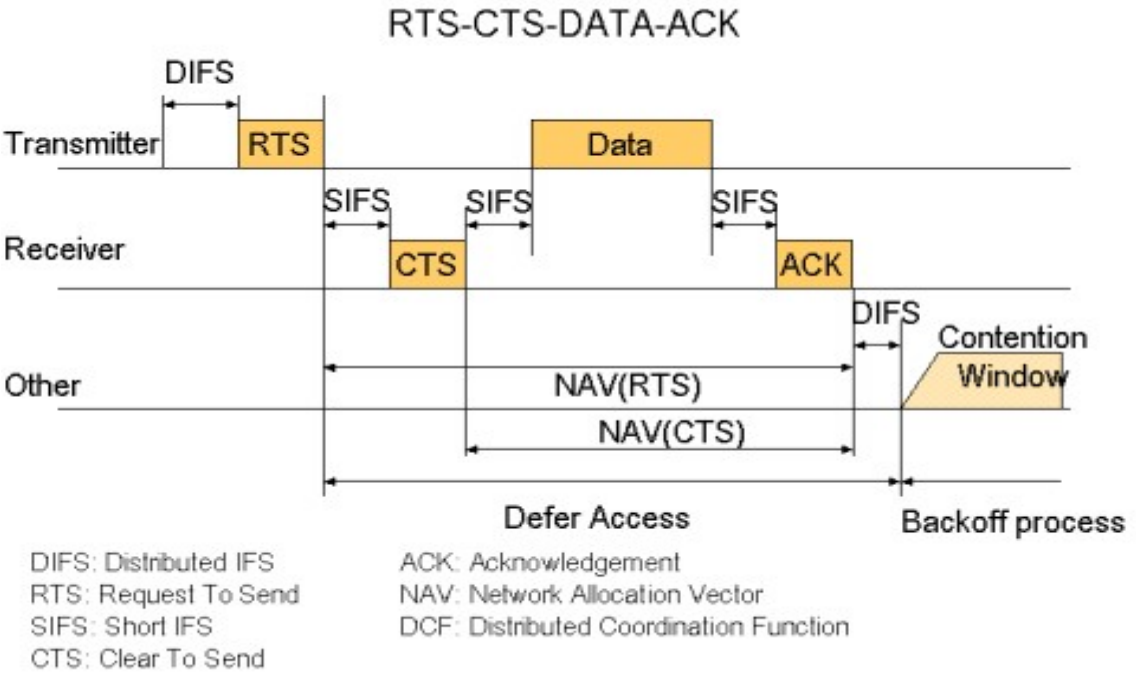


위 그림에서 F는 D에게 데이터를 전송하는 중이다. 이때 G는 E에게 데이터를 전송하고 싶지만, F가 데이터를 전송하고 있으므로 전송을 하지 못하고 기다린다. 그러나 실제로 G가 E에게 데이터를 전송하여도 문제가 없다. 이러한 효율성의 문제가 "노출된 노드 문제"이다.

-> CSMA/CA Protocol에서도 이 문제는 해결되지 않는다.

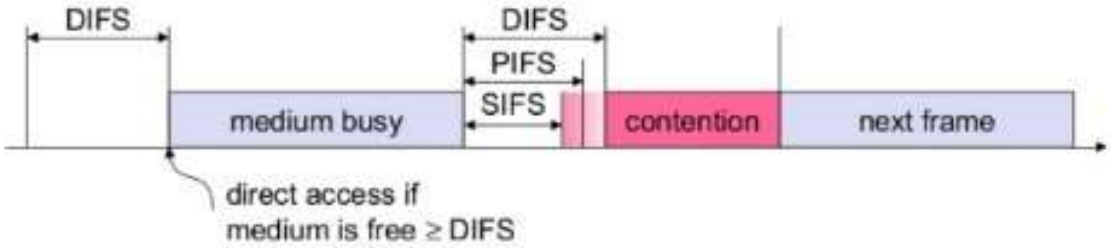
42) 27 page에 CSMA/CA, Hidden Node Problem에 대한 정보는 기술되어 있다.

** CSMA/CA 동작 원리

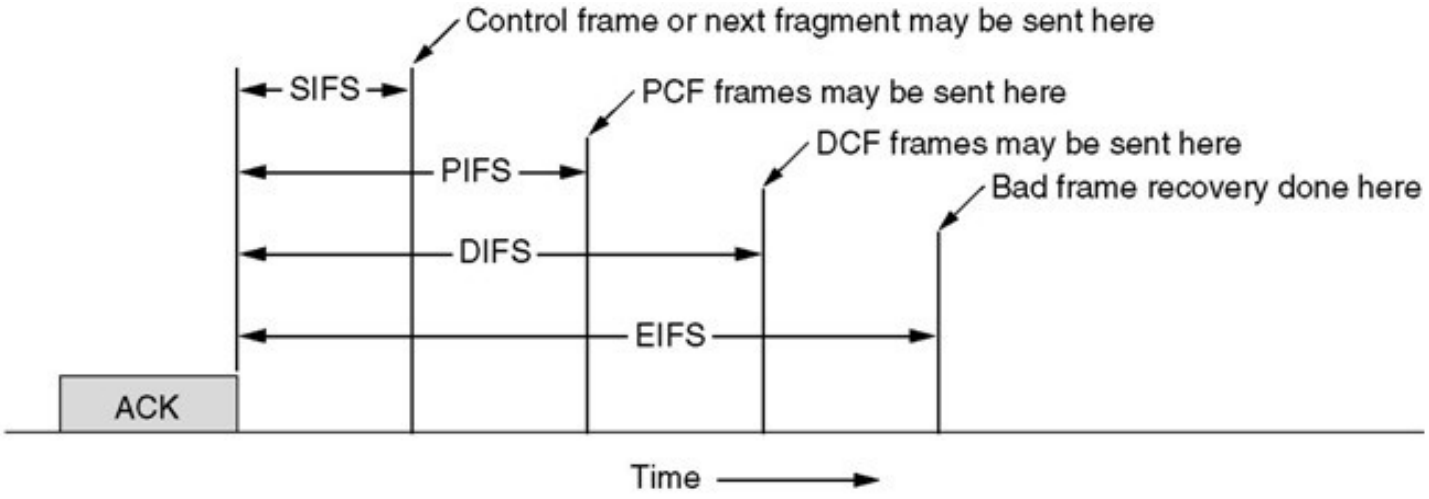


: 간략히 설명하면 Transmitter가 RTS를 Receiver에게 전송하고, 그에 대한 CTS를 받고, Data를 전송한다. 데이터가 모두 전송되면 Receiver가 ACK를 전송하고, 통신을 마친다. 여기서 Transmitter와 Receiver의 영향권 내에 있는 노드들을 NAV(대기 상태)를 가진다. Receiver의 ACK가 도착하면 다른 노드들은 NAV상태가 풀리고, 통신을 할 수 있는 권리를 가지게 된다. 여기서 DIFS(일정 시간의 딜레이)가 지나면 Contention Window 상태가 되는데, 이 상태는 통신의 우선권을 두고 Window들이 경쟁하는 상태이다. 여기서 우선권을 얻는 순서는 SIFS > PIFS > DIFS 순서이다.

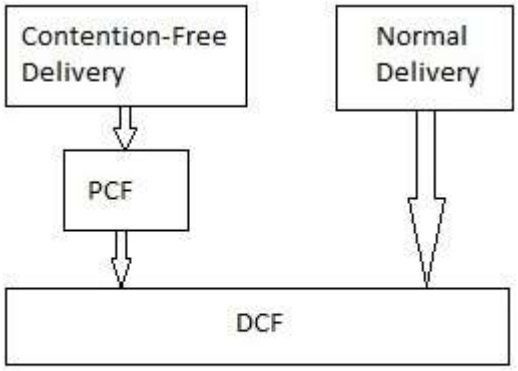
*** IFS [Inter Frame Space]



이 그림은 위 그림에서 ACK가 전송되고 Contention Window 상태가 되었을 때의 상세 과정이다. DIFS(일정 지연 시간)이 지나고 나면 SIFS가 통신에 가장 우선권을 획득한다. SIFS는 Short IFS인데, 이것은 RTS, CTS, ACK, Fragment된 연속 데이터를 위한 IFS이다. 그 다음 우선권은 PIFS인데, 이것은 PCF(Point Coordinate Function) IFS이다. 이 우선권을 획득할 수 있는 노드는 Access Point들이다. Access Point는 PCF MAC을 보유하고 있어서 이 우선권을 얻는 것이 가능하다. 그 다음이 DIFS로, DCF IFS이다. DCF는 Distributed Coordinate Function으로, 이러한 권한은 일반 사용자의 단말기 같은 것들에 부여된다. 위 그림의 이해가 어렵다면 아래의 그림을 보면 좀 낫다.



* MAC subLayer의 구조



802.11 MAC Coordination Functions

MAC Layer는 위처럼 PCF MAC과 DCF MAC을 가질 수 있다. DCF는 필수적으로 가지고 있어야 하지만, PCF는 옵션 사항이다. Access Point는 PCF MAC을 가지고 있기 때문에 PIFS의 우선권을 가질 수 있지만, 일반 단말기는 DCF만 가지고 있기 때문에 DIFS의 낮은 우선권을 가지게 된다.

- Wireless PAN (Personal Area Network)
: IEEE 802.15 위원회에서 규정하는 무선 PAN에 관련된 Protocol
WPAN의 통신거리는 10m 미만이고, 주된 목적은 저속 & 저전력 통신이다. 통신거리도 굉장히 가깝고, 저전력 통신을 이루어야하기 때문에 대부분 Ad-hoc Mode에 기반한다.

* Bluetooth
: IEEE 802.15.1에 기술되어 있는 WPAN 프로토콜로, 거의 무상이용이 가능하기 때문에 현재도 많은 분야에서 이용된다. 중간에 잠시 Zigbee[43]라는 초저전력 802.15.4 프로토콜이 소개되었는데 이것은 고액의 금액을 지불해야 사용이 가능했기 때문에 블루투스 ver.4.0 LE(Low Energy)가 소개되면서 점차 사라져갔다.

* UWB (Ultra-WideBand)
: 간단하게 기존의 확장 대역(Spread Spectrum) 기술을 극대화 시킨 기술이라고 할 수 있다. 매우 넓은 주파수 대역을 사용하여 신호를 분산시켜 신호를 전송하는 것으로 속도를 극대화 시키는 방법이다. [넓게 퍼뜨리는 만큼 각각의 신호 세기는 매우 작다.] 이 기술은 아직 크게 상용화되지 않았지만 이것을 이용하면 GPS를 이용한 위치 추적보다 훨씬 정확한 위치 추적이 가능한 고정밀 위치 추적 시스템을 만들 수도 있다.

※ 중요하진 않지만 블루투스는 Application Layer의 프로토콜에 대한 정리가 매우 잘 되어있어서 기종, 규격에 상관없이 기능만 탑재되어있으면 호환성이 매우 뛰어나다.

43) Zigbee : 주로 Sensor Network에 이용되었던 초저전력 프로토콜이다. Sensor같은 장비는 전송 속도가 중요하지 않고, 배터리 교체 없이 오래 작동되는 것이 목적이기 때문에 이것을 사용하였다.

- 정오표

10페이지 하단

* Low Pass Filter : 저주파가 고주파보다 송수신에 유리하다.
-> 우리는 신호를 보낼 때 고주파 신호를 저주파 신호로 바꾸어주어야 한다.

-> Low Pass Filter를 사용하면 높은 주파수의 신호가 차단되어 저주파 신호를 받아들이는 것이 유리한 것은 맞다. 그러나 신호의 송수신에 있어 Low Pass Filter 이야기가 왜 나온것인지 잘 모르겠다. [교수님은 말씀하셨는데]

* 변조(Modulate) : 고주파 신호 -> 저주파 신호 변환
* 복조(Demodulate) : 저주파 신호 -> 고주파 신호 변환

* 변조(Modulate) : 저주파의 신호를 전달하기 위해 고주파인 Carrier frequency에 담는 것
* 복조(Demodulate) : Carrier frequency에 담겨져 온 신호를 원래의 저주파 신호로 분리하는 것
-> 거의 원래 쓰여진 것의 정반대라고 생각
-> 수업에서 이런 이야기들이 나온 이유는 아마 전화기의 마이크, 방송국의 마이크 등에서 Low Pass Filter를 사용해서 고주파 신호는 전달 이전에 정보 자체가 담기지 않는다. 그래서 변조를 하기 전 송신하려고 하는 신호는 저주파인것들로만 구성된다. 그것들을 전화기의 안테나, 방송국의 송신안테나에서 송신하기 위해서는 고주파 신호로 변조할 필요가 있다. 그래서 신호를 송신하기 전에 변조과정을 거친다. 원래는 공기 자체가 Low Pass Filter의 성질을 가지고 있어서 저주파 신호를 잘 전달하는 것으로 이해하고 있었는데 찾아보니 그게 아니라 Low Pass Filter는 하드웨어적인 것이다. (축전기와 저항의 직/병렬 구조를 이용하여 Low Pass Filter, High Pass Filter가 되는)

22페이지 중간~
** Binary Countdown protocol
-> 노드들을 반으로 갈라서 좌측, 우측에 데이터를 보낼 노드가 있는지 질문한다. 만약, 좌측에 우선순위를 둔다면 좌측에서 또 반으로 갈라 데이터를 보낼 노드가 있는지 질문한다. 이러한 과정을 반복해 데이터를 보낼 노드의 우선순위를 정하고, 순차적으로 데이터를 전송하도록 한다.
-> 그림설명부터 대부분 "The Adaptive Tree Walk Protocol"에 부합함.

-> Binary Countdown Protocol
: 각 노드에 이진화된 Station Address를 부여하여, 그 Station Address를 기반으로 통신망 사용의 우선순위를 정하는 규약이다. 예를 들어, 8개의 Station Address가 111, 110, 101, 100, 011, 010, 001, 000이라고 가정했을 때 가장 왼쪽의 Address number가 1인지 0인지로 우선순위를 매긴다. 만약 1이 우선권을 가진다고 하면, 첫 번째 질의에서 우선권을 가지는 노드는 111, 110, 101, 100이 된다. 마찬가지로 두 번째 Address number를 가지고 분류하면, 111, 110이 우선권을 가지고, 또 이렇게 분류하면 111이 가장 우선적으로 통신권을 얻을 수 있다. 이것으로 우선권 순위를 모두 분류하면, 111 > 110 > 101 > 100 > 011 > 010 > 001 > 000 이다. 이 방법으로 **8개의 노드에 최대 3번의 질의**로 가장 우선적으로 통신권을 얻는 노드를 정할 수 있으므로 "bit-map protocol에서 N개의 노드에 N번의 질문을 해야하지만, Binary countdown protocol에서는 N개의 노드에 log2N 번의 질문을 하면 된다."는 그대로이다.