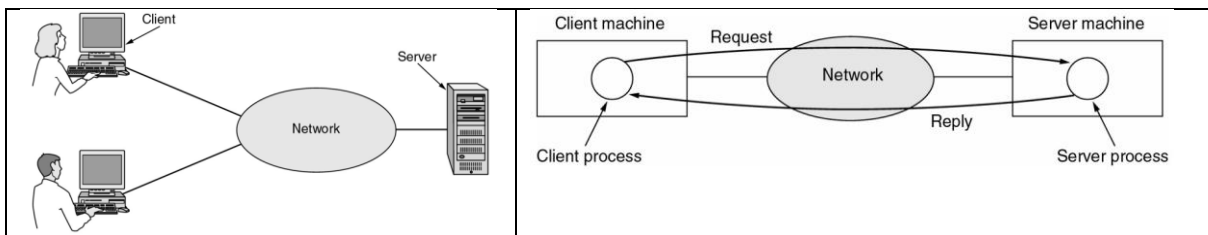


[1. Motivations of Computer Network]

Motivations	Communication Lines, 자원 공유(데이터, 프린터, 프로그램), 정보 교환
Use of Computer Networks	Business application, Home application, Mobile Users

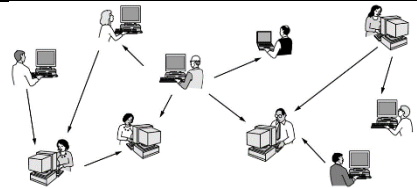
[2. Usages of Computer Network]

<Business Application of Network>



<Home Network Applications>

- ➔ Remote information에 대한 접근
- ➔ 사람 간 커뮤니케이션
- ➔ Interactive entertainment
- ➔ Electronic commerce



<Mobile Network Users>

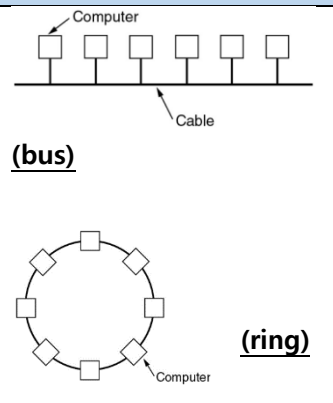
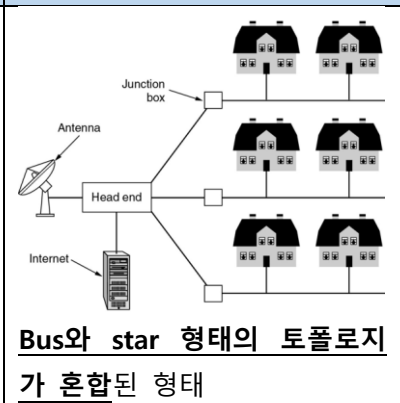
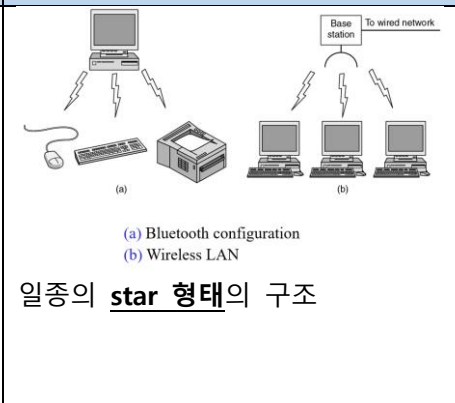
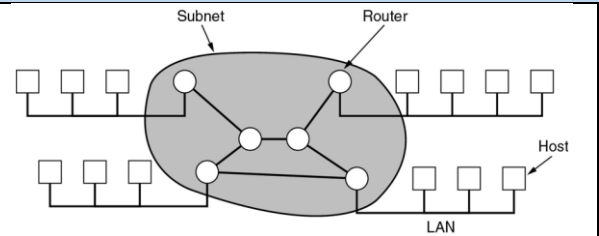
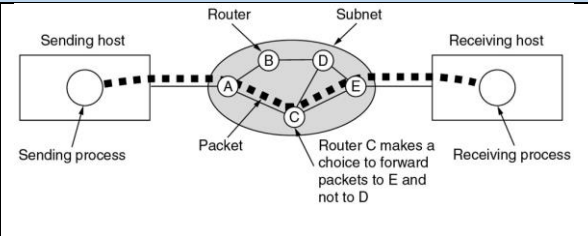
Wireless	Mobile	예시
No	No	회사의 데스크톱 컴퓨터
No	Yes	호텔의 방에서 사용하는 노트북 컴퓨터
Yes	No	Unwired building에서 사용하는 네트워크
Yes	Yes	Portable office, PDA for store inventory

[3. Network Hardware]

Interprocessor distance	Processors located in same	Example
1 m	Square meter	Personal area network
10 m	Room	
100 m	Building	
1 km	Campus	Local area network
10 km	City	
100 km	Country	Metropolitan area network
1000 km	Continent	
10,000 km	Planet	Wide area network
		The Internet

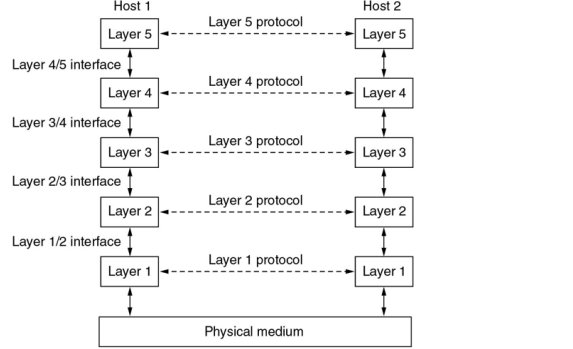
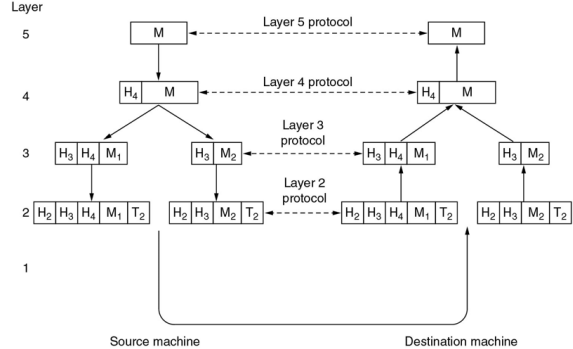
(거리에 따른 네트워크의 분류)

“규모에 따라 사용하는 네트워크 기술이 다르다.”

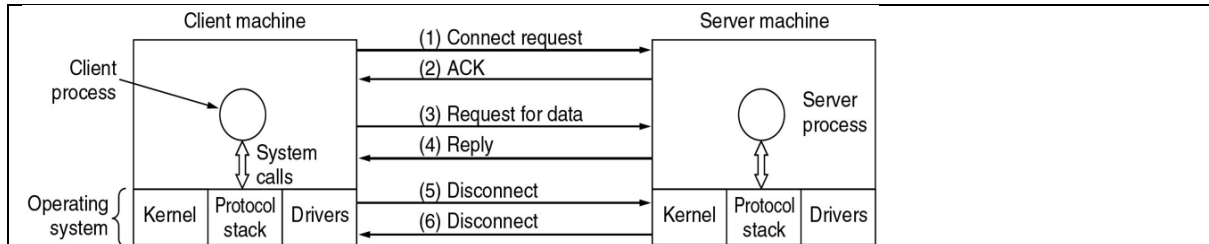
<p>Local Area Networks</p>  <p>(bus)</p> <p>(ring)</p>	<p>Metropolitan Area Networks</p>  <p>Bus와 star 형태의 토폴로지가 혼합된 형태</p>	<p>Wireless Networks</p>  <p>(a) Bluetooth configuration (b) Wireless LAN</p> <p>일종의 <u>star 형태</u>의 구조</p>
<p>Wide Area Networks</p>		
 <p>Bus와 Mesh (또는 star) 형태의 토폴로지가 혼합된 형태</p>	 <p>sender에서 receiver로의 packet stream</p>	

[4. Network Software]

<1. Protocol Architecture: partition & hierarchy>

 <p>→ 위쪽으로 갈수록 소프트웨어 관련, 아래쪽으로 갈수록 하드웨어 관련 계층</p> <p>→ 상위계층이 하위계층의 서비스를 받음</p> <p>→ 각 계층마다 프로토콜이 있고, 이에 따라 정보 교환</p>	 <p>→ 각 계층을 통과할 때마다 header가 붙는다.</p> <p>→ Header는 프로토콜을 구현하기 위한 정보 (가장 중요한 것은 자신과 상대방을 식별하는 정보)를 갖는다.</p>
---	---

<2. Client-Server Protocol>



실제 프로세스들은 **User Level**에서 작동하고, 각종 프로토콜은 응용계층을 제외하고 **OS Level**에서 작동한다. (응용 계층만 User Level, 나머지 모든 계층은 OS Level)

- ➔ 시스템 콜(소켓)으로 OS의 프로토콜 스택을 통해 상대방 서버에 접속
- ➔ 프로토콜 계층이 **User와 OS로 분리됨**

Connectionless	Connection-oriented
<u>바로 데이터를 전송</u>	<u>Connection을 맺은 후</u> connection이 잘 개설되면 데이터 전송

<3. Design Issue for the Layers>

Addressing, Error Control, Flow Control, Multiplexing, and Routing

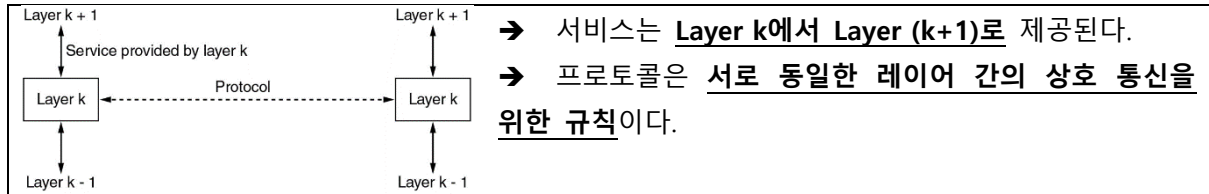
<4. Connection-Oriented and Connectionless Services>

	Service	Example
Connection-oriented	Reliable message stream Reliable byte stream Unreliable connection	Sequence of pages Remote login Digitized voice
Connectionless	Unreliable datagram Acknowledged datagram Request-reply	Electronic junk mail Registered mail Database query

<5. Service Primitives>

Primitive	Meaning
LISTEN	Incoming connection을 기다리는 블럭
CONNECT	Waiting peer와의 connection 개설
RECEIVE	Incoming message를 기다리는 블럭
SEND	peer에게 메시지 전송
DISCONNECT	connection 종료

<6. Service to Protocols Relationship>



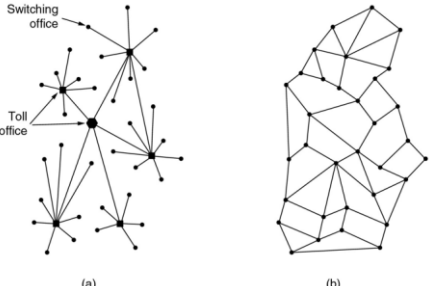
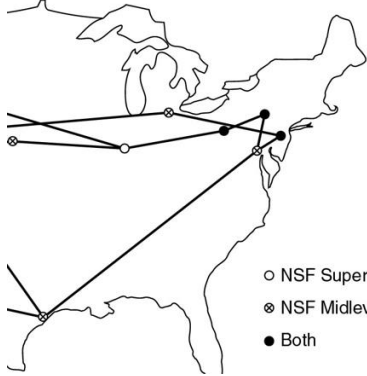
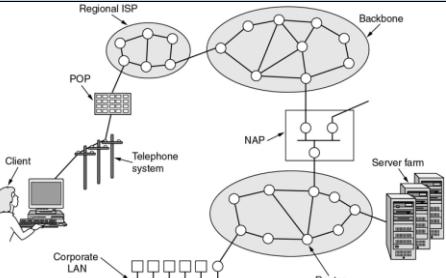
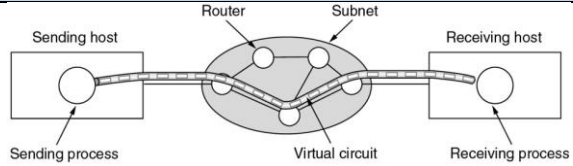
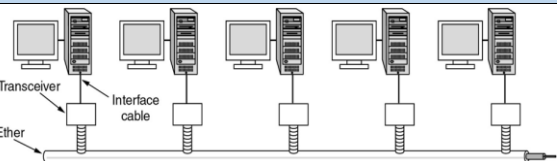
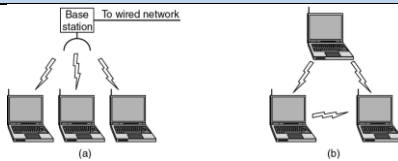
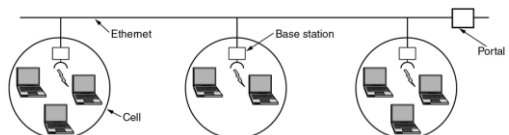
[5. Reference Models]

OSI 7계층	<p>1~3 계층은 subnet으로 하며 이 부분에서는 subnet protocol을 이용한다.</p>																
TCP/IP	<p>Not present in the model</p>	<p>TCP/IP 모델에서는 OSI 모델에 존재하는 <u>5, 6 계층의 기능이 7계층(application)으로 통합</u>되었다.</p> <p>→ <u>구현의 용이성</u> 때문</p> <p>→ 7개 계층을 이용하면 체계적이지만 <u>복잡해서 성능이 떨어짐</u></p>															
Protocols Networks	<table border="1"> <thead> <tr> <th>Layer (OSI Model)</th> <th>Protocols</th> <th>Networks</th> </tr> </thead> <tbody> <tr> <td>Application</td> <td>TELNET, FTP, SMTP, DNS</td> <td></td> </tr> <tr> <td>Transport</td> <td>TCP, UDP</td> <td></td> </tr> <tr> <td>Network</td> <td>IP</td> <td></td> </tr> <tr> <td>DataLink+Physical</td> <td></td> <td>ARPANET, SATNET, Packet radio, LAN</td> </tr> </tbody> </table>	Layer (OSI Model)	Protocols	Networks	Application	TELNET, FTP, SMTP, DNS		Transport	TCP, UDP		Network	IP		DataLink+Physical		ARPANET, SATNET, Packet radio, LAN	
Layer (OSI Model)	Protocols	Networks															
Application	TELNET, FTP, SMTP, DNS																
Transport	TCP, UDP																
Network	IP																
DataLink+Physical		ARPANET, SATNET, Packet radio, LAN															

<Critiques of OSI and TCP/IP Model>

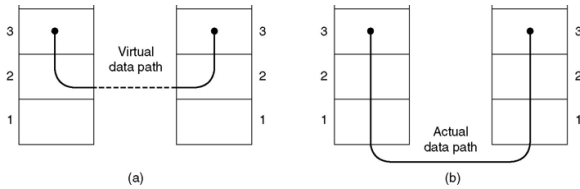
OSI	타이밍, 기술, 구현, 정책의 문제점
TCP/IP	<p>→ 서비스, 인터페이스, 프로토콜이 구분되지 않음</p> <p>→ 일반적인 모델이 아님</p> <p>→ Host-to-network 'layer'는 진짜 layer가 아님</p> <p>→ 물리 계층과 데이터 링크 계층에 대한 설명이 없음</p>

[6. Example Networks]

ARPANET	NSFNET
 <p>(a)의 문제점은 <u>중앙 교환기가 고장 나면 통신이 불가능</u>하다는 것이다. -> 이러한 문제를 해결하기 위하여 (b) 형태의 ARPANET 등장</p>	 <p>○ NSF Supercomputer center ⊗ NSF Midlevel network ● Both</p>
Internet	ATM Virtual Circuits
 <p>→ <u>인터넷 서비스 공급자(ISP)</u>가 제공 (KT, SK, LG 등)</p> <p>■ <u>ISP의 망에는 1~3계층</u>, User의 망에는 1~7계층 또는 1~5계층 존재</p> <p>→ <u>회사에서는 LAN</u>을 사용</p>	 <p>물리적인 회선을 모방한 <u>가상 회선을 이용하여 Source와 Destination을 연결</u>하는 방식</p> <p>→ <u>Connection oriented</u> 방식</p> <p>Bytes 5 48</p> <p>Header User data</p> <p>기본적인 <u>전송 단위는 cell</u>이다.</p>
Ethernet	Wireless LANs
 <p><u>Bus 형태</u>로 되어 있음</p>	 <p>무선랜은 <u>Access Point</u>를 중심으로 하여 단말기를 연결함 (Ad-hoc에서는 Peer-to-Peer)</p>  <p>(Multi-cell 802.11 Network) – 간섭을 방지하기 위해 각각의 셀마다 주파수 채널이 다름</p>

[1. The Data Link Layer]

<Service Provided to Network Layer>



(a) Virtual communication.

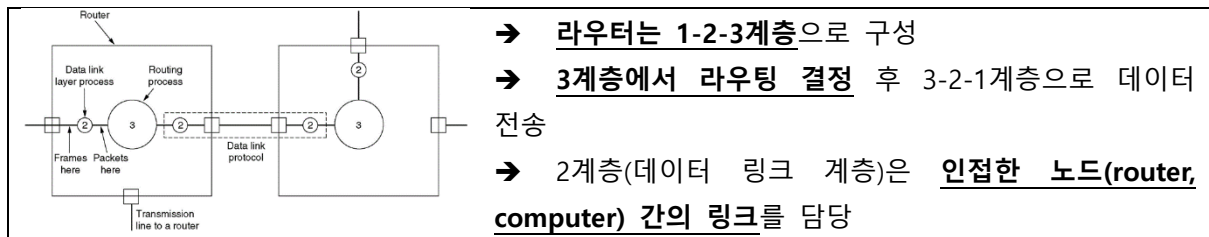
(b) Actual communication.

<Service Type> - 구분 기준: ACK의 유무, Connection Type

Acked CO (connection oriented)	ATM, FR, X.25
Acked CL (connectionless)	WLAN
Unacked CL (connectionless)	Ethernet LAN

→ Local area network는 wide area network보다 안정적

<서비스 구현>



<Network Packet vs. Link Frame>

Sender가 패킷을 나누어서
Receiver에 전송한다.

→ 데이터 링크 계층의 프레임

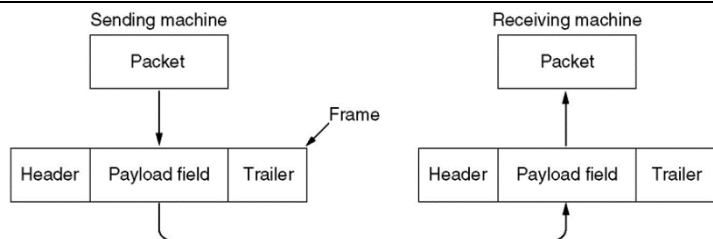
임: header(제어정보: 자신의 주소+상대방 주소 등),

Payload field, Trailer

■ trailer에는 오류 검출용 채널 코드(CRC)가 들어감

◆ CRC는 하드웨어로 검출됨

■ Payload 다음에 trailer가 붙는 이유: header와 payload를 알아야지만 CRC를 계산할 수 있으므로



<데이터 링크 계층의 기능>

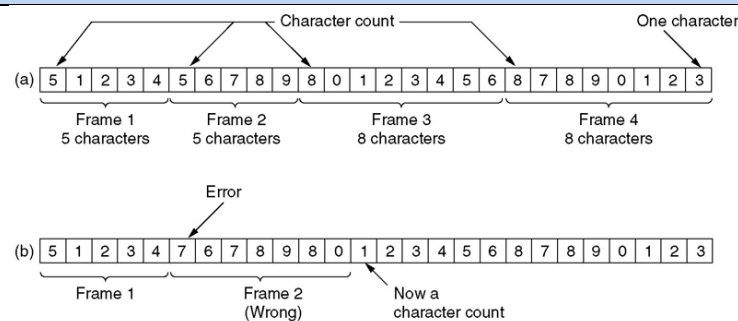
Framing, Error Control, Flow Control, Link Management, Addressing

<데이터 링크 계층의 프로토콜>

Unrestricted Simplex Protocol, Simplex Stop-and-Wait Protocol, Simplex Protocol for a Noisy Channel

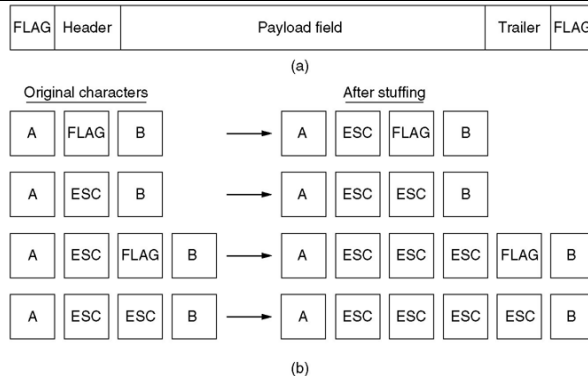
[2. Framing]

프레임의 시작 부분에 character count를 삽입



- (a) 각 프레임의 시작 부분에 해당 프레임의 character count 삽입
- (b) 오류 발생 시 이후의 전체가 깨짐 (다음 프레임으로 계속 전파)

앞뒤로 시작과 끝을 나타내는 flag 사용 (문자 또는 bit 단위)



실제 데이터 부분에 flag가 들어가면 오류가 발생한다.

- **[Sol: stuffing]** Flag 또는 escape 문자가 중간에 들어가면 escape 문자를 앞에 붙여서 표시한다.

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Stuffed bits

Bit stuffing: flag = 01111110

- (a) The original data.
- (b) The data as they appear on the line.
- (c) The data as they are stored in receiver's memory after destuffing.

[flag: 01111110]

비트 단위 stuffing의 경우 1이 6개 나오면 flag이므로 1이 5개 나올 때마다 0을 넣어서 flag와 구분한다.

[3. Error Control]

Channel coding	EDC (Error D etection Code), ECC (Error C orrection Code): FEC
ARQ (automatic repeat request)	SW (Stop-and-Wait), GBN (Go-back-N), SR (Selective-Repeat)

→ ARQ는 자동으로 재전송을 하는 방식이다.

<Error Detection Code: CRC>

CRC: 다항식 코드(Polynomial code)에 속하며, **G(x) : Generator Polynomial**을 이용하여 생성한다.

→ 송신기와 수신기는 같은 G(x)를 이용하여 오류를 검출한다.

- CRC 알고리즘은 **module 2**를 계산한다.
- Receiver에서는 G(x)로 나눈 결과를 계산하여 나머지가 0이면 오류가 없고, 나머지가 있으면 오류가 있다고 판단한다.
- **CRC의 생성 개수**는 Generator의 차수와 같으며, 메시지에 **00**이 **CRC의 개수만큼** 추가된다.

[Frame=**1101011011**, Generator=**10011**] -> **Msg=11010110110000**

1101011011**0000**(2) / 10011(2) = 1100001010(2)(몫) ... **1110**(2)(나머지) 이므로 Transmitted frame = **11010110111110**(2)

<Stop-and-Wait ARQ>

Stop-and-Wait ARQ: 통신환경에 따라 **Simplest(단방향 통신)** 또는 **Half duplex transmission(반이중 통신)** 환경에서 사용

- **ACK 유형:** **Positive(정상)/Negative(오류)** ACK 또는 **Positive ACK Only**
- **1bit의 sequence number**를 사용 (SW ARQ에서는 **1개의 프레임을 전송한 후 ACK가 도착할 때까지 wait**하므로) – Data/ACK frame loss

<Continuous ARQ>

SW-ARQ	효율성 문제 발생
Full duplex transmission	
GBN-ARQ	오류 발생 시 그 프레임과 이전 프레임, 즉 <u>버퍼의 모든 프레임 재전송</u> → 구현이 쉬우므로 사용
SR-ARQ	해당 <u>오류가 발생한 프레임만 선택적으로 전송</u> (더 효율적임) → 구현이 복잡함 (buffering, rendering)

[4. Flow Control]

Flow Control: 송수신기 사이에 성능 차이가 날 때 데이터 흐름을 제어하기 위해 사용. 특히 송신기가 더 빠를 때 송신기의 속도를 늦추기 위하여 사용

<Sliding Window Protocol: for Flow Control>

Sliding Window with SW-ARQ	Window size가 <u>1프레임</u> 으로 제한
Sliding Window with Continuous ARQ	Window size = <u>$N > 1$ frames</u> 수신 버퍼, 송수신 윈도우
Window Size: ACK을 받지 않은 상태에서 프레임을 몇 개까지 전송할 수 있는가? → 프레임 <u>전송</u> 시 <u>Sender의 right of window</u> 증가, <u>ACK</u> 시 <u>left of window</u> 증가 → 프레임 <u>수신</u> 시 <u>Receiver의 window</u> 가 그 크기만큼 이동 → Max window size = f(sequence number bits)	

[5. Algorithms]

Unrestricted Simplex Protocol	Simple Stop-and-Wait Protocol
Sender1: <ol style="list-style-type: none"> 1. Network layer에서 데이터 획득 2. 그 데이터를 s에 복사 3. Physical layer로 데이터 전송 Receiver1: <ol style="list-style-type: none"> 1. Physical layer에서 프레임 획득 2. Network layer로 데이터 전송 	Sender2: <ol style="list-style-type: none"> 1. Network layer에서 데이터 획득 2. 그 데이터를 s에 복사 3. Physical layer로 데이터 전송 4. 다음 이벤트 발생 시까지 대기 Receiver2: <ol style="list-style-type: none"> 1. 다음 이벤트 발생 시까지 대기 2. Physical layer에서 프레임 획득 3. Network layer로 데이터 전송 4. Physical layer로 dummy frame 전송하여 sender를 깨움
A Simplex Protocol for a Noisy Channel	
Sender3: Next_frame_to_send를 0으로 초기화 1번째 packet을 fetch함 <ol style="list-style-type: none"> 1. Physical layer로 데이터 전송 2. 타이머 시작 3. Frame_arrival, cksum_err 또는 timeout 대기 4. 프레임이 도착하면 <ol style="list-style-type: none"> A. Physical layer로부터 ACK 획득 B. ACK==next_frame_to_send이면 <ol style="list-style-type: none"> i. 타이머 중지 ii. Network Layer에서 다음에 전송할 데이터 획득 iii. Next_frame_to_send를 증가 	Receiver3: Frame_expected를 0으로 초기화 <ol style="list-style-type: none"> 1. Frame_arrival 또는 cksum_err 대기 2. 프레임이 도착하면 <ol style="list-style-type: none"> A. Physical layer로부터 프레임 획득 B. R.seq==frame_arrival이면 <ol style="list-style-type: none"> i. Network Layer로 데이터 전송 ii. Frame_expected를 증가 C. S.ack = 1-frame_expected (어떤 프레임이 ACK되었는가?) D. ACK을 physical layer로 전송

[1. Sliding Window Protocol]

	A	초기 상태
	B	첫 번째 프레임이 전송 된 직후
	C	첫 번째 프레임이 수신 된 직후
	D	첫 번째 프레임의 ACK 가 Sender에 도착 한 직후

<Algorithms>

A One-Bit Sliding Window Protocol

1. **Frame_arrival**, **cksum_err** 또는 **timeout** 대기
2. **프레임이 도착하면**
 - A. **Physical layer**로부터 프레임 수신
 - B. **R.seq==frame_expected**이면, 즉 **sequence number**가 프레임 번호와 같으면
 - i. Network Layer로 데이터 전송
 - ii. **Frame_expected**의 값을 증가
 - C. **R.ack==next_frame_to_send**이면, 즉 **ACK**이 다음에 송신할 프레임 번호와 같으면
 - i. Timer를 정지한다.
 - ii. Network layer로부터 packet을 받는다.
 - iii. **Next_frame_to_send**의 값을 증가시킨다.
3. Sequence number를 증가시킨다.
4. **S.ack = 1 - frame_expected**, 즉 마지막으로 수신한 프레임의 sequence number로 한다.
5. Physical layer로 프레임을 전송한다.
6. Timer를 시작한다.

<One-bit sliding window protocol 예시>

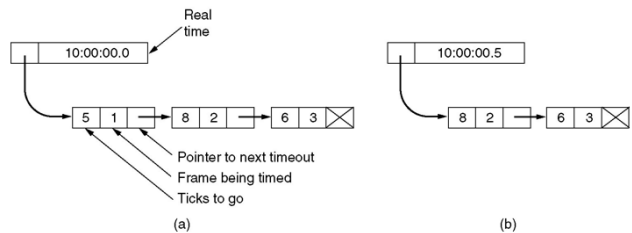
	(a) 정상적인 경우: 양방향 통신
	(b) 비정상적인 경우
	→ B sends (0,0,B0): (0,1,B0)에 대한 ACK가 없으므로 재전송
	→ A sends (0,0,A0): (0,1,A0)에 대한 ACK가 없으므로 재전송
	→ 마찬가지로 (1,0,A1), (1,0,B1)도 각각 (1,1,A1), (1,1,B1)로 재전송
	→ (X,Y,Z)에서 Y가 1로 시작하는 이유: 초기값을 0으로 설정하면 혼동되므로 1로 설정함

*: 패킷 수신

[2. Continuous ARQ]

ARQ	성능	구현	Window Size
Stop-and-Wait (SW)	떨어짐 (window size=1이므로 <u>한번에 하나씩만 전송</u>)		1
Go-back-N (GBN)	떨어짐 (오류가 발생한 프레임 다음의 프레임을 잘 받았는데도 <u>N개만큼 무시하므로 불필요한 재전송 필요</u>)	쉬움	$2^n - 1$
Selective Repeat (SR)	 좋음 (<u>Error가 발생한 프레임만 재전송</u> 하므로)	복잡함 (<u>NAK, buffer, 순서 맞추는 기능</u>)	$\frac{2^n}{2}$
	→ <u>예상 외의 프레임이 들어오면 NAK</u> 을 보낸다. → <u>순서가 다른 프레임을 discard하지 않고 buffer에 저장</u> 하고, 나중에 원하는 프레임이 들어오면 <u>순서를 맞춰야</u> 한다. → 최근에는 구현이 복잡해도 성능이 좋은 SR-ARQ 사용		

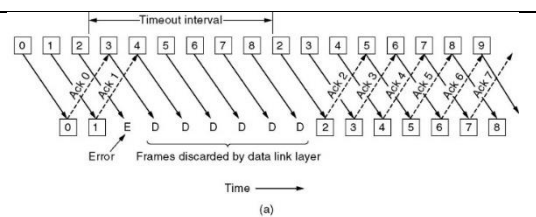
타이머 구현: 프레임을 보낼 때마다 타이머를 프레임 단위로 작동시키면 타이머가 너무 많아지므로 독립적으로 구현하는 것은 오버헤드가 크다. 따라서 시간의 차이를 이용, 포인터로 링크하는 형태로 구현한다.



[Example 1]

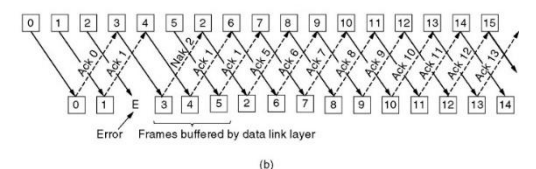
(a) GBN-ARQ 프로토콜

- Sender 측에서는 **0, 1번을 보냈고 이에 대한 ACK**를 받는다. 이후 2~8번을 보내고 2번에서 error가 발생하여, **3~8번은 2가 먼저 들어와야 하므로 discard**된다.
- Timeout이 되면 **N개만큼 back하여 2번으로 되돌아간다**.



(b) SR-ARQ 프로토콜

- 이번에는 **오류가 발생한 2번만 Error가 발생**하고 나머지는 버퍼에 저장된다.
- 3번 프레임이 전송되었을 때 **예상했던 2번 프레임이 아니므로 NAK**을 보낸다.
- 나중에 **2번 프레임이 들어오면 순서를 맞춘다**.



[Example 2]

Sender	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
Receiver	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
	(a)	(b)	(c)	(d) (seqnum : 3 bits)
Window Size = 7 (a) (b)			Window Size = 4 (c) (d)	
0~6의 7개 프레임이 send and receive되었지만 ACK되지 않은 경우			0~3의 4개 프레임이 send and receive되었지만 ACK되지 않은 경우	
→ Sender 측은 ACK를 받지 못했으므로 0~6을 재전송 → Receiver 측은 ACK가 손실된 것을 모르므로 7~0~5를 대기 → 이때 <u>재전송한 데이터 0~5와 새 데이터 0~5를 혼동할 수 있음</u>			→ Sender 측은 ACK를 받지 못했으므로 0~3을 재전송 → Receiver 측은 4~7을 대기 → <u>Sequence number가 중복되지 않음</u>	

<Algorithms>

Sliding Window Protocol Using Selective Repeat

Send_frame:

프레임을 physical layer로 보낸다.
 Data frame이면 타이머를 시작한다.
 ACK timer를 중지한다.

Protocol6:

1. Event를 대기한다.
2. Network_layer_ready 이벤트가 발생하면
 - A. Window를 확장한다.
 - B. Network layer로부터 패킷을 받는다.
 - C. Send_frame(data);
 - D. Upper window edge를 늘린다.
3. Frame_arrival 이벤트가 발생하면
 - A. Physical layer로부터 프레임을 받는다.
 - B. Data frame이면
 - i. (undamaged) Frame_expected가 아니고 no_nak이면 send_frame(NAK)

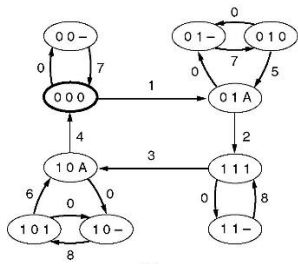
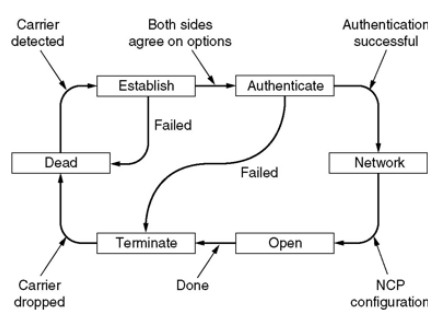
ii. 프레임이 어떤 순서로든 도착할 수 있으면

1. Buffer의 상태를 full로 설정
2. Buffer에 데이터 추가
3. 프레임을 pass하고 receiver의 lower, upper edge를 증가시킨다.
4. ACK timer를 시작한다.
- C. NAK이면 일련번호 확인 후 재전송
- D. 프레임을 버퍼에서 뺀다.
- E. 타이머를 중지한다.
4. Cksum_err 이벤트가 발생하면
 - A. NAK이 아니면 damaged frame이므로 Send_frame(NAK);
5. timeout이면
 - A. Send_frame(data);
6. ack_timeout이면
 - A. Send_frame(ACK);

→ 버퍼에 프레임을 계속 가지고 있는 이유는 재전송하기 위해서이다.

[3. Protocol Verification]

Protocol Verification: 프로토콜에 논리적 오류가 있는지 판단한다.

Finite State Machine	<div>상태 값: (일련번호, 수신번호, 채널상태)</div> <div>➔ 상태 및 상태 간의 transition을 정의</div> <div>➔ 채널상태: n(n번 프레임), A(ACK), -(아무것도 없음)</div> <div><div></div><div><table><tr><th>Transition</th><th>Who runs?</th><th>Frame accepted</th><th>Frame emitted</th><th>To network layer</th></tr><tr><td>0</td><td>-</td><td>(frame lost)</td><td>-</td><td>-</td></tr><tr><td>1</td><td>R</td><td>0</td><td>A</td><td>Yes</td></tr><tr><td>2</td><td>S</td><td>A</td><td>1</td><td>-</td></tr><tr><td>3</td><td>R</td><td>1</td><td>A</td><td>Yes</td></tr><tr><td>4</td><td>S</td><td>A</td><td>0</td><td>-</td></tr><tr><td>5</td><td>R</td><td>0</td><td>A</td><td>No</td></tr><tr><td>6</td><td>R</td><td>1</td><td>A</td><td>No</td></tr><tr><td>7</td><td>S</td><td>(timeout)</td><td>0</td><td>-</td></tr><tr><td>8</td><td>S</td><td>(timeout)</td><td>1</td><td>-</td></tr></table></div></div>	Transition	Who runs?	Frame accepted	Frame emitted	To network layer	0	-	(frame lost)	-	-	1	R	0	A	Yes	2	S	A	1	-	3	R	1	A	Yes	4	S	A	0	-	5	R	0	A	No	6	R	1	A	No	7	S	(timeout)	0	-	8	S	(timeout)	1	-
Transition	Who runs?	Frame accepted	Frame emitted	To network layer																																															
0	-	(frame lost)	-	-																																															
1	R	0	A	Yes																																															
2	S	A	1	-																																															
3	R	1	A	Yes																																															
4	S	A	0	-																																															
5	R	0	A	No																																															
6	R	1	A	No																																															
7	S	(timeout)	0	-																																															
8	S	(timeout)	1	-																																															
HDLC	<div>Bits888≥0168</div> <div><table><tr><td>0 1 1 1 1 1 1 0</td><td>Address</td><td>Control</td><td>Data</td><td>Checksum</td><td>0 1 1 1 1 1 1 0</td></tr></table></div> <div><table><tr><td>Flag</td><td colspan="3">01111110 (8 bits)</td></tr><tr><td>Address</td><td colspan="3">목적지 주소 (multipoint일 때 목적지를 구분하기 위하여 사용)</td></tr><tr><td rowspan="3">Control</td><td>Information</td><td>0+3(seq)+1(P/F)+3(Next)</td><td>용도</td></tr><tr><td>Supervisory</td><td>10+2(type)+1(P/F)+3(Next)</td><td>Flow/Error control 용 ACK, NAK</td></tr><tr><td>Unnumbered</td><td>11+2(type)+1(P/F)+3(Modifier)</td><td>링크 관리(개설, 끊기, 테스트 등)</td></tr><tr><td colspan="4">P/F: 보통은 0, 비상 상황에서는 1</td></tr><tr><td colspan="4">Type: ACK/NAK 구분, Next: ACK 번호</td></tr><tr><td>Data</td><td colspan="3">0비트 이상의 데이터</td></tr><tr><td>Checksum</td><td colspan="3">오류 검사에 사용</td></tr></table></div>	0 1 1 1 1 1 1 0	Address	Control	Data	Checksum	0 1 1 1 1 1 1 0	Flag	01111110 (8 bits)			Address	목적지 주소 (multipoint일 때 목적지를 구분하기 위하여 사용)			Control	Information	0+3(seq)+1(P/F)+3(Next)	용도	Supervisory	10+2(type)+1(P/F)+3(Next)	Flow/Error control 용 ACK, NAK	Unnumbered	11+2(type)+1(P/F)+3(Modifier)	링크 관리(개설, 끊기, 테스트 등)	P/F: 보통은 0, 비상 상황에서는 1				Type: ACK/NAK 구분, Next: ACK 번호				Data	0비트 이상의 데이터			Checksum	오류 검사에 사용												
0 1 1 1 1 1 1 0	Address	Control	Data	Checksum	0 1 1 1 1 1 1 0																																														
Flag	01111110 (8 bits)																																																		
Address	목적지 주소 (multipoint일 때 목적지를 구분하기 위하여 사용)																																																		
Control	Information	0+3(seq)+1(P/F)+3(Next)	용도																																																
	Supervisory	10+2(type)+1(P/F)+3(Next)	Flow/Error control 용 ACK, NAK																																																
	Unnumbered	11+2(type)+1(P/F)+3(Modifier)	링크 관리(개설, 끊기, 테스트 등)																																																
P/F: 보통은 0, 비상 상황에서는 1																																																			
Type: ACK/NAK 구분, Next: ACK 번호																																																			
Data	0비트 이상의 데이터																																																		
Checksum	오류 검사에 사용																																																		
PPP (Point to Point Protocol)	<div>Bytes1111or 2Variable2 or 41</div> <div><table><tr><td>Flag 01111110</td><td>Address 11111111</td><td>Control 00000011</td><td>Protocol</td><td>Payload</td><td>Checksum</td><td>Flag 01111110</td></tr></table></div> <div>➔ Payload는 상위 계층의 패킷</div> <div>➔ Home Personal Computer에서 주로 사용</div> <div>➔ HDLC(bit level)과 달리 byte level, 즉 문자 단위의 flag임</div> <div></div> <div><table><tr><td>Establish</td><td>Network</td></tr><tr><td>Link level에 연결 (LCP: link control protocol 사용)</td><td>Network level 연결</td></tr></table></div>	Flag 01111110	Address 11111111	Control 00000011	Protocol	Payload	Checksum	Flag 01111110	Establish	Network	Link level에 연결 (LCP: link control protocol 사용)	Network level 연결																																							
Flag 01111110	Address 11111111	Control 00000011	Protocol	Payload	Checksum	Flag 01111110																																													
Establish	Network																																																		
Link level에 연결 (LCP: link control protocol 사용)	Network level 연결																																																		

[1. MAC]

MAC(Medium Access Control Sublayer): 전송 매체를 공유하기 위하여 필요한 프로토콜

→ 1:1 독점 형태에서는 불필요, LAN이라는 근거리망(Ethernet)이 들어오면서 필요하게 됨

<채널 할당 방식>

Static Channel Allocation	<p><u>다중 채널을 각 사용자마다 구현</u></p> <p>→ FDMA/TDMA/CDMA 방식</p> <p>→ 데이터가 있을 때뿐만 아니라 전체 주기 동안 채널이 정적으로 할당됨</p> <p>→ 문제점: bursty(간헐적 트래픽 패턴), 효율이 떨어짐(사용하지 않아도 채널이 할당)</p>
Dynamic Channel Allocation	<p>필요할 때(데이터 생성, 전송)만 채널을 동적으로 할당</p> <p>→ Random Access: 랜덤하게 경합을 해서 채널을 할당</p>

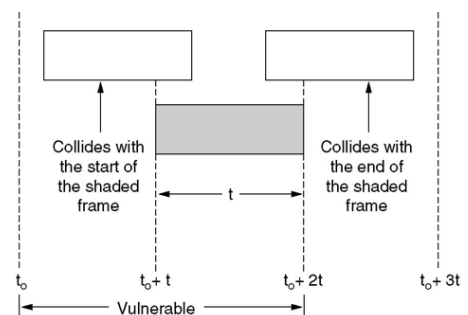
[2. ALOHA 계열 MAC 프로토콜]

Pure ALOHA
<p>데이터가 있으면 <u>일단 보내고 보는 방식</u></p> <p>→ 성공적으로 전송되면 종료, 충돌이 생기면 random delay 후 재전송</p> <p>→ 임의의 시간에 slot 전송 가능</p> <p>→ 성능 평가 기준: Vulnerable Period (충돌이 일어날 수 있는 기간)</p>
Slotted ALOHA
<p><u>Slot 단위의 전송만 가능</u>한 ALOHA 방식</p> <p>→ Pure ALOHA의 throughput이 떨어지는 문제 해결 가능</p> <p>→ 프레임 전송은 <u>항상 slot의 경계에서만 가능</u></p>

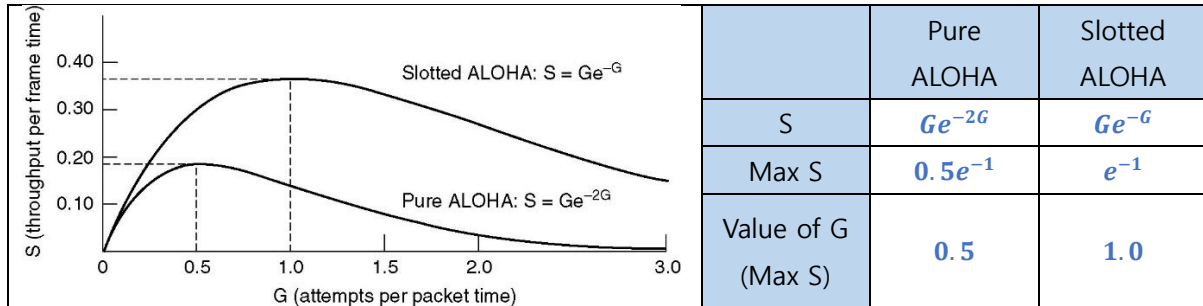
<성능 평가: Poisson Distribution>

	Pure ALOHA	Slotted ALOHA
x (사건 횟수)	$P(X = x) = \frac{e^{-\lambda t} \cdot (\lambda t)^x}{x!}$	
P_s (성공 확률)	$P_s = P(X = 0) = \frac{e^{-\lambda t}}{x!}$	
λ (초당 프레임수)	$\lambda = \frac{G}{t_x}$	
S (throughput)	$S = G \cdot P_s$	
t_v (vulnerable period)	$t_v = 2t_x$	$t_v = t_x$

G 는 Offered Load, t_x 는 프레임 전송 시간



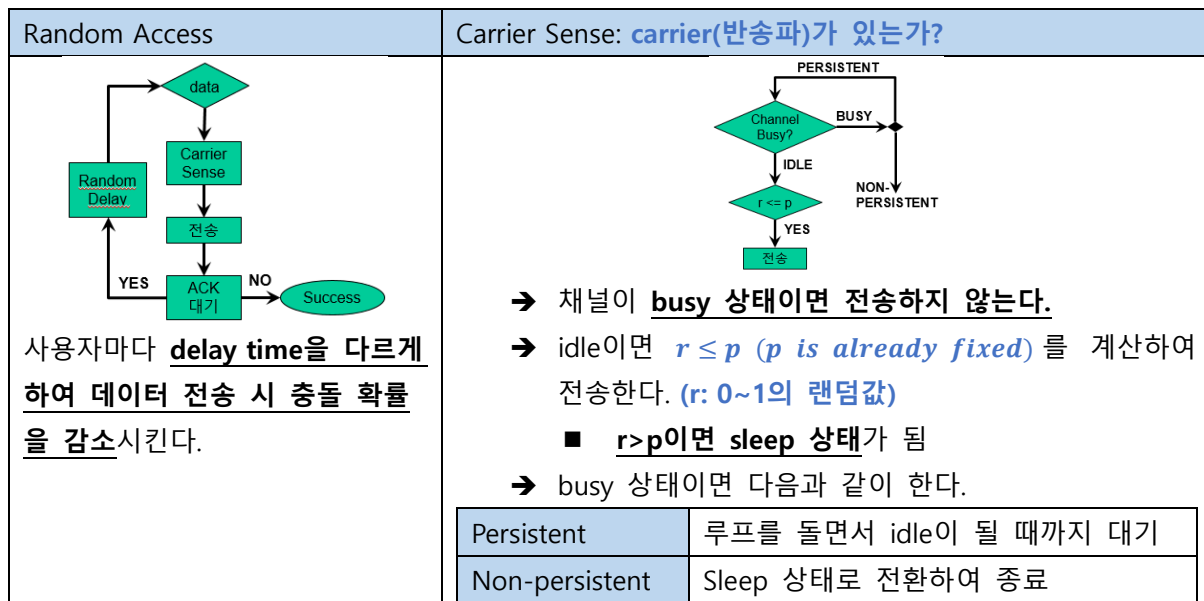
따라서 Pure ALOHA와 Slotted ALOHA의 성능은 다음과 같다.



[3. CSMA 계열 MAC 프로토콜]

CSMA: CS(Carrier Sense)를 이용하는 MAC 프로토콜

- Slot ALOHA도 효율이 좋지 않으므로 도입
- CS(Carrier Sense)가 있으면 CSMA, 그렇지 않으면 ALOHA라고 할 수 있음



<Persistent vs. Non-Persistent CSMA>

	Persistent CSMA	Non-persistent CSMA
예시	P=1 이면 Ethernet (CSMA-CD, 충돌검출가능) P=0 이면 WLAN (충돌 최소화)	WSN
사용	루프를 돌면서 지속적으로 채널 상태를 확인, 전력소모 많음	Sleep상태로 전환하여 종료하므로 전력 소모 적음, 임베디드 디바이스에서 사용

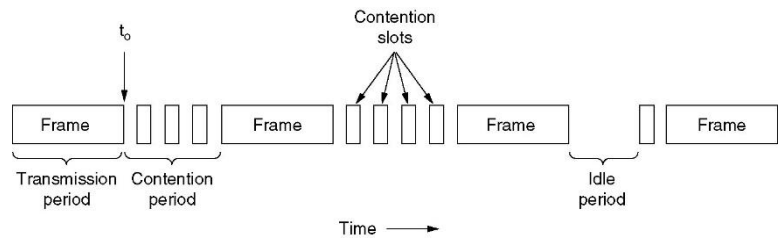
- $p > 0$ 이면 p 의 값이 작아질수록 throughput이 증가한다.
- 1-persistent는 Ethernet에서, Non-persistent는 무선(충돌 최소화)에서 사용한다.

CSMA-CD (CSMA with Collision Detection): 1-persistent CDMA의 성능을 보완하기 위해 사용

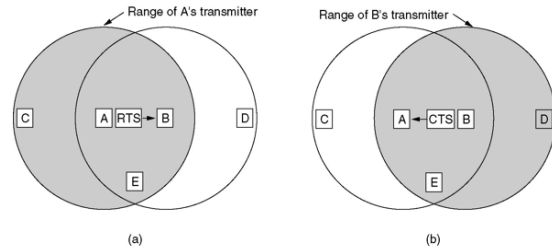
→ 충돌이 생기면 전송을 중단하고 나중에 재전송

→ 충돌 검출을 위해 아날로그 회로 사용, 신호의

세기가 크면 충돌이 발생했다고 판단



Wireless LAN Protocol: (전송 범위의 문제) A의 전송 범위 안에 B, C, E가 들어가므로 A는 B, C, E에 전송할 수 있다.

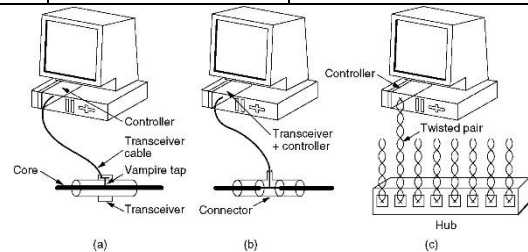


[4. IEEE 802 프로토콜과 Ethernet]

IEEE 802.2	IEEE 802.3	IEEE 802.11	IEEE 802.15	IEEE 802.16
LLC (데이터 링크 프로토콜)	CSMA/CD (Ethernet)	Wireless LAN	Wireless PAN	Wireless MAN

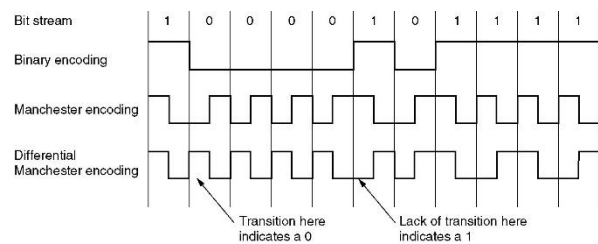
<이더넷 케이블링>

(a), (b)는 bus type, (c)는 star type이다.



Encoding: 디지털 데이터를 전압 펄스 형태로 보낸다.

Binary encoding	원래 bit를 그대로 사용
Manchester encoding	각 bit에서 translation 발생 (장점: 동기를 위한 별도의 clock line 없이 송수신기의 인코딩 방식으로 translation을 이용하여 동기화)
Differential Manchester encoding	각 bit의 시작점에서 해당 bit가 1이면 연속, 0이면 불연속이다.

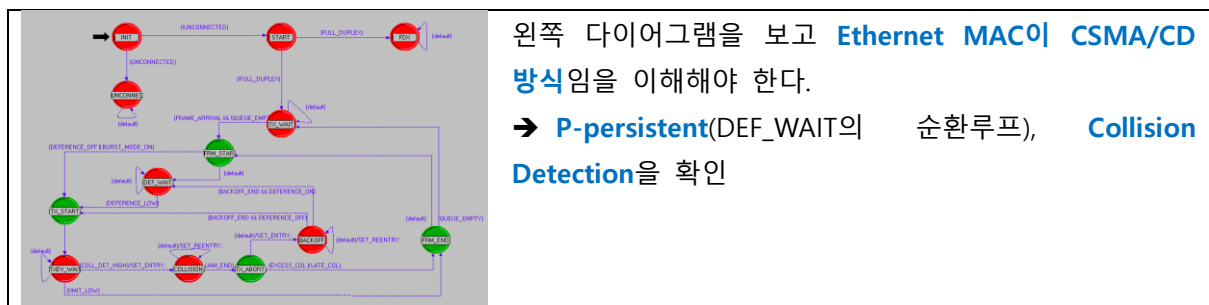
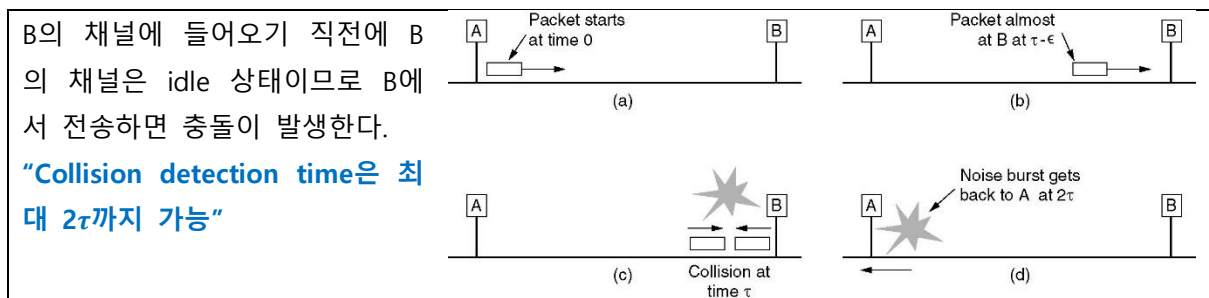


<Ethernet MAC Sublayer Protocol>

DIX Ethernet	Bytes	8	6	6	2	0-1500	0-46	4
(a)	Preamble	Destination address	Source address	Type	Data	Pad	Check-sum	
Preamble: 처음 시작할 때의 프레임 동기를 설정하기 위하여 지정								
IEEE 802.3								
	Preamble	SOF	Destination address	Source address	Length	Data	Pad	Check-sum
SOF: 프레임의 시작을 나타냄								

Pad: null data로, 데이터의 크기가 **46바이트 미만일 때 46바이트 이상이 되게 만들기 위하여** 사용

- ➔ Data가 46바이트 이상이어야 하는 이유: **CSMA/CD Mac Protocol의 최소 프레임 크기 제한**(Preamble를 제외한 64바이트)을 지키기 위하여
- ➔ 다음 실험에서 **64바이트 이상이어야 CSMA/CD 프로토콜에 의한 collision detection이 가능**하기 때문이다.



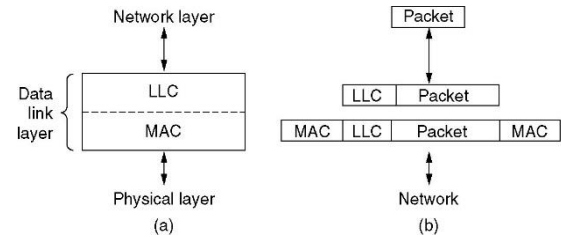
<Switched/Fast/Gigabit Ethernet>

Switched Ethernet	최근 Ethernet의 추세 (단말기들이 star 형태 로 연결)
Fast Ethernet	원래 10Mbps 였는데 802.3u에서 100Mbps 로 증가, 동축케이블 대신 twisted pair 또는 광케이블 사용
Gigabit Ethernet	Two-station 또는 multi-station, 1Gbps 로 속도 증가, 광케이블 중심

IEEE 802.2(LLC): 데이터 링크 계층의 5대 기능 수행

→ MAC이 추가되어 데이터 링크 계층이 LLC, MAC으로 나누어짐

→ 패킷이 들어오면 LLC, MAC에 대해 각각의 헤더가 있으며, 물리 계층으로 들어가기 전에 CRC를 이용한 오류 검출을 위한 trailer가 있음



802.11 Protocol Stack	Data Link Layer에서 공통 LLC를 사용하며, MAC은 CSMA/CA 공통이다. (같은 CSMA이지만 /CD, /CA간 호환은 안됨) → 물리계층은 서로 다른 기술 이용
802.11 MAC Sublayer Protocol	Hidden/exposed station 문제 (전송 가능 범위, 무선 환경에서의 성능저하) <div style="text-align: center;"> <p>(a) The hidden station problem. (b) The exposed station problem.</p> </div>

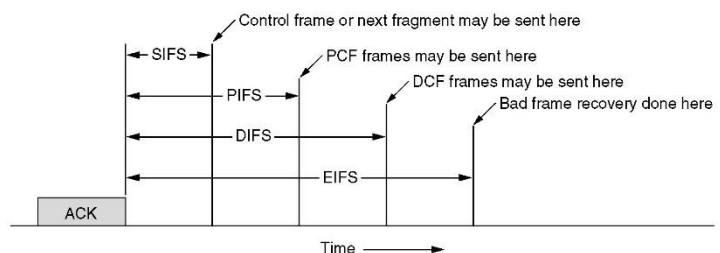
[Hidden/Exposed station problem]

Hidden	C의 전송 범위 밖에 A가 있으므로, A가 Carrier Sensing을 해도 idle로 판단되므로 C가 B에 데이터를 보내면 충돌 가능
Exposed	A에서 B로 데이터를 보내고 있는데 B는 이것 때문에 자신의 채널이 busy하다고 판단하므로 C로 데이터를 보내지 않음 (그러나 보내도 충돌하지 않음)

→ 이 문제를 해결하기 위하여 RTS, DTS 프레임을 교환하여 데이터 전송 전에 주변 node에 알려준다. (NAV: 전송주기)가 지난 후 C가 프레임을 전송한다.

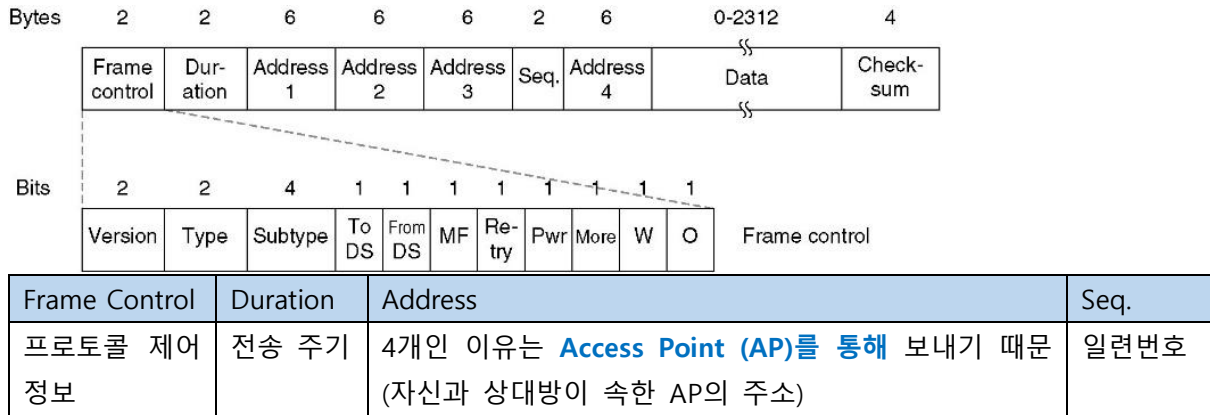
→ 무선 네트워크는 Collision Detection이 불가능하므로 Collision Avoid를 하며, 이는 신호의 세기에 따라 판단한다.

[Interframe Space] 프레임 간의 간격은 SIFS(Control frame에서 사용) < PIFS(PCF 방식에서 사용) < DIFS(DCF 방식에서 사용) < EIFS(특수한 경우 사용)

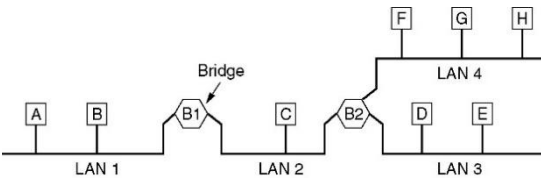


→ 간격이 짧을수록 우선권이 높아서 먼저 전송된다.

[802.11 Frame Structure]



[5. Internetworking]

물리 계층	Repeater, Hub
데이터 링크 계층	Bridge, Switch <div> → Bridge는 서로 다른 종류의 LAN 간의 연결에 사용된다.  </div> → LAN type마다 format이 다르므로 bridge를 통해 format을 맞춰야 한다. → 이중 브릿지를 사용할 수 있는데, 이때 사이클이 발생하여 자원 낭비가 발생할 수 있다. <div> ■ 이를 해결하기 위해 Spanning Tree 알고리즘 적용 </div>
네트워크 계층	Router
전송 계층부터	Gateway

[1. Network Service Model]

Network Service Model의 요구사항: bandwidth, inter-packet timing 방지, loss-free 전송, in-order 전송, congestion feedback to sender

Network Service Model의 유형: virtual circuit 또는 datagram

<Connection Oriented 서비스: Virtual Circuit 형태>

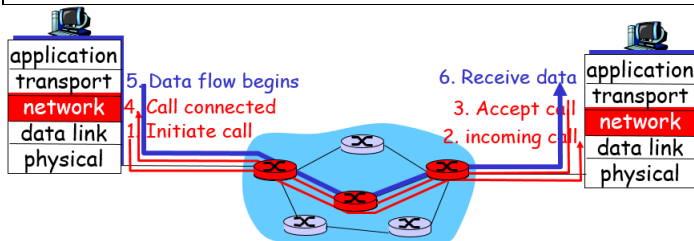
- 데이터를 보내기 전에 **connection**을 맺는다.
- **Source-to-Dest path**는 telephone circuit처럼 **경로를 정하여** 사용한다.
- 성능은 **좋은 편**이며(여러 자원을 할당할 수 있으므로), **Source부터 Destination까지 connection**이 필요하다.

1. **call setup**을 하여 data가 흐르게 한다.
2. 가상회선 구분을 위해 **각 packet은 VC identifier를 carry**한다. (destination host ID가 아님)
3. **Source-dest path**는 각각의 연결에 대하여 **상태를 관리**해야 한다.
 - 전송 계층 연결은 2개의 end system만을 포함한다.
4. 실제 circuit과 같은 성능을 위해 **link, router 자원(bandwidth, buffer)이 VC에 의해 할당**되어야 한다.

<Virtual circuits: Signaling protocols>

Signaling protocol: setup, teardown VC 유지, ATM, frame-relay, X.25에서 사용하며 현재 인터넷에서는 사용하지 않는다.

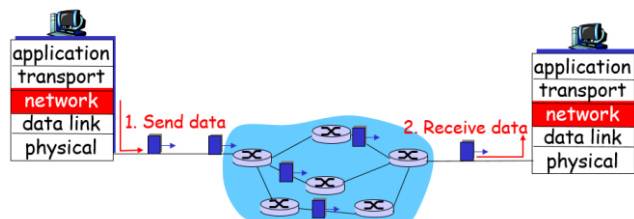
1. call setup을 위하여 **call 요청**을 한다.
2. call 요청이 **accept되면 양쪽으로 signaling을 통해 connection**이 생성된다.
3. 한번 setup되며 종료될 때까지 유지되며, **데이터는 정해진 virtual circuit을 통해 이동**한다.



<CL Service: Datagram (인터넷)>

→ **Call setup이 없다.**

→ 라우터는 end-to-end 연결에 대한 상태가 없다. 즉 **network level의 connection** 개념이 없다.



→ 패킷은 **목적지 주소를 이용하여 포워딩**된다. 따라서 **서로 같은 Source, Destination을 갖는 패킷의 이동 경로가 다를 수 있다.**

<Datagram or VC network>

Internet		ATM
Connectionless 방식으로 elastic(flexible) 하다. → 우회 경로를 찾는다. → 타이밍 요구 사항을 만족하는 것은 어려움 일반적인 컴퓨터(smart end system) 를 사용하므로 성능이 좋다.		가상 회선 을 사용하고, 기존의 전화망과 상당히 유사 → 타이밍 요구사항 만족 → Bandwidth 보장 가능 → 단말기는 기능이 약함(dumb)
Issue	Datagram subnet	Virtual-circuit subnet
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC number
State information	Routers do not hold state information about connections	Each VC requires router table space per connection
Routing	Each packet is routed independently	Route chosen when VC is set up; all packets follow it
Effect of router failures	None, except for packets lost during the crash	All VCs that passed through the failed router are terminated
Quality of service	Difficult	Easy if enough resources can be allocated in advance for each VC
Congestion control	Difficult	Easy if enough resources can be allocated in advance for each VC

네트워크의 3대 주요 기능: **라우팅(routing)**, 혼잡 제어(congestion control), 인터넷워킹

[2. Routing]

Routing(라우팅): **Source to Dest**로 가는 '**좋은**' 경로를 찾는 것

- 일반적으로 **그래프 이론**을 사용하며, **node**는 **router**, **edge**는 **물리적 link**이다.
- **Good Path**: 일반적으로 **최소의 cost**가 발생하는 path로 정의한다.

<Routing Factors>

Where	Source, 각 node(인터넷에서 사용), 중심 node(중앙 집중 방식)
How	Static vs. Dynamic
When	Virtual Circuit vs. Datagram
Criteria	Hop Count, delay, cost, load, bandwidth

<Global vs. Decentralized>

Global	모든 node가 link cost 정보를 포함하여 완전한 topology를 갖는다. "link state" 알고리즘
Decentralized	각 node가 물리적으로 연결된 neighbor와 그들 간의 cost를 알 때 → 계산을 반복하고 이웃과 정보를 교환한다. → 대표적으로 "Distance Vector" 알고리즘이 있다.

<Static vs. Dynamic>

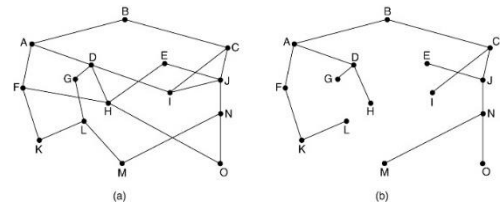
Static	라우팅 경로가 시간이 지남에 따라 조금씩 변한다.
Dynamic	라우팅 경로가 빠르게 변한다. → Link cost change에 따라 주기적으로 업데이트

<Sink Tree>

Optimality Principle(최적 원리): Bellman의 최적 원리에

기반한 Q-learning 기법을 사용한다.

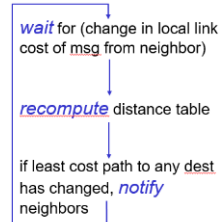
- Dijkstra 알고리즘 등의 최단거리 알고리즘을 통해 최단거리를 나타낸 sink tree를 생성



- Sink Tree에서 전체 경로의 부분 경로들도 최적임 (즉 최단거리가 부분집합에도 적용)

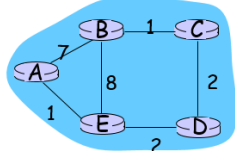
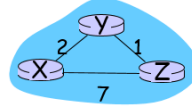
<Distance Vector Routing> - Bellman-Ford 알고리즘 사용

Iterative	어떤 node도 더 이상 정보를 교환하지 않을 때까지 반복 → Stop signal이 없음
Asynchronous	Lock step에서는 node가 정보를 교환하지 않아도 됨
Distributed	각 node는 직접 연결된 node들과만 통신
Distance Table 자료구조	→ 각 node는 자신의 distance table을 갖는다. → 각 행은 가능한 destination을, 각 열은 직접 연결된 이웃 node를 나타낸다. → Loop 발생 가능
핵심 수식	$D^X(Y, Z): \text{distance from } X \text{ to } Y, \text{ via } Z \text{ as next hop}$ $D^X(Y, Z) = \text{cost}(X, Z) + \min_w \{D^Z(Y, w)\}$ $(X \rightarrow Y) = (X \rightarrow Z) + \min \{Z \rightarrow Y\}$ <p>→ 목적지 Y에 도착할 때까지 recursive하게 최단경로 탐색</p> <p>→ 주기적으로 이웃 node와 정보 교환 시 업데이트</p>



- Network diameter에 따라 hop 수가 정해진다.

<Distance Vector Routing에서 Routing Table 생성>

 $D^E(C,D) = c(E,D) + \min_w \{D^D(C,w)\} = 2+2 = 4$ $D^E(A,D) = c(E,D) + \min_w \{D^D(A,w)\} = 2+3 = 5$ $D^E(A,B) = c(E,B) + \min_w \{D^B(A,w)\} = 8+6 = 14 \text{ loop!}$	<p>cost to destination via</p> <table><tr><th>$D^E()$</th><th>A</th><th>B</th><th>D</th></tr><tr><td>A</td><td>1</td><td>14</td><td>5</td></tr><tr><td>B</td><td>7</td><td>8</td><td>5</td></tr><tr><td>C</td><td>6</td><td>9</td><td>4</td></tr><tr><td>D</td><td>4</td><td>11</td><td>2</td></tr></table> <p>Outgoing link to use, cost</p> <table><tr><th></th><th>A</th><th>B</th><th>D</th></tr><tr><td>A</td><td>A,1</td><td></td><td></td></tr><tr><td>B</td><td></td><td>D,5</td><td></td></tr><tr><td>C</td><td></td><td></td><td>D,4</td></tr><tr><td>D</td><td></td><td></td><td>D,2</td></tr></table> <p>Routing table</p>	$D^E()$	A	B	D	A	1	14	5	B	7	8	5	C	6	9	4	D	4	11	2		A	B	D	A	A,1			B		D,5		C			D,4	D			D,2	 <p>cost via</p> <table><tr><th>D^X</th><th>Y</th><th>Z</th></tr><tr><td>Y</td><td>2</td><td>∞</td></tr><tr><td>Z</td><td>∞</td><td>7</td></tr></table> <p>cost via</p> <table><tr><th>D^Y</th><th>X</th><th>Z</th></tr><tr><td>X</td><td>2</td><td>∞</td></tr><tr><td>Z</td><td>∞</td><td>9</td></tr></table> <p>cost via</p> <table><tr><th>D^Z</th><th>X</th><th>Y</th></tr><tr><td>X</td><td>7</td><td>∞</td></tr><tr><td>Y</td><td>∞</td><td>1</td></tr></table> <p>cost via</p> <table><tr><th>D^X</th><th>Y</th><th>Z</th></tr><tr><td>Y</td><td>2</td><td>8</td></tr><tr><td>Z</td><td>3</td><td>7</td></tr></table> <p>cost via</p> <table><tr><th>D^Y</th><th>X</th><th>Z</th></tr><tr><td>X</td><td>2</td><td>8</td></tr><tr><td>Z</td><td>9</td><td>1</td></tr></table> <p>cost via</p> <table><tr><th>D^Z</th><th>X</th><th>Y</th></tr><tr><td>X</td><td>7</td><td>3</td></tr><tr><td>Y</td><td>9</td><td>1</td></tr></table> $D^X(Y,Z) = c(X,Z) + \min_w \{D^Z(Y,w)\} = 7+1 = 8$ $D^X(Z,Y) = c(X,Y) + \min_w \{D^Y(Z,w)\} = 2+1 = 3$	D^X	Y	Z	Y	2	∞	Z	∞	7	D^Y	X	Z	X	2	∞	Z	∞	9	D^Z	X	Y	X	7	∞	Y	∞	1	D^X	Y	Z	Y	2	8	Z	3	7	D^Y	X	Z	X	2	8	Z	9	1	D^Z	X	Y	X	7	3	Y	9	1
$D^E()$	A	B	D																																																																																													
A	1	14	5																																																																																													
B	7	8	5																																																																																													
C	6	9	4																																																																																													
D	4	11	2																																																																																													
	A	B	D																																																																																													
A	A,1																																																																																															
B		D,5																																																																																														
C			D,4																																																																																													
D			D,2																																																																																													
D^X	Y	Z																																																																																														
Y	2	∞																																																																																														
Z	∞	7																																																																																														
D^Y	X	Z																																																																																														
X	2	∞																																																																																														
Z	∞	9																																																																																														
D^Z	X	Y																																																																																														
X	7	∞																																																																																														
Y	∞	1																																																																																														
D^X	Y	Z																																																																																														
Y	2	8																																																																																														
Z	3	7																																																																																														
D^Y	X	Z																																																																																														
X	2	8																																																																																														
Z	9	1																																																																																														
D^Z	X	Y																																																																																														
X	7	3																																																																																														
Y	9	1																																																																																														
$D^X(Y,Z)$ = cost(X,Z) + min _w {D ^Z (Y,w)} 를 이용한 계산 (Distance Table 생성)	Distance Table을 이용한 라우팅 테이블 생성 (각 열에서 최솟값을 선택)	Distance Table을 모든 이웃 node에 전송한다.																																																																																														

<Distance Vector Algorithm>

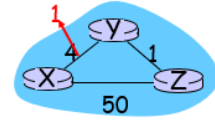
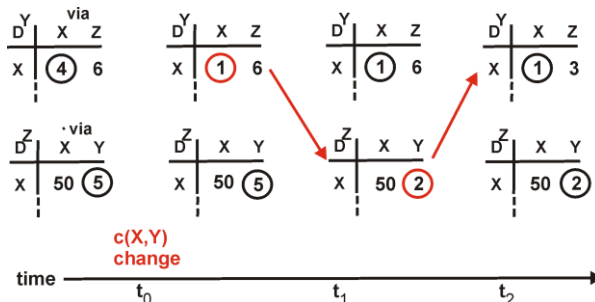
At all nodes, X:

- 1 Initialization:
- 2 for all adjacent nodes v:
- 3 $DX(*,v) = \text{infinity}$ /* the * operator means "for all rows" */
- 4 $DX(v,v) = c(X,v)$
- 5 for all destinations, y
- 6 send $\min_w DX(y,w)$ to each neighbor /* w over all X's neighbors */
- 8 loop
- 9 wait (until I see a link cost change to neighbor V
- 10 or until I receive update from neighbor V)
- 11
- 12 if $(c(X,V)$ changes by d)
- 13 /* change cost to all dest's via neighbor v by d */
- 14 /* note: d could be positive or negative */
- 15 for all destinations y: $DX(y,V) = DX(y,V) + d$
- 16
- 17 else if (update received from V wrt destination Y)
- 18 /* shortest path from V to some Y has changed */
- 19 /* V has sent a new value for its $\min_w DV(Y,w)$ */
- 20 /* call this received new value is "newval" */
- 21 for the single destination y: $DX(Y,V) = c(X,V) + \text{newval}$
- 22
- 23 if we have a new $\min_w DX(Y,w)$ for any destination Y
- 24 send new value of $\min_w DX(Y,w)$ to all neighbors
- 25
- 26 forever

[1. Routing]

<Link State Changes>

1. node는 local link cost가 바뀐 것을 탐지한다.
2. distance table을 업데이트한다.
3. cost가 **least cost path**에 대하여 바뀌면 이웃 node에 알린다.

**[Good News Travels Fast]**

After t1:

$$D^Z(X, Y) = \text{cost}(Z, Y) + \min_w D^Y(X, w) \\ = 1 + \min(1, 6) = 1 + 1 = 2$$

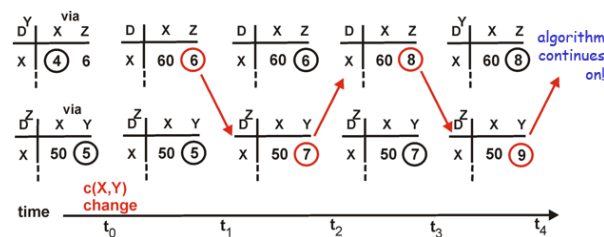
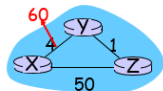
After t2:

$$D^Y(X, Z) = \text{cost}(Y, Z) + \min_w D^Z(X, w) \\ = 1 + \min(50, 2) = 1 + 2 = 3$$

[Bad News Travels Slow] – “Count to Infinity Problem”

Link cost changes:

- a) good news travels fast
- b) bad news travels slow - “count to infinity” problem!



After t1:

$$D^Z(X, Y) = \text{cost}(Z, Y) + \min_w D^Y(X, w) \\ = 1 + \min(60, 6) = 1 + 6 = 7$$

After t2:

$$D^Y(X, Z) = \text{cost}(Y, Z) + \min_w D^Z(X, w) \\ = 1 + \min(50, 7) = 1 + 7 = 8$$

After t3:

$$D^Z(X, Y) = \text{cost}(Z, Y) + \min_w D^Y(X, w) \\ = 1 + \min(60, 8) = 1 + 8 = 9$$

<Link State Routing>

단계	Distance Vector와 공통
1. 이웃 node를 찾아서 그들의 network address 를 찾는다.	
2. 각 neighbor의 delay 또는 cost 를 측정한다.	
3. 이웃 node의 링크 상태(delay, cost 정보) 를 의미하는 패킷을 만든다.	
4. 이 패킷을 모든 router로 전송 한다.	
5. Dijkstra 알고리즘 을 적용하여 다른 모든 라우터와의 shortest path 를 계산한다.	

➔ “전체 network의 topology를 알고 적용하는 방법”

<Dijkstra's Algorithm>

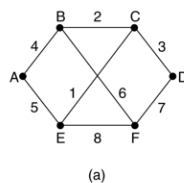
1. 자신 주변의 **이웃 node**를 찾는다. (hello packet 전송)
2. **cost**를 **측정**한다. (패킷을 전송하여 반사시킨 후 시간을 반으로 나눔)
3. **Link state packet**을 만든다. (각 node마다 패킷을 만들어서 이웃 node와의 링크 상태 표현)

→ **(Source node - 일련번호 - 수명 - 각 node로 갈 때의 cost)**

→ 각 node는 **LS packet**을 만들어서 방송한다.

→ **Acknowledge**를 하는 이유: **못 받으면 재전송**하기 위해

- **Reliable broadcast**, 즉 방송을 하되 받지 못하면 재전송하는 방법을 이용한다.
- **하나라도 받지 못하면 topology가 깨지기** 때문



Link		State		Packets	
A	Seq.	B	Seq.	C	Seq.
Age	Age	Age	Age	Age	Age
B 4	A 4	B 2	C 3	A 5	B 6
E 5	C 2	D 3	F 7	C 1	D 7
	F 6	E 1		F 8	E 8

(a)

(b)

[Dijkstra's Algorithm]

1. network topology와 link cost는 **모든 node가 알고 있다**.
 - **Link state broadcast**를 이용하여 구현 가능
 - 모든 node가 서로 같은 정보를 갖는다.
2. 어떤 노드에서 **모든 다른 노드와의 least cost path**를 계산한다.
 - 해당 node에 대한 routing table을 생성할 수 있게 한다.

[Notations]

$c(i,j)$	Node i에서 j로의 link cost (직접 연결되지 않으면 무한대)
$D(v)$	Source에서 destination v로의 현재 cost 값
$p(v)$	Source->v path에서의 predecessor node (next v)
N	Least cost path가 알려진 node 의 집합 (즉 영구 node 의 집합)

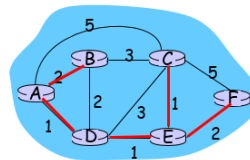
[Program]

```

1 Initialization:
2 N = {A}
3 for all nodes v
4   if v adjacent to A
5     then D(v) = c(A,v)
6   else D(v) = infinity
7
8 Loop
9   find w not in N such that D(w) is a minimum
10  add w to N
11  update D(v) for all v adjacent to w and not in N:
12    D(v) = min( D(v), D(w) + c(w,v) )
13    /* new cost to v is either old cost to v or known
14       shortest path cost to w plus cost from w to v */
15  until all nodes in N
  
```

[Example]

Step	start N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
→0	A	2,A	5,A	1,A	infinity	infinity
→1	AD	2,A	4,D		2,D	infinity
→2	ADE	2,A	3,E			4,E
→3	ADEB		3,E			4,E
→4	ADEBC					4,E
→5	ADEBCF					



<Algorithm Complexity: n nodes>

- 각 iteration에서 N에 있는 것을 제외한 모든 node를 탐색한다.
- **$n(n+1)/2$ 회의 비교**가 필요하므로 Complexity는 **$O(n^2)$** 이다. (더 효율적인 경우 **$O(n \log n)$**)

<Comparison of LS(Link State) and DV(Distance Vector) algorithms>

	Link State Algorithm	Distance Vector Algorithm
Message Complexity	n개의 node와 E개의 link가 있는 경우 $O(nE)$ 개의 메시지 전송	이웃 간에만 메시지를 교환
Convergence Speed	$O(n^2)$ 알고리즘을 사용하는 경우 $O(nE)$ 개의 메시지가 전송된 후 수렴	Convergence time의 편차가 심함 → routing loop 존재 → count-to-infinity 문제점
Robustness (라우터 오작동 시)	→ 부정확한 link cost를 광고할 수 있다. → 각 node는 해당 node에 대한 table만을 계산한다.	→ DV node가 부정확한 path cost를 광고할 수 있다. → 각 node의 table은 다른 node가 사용할 수 있다.

→ Distance Vector에서는 hop 수를 이용하는데, cost 관점에서는 열악하고 부정확하다.

→ Link State 방식을 주로 사용하는 이유: Distance Vector는 수렴 속도의 문제가 크고, Link State는 link의 cost를 다양하게 표현할 수 있다.