정보통신 월말 정리

수업 멘토: 2018107161 변세정

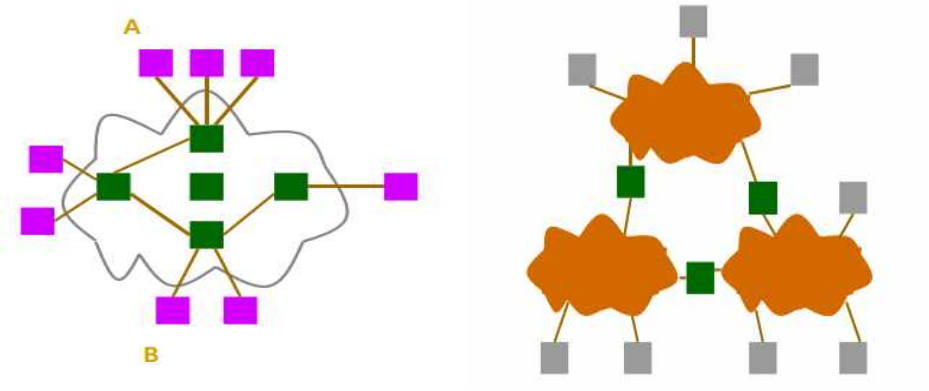
본 자료는 수업 자료를 토대로 작성되었습니다.
그 외 부분은 기타 필요한 정보를 소개하는 것이고 실제 수업 범위는 아닙니다.

컴퓨터 통신과 네트워크

컴퓨터 통신이란, 객체가 프로토콜에 따라서 매체를 통해 정보를 전달·수신하는 것을 말한다.

여기서 객체는 사람, 컴퓨터와 같이 정보를 보내는 주체이고, 매체는 구리선, 광케이블과 같이 정보를 보내는 통로, 그리고 프로토콜은 정보를 보낼 때, 송신·수신자 간의 약속이다. 보내는 정보는 비트로 보내지고, 용량에 따라 Bps, Kbps 등으로 있다.

인터넷의 성장, 기술의 발전 등에 따라 네트워크의 고속화 기술, 보급이 널리 설정된 상태이다.



핑크색 = 호스트	초록색 = 게이트웨이
초록색 = 교환기	주황색 = 네트워크

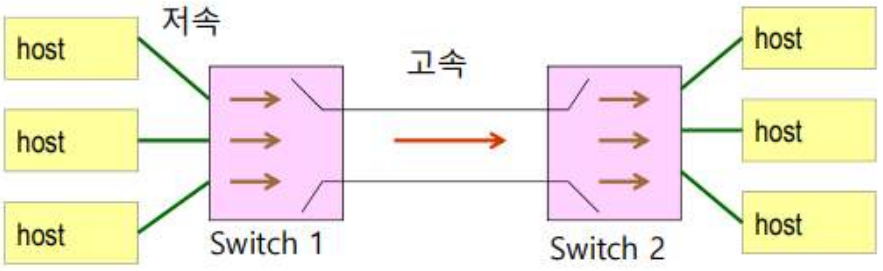
노드·링크로 구성된 네트워크..

연결에 대해 여러 노드가 병렬적으로 연결이 되거나 하나씩 연결되는 경우 등이 있다. 교환기라는 것을 통해서 여러 노드를 연결하고 사이의 정보를 교환하는 역할을 한다.
실질적으로 내부 상황을 명확하기 어렵기에 구름 모양으로 표현한다. 이를 클라우드라고 한다. 이러한 네트워크상 통신은 아래와 같이 호스트의 개수에 따라 표현하기도 합니다.

- unicast(유니캐스트) : 지정된 한 호스트에 통신하는 방식
- multicast(멀티캐스트) : 특정한 여러 호스트에 통신하는 방식
- broadcast(브로드캐스트) : 모든 호스트에 통신하는 방식(보통 같은 네트워크 안에 있는 모든 호스트)

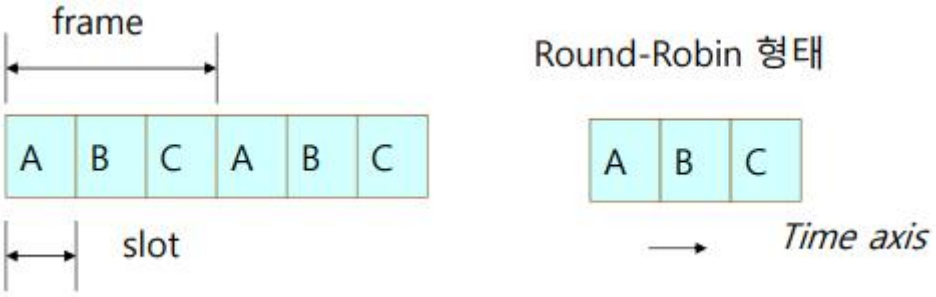
교환기와 통신

교환기를 통해 호스트들의 메시지를 저속으로 받고 이를 고속으로 다른 교환기를 통해 원하는 다른 호스트에게 전송한다.



이때 교환기가 모든 호스트의 메시지를 동시에 처리할 수 없다. 따라서 호스트에게 받은 메시지를 처리하는 방식으로 시분할, 주파수 분할, 통계적 시분할 방식이 있다.

시분할 방식



교환기에 들어온 호스트의 작업에 순서를 부여해 반복하는 형식이다. 위에서 A->B->C 순으로 작업을 하고 다시 A->B->C->A... 순으로 통신 작업을 한다.

주파수 분할 방식

한 매체의 주파수를 달리해서 전송한다. (간섭이 없도록 주파수 설정을 한다.)

통계적 시분할 방식

시분할 방식에서 슬롯 낭비(통신이 필요 없어도 슬롯이 할당되는 것)를 줄이기 위해 선입 선출으로 슬롯을 할당하고 슬롯마다 사용자를 명시하여 사용하는 방식이다.

이렇게 여러 방식이 있지만, 이외에도 메시지 전송에 문제가 생길 수도 있다. 즉, 신뢰성이 있어야 하고, 빨리 통신하게 시간 제약성도 있다. 해당하는 오류는 다음과 같다.

-신뢰성을 평가할 때 네트워크 전송오류

#BER(Bit Error Rate)
: 비트 전송 시 오류가 발생할 확률
#감쇄
: 거리에 따라 전기 신호가 약해짐으로 데이터가 손실되는 현상
#지연 왜곡
: 주파수에 따라 전송속도가 달라서 데이터가 왜곡되는 현상
#기타
: 누화(crosstalk), 잡음 등등

-시간 제약성을 평가할 때 성능 요소

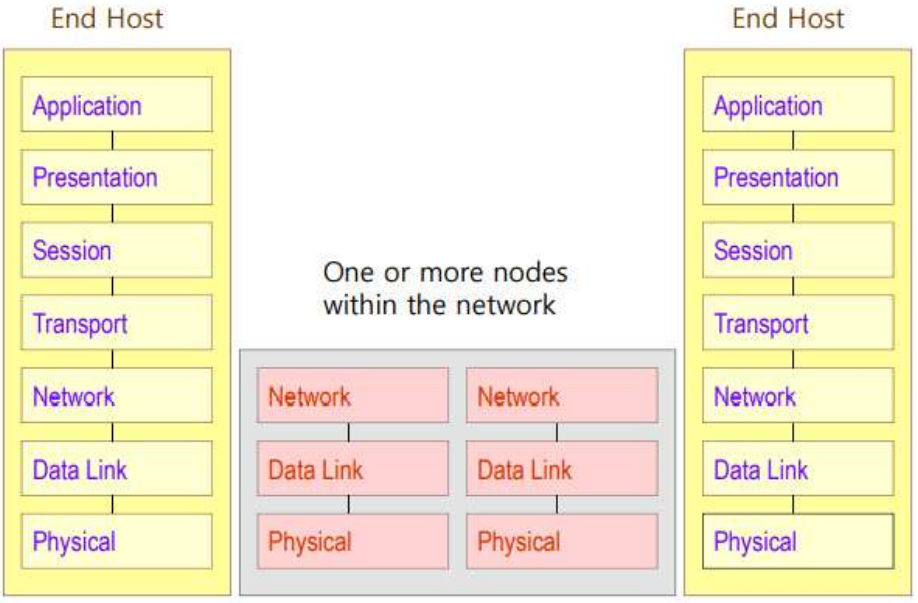
#대역폭 : 일정 시간 동안 전송될 수 있는 비트 수. (매체마다 다르다)
#지연시간 : 매체를 타고 한 비트가 전송되는 시간(광속에 비례, 매체에 무관)
#큐잉시간 : 교환기의 큐에서 대기 중인 시간
#RTT (Round Trip Time) : 왕복 시간

프로토콜과 OSI 계층

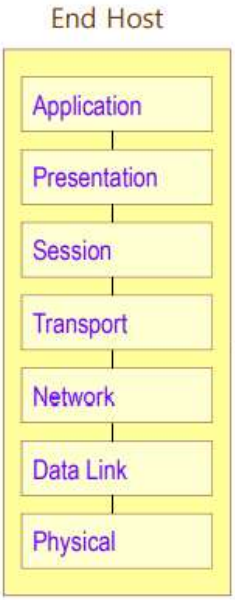
프로토콜: 통신하는 쌍방간 규약된 약속

프로토콜은 데이터 전송 시 어떻게 보낼 것인가 등 여러 정보를 약속해둔 것으로 호스트와 호스트 사이의 언어와 같다. 구현에 있어서 기능마다 나누고 모델을 정의하게 된다. 계층화를 통해서 통신의 기능을 계층적으로 볼 수 있게 된다. 여러 모델 중 OSI 7계층은 ISO 기관에서 네트워크의 기능을 7개의 계층으로 분할한 것이다. peer은 상대방의 같은 계층을 의미하고, peer-to-peer은 따로 서버를 거치지 않고 노드끼리 전송한다는 의미이다. 즉, 같은 계층끼리 통신이 가능하고 해석이 된다는 것이다.

*RM(Reference model) :참조 모델



위 그림과 같이 OSI 7 계층이 있다. 왼쪽에서 오른쪽으로 통신을 보낸다고 생각했을 때, 데이터는 왼쪽 Application 계층에서 physical->오른쪽 physical->Application 순으로 지나간다. 우선 각 계층은 다음과 같은 기능으로 이루어진다.



이 위부터 상위계층이고 아래로 갈수록 하위계층이다.

이러한 계층 중 상위계층에서 하위계층으로 데이터를 처리하도록 보내는데, **계층으로 내려갈 때마다 헤더가 붙어서 다음 계층에서는 한 데이터로 처리하게 된다.**

반대로 하위계층에서 상위계층으로 올라갈 때 헤더를 읽고 데이터를 처리한다.

응용 계층	사용자 인터페이스를 제공, 데이터를 생성하고 서비스한다.
표현 계층	두 호스트 간 표현 방식의 통일
세션 계층	한 세션에 속한 다수 연결의 동기화, 결합
트랜스포트 계층	프로세스 대 프로세스 통신을 위한 연결제공.
네트워크 계층	경로 제어. 가장 좋은 경로를 찾고 올바르게 이동할 수 있도록 한다.
데이터 링크 계층	비트 흐름을 프레임으로 변환한다. 부 계층으로 LLC와 MAC이 있다. 해당하는 비트에 대한 오류제어와 흐름제어의 기능을 한다.
물리적 계층	전기, 물리적 특성의 정의로 신호의 방식, 크기, 부호화(encoding) 등을 한다. 즉, 프레임인 비트를 이동시키는 기능으로 하드웨어적 성격이 강하다.

LLC : 두 호스트 간의 논리적 채널(실제로는 없지만 개념적으로 있다.)
MAC : 공유된 매체에 대한 접근

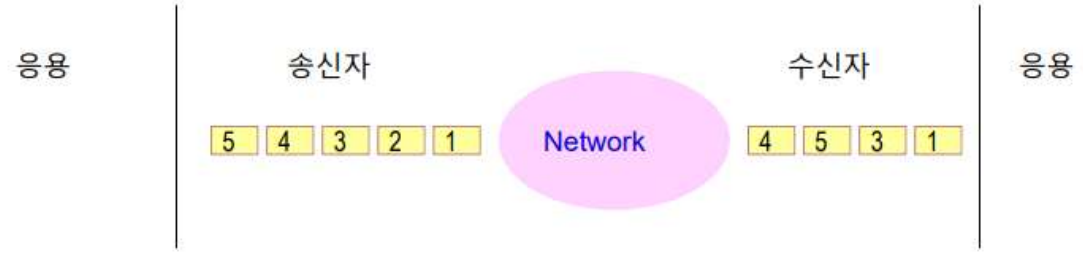
프로토콜의 기능 개요

NET : 각 네트워크(이더넷 등)
IP : 경로제어(네트워크에서 해당 호스트를 찾기 위한 프로토콜)

TCP : 프로세스 간 신뢰성있는 통신, 속도가 느리다.
UDP : 프로세스 간 비 신뢰적 통신, 속도가 빠르다.

상위
  FTP, HTTP, ssh over TCP
  NV, TFTP over UDP

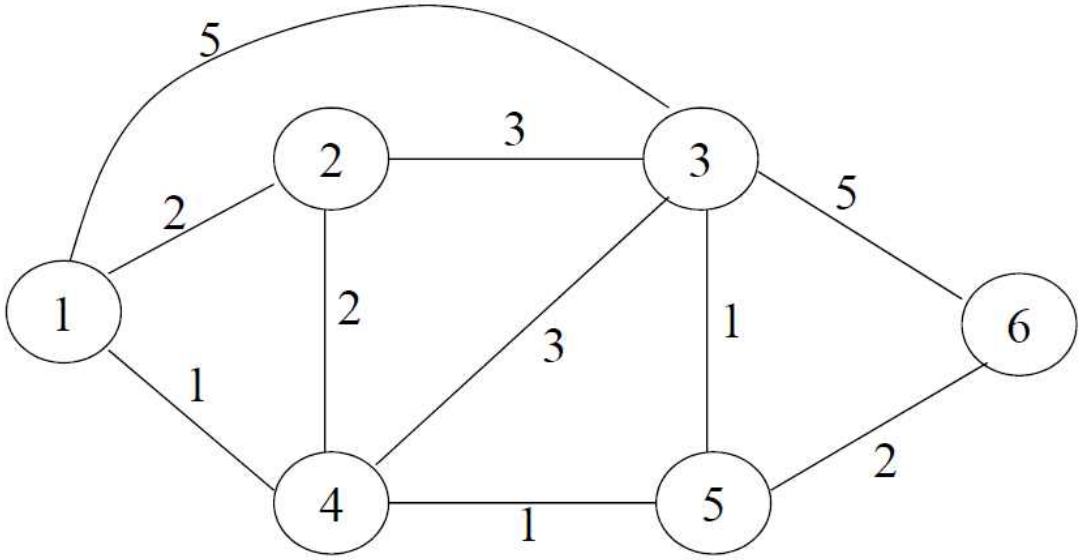
신뢰성



송신자가 데이터를 어떻게 보내도 네트워크의 일부 패킷 손실과 순서가 바뀔 수 있는데 수신자는 다시 정렬 및 재전송하여 송신 순서대로 읽어서 처리한다.

다익스트라 알고리즘(Dijkstra)

다익스트라 알고리즘은 출발점에서 모든 지점간의 최단 경로를 찾아 업데이트 하는 알고리즘입니다. 중앙 집중식 경로 찾기 알고리즘으로 노드(컴퓨터)의 수가 적을 때 사용가능하다. 이 알고리즘의 그래프를 인접행렬, 인접리스트로 표현 가능하다.



이 알고리즘은 간단한 개념을 통해 구현된다. 위 그림 1번에서 6번으로 간다는 예시를 들면, 1->3또는 1->5로 가서 6으로 가야한다. 한눈에 보면 당연히 1->4->5->6이 최적의 경로이다. 하지만, 컴퓨터는 연산 과정을 통해 어떤 경로가 최적인지 구한다. 따라서 다익스트라 알고리즘은 1->6일 때 비용을 dist[1][6]=Math.min(dist[1][6], (dist[1][k]+dist[k][6]))으로 이미 계산된 비용, 1과 6사이 k노드가 있고 출발지에서 k를 거쳐서 목적지로 가는 경우.. 둘 중 최소값을 비용으로 업데이트하는 것을 기본으로 합니다. 이를 1과 가까운 노드부터 DFS 또는 BFS 방식으로 탐색하여 최적의 경로를 찾는 것이 다익스트라 알고리즘입니다.

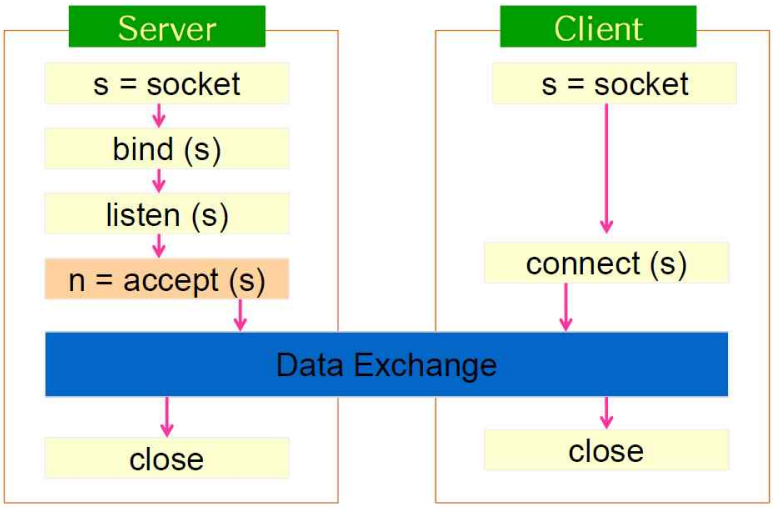
소켓 프로그래밍(Socket Programming)

소켓이란? 소켓은 다양한 데이터가 도착하는 종착지로 end-to-end communication이라고 불린다. (소켓=IP주소+port 번호)

- 호스트의 프로세스와 프로세스 간 통신.
- Socket Interface 통신을 하는 시스템 호출의 집합.

이렇게 소켓을 통해
프로그램 간 통신 시 두 프로그램 모두 소켓이 구성되어 있어야 통신가능.

- TCP 호출과정.

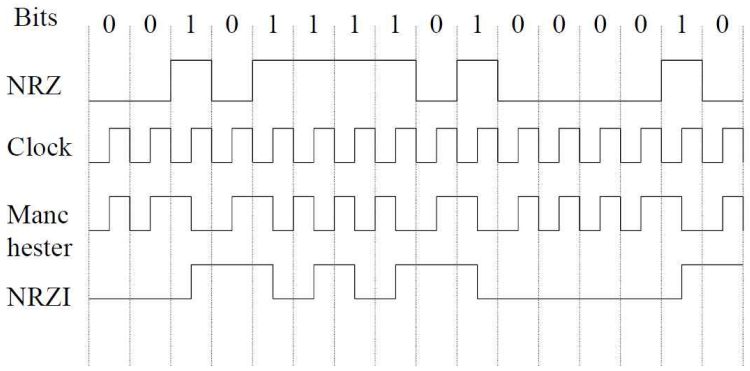


*블로킹(blocking) 호출 = 무한히 대기하지 않는 호출
*Non blocking 호출 = 무한히 대기하는 호출 (ex: getchar(); 등)

위의 과정은 광섬유, 구리선 등의 케이블을 통해서 데이터가 전달된다. 그렇다면 받은 비트를 어떻게 해석할까?

Encoding

비트를 보내는 방식은 0과 1이지만, 이를 받는 수신자는 이게 어떻게 해석하면 좋을지 모른다. 이 비트를 데이터로 역변환하는 방식을 인코딩(Encoding) 이라하고 인코딩방식은 다음과 같다.



NRZ	비트대로 클럭 신호가 결정된다.
NRZI	비트가 1일 때마다 신호가 바뀐다.
Manchester Encoding	1: 1->0, 0: 0->1 의 규칙을 가진다. **대역폭의 반만 사용 가능.** 주기 당 최소 한번 전이
4B/5B	4비트를 5비트로 바꿔서 전송(5 비트는 최대 1개의 0) 00000:회선 고장, 11111: 회선 미사용

HDLC(High Level Data-link Control) 전송모드

HDLC는 각 프레임(정보 프레임, 감시 프레임, 비 번호 프레임)에 대한 제어를 의미합니다. 전송 방식으로는 3가지로 다음과 같습니다.

1. NRM 정규응답모드
   종국은 주국의 허가가 있을 때 송신.(불균형 링크)
2. ABM 비동기 균형 모드
   허가 없이 언제나 전송할 수 있도록 설정되는 것. (균형 링크)
3. ARM 비동기 응답모드
   종국은 주국의 허가 없이 송신가능하지만, 링크, 오류 복구 등 제어는 주국만 가능하다.

이런 다양한 전송방식으로 비트가 전송되는데, 전송되는 데이터 블록(= 프레임)에 오류가 있는지와 어디가 시작이고 어디가 끝인지 알아야 한다. 이를 알기 위해 Bit Stuffing을 하게 된다.

Bit stuffing

프레임의 시작과 끝은 01111110이다. 하지만, body 부분에 똑같은 비트가 나올수 있기 때문에 다음과 같이 진행한다.

송신: 011111 이후 무조건 0을 삽입.
수신: 011111 수신 후 뒤가...
10 이면 끝을 인지
0이 나오면 0을 제거
11이 나오면 네트워크 오류

01100111111101111101111101000011111111

01100111110110111110101111100100001111101111

프레이밍에 대해서는 다른 바이트 중심 프로토콜, 클럭 기반 프레이밍이 있습니다.

프레이밍으로 데이터를 인식된다면, 이제 그 데이터가 오류가 나지 않았는지 탐지해야합니다. 이런 오류탐지에는 패리티, CRC, Hamming code 등이 있습니다.

패리티(Parity)

패리티는 송신된 데이터의 오류를 탐지하는 기법중 하나로, 송수신자 양단간의 합의로 정해진다. 이는 몇 비트마다 묶어서 해당하는 그룹에 1의 개수로 부가적인 비트를 데이터에 포함시킨다. 1의 개수가 짝수인걸 기준으로 하는 것을 짝수 패리티, 홀수 기준을 홀수 패리티로 한다.

예를 들면...

> 00010101010110101001O 비트에, 7비트 짝수 패리티라면 다음과 같은 과정을 거친다.
>
> 1) 7비트로 나누기 0001010 1010110 1010010
> 2) 각 묶음 당 1의 개수 세기 2, 4, 3 //↓짝수 패리티라서 짝수라면 0, 홀수면 1이다.
> 3) 해당하는 값 추가 [00010100 10101100 10100101]
>
> 결과: 000101001010110010100101

이러한 패리티는 몇 비트를 묶는 것이 좋을까...?

짧은 묶음 vs 긴 묶음

짧은 묶음	긴 묶음
부가적 데이터↑ = 비트 처리↑	부가적 데이터↓ = 오류 탐지↓

-문제점들

짝수 패리티에서 짝수개의 비트가 오류가 난다면 오류탐지를 못한다.

(0011 -> 1010로 오류가 나도 짝수패리티는 똑같이 0이다.)

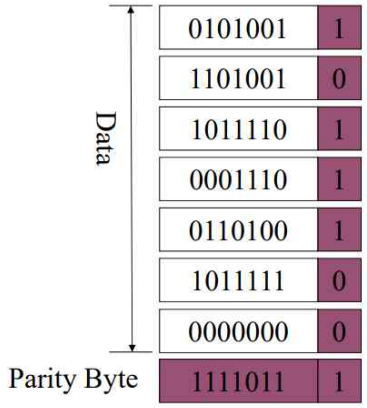
n 비트 중 a 비트 동시 오류 확률

= nCa p^a (1-p)^(n-a) //p는 BER(비트 오류확률)

if n 비트 중 동시 오류가 날 확률이라면 위 a가 2, 4, .. 일 때 모든 확률을 더하면 됩니다.

2차원 패리티

각 행,열에 대한 패리티는 자주색 부분으로 이러한 형태를 가집니다.

	해당 구조는 오류가 났음에도 오류탐지를 못하는 경우가 발생합니다. -오류발생 Data 부분에 사각형 네 모서리에서 오류가 동시에 발생하는 경우. 해당 네모서리에서 오류동시 발생 확률은... ${}_8C_2\,{}_8C_2\,p^4\,(1-p)^{60}$ //p는 BER(비트 오류확률)

CRC(Cyclic Redundancy Code)

CRC의 경우 CRC 다항식으로 나눈 나머지가 0이 되도록 하는 오류코드를 구하는 방식이다.

예를 들어, CRC 다항식 3차 비트가 있을 때..

$$10011010 = x^7 + x^4 + x^3 + x^1$$

$$\text{CRC 다항식} = x^3 + x^2 + 1$$

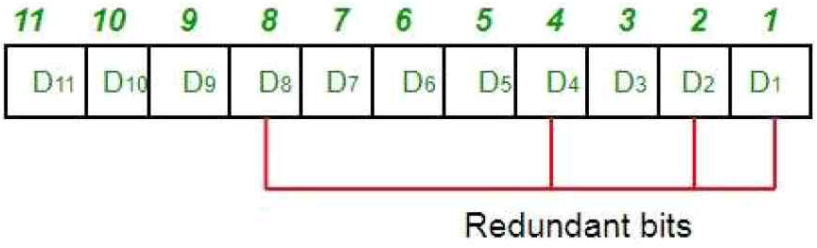
11111001 1101) 10011010000 1101 1001 1101 1000 1101 1011 1101 1100 1101 1000 1101 101	왼쪽과 같은 과정을 가진다. 1. 비트에 CRC다항식의 차수만큼 0을 추가한다. (10011010->10011010**000**) 2. 이를 CRC 다항식으로 나눈다. 이때, 나누기를 할 때, 빼기 연산은 XOR로 연산한다. 3. 비트 끝까지 나누기를 한 뒤 나온 나머지를 원래 비트에 붙여서 처리한다.(10011010->10011010**101**)

CRC 다항식은 양단간의 합의로 결정되고 오류탐지를 못할 확률 계산이 난이하다. 경험적으로 좋은 CRC 다항식이 존재한다. 이러한 CRC는 XOR 연산 등이 있어 하드웨어로 구현된다.

Hamming code

오류탐지를 위해 m 비트를 보내려고 할 때 r개의 비트가 추가하여 보내는 방식이다.

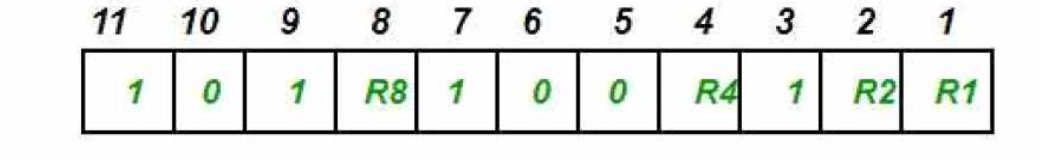
$2^r \geq m + r + 1$ 옆의 공식이 성립하도록 r이 설정된다.
예를 들어 7비트를 보낸다면, 2^r≥8+r로 공식이 나온다. r=3이라면 공식이 성립하지 않기 때문에 r=4인 것이 좋다.



위 그림과 같이 추가된 r개의 비트는 2^n 위치에 추가된다. 이 위치에 추가되는 이유는 이 해밍코드를 구하는 과정에서 이진수로 나타내서 구하기 때문이다.(공식이 설정된 것도 이를 이용하기 위해서이다.)

그렇다면 추가된 r개의 비트는 0일까 1일까?

이에 대한 질문은 패리티를 이용해 결정된다.
만약 1011001이라는 비트가 들어와서 추가되는 비트들을 결정할 때, 들어온 비트들을 이진수로 설정하여 패리티 방식으로 구하게 된다.



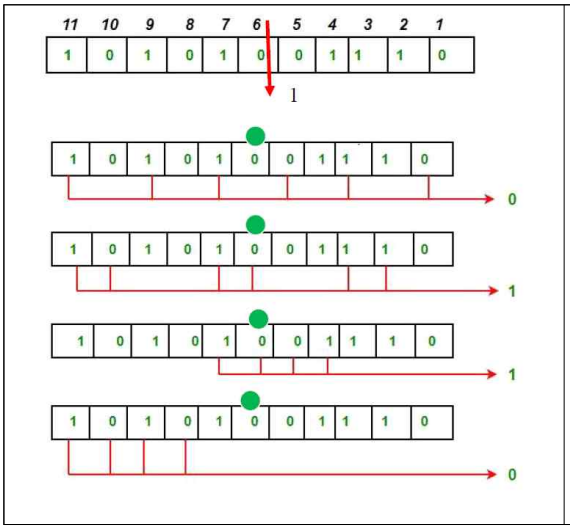
공식으로 구한 r은 사실 각 비트를 이진수로 바꾸었을 때 자리수와 같다. 즉, 여기서 가장 큰 11 은 이진수로 바꾸었을 때 4자리를 벗어나지 않는다는 뜻이다.(11-> 1011)

추가된 비트는 각 자리에 맞게...
R1: 각 비트를 이진수로 변환했을 때, 2^0이 1인 것들과 패리티 =>(1, 3, 5, 7, 9, 11)
R2: 각 비트를 이진수로 변환했을 때, 2^1이 1인 것들과 패리티 =>(2, 3, 6, 7, 10, 11)
R3: 각 비트를 이진수로 변환했을 때, 2^2이 1인 것들과 패리티 =>(4, 5, 6, 7)
R4: 각 비트를 이진수로 변환했을 때, 2^3이 1인 것들과 패리티 =>(8, 9, 10, 11)

이를 통해 얻는 해밍코드는... 0110이다! 따라서 데이터를 보낼 때 기존 비트에 추가하여 10101001110을 보내면 된다.

- **오류발생과 보정**

만약 비트 1자리가 오류가 나오면 다음과 같이 보정할 수 있다.



옆의 그림과 같이 6번째 자리의 비트가 오류가 난다면, 6번째 자리를 이용해 구한 해밍코드 역시 오류가 난다. 오류가 나면 1로 오류 난 것을 통합하면 0110이 된다.

0110은 10진수로 6이고 이는 오류가난 비트의 자리수를 가리킨다..!

이 경우 해당 자리의 비트를 보정할 수 있다.

이처럼 1자리에 대한 오류는 보정이 가능하지만, 2자리 3자리 등 여러 비트에 오류가나면 보정이 불가능하고 오류탐지만 가능하다.

해밍거리

해밍 거리는 쉽게 말해 다른 비트의 개수를 의미한다.(비트뿐만 아니라 문자열 등도 가능)
'1011101'과 '1001001' 사이의 해밍거리는 2이다.(1011101, 1001001)

만약 오류가 났을 때, 송신한 비트와 오류가난 비트를 위와 같이 비교하여 얼마나 훼손되었는지 알 수가 있다. 서로 중복되는 것(Redundancy)가 증가될수록 탐지수와 보정 수는 증가한다.

따라서 해밍거리가 최소가 되는 최소해밍거리(Dmin)로 만족되게 배치하면, 그만큼 오류 보정도 잘된다. 이러한 배치방식은 복잡하다. 만약 배치가 성공하여 최소해밍거리가 k일 때, k-1까지 오류 탐지가 된다. 그리고 (k-1)/2 까지 오류가 보정된다.

오류제어

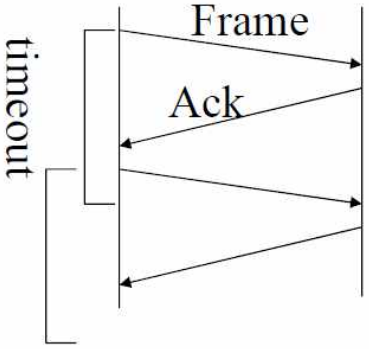
오류제어란, 오류탐지 후 이를 극복하기 위한 기법이다.
수신자가 오류탐지 후 송신자에게 이를 알리고 재전송한다. 이때, Acknowledge와 Timeout을 사용한다. 이러한 재전송을 하기 위한 기법으로는 여러 가지가 있다.

기법

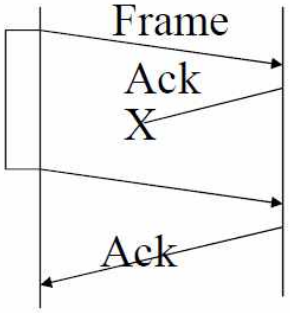
- 정지 대기(Stop and Wait)
- go-back-N
- Selective repeat

정지 대기(Stop and Wait)

한 프레임마다 잘 받았는지 신호(Ack)를 기다리고 다음 프레임을 전송하는 방식.
Ack를 받은 이후에만 다음 프레임을 전송할 수 있다. 단순하지만 낭비시간이 크다.
=> 속도가 느리지만, 오류가 많은 네트워크에서 좋다.



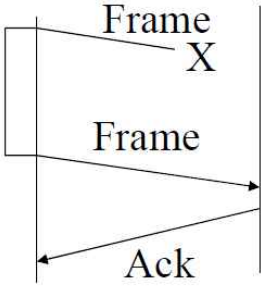
정상적인 경우



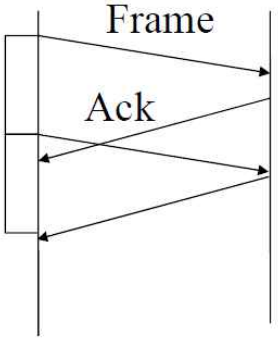
Ack에 손실 발생 ->
Timeout 발생 후 재전송

이때 어떤 프레임을 보내야하는지 알기 위해, 프레임 번호가 필요하다.
프레임 번호가 다 부여되면 송신자는 프레임을 보낼 때 프레임 번호를 기억해두어야 한다. (그래야 재전송이 가능) 정상적인 경우 다음 프레임으로 넘어간다.!

만약 Ack 신호가 안 오면 자연스레 Timeout 하게 되는데, Timeout 시간을 어떻게 설정하는 것이 좋을까?



Frame 전송 오류
timeout후 재전송



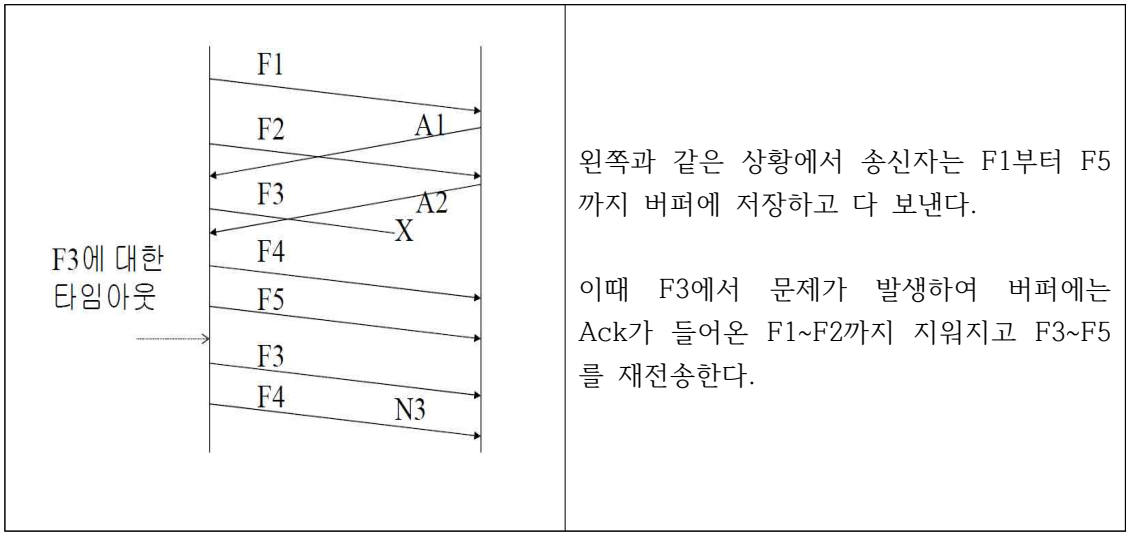
Timeout이 너무 짧은 경우
정상 수신에도 재전송
중복 프레임 수신

시간이 너무 길면 오류가 발생해서 재전송하기까지 너무 늦는다는 단점이 있고, 위 그림과 같이 너무 짧으면 정상수신의 경우에도 다시 전송하는 경우가 있다.

즉, 뭐든 적당히 설정해야한다...!

Go-Back-N

송신자가 버퍼를 이용해 Ack 확인 없이 전부 보낸 뒤 Ack 확인이 되면 버퍼에서 지우고, 재전송이 필요하다면 필요한 번호부터 다시 보내는 방식.
=>오류가 없으면 아주 빠른 방식이다.



왼쪽과 같은 상황에서 송신자는 F1부터 F5까지 버퍼에 저장하고 다 보낸다.

이때 F3에서 문제가 발생하여 버퍼에는 Ack가 들어온 F1~F2까지 지워지고 F3~F5를 재전송한다.

이러한 경우 나머지 F4~F5는 오류가 없어도 재전송이 되는, 프레임 낭비가 발생한다. 거기다 만약 Timeout이 긴 경우 낭비가 더 심해진다. 이러한 문제점으로 오류인 프레임만 재전송하기 위해 슬라이딩 윈도우 등 여러 가지 방법이 있다.

Selective repeat

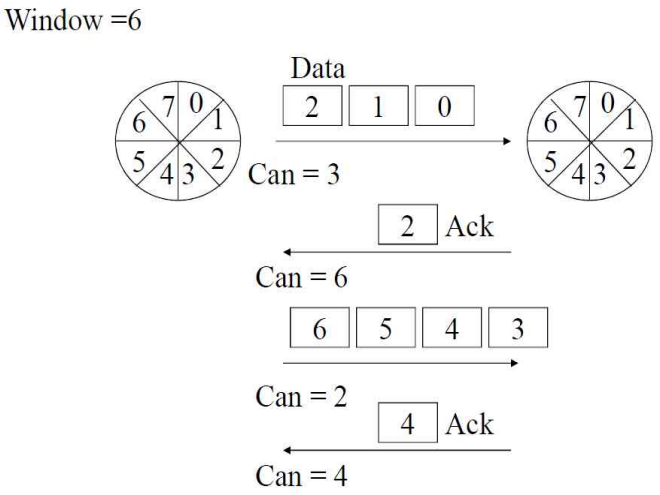
Selective repeat는 흐름제어와 함께 슬라이딩 윈도우에 병합되었다.

그렇다면 슬라이딩 윈도우와 흐름제어는 무엇일까?

Sliding Window

슬라이딩 원도우는 윈도우(수신자의 확인 없이 보낼 수 있는 양) 만큼 전송하고 Ack를 받는다. 만약 오류가 난 경우 수신자 측에서 오류가 난 프레임 바로 직전의 프레임 Ack신호를 보내고 송신측은 해당 Ack 이후 프레임부터 다시 보내게 된다.

예를 들면...



1. 윈도우가 6인 상태에서 데이터를 3개정도 보낸다.
2. 수신측에서 2 프레임에 대한 Ack를 송신측에 보내고, 송신측은 0~2까지의 정보를 버퍼에서 지운다. (오류가 없음)
3. 이제 버퍼에 남은 데이터를 보낸다.
4. 수신측에서 4 프레임에 대한 Ack를 송신측에 보내고, 송신측은 3~4까지의 정보를 버퍼에서 지운다. (5 프레임에서 오류가 있음)
5. 5프레임만 재전송 한다.(6프레임은 재전송 X,)

이때 만약, 5 프레임에만 오류가 발생한 경우에 수신측에서는 6 Ack를 보낸다. 하지만, 6프레임에도 오류가 있는 경우 수신측에서 5 Ack를 보낸다. 이 경우 다신 6 프레임을 보내야한다.

▷ 윈도우는 6인데 왜 3개만 보낼까?

윈도우가 6이더라도 너무 빨리 보내면(송신자, 수신자의 속도차이) 수신자 측에서 받기 힘들어 한다. 이 경우 일정 비율을 정해서 보내는 비율 제어가 필요하다.

이렇게 작동되는 전송 방식에서 몇 가지 제어가 필요하다. 윈도우에 관한 비율제어부터 흐름제어, 폭주제어가 있다.

흐름제어

양단간의 송수신 제어.

수신자가 받을 수 있는 양만큼만 전송(서로 주고받을 수 있는 양)하는 것으로 제한요소로는 버퍼, cpu 속도(명령어 해독을 위해..) 등이 있다.
정지 대기의 경우 1 프레임씩 보내고, 슬라이딩 윈도우의 경우 윈도우만큼 보내게 된다.

-흐름제어: 흐름제어
-폭주제어: 네트워크 전체적인 제어
-비율제어: 윈도우 내 비율 제어

PiggyBack Scheme

각 프레임마다 Ack 프레임을 보내면 낭비가 발생한다. (헤더를 붙이는 과정 등 시간 소비) 따라서 상호데이터를 송수신하는 과정에 Ack를 데이터에 넣어 보내는 방식을 의미한다.
=>일방 전송 시는 Ack프레임을 따로 전송, 쌍방 전송 시 가능

Ack를 따로 하는 것보다 자신이 보낼 메시지에 같이 보냄으로 더 효율적으로 이용가능하게 한다.