

#1-1

1. 인터넷(Internet) : TCP/IP 기반의 네트워크가 세계적으로 확대되어 연결된 네트워크들의 네트워크를 의미한다. 네트워크를 묶은 거다.
 - 수십억이 컴퓨터 디바이스를 통해 연결되게 해줌.
 - RFC, IETF 등의 표준 규격이 있음.
2. 프로토콜(Protocol) : Network 의 Entity 간 송수신되는 메시지의 순서와 우선순위를 정해놓은 것. 대표적인 프로토콜로 웹에서 사용되는 HTTP, IP, MAC, ARP 등이 있다.

네트워크 구조

1. 네트워크 가장자리(network edge)

네트워크 edge 는 이름대로 네트워크의 가장자리(종단)를 의미한다. 이러한 종단 시스템은 스위치와 라우터를 걸쳐 ISP 를 통해 수많은 네트워크와 연결된다. hosts, access net, physical media 등이 예이다.

- ISP 는 뭔데? (Internet Service Provider) : 인터넷에 접속하는 수단을 제공하는 주체를 가리키는 말이다. 그 주체는 대부분 영리를 목적으로 하는 사기업이지만 비영리를 목적으로 하는 비영리 단체일수도 있다. Ex) KT, SKT, LG U+...

ISP 는 크게 3 가지로 분류가 가능하다.

1. 인터넷 통신망을 보유하여 인터넷 회선과 IP 할당까지 담당하는 회사
 2. 인터넷 통신망을 보유하고 있지만, 회선만 임대하고 IP 할당은 하지 않은 회선 임대료만으로 수익을 내는 회사
 3. 자체적으로 보유하는 통신망을 없지만, 다른 회사의 통신망을 임대받고 말 그대로 인터넷 서비스만 하는 회사
- 대한민국의 경우 통신망을 보유하는 회사가 인터넷 서비스도 제공하기 때문에 인터넷 서비스만 제공하는 회사는 상당히 드문 편이다.

- Host 는?

- host 는 function(packet)을 sending 한다.
- Packet 이라 불리는 작은 단위로 나누어 L 비트 길이 단위로 수신한다.
- 패킷을 Transmission Rate R 의 길이의 네트워크로 송신한다.

기본적으로 Host 의 패킷 전송 지연(Transmission delay) 속도는 Packet 의 길이 L(bits) / Transmission rate 의 길이 R (bits/sec)로 불린다.

$$\text{packet transmission delay} = \text{time needed to transmit } L\text{-bit packet into link} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

2. 접속 네트워크(access network)

접속 네트워크란 말 그대로 네트워크에 접속하는 네트워크를 말한다. 즉 종단 시스템을 그 종단 시스템으로부터 다른 먼거리의 종단 시스템까지의 경로상에 있는 첫번째 라우터에 연결하는 네트워크이다. 말로는 뭔가 애매한데 그냥 단순하게 랜선을 떠올리면 된다. 유선으로 네트워크에 접속하기 위해선 랜선을 꼽아서 사용해야 하고 무선으로 네트워크에 접속하기 위해서는 와이파이를 이용하여 네트워크에 접속해야한다.

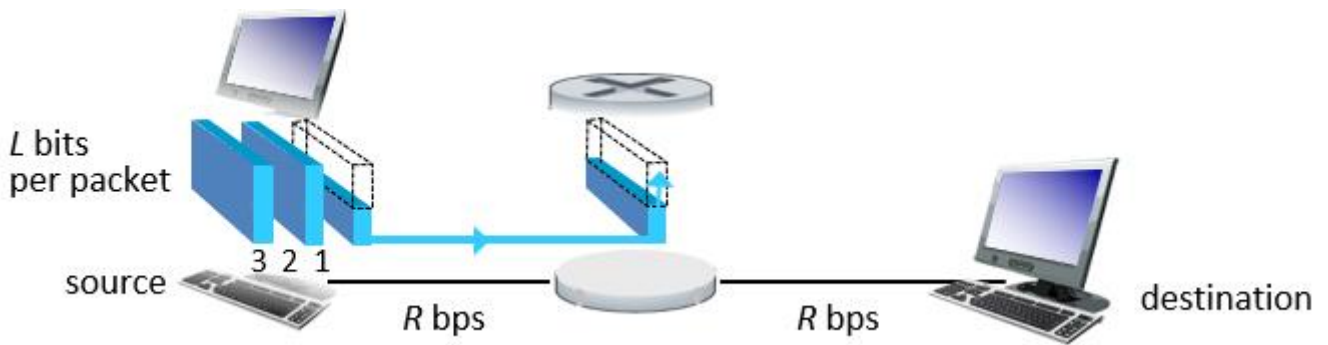
가정이 인터넷에 어떻게 연결되는지를 살펴보면 가장 널리 보급된 광대역 가정 접속 유형은 DSL 과 케이블이다. 일반적으로 가정은 유선 로컬 전화 서비스를 제공하는 같은 지역 전화 회사로부터 DSL 인터넷 접속 서비스를 받는다.

DSL 은 뭔데? Digital subscriber line

3. 네트워크 코어(network core)

네트워크 코어는 종단에 흩어져있는 시스템을 연결시켜주는 중간부분이다. 네트워크에 돌아다니는 수많은 패킷들을 패킷 교환 방식(Packet Switching)과 회선교환 방식(Circuit Switching)을 이용해 연결시켜준다.

- 패킷 교환 방식(packet switching)



저장-후-전달(Store-and-forward) 방식을 이용한다. 저장-후 전달은 쉽게 말하면 선저장 후전달 방식인데, 말 그대로 그림을 보면 가운데에 있는 라우터가 해당 패킷을 모두 받아야지만(자세히 말하자면 라우터의 버퍼) 출력 링크를 통해 목적지로 패킷을 전송시킬수 있는 방식을 말한다. 위 그림에서는 1 번 패킷이 라우터로 전송되고 있다. 한 절반 정도까지만 전달 기 때문에 이 1 번 패킷이 출력되기 위해서는 나머지 절반을 모두 다 받아야지(즉 1 번 패킷의 모든 비트를 다 받아야지만) 내보낼수 있다는 소리이다.

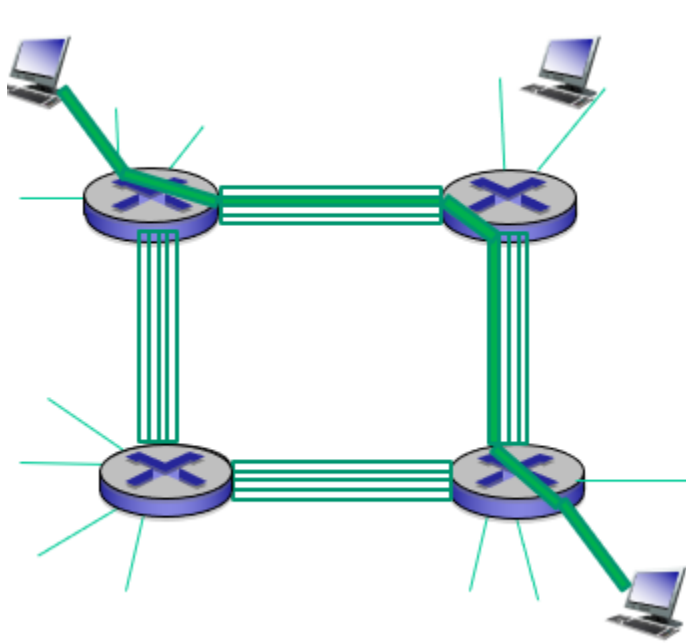
저장-후-전달 전송에 대한 이해를 돕기 위해 송신 시스템에서 패킷을 송신하기 시작해서 전체 패킷을 수신 시스템에서 수신할 때까지 경과된 시간을 계산 해 보자. (전파지연은 무시하고) 송신 시스템이 라우터를 향해 모든 패킷을 전송하는데 L/R 초 시간이 걸렸고 라우터에서 목적지까지 L/R 초 시간이 걸렸으므로 총 송신-수신 까지의 걸린 시간은 $2L/R$ 초 만큼 걸린다.

이렇게 $2L/R$ 가 지나게 되면 1 번 패킷은 도착하고 2 번패킷은 라우터에 있다. L/R 초가 더 지나면 2 번패킷도 수신 시스템에 도착하고 3 번 패킷은 라우터에 도착한다. 마지막으로 L/R 초가 더 흐르게 되면 마지막 3 번 패킷이 라우터를 빠져나가 수신시스템에 도착하게 된다.

따라서 최종적으로 3 개의 모든 패킷은 $4L/R$ 초 만에 송신시스템에서 출발하여 수신시스템에 도착하게 된다. 하지만 이건 패킷이 하나도 막히지 않는 이상적인 상태를 말하고 실제 네트워크는 추석날 귀성길마냥 지연 시간이 존재한다. 이를 큐잉 지연(queueing delay)이라고 한다

각 패킷 스위치는 접속된 여러 개의 링크를 갖고 있다. 각 링크에 대해 패킷 스위치는 출력 버퍼를 갖고 있으며, 그 링크로 송신하려고 하는 패킷을 저장하고 있다. 만약 버퍼에 도착한 패킷이 출력 링크로 나가야 하는데 그 해당 링크가 이미 다른 패킷을 내보내고 있다면 도착한 패킷은 버퍼에서 대기해야한다. 따라서 저장-후-전달 방식에서의 전체 패킷이 버퍼에 도착해야지만 출력 링크로 나갈 수 있는 지연시간과 출력 버퍼에서 대기해야 하는 큐잉 지연을 겪게 된다. 이 지연은 가변적이고 네트워크의 혼잡 정도에 따라 다르다. 마치 톨게이트에서 기다리는 수많은 차량을 보는 듯 하다. 또한 버퍼 공간의 크기는 유한하기 때문에 버퍼가 만약 꽉차 있으면 버퍼에 들어가지 못하고 패킷이 버려지는 패킷 손실이 발생한다.

- 회선 교환 방식



굵은 초록색의 회선을 두 호스트가 사용하게 되면 다른 호스트 들은 이 회선을 사용하지 못한다. 각 링크가 4 개의 회선을 가지므로 라우터 간에 1Mbps의 전송속도를 갖는다면 각 회선 교환 연결은 250kbps의 속도를 얻게 된다.

링크와 스위치의 네트워크를 통해 데이터를 이동시키는 방식에는 패킷 교환 방식 말고 회선교환 방식이라는 것이 있다. 회선교환 방식에서는 종단 시스템 간에 통신을 제공하기 위해 경로상에 필요한 자원은 세션이 유지되는 동안에는 예약 되어야 한다. 회선 교환

네트워크의 예로 전통적인 전화망이 있다. 어떤 사람이 전화망을 통해 다른 사람에게 정보를 보내려고 할 때 어떤 일이 일어나는지 생각하자.

일단 내가 친구한테 전화를 하려면 수화기를 들고 번호를 입력한다. 번호를 입력하고 친구가 전화를 딱 받는 순간! 해당 전화망은 아무도 쓰지 못하고 오로지 나만 쓸수 있다는 것이다. 이렇게 연결이 예약된 상태에서 전화를 하게된다면 송신자는 수신자에게 보장된 일정 전송률로 데이터를 보낼수 있다는 장점이 있다. 하지만 연결 동안에는 다른 사람이 사용하지 못하는 단점이 있다. 전화를 하는 도중 다른 친구가 나한테 전화를 해도 상대방이 통화중입니다 라는 소리가 나오는 것처럼 말이다.

패킷 교환 방식 vs 회선 교환 방식

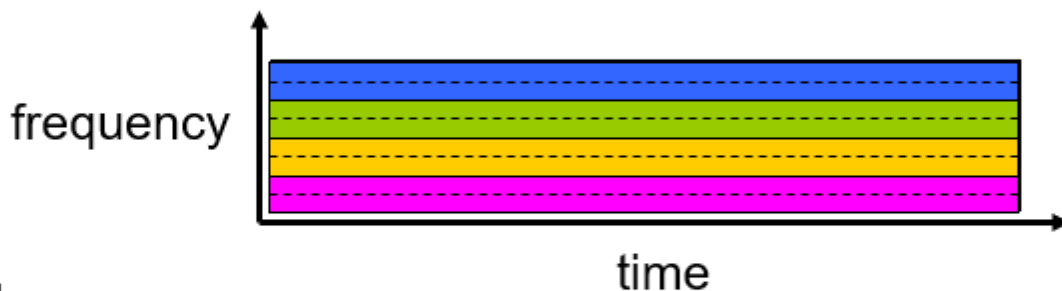
Circuit switching: FDM versus TDM

Example:

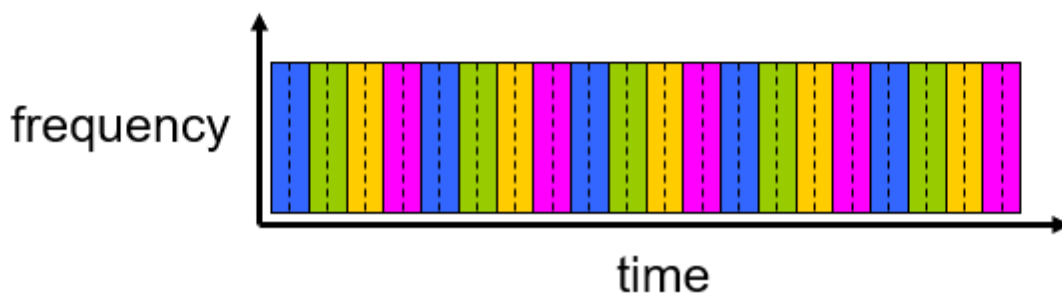
4 users



FDM



TDM



패킷 교환을 반대하는 사람은 가변적이고 예측할 수 없는 종단간의 지연(주로 불규칙적이고 예측할 수 없는 큐잉 지연에서 발생) 때문에 패킷 교환이 실시간 서비스(전화통화와 비디오 회의 통화 같은..)에는 적당하지 않다고 주장했다. 반면에 패킷 교환 옹호자는 다음과 같이 주장한다.

- 1) 패킷 교환이 회선 교환보다 전송용량의 공유에서 더 효율적이다.
- 2) 패킷 교환이 더 간단하고, 효율적이며, 회선 교환보다 구현 비용이 적다.

회선 교환 방식은 회선 요구에 관계없이 그냥 미리 전송 링크의 사용을 할당하고 사용하는 반면에 패킷 교환은 요구할 때만 링크의 사용을 할당하고 그렇지 않을 경우는 할당하지 않아 다른 사람들도 사용할 수 있다는 장점이 있다. 패킷 교환과 회선 교환이 현재 전기통신 네트워크에서 널리 이용되지만, 그 추세는 패킷 교환으로 바뀌고 있다고 한다. 특히 전화망은 비싼 해외 통화 부분을 패킷 교환 방식을 이용한다고 한다.

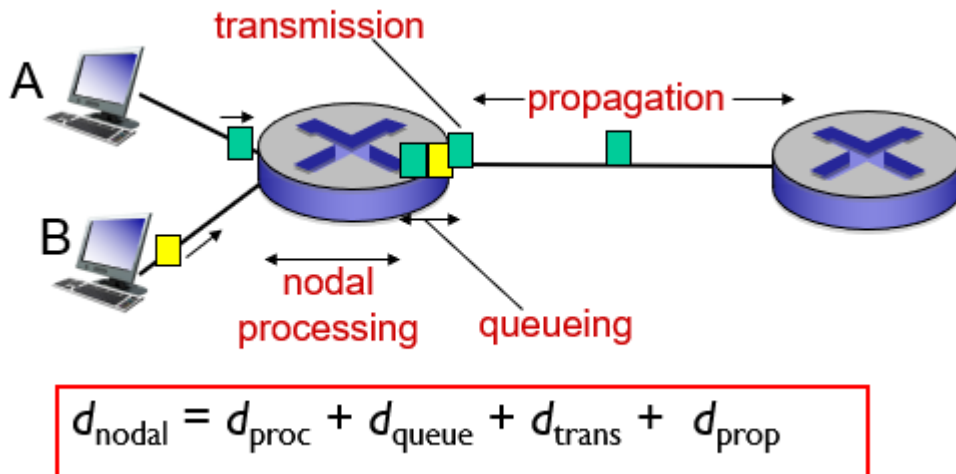
네트워크의 네트워크(Networks Network)

-> 글로벌 시장을 위해, 각 접속 ISP를 글로벌 ISP로 연결시키는 것. 원래 제공자였던 ISP는 글로벌 ISP의 고객(customer)이 된다. 이는 2-계층 구조이다.

1-2. 패킷 교환 네트워크에서의 지연과 손실, 처리율(Delay, Loss, Throughput)

컴퓨터 네트워크는 두 종단(edge)시스템 사이의 처리율을 제한하여 종단 시스템 간의 지연을 야기한다.

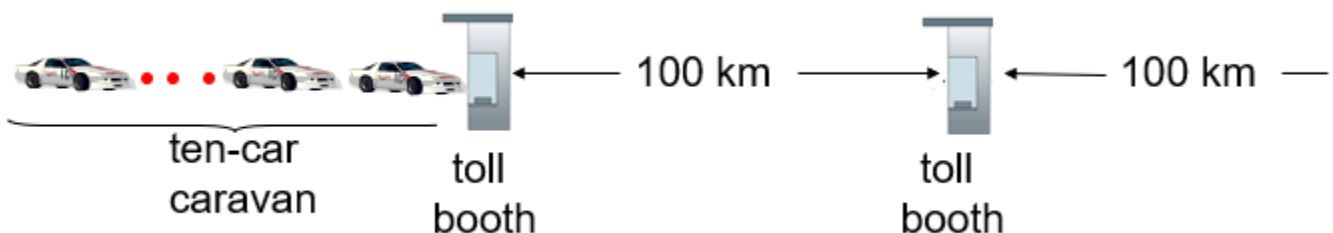
지연 유형의 종류 - 라우터 A에서 라우터 B로 데이터를 보낼 때, 라우터 A에서 일어나는 노드 지연의 특성들을 의미한다. 지연 유형의 종류는 4가지가 있다.



1. 처리 지연(processing delay) - 패킷 헤더를 조사하고, 그 패킷을 어디로 보낼지를 결정하는 시간.
2. 큐잉 지연(Queuing delay) - 패킷에 있는 큐에서 전송되기를 기다리는 시간. 주어진 패킷의 지연은 패킷의 상태마다 상당히 다르다.
3. 전송 지연(Transmission delay) - 패킷은 큐(선입선출)형태로 전송되기 때문에, 한 패킷은 앞서 도착한 다른 모든 패킷이 전송되어야 전송될 수 있다. 이때 패킷의 길이를 L bit, 라우터 A에서 B까지 링크의 전송률은 R bps로 생각해보자. 여기서 전송 지연은 L/R 이다. 이는 패킷의 모든 비트를 링크로 밀어내는 데(전송하는 데) 필요한 시간이다. 전송 지연은 일반적으로 수 마이크로초 ~ 수 밀리초이다.
4. 전파 지연(propagation delay) - 비트가 링크에 전해지면 라우터 B까지 전파되어야 한다. 링크의 처음부터 라우터 B까지의 전파에 걸리는 시간이 전파 지연이다. 전파속도는 두 라우터 사이의 거리 d 를 전파속도 s 로 나눈 것이다. (d/s). 전파속도 s 는 빛의 속도와 비슷하지만, 광역 네트워크같이 먼 거리의 라우터로 보낼 때 전파 지연은 일반적으로 수 초에 이른다.

- 전송 지연과 전파 지연의 차이점

전송 지연은 라우터 사이의 거리와 상관없이, 라우터가 패킷을 보내는 데 필요한 시간이다. 반면 전파 지연은 비트가 한 라우터에서 다음 라우터로 전파되는 데 걸리는 시간이다.



부스와 부스 사이의 거리가 전파 지연, 통게이트를 기다리는 데 걸리는 시간이 전송 지연이다.

노드 지연은 위 4가지를 합친 속도다. ($d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$)

이때 지연 요소의 기여도는 상황에 따라 차이가 있다.

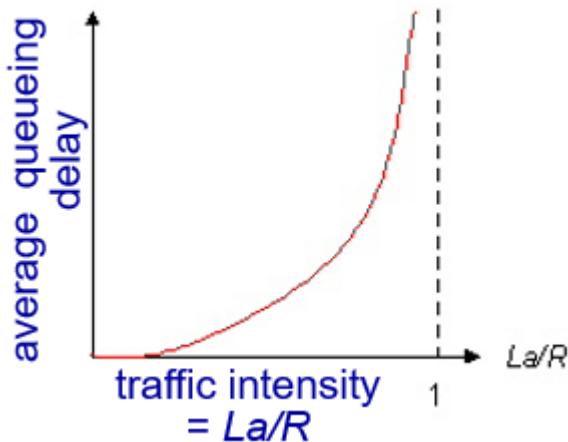
- d_{prop} 은 같은 대학교 캠퍼스 내부의 두 라우터 같이 가까운 장소를 연결하는 링크에서는 거의 무시되지만, 광역 네트워크 간 거리에서는 상당한 거리가 소모되어 중요한 요소가 된다.
- d_{trans} 는 10Mbps 나 그 이상의 전송률인 경우(LAN 등)는 거의 무시할 수 있으나 저속 다이얼업 모델 링크에서 보내지는 커다란 링크일 경우 큰 시간이 소요된다.
- d_{proc} 은 보통 무시된다. 하지만 라우터의 속도가 늘어나면 늘어날수록 상대적으로 큰 영향을 끼친다.

큐잉 지연과 패킷 손실 - 다른 세 개의 지연(d_{prop} , d_{trans} , d_{proc})과는 다르게, 큐잉 지연은 패킷마다 다를 수 있다. 한 패킷이 비어 있는 큐에 도달한다면 바로 출발할 수 있고, 10개가 채워져 있는 큐에 들어간다면 앞의 10개가 전송되기까지 기다릴 것이다. 이런 큐잉 지연 시간을 어떻게 파악할 수 있을까? 몇 가지 가정을 해보자.

일단, 큐는 무한 비트를 가진다고 가정하자. R 은 비트가 큐에서 사라지는 전송률(bps), α 는 패킷이 큐에 도착하는 평균 시간, 패킷의 길이를 L 이라고 생각했을 때

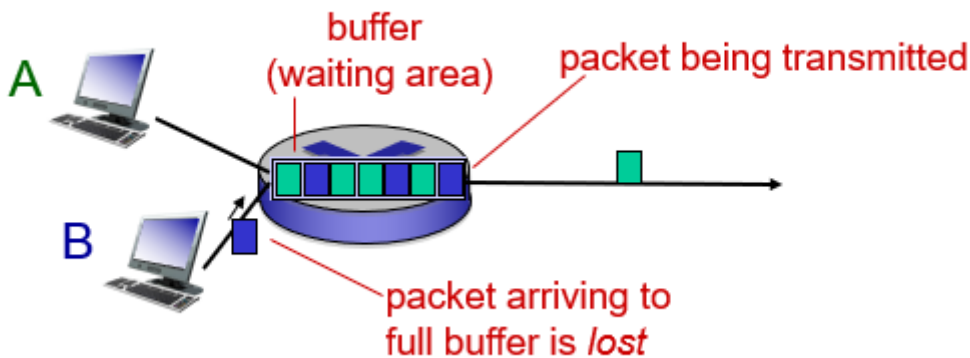
트래픽 강도(traffic intensity)는 $\alpha L / R$ 이라고 정의된다. 트래픽 강도가 바로 큐잉 지연을 측정하는 기준이다. 트래픽 강도가 0 에 가까우면 큐잉 딜레이는 거의 없는 것이고, 1 을 넘으면 비트가 큐에 도착하는 평균율이 비트가 큐에서 전송되는 비율을 초과해 큐가 넘쳐흐르게 된다.(물론 무한하기 문에 넘치진 않겠지만.) 따라서 트래픽 강도는 1 을 넘으면 안된다.

일반적으로 n 번째 전송되는 패킷이 겪는 큐잉 지연은 $(n-1)L/R$ 초일 것이다. 하지만, 패킷은 일반적으로 큐에 도착하는 프로세스는 Random 하기 때문에, 똑같은 시간에 파박파박 오지 않고, 짧은 시간에 중점적으로 들어온다면 큰 큐잉 지연이 생길 것이다. 트래픽 강도가 1 에 근접할수록 평균 큐잉 지연은 급속하게 증가한다. 이는 도로에 차가 많아질수록 교통체증이 걸려 걸리는 시간이 급등하는 것과 같다.



패킷 손실(packet loss)

현실적으로 큐는 무한대의 용량을 가질 수 없기 문에, 계속해서 트래픽 강도가 1 이 넘는 상황(큐에 쌓이는 상황)이 지속된다면 큐가 넘쳐흐르게 되고, 라우터는 패킷을 새로 저장할 수 없게 된다. 이때 라우터는 그 패킷을 잃어버린다.(lost) 잃어버린 패킷은 다시 전송되거나, 전송되지 않고 없어질 수도 있다.



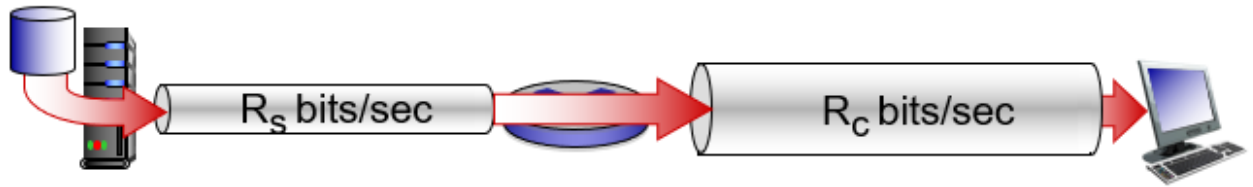
처리율(Throughput) : 종단간 처리율 - rate(bits/time)을 의미한다.

지연 / 패킷 손실과 함께, 처리율로도 컴퓨터 네트워크의 성능을 알 수 있다.

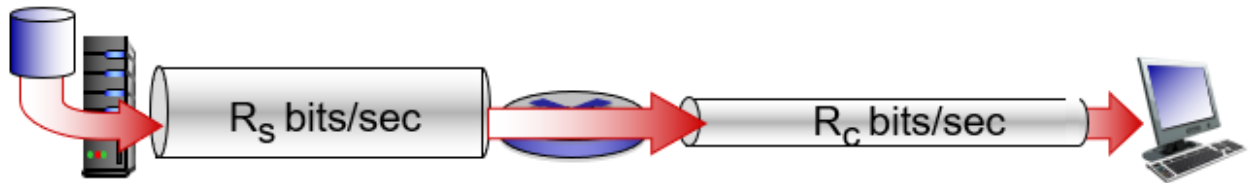
순간적인 처리율(instantaneous throughput) - 호스트가 어느 한 순간 큰 파일을 단위 시간(초)당 수신하는 비율(bit/sec).

평균 처리율(average throughput) - 파일의 크기를 모든 파일을 수신할 때에 걸리는 시간으로 나눈 값.(bit / sec)

■ $R_s < R_c$ What is average end-end throughput?



■ $R_s > R_c$ What is average end-end throughput?



$R_s < R_c$ 일 경우 서버가 배출한 비트는 지연 없이 클라이언트로 흘러갈 것이고,
 $R_s > R_c$ 인 경우 서버가 배출한 비트는 클라이언트로 빠르게 전송되지 못하고 쌓임
 -> 아래의 상황일 경우 필요한 것이 병목 링크(bottleneck link)
 이럴 경우 병목 링크의 전송률이 처리율이 된다.

1-3. 프로토콜 계층구조와 서비스 모델(Protocol Layers, Service models)

네트워크는 복잡하기 때문에 많은 계층으로 나눈다. 이를 프로토콜 계층화(Protocol Layer)라고 한다. 또한 한 계층이 상위 계층에 제공하는 서비스를 계층의 서비스 모델(Service Model)이라고 한다.

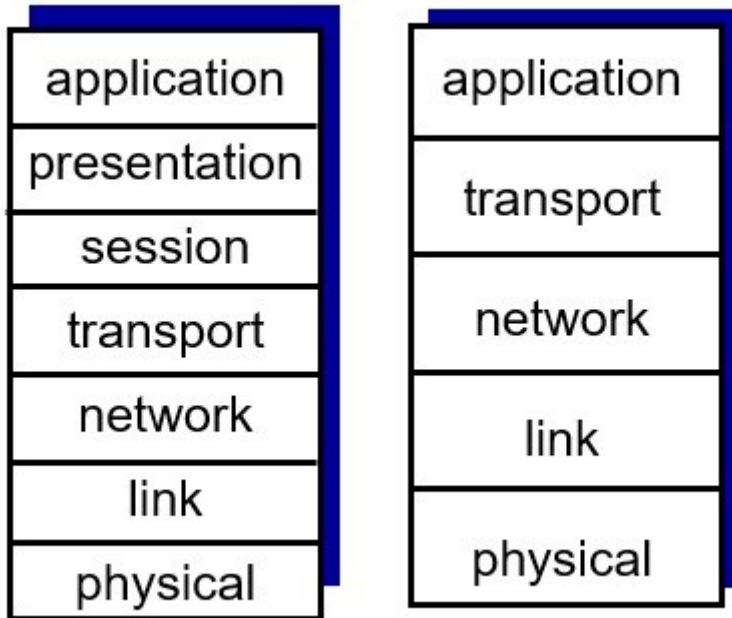
인터넷 프로토콜 스택(Internet protocol stack models) : 5 개 계층으로 이루어진 계층화 모델

1. 애플리케이션 계층(Application Layer) : 네트워크 애플리케이션과 애플리케이션 계층 프로토콜이 있는 곳이다. HTTP, FTP, SMTP 등이 포함된다.
2. 트랜스포트 계층(Transport Layer) : 클라이언트와 서버 간 애플리케이션 계층 메시지를 전송하는 서비스를 제공한다. 인터넷에는 TCP 와 UDP 라는 트랜스포트 프로토콜이 존재한다.
3. 네트워크 계층(Network Layer) : 한 호스트에서 다른 호스트로 데이터그램(datagram)을 라우팅(routing)하는 서비스를 제공한다. 한 노드에서 다른 노드로 이동하기 위해 링크 계층에 의존해야 한다. 인터넷에는 단 하나의 IP Protocol 과 다수의 Routing Protocol 모델이 존재한다.
4. 링크 계층(Link Layer) : 출발지와 목적지 간 일련의 패킷 스위치(라우터)를 통해 데이터그램을 라우팅한다. 프레임(frame)이라고도 불린다. 이더넷, 와이파이, DOCSIS 프로토콜 등이 해당된다.
5. 물리 계층(Physical Layer) : 프레임 내부의 각 비트를 한 노드에서 다음 노드로 이동하는 서비스를 제공한다. 이 계층의 프로토콜들은 링크, 더 나아가 링크의 실제 전송 매체(코일쌍선, 광케이블)에 의지한다.

OSI 모델 - OSI 모델은 5 계층이 아닌 2 계층을 더한 7 계층을 주장하였다.

프리젠테이션 계층 - 애플리케이션 계층과 세션 계층 사이에 있는 계층. 통신하는 애플리케이션들이 교환되는 데이터의 의미를 해석/압축/암호화하도록 하는 서비스를 제공한다. - 애플리케이션이 데이터가 저장되는 내부 포맷을 신경쓰지 않게 하기 위해 제작됨

세션 계층 - 프리젠테이션과 트랜스포트 계층 사이에 존재한다. 데이터 교환의 경계와 동기화를 제공한다. 체크포인트와 Recovery 를 담당한다.



계층에서의 캡슐화(encapsulation)가 중요하다.

2-1. 어플리케이션 계층(application layer)

네트워크 애플리케이션은 컴퓨터 네트워크의 정수이다. 인터넷, 게임, 메신저, 유튜브 등의 애플리케이션이 없다면, 복잡한 네트워킹 기술은 필요하지도 않았을 것이다. 반대로 수많은 애플리케이션이 만들어 진 이유 또한 네트워크의 발전 덕분이다.

애플리케이션 기획자는 소프트웨어를 작성할때 Network-core device 의 작동을 염두에 둘 필요가 없다. 네트워크 코어 장비는 애플리케이션 계층에서는 동작하지 않고 네트워크 및 그 하위 계층에만 작동하기 때문.-> 본인이 할 수 있는 일에만 집중할 수 있다!

애플리케이션 구조(application architecture) - 서버와 어플리케이션의 관계에서, 어플리케이션이 다양한 종단 시스템에서 어떻게 조직되어야 하는지를 지시하는 구조. 대표적으로 클라이언트-서버(client-server)구조와 P2P 구조가 있다.

1. 클라이언트-서버 구조(Client-Server atchitcture)

클라이언트-서버 구조에서 서버는 항상 켜져있는 호스트를 의미하는데, 이 호스트는 클라이언트라는 다른 수많은 호스트로부터 요청을 받는다. 클라이언트의 호스트는 서버와 달리 가끔씩만 켜져있을 수도 있다.

- 대표적인 예 : 웹 어플리케이션(서버가 고정 IP 주소를 가지고 있고, 클라이언트가 서버 주소로 패킷을 보내면 서버가 다운되지 않는 한 항상 서버에 연결할 수 있다)
 , 파일 전송, 원격 로그인, 전자메일 등등..

2. P2P 구조(Peer-to-Peer)

P2P 구조는 항상 켜져있는 기반구조 서버를 거의 사용하지 않거나, 사용하더라도 최소한도만 사용한다. 대신에 애플리케이션은 Peer 라는 간헐적으로 연결된 호스트 쌍이 직접 통신하도록 한다. 특정 서버를 중심으로 한 중앙집권 체제가 아닌, 분산 체제이다. 자기 확장성(self-scalability)을 가지고 있어 중앙 서버에 큰 부하가 가지 않고 각 피어끼리 자원을 공유하기 문에 비용적으로 효율적이지만, 보안/성능/신뢰성에 문제가 생길 수도 있다.

- 대표적인 예 : 비트토েন্ট(파일 공유), 인터넷 전화(스카이프), 피어-지원 다운로드 가속기(선레이) 등이 있다. 온디스크 등의 수많은 다운로드 사이트도 P2P 구조를 이용한다(그래서 유저 컴퓨터의 속도 하락에 일조한다..하하 -_-)

또한 1,2 번을 결합한 하이브리드 구조를 가지는 경우도 있다. 예를 들어, 많은 메시지를 운용하는 어플리케이션에서 서버는 사용자의 IP 주소만 찾고 사용자 간 메시지는 사용자 호스트 사이에 이뤄지는 식이다.

프로세스 간 통신(Processes Communicating)

프로세스 : 종단 시스템에서 운영되는 프로그램. 운영체제에서 실제 통신하는 것은 프로그램이 아니라 프로세스(process)이다. 통신 프로세스가 같은 종단 시스템에서 실행될 때 프로세스 간 통신한다(Processes Communicate)라고 정의한다.

- 프로세스 간 통신을 위한 규칙은 종단 시스템의 운영체제(OS)에 의해 관리된다.
- 서로 다른 종단 시스템에서 프로세스는 컴퓨터 네트워크를 통한 메시지 교환으로 서로 통신한다.

클라이언트-서버 프로세스(Client - Server Process)

클라이언트-서버 구조에서는 클라이언트와 서버의 프로세스가 서로 정보를 교류한다. 예를 들어, 웹 어플리케이션에서 클라이언트 브라우저 프로세스는 웹 서버 프로세스와 메시지를 교환한다. 프로세스는 클라이언트와 서버 두 가지의 역할을 모두 다 할 수 있는데, 두 프로세스는 다음과 같은 기준으로 구분한다.

- 두 프로세스 간의 통신 세션에서 통신을 초기화하면 프로세스 클라이언트, 세션을 시작하기 위해 접속을 기다리면 프로세스 서버라고 한다.

P2P의 경우에서, 종단 시스템의 프로세스는 클라이언트와 서버 둘 다 될 수 있다.

소켓 - 호스트의 애플리케이션과 전송 계층 간의 인터페이스. 애플리케이션과 네트워크 사이의 API라고도 한다.

- 프로세스는 소켓을 통해 메시지를 보내고 받는다.
- 프로세스가 마치 방(house)이라면, 소켓은 문(door)로 표현된다.
- 애플리케이션 개발자는 소켓의 애플리케이션 계층은 통제할 수 있지만, 전송 계층은 거의 손대지 못한다.

어떤 Transport Service가 애플리케이션에 필요한가?

- data integrity - 신뢰적 데이터 전송(데이터가 중간에 새면 안된다)
- timing - 시간을 보장해야 한다. 실시간 애플리케이션(게임)은 더더욱!
- throughput - 처리량. 대역폭은 많으면 많을수록 좋지만 탄력적 애플리케이션(elastic application)보다 대역폭 민감 애플리케이션(bandwidth-sensitive application)에서 더욱 많이 필요하다.
- security - 보안은 당연히 지켜져야 한다.

TCP(Transmission Control Protocol)와 UDP(User Datagram Protocol)

- 인터넷(또는 일반적인 TCP/IP Network)이 제공하는 두 프로토콜
- 두 프로토콜은 애플리케이션에게 서로 다른 서비스 모델을 제공한다.
- 기본적으로 암호화는 제공되지 않는다.
- SSL(Secure Socket Layer) - TCP에 보안을 추가한 패킷.

TCP 서비스 - 애플리케이션이 TCP 전송 프로토콜을 사용하면, 연결지향형 서비스와 신뢰적 데이터 전송 서비스, 혼잡제어 방식 서비스를 받게 된다.

UDP 서비스 - 최소의 서비스 모델을 가진 간단한 전송 프로토콜이다. 비연결형이므로 두 프로세스가 통신하기 전에는 핸드셰이킹을 하지 않는다. 또한 혼잡제어도 하지 않는다.

2-2. Web page와 Http

HTTP(HyperText Transfer Protocol) - 대표적인 클라이언트-서버 프로그램.

- 웹 클라이언트가 웹 서버에게 웹 페이지를 요구하거나 서버가 클라이언트로 어떻게 전송하는지를 정의한다.

Web page - 객체(object)들로 구성되어 있는 문서. 여기서 객체는 단일 URL로 지정할 수 있는 하나의 파일이다. 대부분의 웹 페이지는 기본 HTML 파일과 여러 참조 객체로 구성된다. TCP를 전송 프로토콜로 사용한다.

비지속 연결과 지속 연결 - 각 요구/응답이 분리된 TCP 연결상으로 보내지는걸 비지속 연결, 모든 요구와 응답이 같은 TCP 연결상으로 보내지는 것을 지속 연결이라고 한다. HTTP에서는 비지속/지속 모두 사용가능하다.

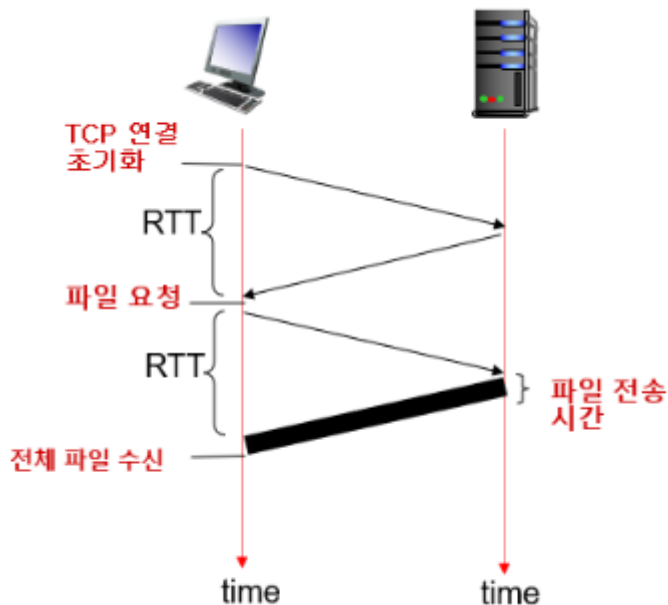
- 비지속연결 HTTP(Non-persistent HTTP)

웹 페이지를 서버에서 클라이언트로 전송하는 단계를 살펴보자. 페이지가 기본 HTML 파일과 5개의 JPEG 이미지로 구성되고, 이 6개의 객체가 같은 서버에 있다고 가정하자. 기본 HTML 파일의 URL은 다음과 같다.

`http://www.someSchool.edu/someDepartment/home.index`

1. HTTP 클라이언트는 HTTP의 기본 포트 번호 80을 통해 `www.someschool.edu` 서버로 TCP 연결을 시도한다.
2. HTTP 클라이언트는 1단계에서 설정된 TCP 연결 소켓을 통해 서버로 HTTP 요청 메시지를 보낸다. 이 요청 메시지는 `/someDepartment/home.index` 경로 이름을 포함한다.

3. HTTP 서버는 1 단계에서 설정된 연결 소켓을 통하여 요청 메시지를 받는다. 저장장치로부터 /someDepartment/home.index 객체를 추출하고 HTTP 응답 메시지에 그 객체를 캡슐화한다. 그리고 응답 메시지를 소켓을 통해 클라이언트로 보낸다.
4. HTTP 서버는 TCP 에게 TCP 연결을 끊으라고 한다.(실제로는 TCP 클라이언트가 응답 메시지를 올바로 받을때 까지 연결을 끊지 않는다.)
5. HTTP 클라이언트가 응답 메시지를 받으면, TCP 연결이 중단된다. 메시지는 캡슐화된 객체가 HTML 파일인 것을 나타낸다. 클라이언트는 응답 메시지로부터 파일을 추출하고 HTML 파일을 조사하고 5 개의 JPEG 객체에 대한 참조를 찾는다.
6. 그 이후에 참조되는 각 JPEG 객체에 대하여 처음 4 단계를 반복한다.



위 그림을 보면 쉽게 이해 될 것이다. 우선 RTT 라는 것은 패킷이 클라이언트로부터 서버까지 가고, 다시 클라이언트로 되돌아오는 데 걸리는 시간을 뜻한다. 처음에 TCP 연결을 해야 한다(3-way 핸드셰이킹). 첫번째 RTT 가 이 TCP 연결을 위한 세팅까지 걸리는 시간을 뜻한다. TCP 연결이 세팅이 되면 그때부터 파일을 요청한다. 하나의 파일에 대해서 서버가 요청을 받고 해당 요청을 처리하는데 (요청하는 파일을 전송하는데) 걸리는 시간을 TR 이라고 하자. 위의 검은색 부분이 TR 을 뜻한다. (요청을 하고 요청에 대한 응답이 올때까지 아무 동작도 못한다.)

결국 하나의 파일에 대해서 1.TCP 연결 2. 파일 요청/응답 에 걸리는 총 시간을 구해보면 $RTT+RTT+TR$ 이다.

비지속 연결이기 때문에 서버는 위 과정이 완료되면 연결을 종료한다. 현재 HTML 문서 하나를 요청한 것에 대한 과정이라고 보자. 우리는 1개의 HTML 문서와 5 개의 이미지를 요청하는 것이기 때문에 나머지 5 개의 파일을 위 과정을 똑같이 거쳐서 요청/응답해야 한다.

첫번째 이미지 파일에 대해서도 똑같이 TCP 연결 부터 시작해야 하기 때문에 총 5 개의 파일도 똑같이 $(2RTT+TR)$ 씩 걸리고 최종적으로 6 개의 파일전송에 요청 해서 응답받는데 까지 걸리는 시간은 $(2RTT+TR)*6=12RTT+6TR$ 이다.

2. 지속연결 HTTP(Persistent HTTP)

위에서 비지속 연결에 대해서 설명했는데 딱봐도 뭔가 단점이 확 보이지 않는가? 매번 파일을 요청할때마다 TCP 연결을 요청해야 해야 하기 때문에 많은 부하가 걸릴것이다. 따라서 맨처음 파일 요청을 위한 TCP 연결이 세팅된 후에 부터는 또다시 TCP 연결은 하지 않고 처음에 연결된 TCP 연결을 이용하여 나머지 파일에 대해서 클라이언트와 서버가 요청/응답을 하는 것이 바로 지속 연결 HTTP 이다. 또한 요청에 대한 응답을 기다리지 않고 한번에 연속해서 요청을 할수가 있는데 이를 파이프라이닝 이라고 한다. 서버가 만약 연속된 요구를 수신하게 되면, 서버는 객체를 연속해서 보낸다. HTTP 의 디폴트 모드는 파이프라이닝을 이용한 지속 연결을 사용한다.

1 번에서의 RTT 를 지속연결로 해서 계산해 보자.

처음에는 똑같이 HTML 문서의 요청 및 응답까지의 수신에는 $2RTT+TR$ 이 걸린다. 하지만 그다음 5 개의 이미지 파일은 TCP 연결을 하지 않고 현재 연결된 TCP 를 통해 통신을 하기 때문에 $RTT+TR$ 만큼만 시간이 걸린다. 따라서 총 시간은 $2RTT+TR + (RTT+TR)*5=7RTT+6TR$ 비지속 연결에서 걸린 시간과 비교 해 봤을 때 확실히 더 빠른 시간안에 요청정보를 수신 받을 수 있다.

HTTP Request Message : HTTP Message 는 메시지 포맷을 정의한다.

요청 메시지(HTTP Request Message) - 일반 ASCII text 로 쓰여진다.

request line
(GET, POST,
HEAD commands)

header
lines

carriage return,
line feed at start
of line indicates
end of header lines

carriage return character
line-feed character

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

요청 메시지의 첫 줄은 Request Line 이라 불리며, GET,POST,HEAD,DELETE 값을 가지는 Method Field 과, URL Field, HTTP Version Field 를 가진다. 위 사진 같은 경우에는 Get, /index.html, HTTP/1.1, www 세 가지를 의미한다.

다음 줄부터는 헤더 라인이라고 불리며, 여러가지가 있다.
Host 는 객체가 존재하는 호스트를 명시하고 있다.

응답 메시지(HTTP Response Message)

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
1\r\n
\r\n
data data data data data ...
```

응답 메시지는 크게 Status line, Header lines, data lines(entity body)로 나뉘어진다. Status Line 은 응답 메시지를 의미하며, 각 숫자가 상태 메시지를 가진다.

■ some sample codes:

200 OK

- request succeeded, requested object later in this msg

301 Moved Permanently

- requested object moved, new location specified later in this msg (Location:)

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

-> 404 Error 가 여기서 나왔다.

쿠키 (User-serer state : Cookie)

- Http 서버가 상태를 유지하지 않기 문제(지나간 상태는 저장하지 않기 문제) 성능의 향상을 피할 수 있었지만, 사용자를 확인할 수 없는 문제가 생겼다.

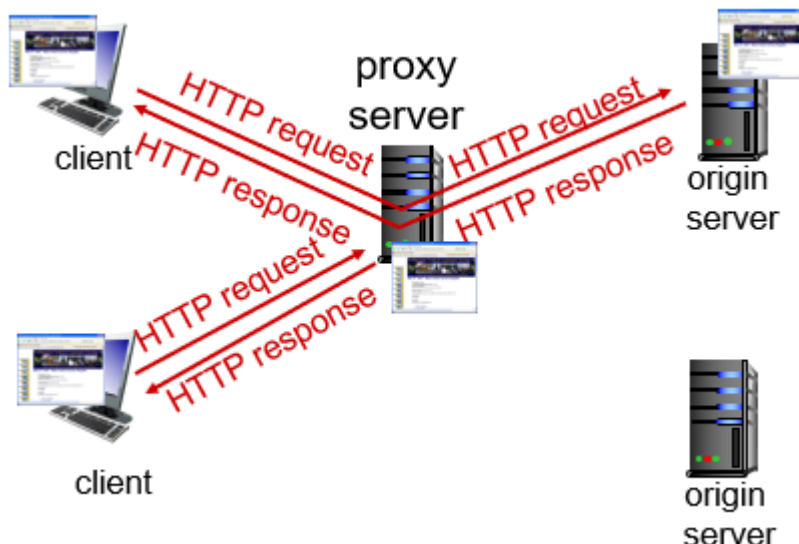
- 이를 해결하기 위해 만든 것이 쿠키다. 대부분의 상용 웹 사이트가 쿠키를 사용.

- 사용자가 HTTP Response Message 를 받았을 때, 서버가 User 의 활동을 추적하고 User 의 컴퓨터 저장하는 것.(웹 서버에 저장하는 경우는 세션 Session 이라고 부른다.) 개인 정보나, 사용자 맞춤형 정보를 입력하는 데에 유용하다.

웹 캐싱(Web Caching) - 프록시 서버(proxy server)라고도 함.

가끔씩 인터넷이 끊겼을 때도, 과거에 켜던 웹을 볼 수 있는 것이 이것 덕분에 가능.

원출처의 웹 서버를 대신해 HTTP 요구를 충족시키는 네트워크 개체이다. 이전에 방문한 웹 사이트의 객체를 저장한다.

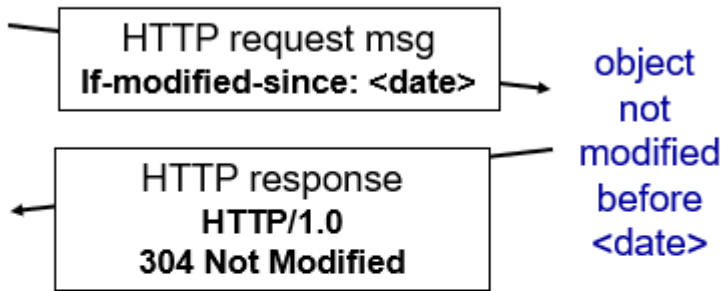


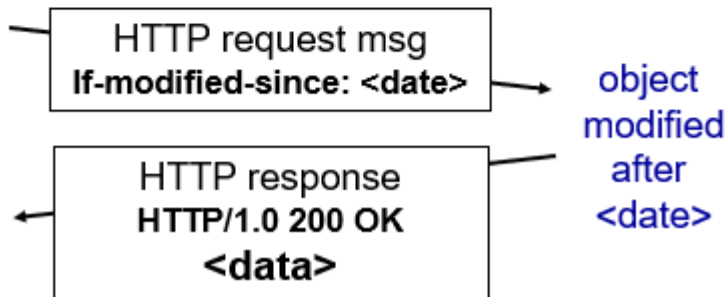
웹 캐시를 사용하는 것만으로 클라이언트의 요구에 따른 응답 시간을 줄일 수 있다. 특히 클라이언트와 원 출처 서버 사이의 병목 대역폭이, 클라이언트와 캐시 사이의 병목 대역폭에 비해 매우 작을 때 더욱 효과적이다.

조건부 GET(Conditional GET)

웹 캐시 내부에 있는 객에 복사본이 Origin server 와 다를 때만 최신화하는 기법.

HTTP 요청 메시지가 GET 방식을 사용하고 있고, If-Modified-Since 헤더를 사용하고 있다면, 그것이 조건부 GET 메시지이다.

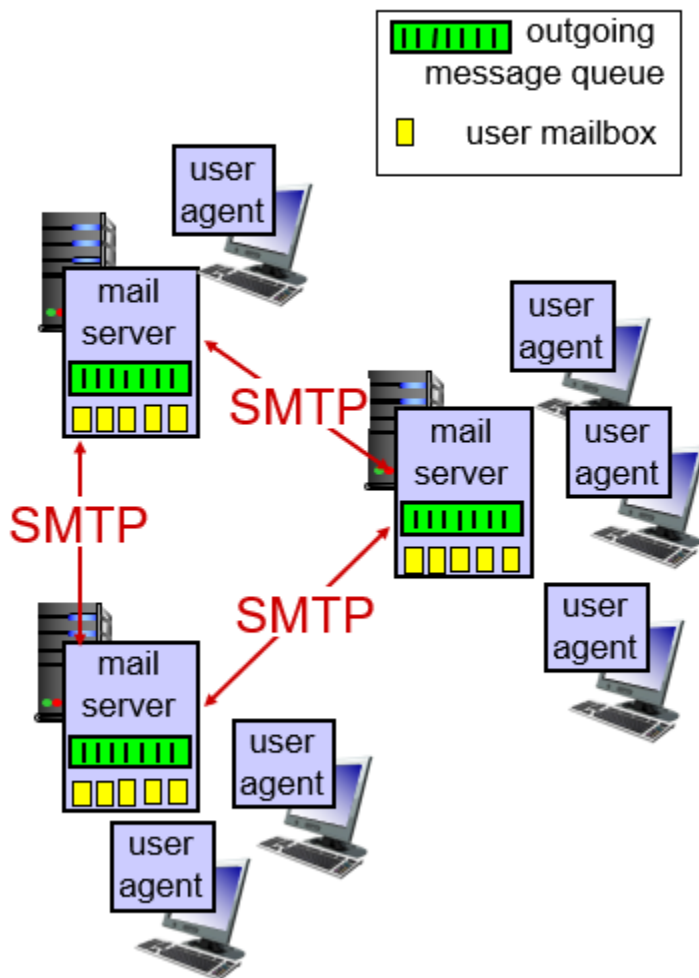




2-3. Internet E-mail

전자메일은 인터넷 중에서도 가장 널리 사용되는 어플리케이션 중 하나이다.

전자메일은 사용자 에이전트, 메일 서버, SMTP(Simple Mail Transfer Protocol)을 이용해 통신한다.



메일 서버는 각 메일 박스를 가지고 있다. 메일박스는 메시지를 관리하고, 메시지를 보낼 수 없는 상황에 메시지 큐에 메시지를 보관하고 나중에 다시 송신한다. 여러 번 시도해도 성공하지 못하면 송신을 취소하고 발신자에게 통보한다.

SMTP는 인터넷 전자메일을 위한 주요 애플리케이션 계층 프로토콜이다. SMTP는 TCP를 이용해 전송한다.

2-4. DNS와 P2P, Video Streaming

네트워크는 서로 다른 종단 시스템에 연결되는 클라이언트와 서버 프로그램으로 구성되고, 서버와 클라이언트 코드를 모두 작성하는 것이 개발자가 해야 할 일이다.

어플리케이션을 개발하는 동안 개발자가 유념해야 할 것은 그 애플리케이션이 TCP를 쓰는지 UDP를 쓰는지 알아야 한다는 것이다. TCP는 연결지향형 서비스며 신뢰적 바이트 스트림을 제공하는 데 반해, UDP는 클라이언트가 서버가 연결되지 않은 비연결형이고 한 종단 시스템에서 다른 곳으로 패킷을 보낼 때 전송에 대한 보장을 하지 않는다.

1. UDP를 이용한 클라이언트-서버 통신

server (running on serverIP)

create socket, port= x:
`serverSocket =
socket(AF_INET, SOCK_DGRAM)`

read datagram from
`serverSocket`

write reply to
`serverSocket`
specifying
client address,
port number

client

create socket:
`clientSocket =
socket(AF_INET, SOCK_DGRAM)`

Create datagram with server IP and
port=x; send datagram via
`clientSocket`

read datagram from
`clientSocket`

close
`clientSocket`

소켓을 이용해, 위 그림과 같은 방식으로 통신하게 된다.(매우 간단한 과정이지만)
이를 파이썬 코드화해보자.

UDPClient.py

```
import socket #소켓 라이브러리

serverName = 'MyHostName'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)#IPv4, UDP 소켓 생성
message = raw.input('Input lowercase sentence 빨리빨리 :')
clientSocket.sendto(message.encode(), (serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode()) #client로부터 받은 메세지
clientSocket.close()
```

UDPServer.py

```
import socket
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print('The server is ready to receive')
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

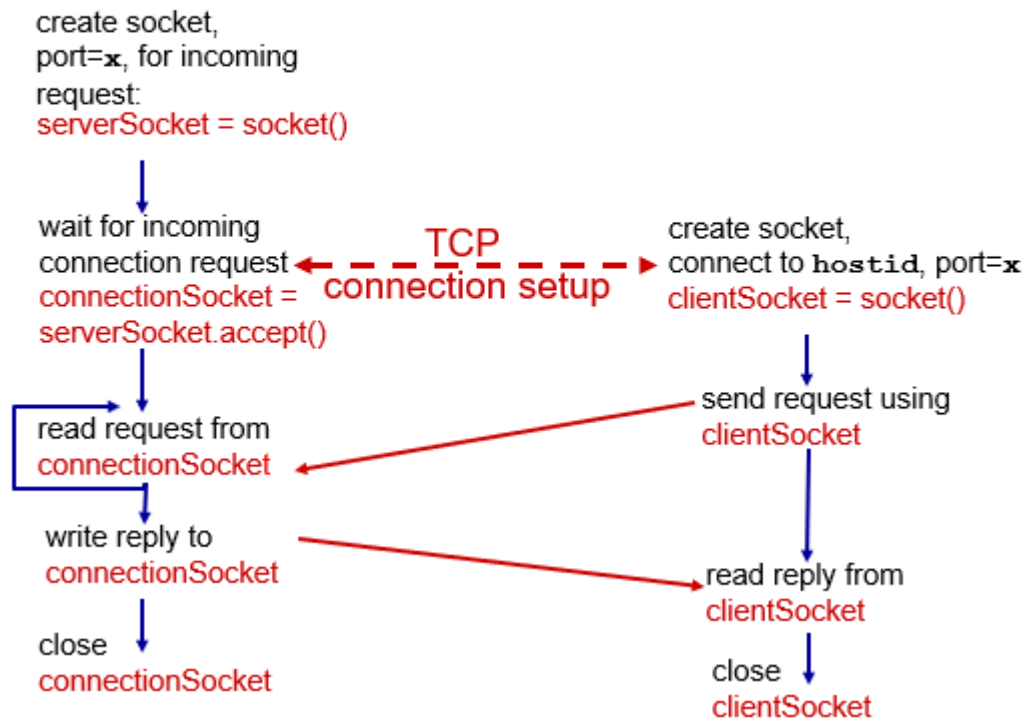
2. TCP를 이용한 클라이언트-서버 통신

TCP는 연결지향 프로토콜로, 클라이언트와 서버가 서로 데이터를 보내기 전에 먼저 TCP 연결을 설정(handshake and establish)해야 한다.

TCP 연결을 설정하면, 한 쪽에서 다른 쪽으로 데이터를 보낼 때 Socket을 통해 데이터를 TCP로 보낸다. TCP는 클라이언트와 환영 소켓, 그리고 또 서버 간 파이프를 운용한다. 그리고 이 세 방향 핸드셰이크를 통해 서버와 클라이언트를 연결한다. 이는 신뢰적(reliable) 프로세스를 가능하게 한다.

server (running on hostid)

client



코드로 살펴보자.

TCPClient.py

```
import socket
serverName = servername
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM) #IPv4, TCP 소켓 생성
clientSocket.connect((serverName,serverPort)) #미리 TCP 연동
sentence = raw_input( Input lowercase sentence: )
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ( From Server: , modifiedSentence.decode())
clientSocket.close()
```

TCPServer.py

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(( ,serverPort))
serverSocket.listen(1)
print The server is ready to receive
while True:
```

```
connectionSocket, addr = serverSocket.accept()
sentence = connectionSocket.recv(1024).decode()
capitalizedSentence = sentence.upper()
connectionSocket.send(capitalizedSentence.encode())
connectionSocket.close()
```

3-1. Transport Layer - 다중화와 역다중화(Multiplexing & Demultiplexing)

Transport Layer

- 네트워크 구조의 핵심 구조. Application process 간 논리적 통신(logical communication)을 제공한다.
- Transport Layer 는 5 계층 모델에서 Application Layer 과 Network Layer 사이에 존재한다.
- 인터넷(TCP/IP Network)에는, TCP 와 UDP 라는 두 가지 Transport Layer Protocol 이 존재한다.

트랜스포트 계층(Transport Layer)과 네트워크 계층(Network Layer)의 관계

network layer: logical communication between hosts(호스트)

transport layer: logical communication between processes(프로세스)

- relies on, enhances, network layer services

앞에서 배운 트랜스포트 계층의 가장 대표적인 프로토콜인 TCP 와 UDP 패킷은 세그먼트(segment)라고 불린다.

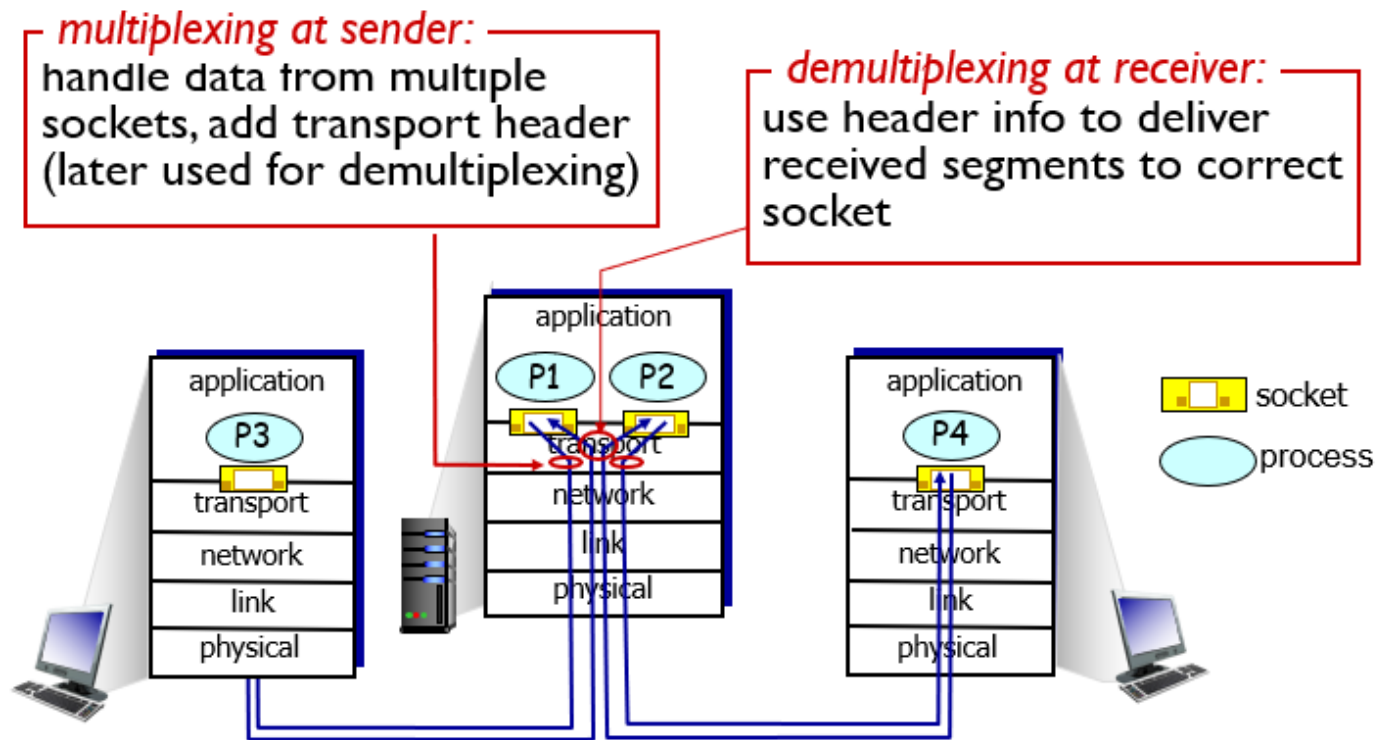
UDP 와 TCP 의 가장 기본적인 기능은 종단 시스템 간 IP 서비스를 종단 시스템에서 동작하는 두 프로세스 간 전달 서비스로 확장하는 것이다. 네트워크 계층에서 진행되는 호스트 대 호스트 전달을 트랜스포트 서비스에서 제공하는 프로세스 대 프로세스 전달로 확장하는 것을 트랜스포트 다중화(Transport multiplexing)와 역다중화(demultiplexing)라고 부른다.

다중화와 역다중화(Multiplexing & Demultiplexing)

사실 다중화와 역다중화는 모든 컴퓨터 네트워크에서 필요한 작업이지만, 여기서 말하는 다중화와 역다중화는 트랜스포트 계층에서만 한정된다.

목적지 호스트에서의 트랜스포트 계층은 네트워크 계층으로부터 segment (UDP/TCP)를 수신한다. 트랜스포트 계층은 호스트에서 동작하는 애플리케이션 프로세스에 이 세그먼트의 데이터를 수신해야 한다. 여기서 필요한 것이 다중화와 역다중화이다.

트랜스포트 계층 세그먼트의 데이터를 어플리케이션의 올바른 소켓으로 전달하는 과정을 역다중화(demultiplexing)라고 한다. 또한 출발지 호스트에서 소켓으로부터 데이터를 모으고, 이에 대한 세그먼트를 생성하기 위해 각 데이터에 헤더 정보로 캡슐화하고, 그 세그먼트들을 네트워크 계층으로 전달하는 작업을 다중화(multiplexing)라고 한다.



다중화와 역다중화를 위해, 소켓은 특별한 필드 두 가지를 가지는데, 1. 출발지 포트 번호 필드(source port number field)와 2. 목적지 포트 번호 필드(destination port number field)이다. 각각의 포트 번호는 0~65535의 16 비트 정수이다. 그 중에서 0~1023(2^{10})까지는 잘 알려진 포트 번호라고 해 사용을 엄격하게 제어하고 있다.

연결형/비연결형(connection / connectionless) 다중화와 역다중화

UDP(비연결형)에서의 역다중화 방식 : UDP 에서, Host 의 각 Socket 은 port number 를 할당받는다. 그리고 Segment(UDP)가 Host 에 도착하면, Transport 계층은 Segment 안의 목적지 포트 번호를 검사하고 상응하는 Socket 으로 Segment 를 보내게 된다. 여기서 UDP 소켓은 목적지 IP 와 목적지 포트 번호 두 가지 요소로 식별되게 된다.(출발지 IP 와 출발지 포트 번호는 고려하지 않는다.)

TCP(연결형)에서의 역다중화 방식 : TCP 역다중화를 수행하기 위해서는, UDP 와 달리 4 개 요소들의 집합(four-tuple)인 출발지 IP 주소, 출발지 포트 번호, 목적지 IP 주소, 목적지 포트 번호에 의해 식별된다는 것이다. 네트워크로부터 호스트에 TCP Segment 가 도착하면 호스트는 Segment 를 전달(역다중화)하기 위해 4 개 값을 모두 사용한다. UDP 와는 다르게 다른 출발지 주소 or 다른 포트 번호를 가지고 도착하는 2 개의 TCP Segment 는 2 개의 다른 소켓으로 향하게 된다.

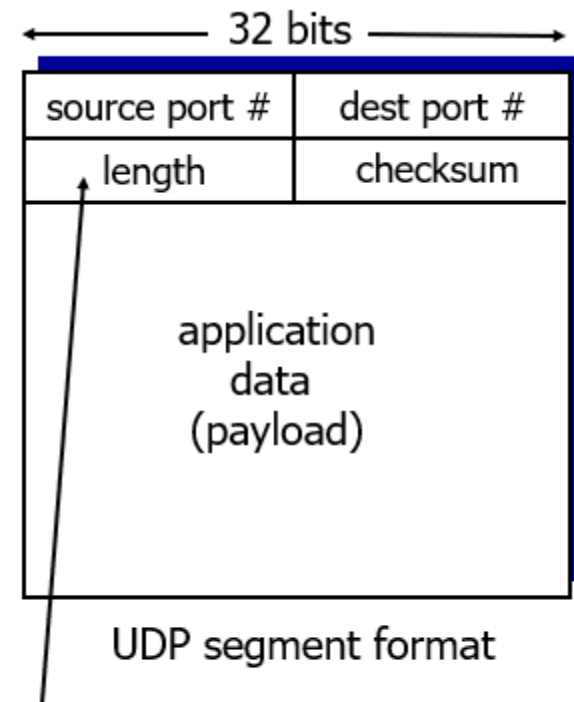
3-2. UDP - 비연결형 트랜스포트(Connectless Transport)

다시 한번 되짚어보자. 트랜스포트 계층의 가장 큰 목적은 네트워크 계층으로부터 받는 데이터를 역다중화해서 애플리케이션에 보내거나, 애플리케이션의 요청 정보를 다중화해서 네트워크에 보내는 것이다.

UDP 는 트랜스포트 계층 프로토콜이 할 수 있는 최소한의 일만 한다. UDP 는 다중화/역다중화 기능과 몇몇 오류 검사를 제외하면 아무것도 하지 않는다. 거기에 UDP 는 TCP 에 비해 혼잡제어 / 신뢰적 데이터 전송 시스템을 제공하지 않기 때문에 현대에서 사용되지 않을 것 같지만, 사실 요즘도 많이 쓰인다고 한다. 그 이유는 뭘까?(마치 Python/Java 가 있는데 왜 C 를 쓰냐고 묻는 것 같다..하하 =_=)

1. 무슨 데이터를 언제 보낼지에 대해 애플리케이션 레벨에서 정교하게 제어 가능
- 혼잡제어 / 신뢰적 데이터 전송이 없기 때문에, 프로그래머가 구현 가능.
2. 연결 설정이 없다 - TCP 의 Three-way handshake 등의 과정이 없기 때문에, 연결을 설정하기 위한 어떤 지연도 없다.
3. 연결 상태가 없다 : 더 많은 클라이언트를 수용할 수 있다.
4. 작은 패킷 헤더 오버헤드(Segment 당 8 바이트로 가볍다.)

UDP Segment 구조와 Checksum



UDP 세그먼트의 구조.

애플리케이션 데이터는 UDP 데이터그램의 Data Field에 load된다. UDP 헤더는 2 바이트씩 구성된 4 개의 헤더를 가진다. 1. 출발지를 의미하는 source port, 2. 도착지를 의미하는 dest port, 3. UDP Segment의 길이를 의미하는 length, 4. Checksum 기능을 가진 Checksum. Checksum 이 뭐지??

Checksum - UDP에서의 Checksum은 오류 검출을 의미한다. Checksum은 Segment가 출발지로부터 목적지로 이동했을 때, UDP Segment 안 bit의 변경사항이 있는지 체크한다. Checksum은 어떻게 구할까?

1. 수신측에서 IP 헤더를 16 비트씩 나눈다
2. 나눈 비트 중 CheckSum 값을 제외하고 나머지를 모두 더한다.
3. Carry 값(새로운 비트로 넘어가는 값)이 발생하면 윤회식 자리올림을 한다(마지막 bit로 잘처리한다)
4. 1의 보수를 취한다
5. 구한 값과 전달받은 Checksum을 비교한다.

예를 들어보자.

4 바이트의 데이터 4 개가 있다 - 0x25, 0x62, 0x3f, 0x52(이진법은 너무 길어)

1. 모든 바이트를 더하면 0x118 이 된다.
2. 이를 이진법으로 바꾸고, Carry 값을 윤회식 자리올림을 하면 0001 0001 1000 -> 0001 1001 이 된다.
3. 1의 보수를 취하면 1110 0110 이 된다.
4. 이를 16 진수로 바꾸면 0xE7 이 된다. Checksum Byte가 만들어졌다.
5. 원래 값과 16 진수를 더하면 11111111 이 된다. 문제가 없다는 것을 체크할 수 있다.

왜 굳이 링크 프로토콜에서 체크하지 않고, UDP에서 체크하는가? -> 출발지와 목적지 사이의 모든 링크가 오류 검사를 제공한다는 보장이 없기 때문에, segment들이 정확하게 링크를 통해 전송되어도, 라우터의 메모리에 저장될 때 오류가 있을수도 있다. 이것이 종단간 원리(End-End principal)의 한 예이다. UDP는 오류 검사를 제공하지만, 오류를 회복해주지는 않는다. 일부 UDP는 손상된 Segment를 그냥 버리고, 다른 곳은 경고와 함께 손상된 Segment를 넘겨준다.

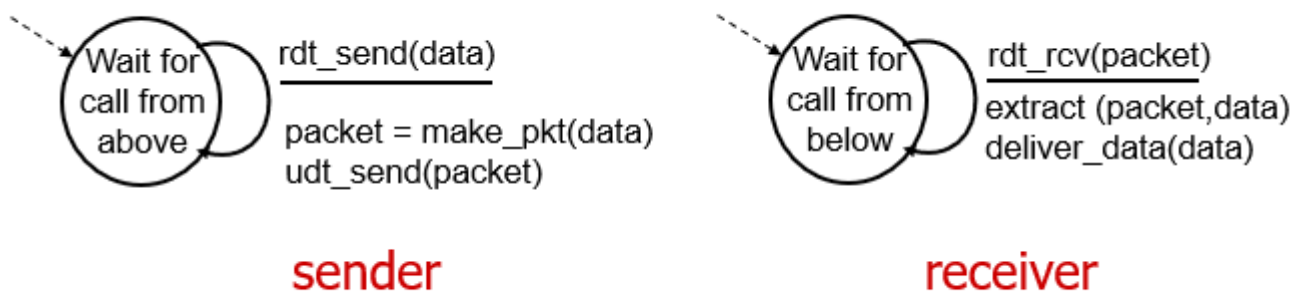
3-3. 신뢰적 데이터 전송 프로토콜(RDT, Reliable data transfer protocol)

1 신뢰적 데이터 전송 프로토콜은 TCP 가 UDP 와 구분되는 가장 큰 특징이며, 트랜스포트 계층에서 매우 중요한 문제이지기도 하지만, 사실 신뢰적 데이터 전송이라는 개념은 좀 더 일반적인 개념이다. 그렇기 때문에 TCP 개념을 배우기 전 신뢰적 데이터 전송 프로토콜(RDT)를 다뤄보도록 한다.

신뢰적 데이터 전송 : 네트워크에서 제일 중요한 개념 중 하나. 사실 가장 중요한 개념으로 봐도 무방. 하위 계층에서 상위 계층에서 데이터를 전송할 때 전송된 데이터가 손상되거나 손실되지 않게 보장하는 개념. 이러한 서비스 추상화를 구현하는 것은 신뢰적 데이터 전송 프로토콜(Reliable data transfer protocol)의 의무이다. 아래에 있는 계층이 쌓이면 쌓일수록, 아래에 있는 계층이 신뢰적이지 않을 가능성이 높아지기 때문에 어려워진다. 아래는 신뢰적 데이터 전송의 단계적 예이다.

Reliable Data Transfer(RDT 의 과정)

RDT 1.0 - 완전히 신뢰적인 채널 상에서의 신뢰적 데이터 전송 : 비트 오류가 존재하지 않고, 하위 채널을 완벽하게 신뢰할 수 있다고 가정해, 본인의 경우만 고려하면 된다. 이에 대한 FSM (Finite State Machine 유한 상태 머신, 컴파일러에서 배움)은 아래와 같다.



`rdt_send` 는 상위 계층으로부터 데이터를 받고 데이터를 포함한 패킷을 생성한 후(`make_pkt(data)`), 이를 패킷에 송신한다(`udt_send(packet)`)

`rdt_receiver` 는 하위의 채널로부터 패킷을 수신하고, 패킷에서 데이터를 추출한 후(`extract(packet_data)`) 이를 상위 계층으로 전달한다(`deliver_data(data)`)

사실 여기서는 패킷이 왜 필요하나 싶을 정도로 데이터와 패킷의 차이점이 없다. 또한 완전히 신뢰적인 상황에서는 오류가 있을 수 없으므로 수신 측이 송신 측에게 어떤 피드백(feedback -> 잘됐다,잘못됐다,전달이 안됐다..)도 제공해 줄 필요가 없다.

RDT 2.0

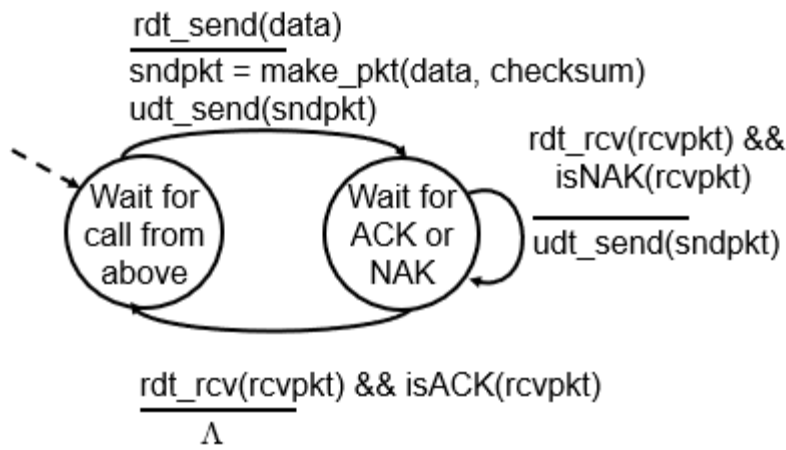
2.0 부터는 신뢰적 데이터 송신을 위해 오류를 검출하고 자동으로 재송신하는 데이터 검출 프로토콜인 ARQ Protocol(Automatic Repeat reQuest, 자동 재전송 요구) 를 사용한다. 비트 오류를 처리하기 위해서는 기본적으로 다음 세 가지 기능들이 요구된다.

오류 검출(Error detection) : 비트 오류가 발생했을 때 수신자가 검출할 수 있는 기능이 필요하다. UDP 의 경우에는 Check-Sum Field 가 이를 수행하였다. 이는 추후에 다루도록 한다.

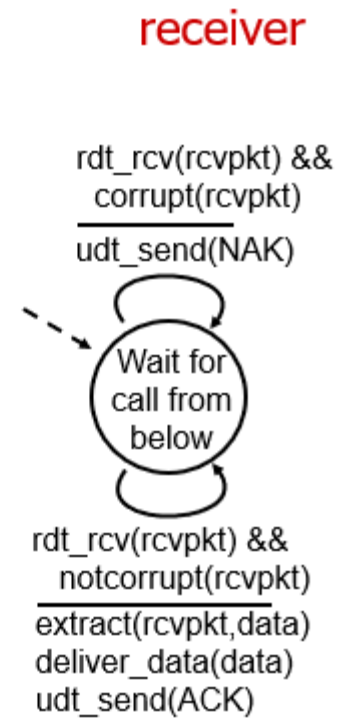
수신자 피드백(Receiver feedback) : 송신자와 수신자가 일반적으로 수천 킬로미터 떨어진 다른 종단 시스템에서 동작하므로, 송신자가 수신자의 상태를 아는 방법은 피드백(괜찮은지,아닌지)을 받는 것이다. 메시지 명령은 긍정 확인응답(ACK, Acknowledgements)과 부정 확인응답(negative acknowledgements)로 나뉜다. 수신자는 송신자로 이러한 응답을 보낸다. 원칙적으로 이런 응답은 이진법(0,1)로 나타낼 수 있다.

재전송 : 수신자에서 오류를 가지고 수신된 패킷은 송신자에 의해 재전송된다.

이 세 단계에서, 송신자는 수신자가 패킷을 정확하게 수신했다는 것을 확인하기 전까지 새로운 데이터를 전달하지 않을 것이다. 이러한 행동 때문에 `rdt 2.0` 프로토콜은 전송-후-대기(stop-and-wait)프로토콜로 알려져 있다.

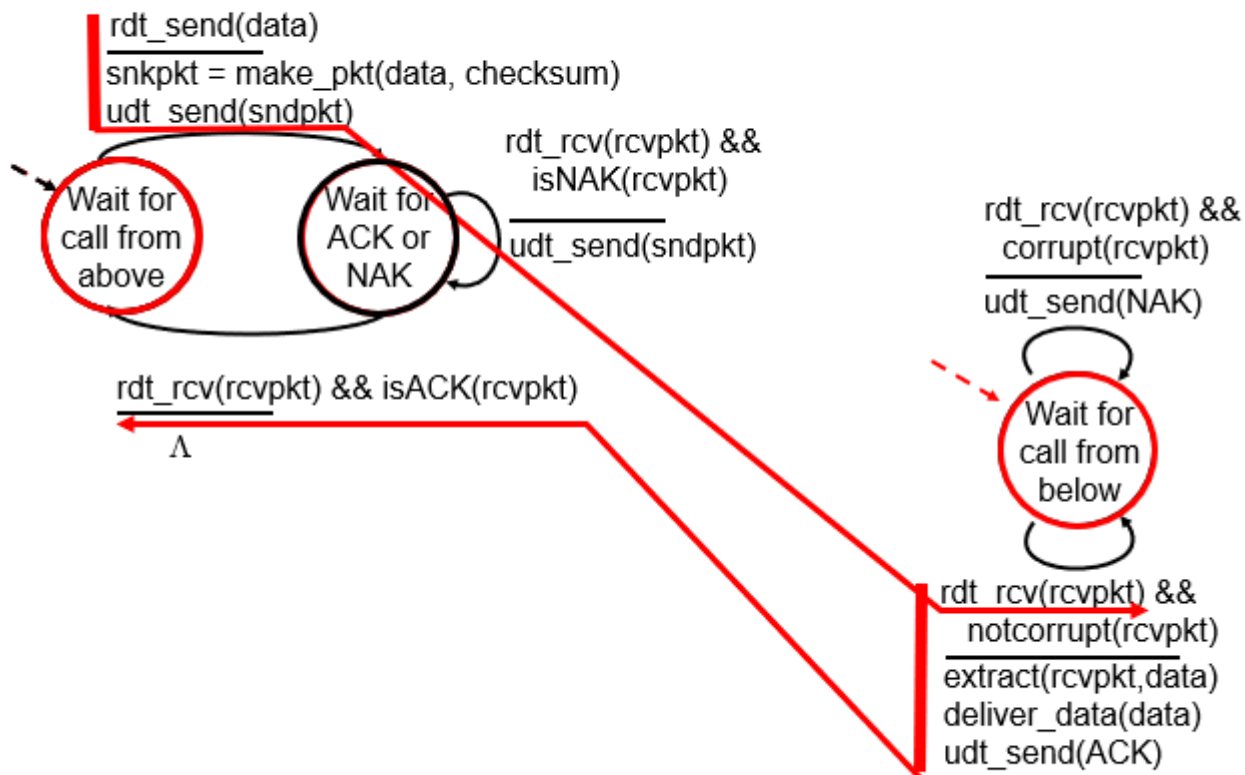


sender

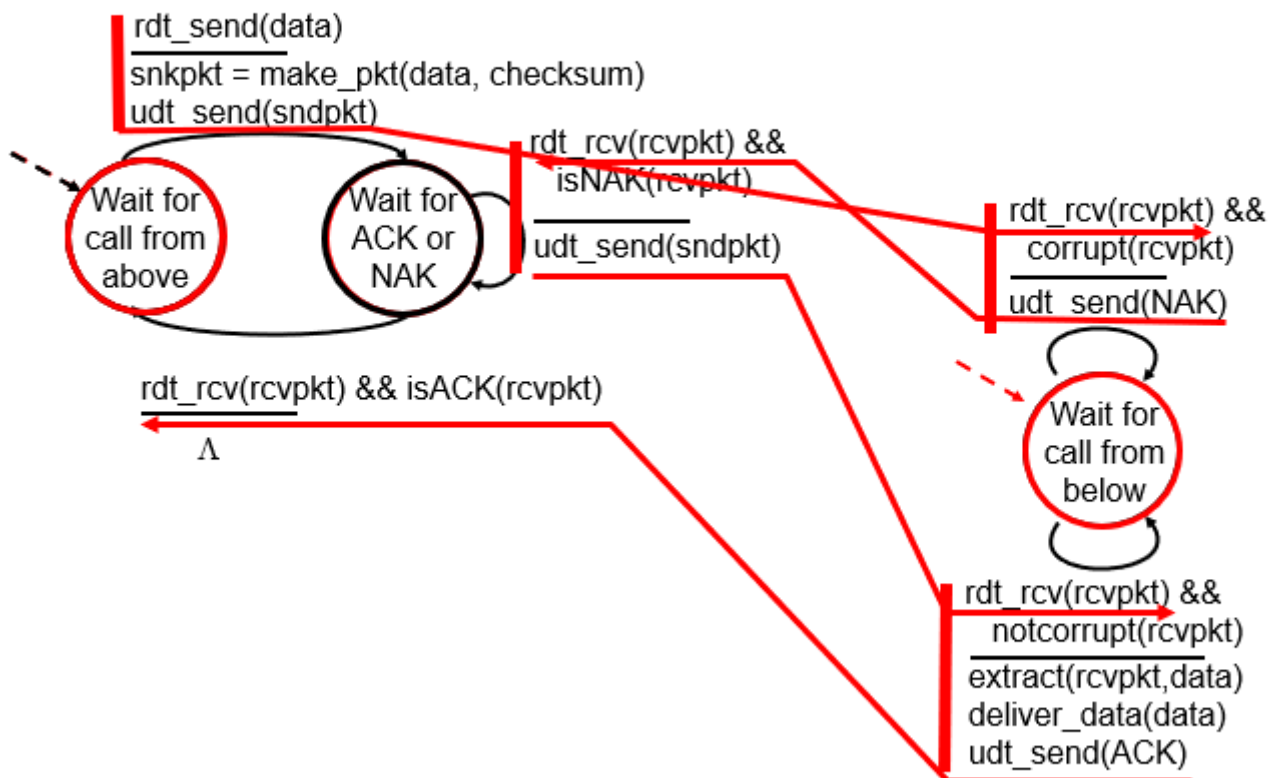


송신자 FSM은 두 가지의 상태를 가진다. 송신자 FSM는 파일을 보내주고 ACK, NAK 상태를 수신자로부터 기다린 이후, 오류가 없다면 상위 계층에서 데이터를 기다리는 상태로 돌아간다. NAK가 수신된다면 프로토콜은 데이터를 재전송하고 또다시 ACK, NAK 상태를 기다린다. 수신자 FSM은 단일 상태를 가진다. 패킷이 도착했을 때 수신자는 수신된 패킷이 손상되었는지 여부에 따라 ACK, NAK로 응답하고 오류가 검출되거나 검출되지 않는 이벤트에 대응한다.

rdt2.0: operation with no errors



rdt2.0: error scenario



Transport Layer 3-31

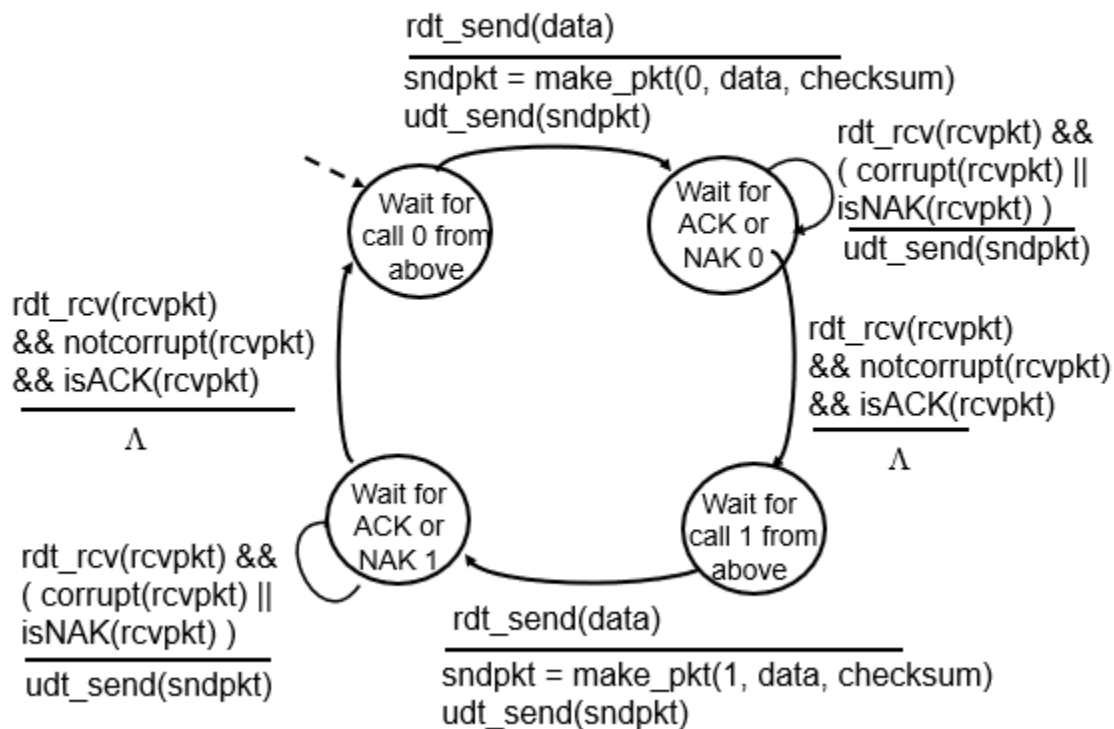
NCK(Error 상태)

rdt 2.0은 잘 동작하는 것 같지만 쓰다보면 치명적인 결함이 발생하는데, 첫 번째로, ACK와 NCK 패킷 자체가 손상될 수 있다는 가정을 하지 않은 것이다. ACK와 NCK를 헛갈려 멀쩡한 비트를 문제가 있다고 판별하거나, 멀쩡하지 않은 비트를 멀쩡하다고 메시지를 보내버리면 문제가 심각해진다. 최소한 이 상태에서만은 체크섬 비트를 추가할 필요가 있다. 또 다른 문제는 어떻게 프로토콜이 ACK 또는 NAK 패킷 오류로부터 복구되는가이다. 만약 ACK 또는 NAK가 손상되었다면, 송신자는 수신자가 전송된 데이터의 몇번째까지 마지막 부분을 올바르게 수신했는지 알 수 없다는 것이다. 예를 들어보자, 수신자가 원래 ACK를 보내야 했는데 이것이 손상되어 송신자에게 NCK로 들어갔다고 치자. 송신자는 잘못된 정보가 보내진 줄 알고 쓸모없는 패킷 정보를 하나 더 보낼 것이고, 혹시 최악의 경우 이 응답이 왜곡될지도 모르는 카오스 상태가 되버린다. 이를 해결하는 것이 rdt 2.1이다.

RDT 2.1

RDT 2.1은 RDT 2.0에 비해 n 배 많은 상태를 가진다. 이는 각각 순서번호 0, 1..n에 대한 ACK, NAK로 나뉘어진다. 순서번호를 가지고 있는 이유는 각 패킷에 대한 순서를 정함으로써 이 패킷이 새로운 데이터를 전송하는지, NAK를 받아 재전송하는 것인지를 구분할 수 있는번호를 추가하는 것이다. RDT 2.0과 달리, 만약 NAK 1이라는 데이터를 송신자가 받는다면, 송신자는 1번 패킷이 문제가 있다는 것을 알아챌 것이다.

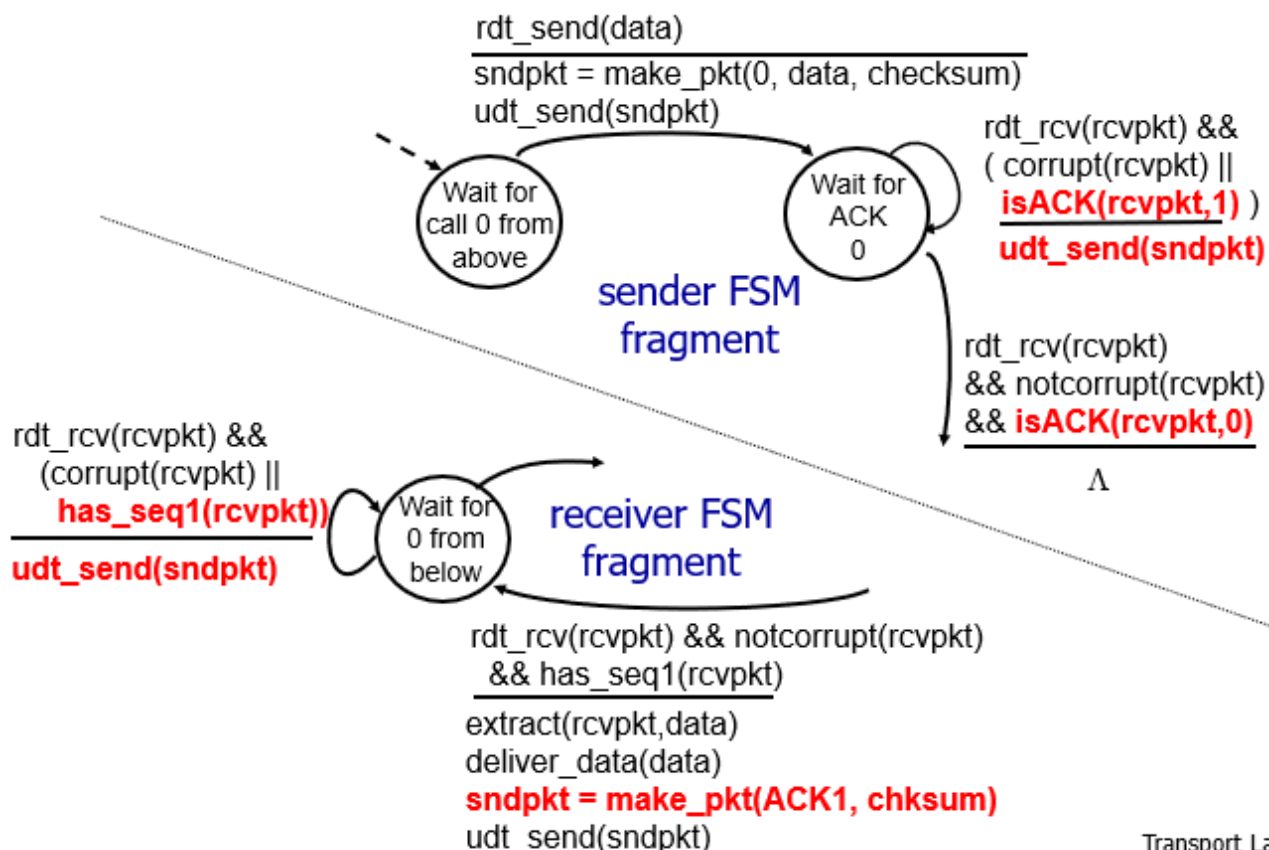
rdt2.1: sender, handles garbled ACK/NAKs



RD 2.2 : NAK - Free protocol

RD 2.2에서는 부정 수신응답인 NAK를 삭제해버린 대신, 몇 번 패킷까지의 정보까지가 ACK 인지를 송신해줌으로써, NAK를 송신하는 것과 같은 효과를 가진다.

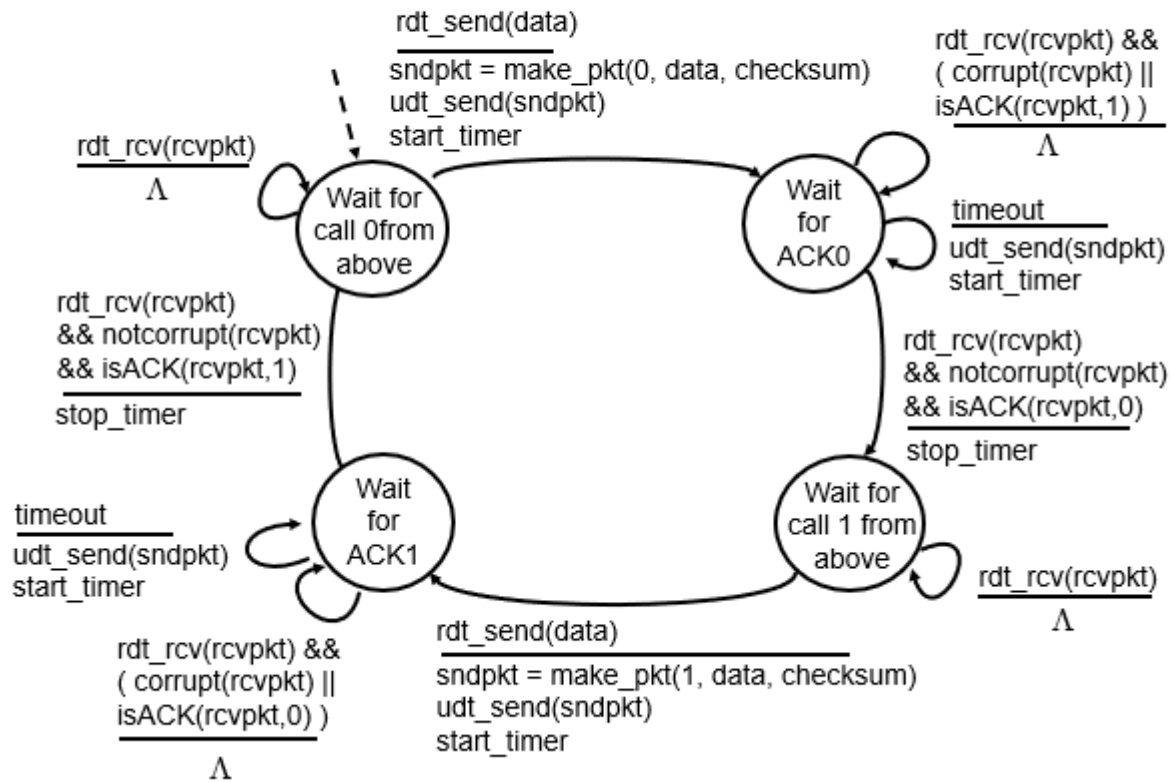
rdt2.2: sender, receiver fragments



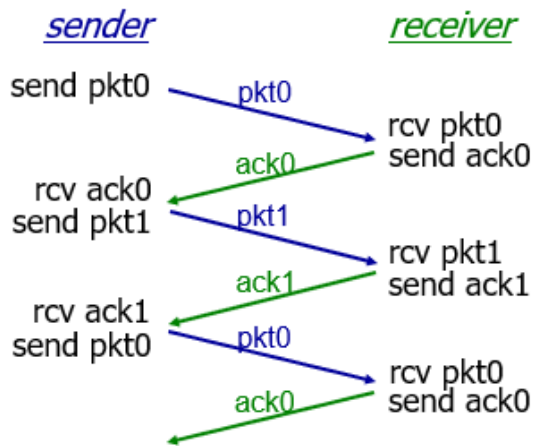
RD T 3.0

최근 만들어진 복잡한 인터넷 세계에서 프로토콜의 패킷 손실을 빠르게 회복하는 것은 매우 중요한 이슈이다. 송신자의 관점에서 재전송은 매우 간단하게 문제를 회복할 수 있는 기능이지만, 이는 시간적으로 너무 많은 낭비를 불러일으킬 수 있다. RD T 3.0 은 Timeout 기능을 추가해 발신자가 주는 ACK 를 '합리적인 시간' 동안 기다리고, ACK 가 딜레이된다면 타이머를 멈춘다(timeout).

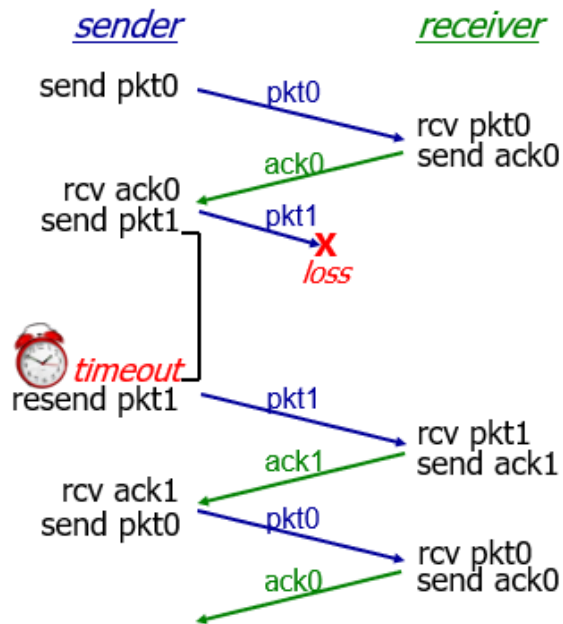
rdt3.0 sender



rdt3.0 in action



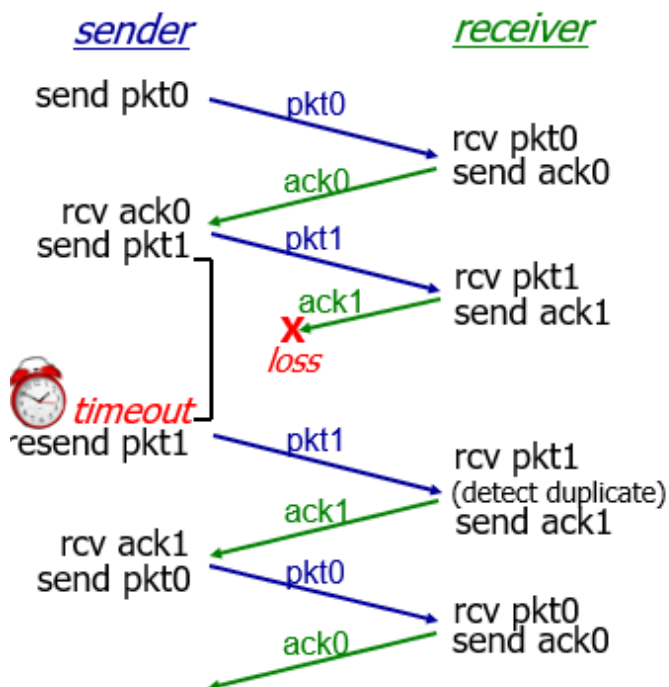
(a) no loss



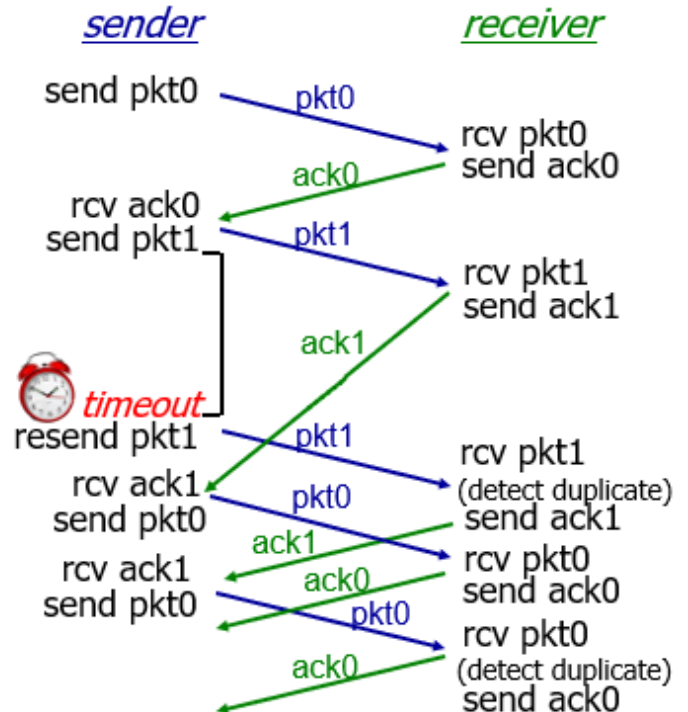
(b) packet loss

Transport Layer 3-40

rdt3.0 in action



(c) ACK loss



(d) premature timeout/ delayed ACK

Transport Layer 3-41

RDT 3.0의 출현으로 네트워크에서는 괄목할만한 성능 향상이 일어났지만, 현대적인 고속 네트워크의 요구를 모두 만족시키지는 못하였다. 이는 RDT가 기본적으로 전송-후-대기(stop-and-wait) 프로토콜이기 때문인데, 이는 다음 장에 배울 파이프라인된 RDT에서 해결할 수 있을 것이다.

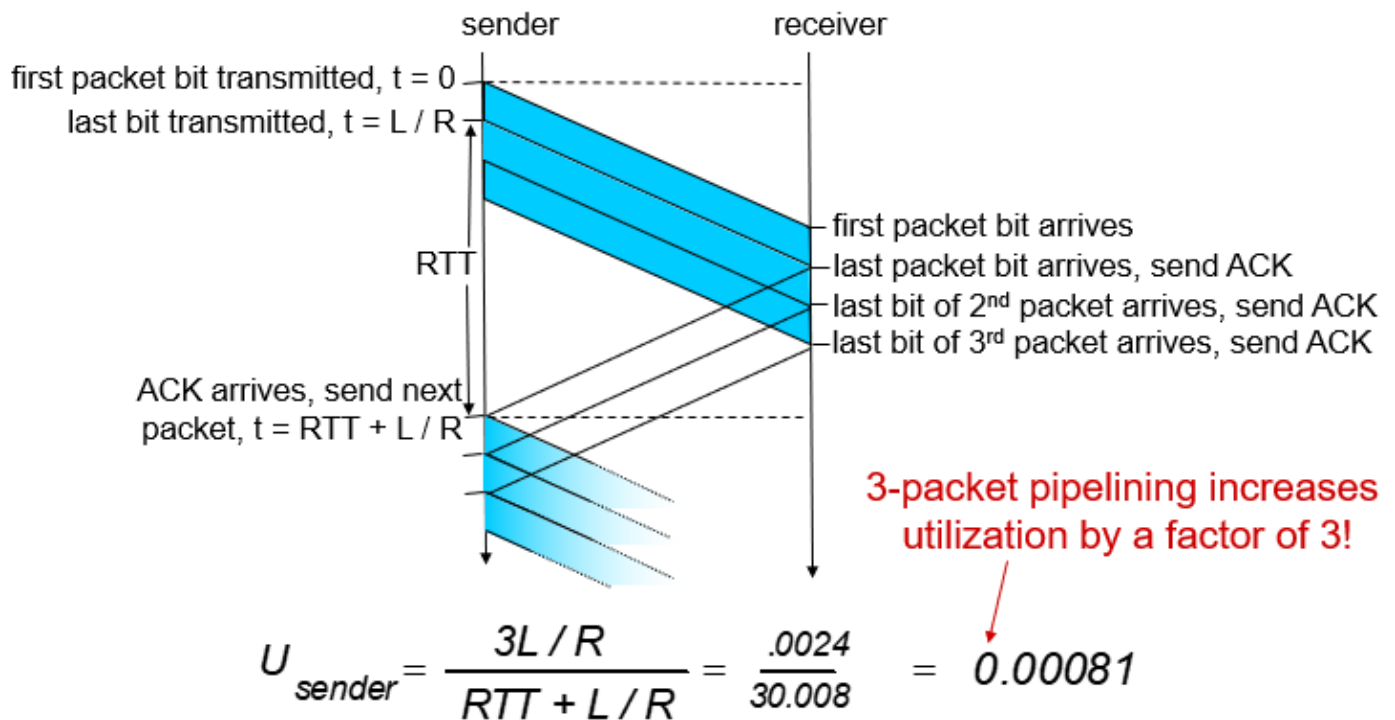
3-4. 파이프라인된 신뢰적 전송 프로토콜(Pipelined RDT) - GBN(Go-Back-N), SR(Selective-Repeat)

Pipelined Protocols

프로토콜 rdt 3.0의 개발로 네트워크에서는 괄목할 만한 기술 성능 향상이 일어났지만, 요즘 네트워크는 그보다 더 빠른 속도를 원한다. rdt 3.0의 핵심적인 성능 문제는 이 또한 여전히 전송-후-대기(stop-and-wait)방식 프로토콜이라는 것이다. 미국에 있는 두 종단 호스트를 예로 들어보자.

- 두 종단 시스템 사이의 광속 왕복 전파지연(RTT)은 30ms이다.
- 두 종단 시스템을 연결하는 채널은 10^9 비트(1Gbps)의 전송률을 가지고 있다.

헤더 필드와 데이터를 패킷당 1000 바이트(8000bit)의 패킷 크기를 가지고 1Gbps 링크로 패킷을 실제로 전송하는데 필요한 시간은 $1000\text{byte}/10^9\text{bits} = 8\text{micro sec}$ 이다. 이를 전송-후-대기 프로토콜을 가지고 데이터를 전송하기 시작한다면, 아무리 작은 데이터를 전송한다고 해도(RTT가 30ms) ACK를 기다리는데 엄청난 시간(30ms)이 들게 된다.



이러한 성능 문제에 대한 해결책이 바로 파이프라이닝이다. 파이프라이닝 방식은 신뢰적인 데이터 전송 프로토콜에서 다음과 같은 중요성을 가지고 있다.

- 순서번호의 범위가 커져야 한다. 각각 전송중인 패킷은 유일한 순서번호를 가져야 하고, 거기에 확인응답이 되지 않은 여러 패킷이 있을 수도 있기 때문이다.
- 프로토콜의 송신/수신 측은 한 패킷 이상을 버퍼링해야 한다. 최소한 송신자에게는 전송되었으나 확인응답 되지 않은 패킷은 버퍼링해야 한다.
- 파이프라인 오류 회복의 기본적인 접근방법으로 N부터 반복(Go-Back-N, GBN)과 선택적 반복(Selective Repeat, SR)이 있다.

GBN 프로토콜 - Go-Back-N, N부터 반복이라는 뜻의 프로토콜이다. GBN은 두 가지의 조건을 갖추어야 제대로 기능할 수 있다.

- 송신자가 확인응답을 기다리지 않고 여러 패킷을 전송 가능할 때만 가능하다.
- 파이프라인에서 확인응답이 안된 패킷이 최대 허용 수 N보다 크지 않아야 한다.

GBN 프로토콜에서 각 패킷은 1.Already ack'ed(이미 응답됨), 2.usable, not yet sent(사용 가능하나 전송 안됨), 3.sent, not yet ack'ed(전송 후 응답안됨), 4.not usable(사용 불가) 4가지로 분류된다.

존재하지 않는 이미지입니다.

여기서 usable, not yet sent와 sent, not yet ack'ed 두 가지를 합쳐서 window size.N이라고 부르며, GBN 프로토콜은 Sliding-window protocol로도 불린다.

GBN은 다음과 같은 세 가지 이벤트에 대해 반응해야 한다.

존재하지 않는 이미지입니다.

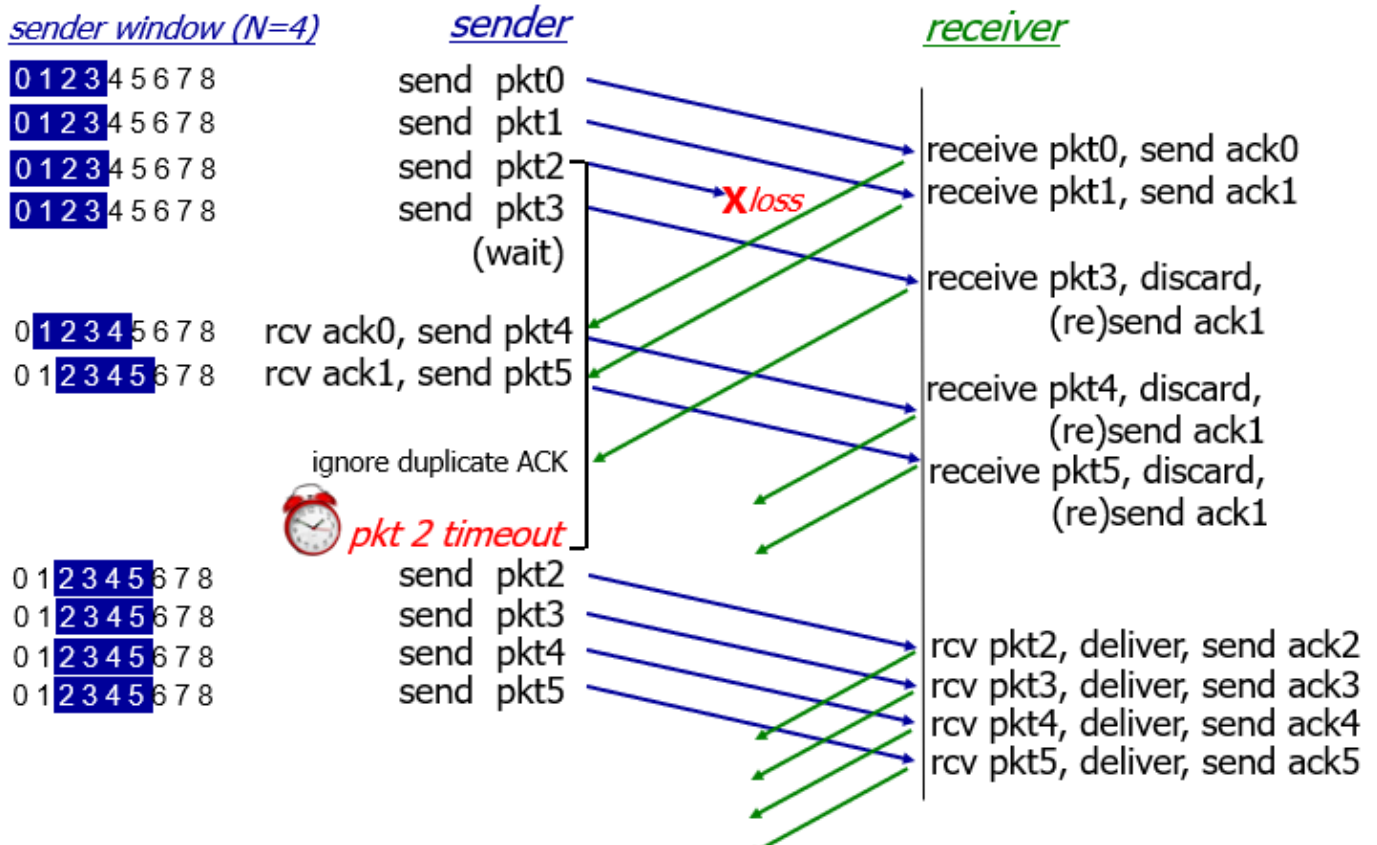
상위 호출 : rdt_send 가 들어오면, 송신자는 윈도우가 가득 찼는지(N 개의 확인응답 되지 않는 패킷이 있는지 없는지) 확인한다. 만약 윈도우가 가득 차 있다면 상위 데이터로 윈도우가 가득 찼다는 정보를 반환하고, 상위 계층이 나중에 다시 하도록 처리한다.

ACK 수신 : GBN 에서 순서번호 n 을 가진 패킷에 대한 확인응답은 누적 확인응답(cumulative acknowledgement)로 인식된다. 이는 확인응답의 n 까지 순서번호를 가진 배열이다.

타임아웃 이벤트 : 전송-후-대기 프로토콜과 같이, 타임아웃이 발생한다면 송신자는 sent, not yet ack'ed 의 모든 패킷을 다시 송신한다. 만약 아직 확인응답 안된 패킷이 없다면, 타이머는 멈춘다.

수신자는 n 에 대한 ACK 를 송신하고 상위 계층에 패킷의 데이터 부분을 전송한다. ACK 가 아닌 경우(NCK 는 없지만) 패킷을 버리고 가장 최근 재대로 수신된 패킷에 대한 ACK 를 재전송한다. 패킷의 순서가 잘못되었을 시에도 잘못된 패킷은 모두 버린다. 수신자 버퍼링의 접근이 간단하다는 장점이 있다.

GBN in action



위 그림은 윈도우 크기가 4 인 경우에 대한 GBN 프로토콜의 예이다. 패킷 2 가 잘못 송신되자 수신자(receiver)는 3,4,5 에 대한 패킷을 모두 버리고 2 부터 다시 받기 시작한다. 송신자는 ack2 를 받지 못하자 pkt 2 부터 다시 보낸다.

선택적 반복(Selective Repeat, SR) - GBN 프로토콜 또한 문제를 가지고 있다. 윈도우 크기가 커지고, 밴드폭 지연(bandwidth-delay)공의 결과가 클 수록 많은 패킷들이 파이프라인 안에 있을 수 있다. 결론적으로 한 패킷의 에러만으로 주변의 모든 패킷들을 재전송해야 하기 때문에 느려지는 것이다. 이를 대체하는 것이 Selective Repeat 이다. 이름에서 알 수 있듯이, SR 프로토콜은 수신자에서 발생한 패킷만 개별적으로 다시 보내도록 한다.

송신자 - 상위에서 데이터가 수신될 때 송신자는 패킷의 다음 순서번호를 검사한다. 순서번호가 윈도우 내에 있으면 데이터는 패킷으로 송신되고, 없으면 GBN 처럼 버퍼에 저장하거나 나중에 전송하기 위해 상위 계층으로 돌려놓는다

타이머는 손실된 패킷을 보호하기 위해 사용된다. 각 패킷은 자신만의 논리 타이머를 가지고 있다. ACK 가 수신될 때 SR 송신자는 그 ACK 가 윈도우에 있다면 그 패킷을 수신된 것으로 표기한다. 만약 패킷 순서번호가 send_base 와 같다면 윈도우 베이스는 가장 작은 순서번호를 가진 아직 확인응답되지 않은 패킷으로 옮겨진다.

수신자 - 수신자는 패킷의 순서와는 무관하게 순서없이 수신된 패킷에 대한 확인응답을 할 것이다. 순서가 틀린 패킷은 빠진 패킷이 들어올 때까지 버퍼에 저장하고, 빠진 패킷이 들어오면 순서를 맞춰서 순서대로 상위 계층에 전달할 수 있다.

sender window (N=4)

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 []


0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

sender

send pkt0
 send pkt1
 send pkt2
 send pkt3
 (wait)

rcv ack0, send pkt4
 rcv ack1, send pkt5

record ack3 arrived
 **pkt 2 timeout**

send pkt2
 record ack4 arrived
 record ack5 arrived

receiver

receive pkt0, send ack0
 receive pkt1, send ack1

receive pkt3, buffer,
 send ack3

receive pkt4, buffer,
 send ack4

receive pkt5, buffer,
 send ack5

rcv pkt2; deliver pkt2,
 pkt3, pkt4, pkt5; send ack2

Q: what happens when ack2 arrives?

위 그림에서, 수신자는 패킷 3,4,5를 버퍼에 저장하고 마지막으로 패킷 2가 수신되었을 때 모든 패킷을 상위 계층에 전달한다.

Selective Repeat - Dilemma

SR 프로토콜에서 송신자와 수신자의 윈도우는 항상 같지 않다. 다시 말해서 수신자가 보내는 ACK를 송신자가 잘못 해석할 여지가 있다는 것이다. 송신자와 수신자 윈도우 사이의 동기화 부족은 순서번호가 작을 때 문제가 생길 수 있다.

Selective repeat: dilemma

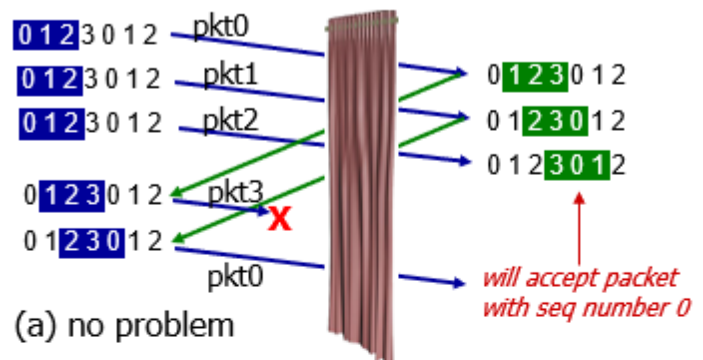
example:

- seq #'s: 0, 1, 2, 3
- window size=3
- receiver sees no difference in two scenarios!
- duplicate data accepted as new in (b)

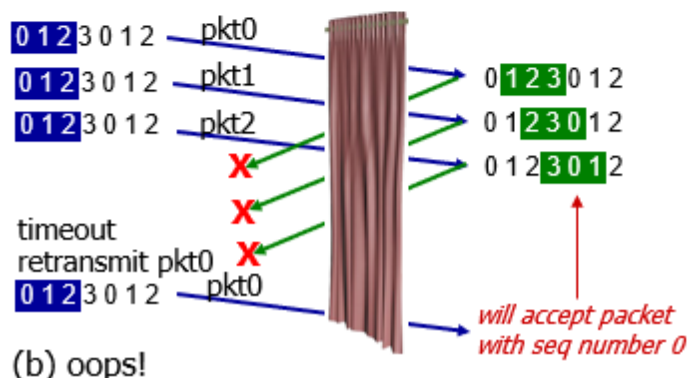
Q: what relationship between seq # size and window size to avoid problem in (b)?

sender window (after receipt)

receiver window (after receipt)



receiver can't see sender side.
 receiver behavior identical in both cases!
something's (very) wrong!



위 그림을 보자. 송신자는 0 1 2 3 0 1 2 라는 패킷을 보내고 있다. 수신자는 pkt 3 을 잃어버리고, ACK 3 을 주지 못한 채 0 을 가진 패킷이 다시 도착한다. -> 여기서 수신자는 다섯 번째 패킷 0 의 원래 전송과 첫번째 패킷의 재전송을 구분할 수 없다.

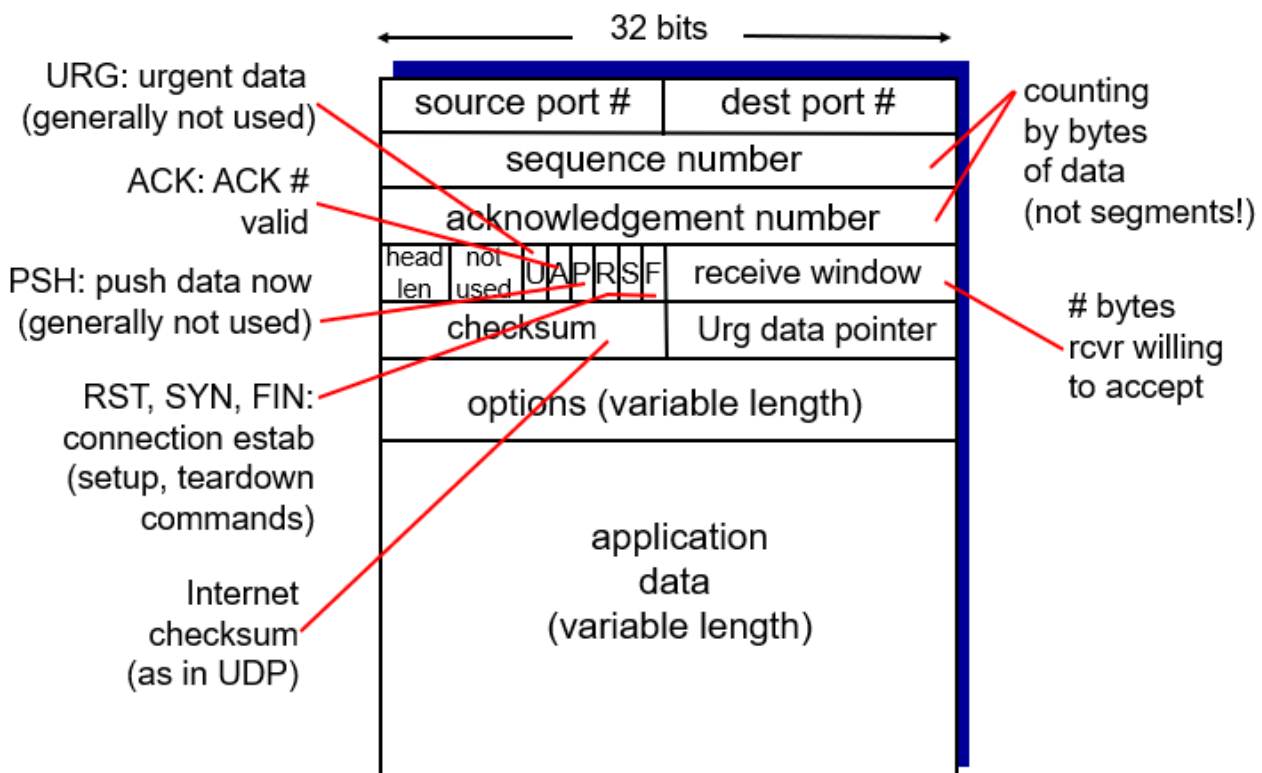
이를 해결하기 위해서는 윈도우 크기가 '충분히' 작아야 한다. 윈도우의 크기는 SR 프로토콜에 대한 순서번호 공간 크기의 절반보다 작거나 같아야 한다.

3-5. 연결지향형 트랜스포트(Connected-Oriented Transport) - TCP

지금까지 TCP 에서 배운 특징들을 다시 복기해보자.

1. TCP 는 두 프로세스가 연결하기 전에 먼저 handshake 를 해야 하므로 연결지향형(connection-oriented)이다. 즉, 데이터 전송을 보장하는 파라미터들을 각자 설정하기 위한 사전 segment 들을 보내야 한다.
2. TCP 프로토콜은 오직 종단 시스템에서만 동작하고 **중간의 네트워크 요소(router/bridge)에서는 작동하지 않으므로, 중간의 네트워크 요소들은 TCP 연결 상태를 유지하지 않는다.**
3. TCP 연결은 **전이중(full-duplex)** 서비스를 제공한다. A 에서 B 로 갈 수 있는 TCP 연결이 있다면, 애플리케이션 계층 데이터는 B 에서 A 로도 자유롭게 이동할 수 있다.
4. TCP 연결은 항상 **단일 송신자와 단일 수신자를 가지는 점대점(point-to-point)연결 방식**이다. 단일 송신자와 여러 수신자를 가지는 멀티캐스팅(multicasting)은 TCP 에서는 불가능하다.
5. 클라이언트 프로세스는 서버의 프로세스에 연결을 원한다고 요청한다. 서버는 두 번째 특별한 TCP Segment 로 응답한다. 마지막으로 클라이언트가 payload 를 포함한 세 번째 특별한 Segment 로 다시 응답한다. 처음 2 개의 Segment 에는 application 계층의 번호(payload)가 없지만, 세번째 Segment 는 Application 계층의 번호인 Payload 를 포함할 수 있다. 두 호스트 사이에서 세 개의 Segment 가 보내지기 때문에, **세 방향 핸드셰이크(Three-way handshake)**라고 불린다.
6. TCP 연결이 설정되면 두 애플리케이션 프로세스는 서로 데이터를 보낼 수 있다. 클라이언트 프로세스는 Socket 을 통해 데이터의 스트림을 전달한다. 데이터가 관문을 통해 전달되면, 데이터는 클라이언트에서 동작하는 TCP 에 맡겨진다.
7. Segment 의 크기는 정확하게 정해지지 않고 TCP 자율로 데이터를 전송할 수 있지만, 최대 세그먼트 크기(Maximum Segment Size,MSS)에는 제한을 받는다.
- MSS 는 로컬 송신 호스트에 의해 전송될 수 있는 가장 큰 링크 계층 프레임의 길이(최대 전송 단위)에 의해 첫번째로 결정되며, 그 후 TCP Segment 와 TCP/IP 헤더 길이(통상 40 바이트)가 단일 링크 계층 프레임에 딱 맞도록 하여 정해진다.
8. TCP 는 TCP 헤더와 클라이언트 데이터를 하나로 합쳐 TCP Segment 를 형성한다. Segment 는 네트워크 계층의 IP 데이터그램 안에 각각 캡슐화된다. TCP 가 상대방에게서 Segment 를 수신했을 때, Segment 의 데이터는 TCP 연결의 수신 버퍼에 위치한다. Application Layer 에서는 이 버퍼에서 데이터의 스트림을 읽는다.

- TCP Segment 구조



TCP Segment 는 크게 헤더와 데이터 필드로 나뉘어진다.

- 데이터 필드는 애플리케이션 데이터를 담는다.
- MSS 는 Segment 의 데이터 크기를 제한한다.
- TCP 는 MSS 의 크기대로 파일을 쪼개 Segment 를 제작하지만, 많은 어플리케이션에서 MSS 의 크기보다 작게 Segment 를 제작하는 경우도 있다.
- 헤더는 상위 계층 Application 으로부터 다중화와 역다중화를 하는데 사용하는 출발지 포트(source port)와 목적지 포트(dest port)를 가진다.
- UDP 와 같이, Checksum 필드 또한 포함한다.
- 32 비트 순서번호(Sequence number)와 확인응답번호(Acknowledgement number) 필드는 신뢰적 데이터 전달을 위해 포함된다.
- 16 비트 수신 윈도우(Receive window) 필드는 흐름제어에 사용된다. 이는 수신자가 받아들이는 바이트의 크기를 나타낼 때 사용된다.
- 4 비트 헤더 길이(Header length) 필드는 32 비트 워드 단위로 TCP 헤더의 길이를 나타낸다. TCP 는 일반적으로는 20 바이트인 가변적 길이로 되어있다.
- 가변적인 길이의 옵션(option)필드는 송신자와 수신자의 MSS(최대 세그먼트 길이)를 협상하거나 고속 네트워크에서 사용하기 위한 윈도우 확장 요소, 타임스태프 옵션 등을 정의한다.
- 플래그(flag) 필드는 아래의 작은 6 개의 비트를 포함한다.
 1. URG(U)비트는 세그먼트 송신 상위 개체가 긴급으로 표시하는 데이터인지 표시한다.(사실 거의 사용되지 않는다)
 2. ACK(A) 비트는 확인응답 필드값이 성공적으로 전달되었는지 확인한다.
 3. PSH(P)비트는 수신자가 데이터를 상위 계층에 즉각적으로 전달되는 데이터인지 표시한다(이것도 사실 거의 안쓴다)
 4. RST, SYN, FIN 비트는 연결 설정과 해제에 사용된다.

위 구성요소 중 순서번호(Seq number)와 확인응답 번호(Ack number)는 TCP 에서 신뢰적 데이터 전송을 위해 가장 중요한 필드이다. 세그먼트에 대한 순서번호(Seq Num)은 보통 Segment 의 첫 번째 자리에 위치해있는 경우가 많다. 확인응답 번호는 조금 복잡한데, A 에서의 확인응답 번호는 호스트 A 가 호스트 B 로부터 받는 다음 바이트의 순서번호이다. 간단하게 말해서 호스트 A 가 B 로부터 535 까지의 번호가 붙은 바이트를 수신하면, 호스트 A 는 536 번째 바이트를 확인응답 번호로 놓고 그 다음 바이트를 기다린다.

- 왕복시간(RTT) 예측과 타임아웃

RTT 는 손실 세그먼트에 대한 재전송 시간(round trip time)을 의미한다. 즉 Segment 가 전송된 시간부터 긍정 확인응답 될 때까지의 시간을 의미한다. Timeout 이 당연히 RTT 보다 '적당히' 커야 불필요한 재전송이 발생하지 않을 것이다. 그렇다면 얼마나 '적당히' 커야 할까? 여기서 가장 큰 문제는 모든 RTT 의 길이는 다르다는 것이다.(라우터에서의 혼잡과 종단 시스템에서의 부하 변화 때문에) 사실 모든 Segment 에 대해 동일한 Timeout 을 적용하는 것은 굉장히 귀찮기 때문에 평균값을 내서 Sample-RTT 를 만든다. 대체로는 RTT 의 평균값을 채택한다. 보통 1/8 의 가중치를 가지고 RTT 를 계속해서 갱신해간다. 이는 가중평균이라고 부르는데, 최근의 샘플들이 네트워크상의 현재 혼잡을 더 잘 반영하기 때문이다.

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$

통계에서 이런 평균은 지수적 가중 이동 평균(exponential wrighted moving average)라고도 불린다. 이 값을 기준으로 아래와 같이 Timeout 값을 정한다.

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$



↑
estimated RTT

↑
"safety margin"

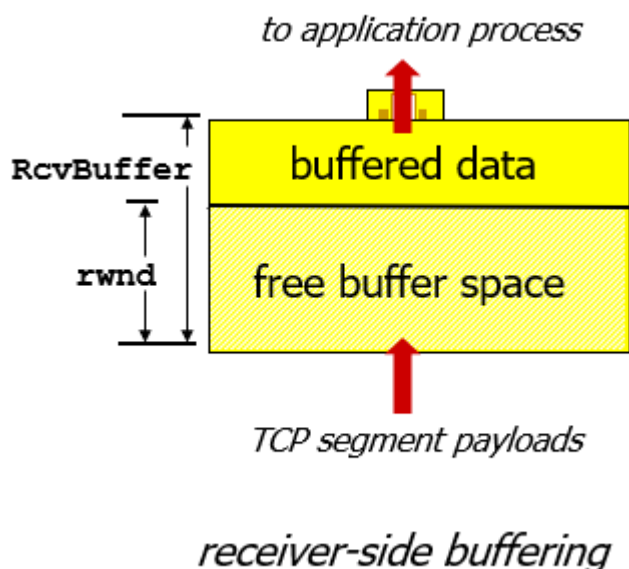
- TCP 에서의 신뢰적 데이터 전달(RDT)

인터넷 프로토콜은 데이터그램 전달을 보장하지 않고, 데이터그램이 순서대로 전달된다는 것을 보장해주지 않는다. 또한 데이터그램의 무결성도 보장해주지 않는다. IP 서비스에서 데이터그램은 라우터의 버퍼를 오버플로 상태로 만들수도 있고, 목적지에 도달하지 않을수도 있고, 데이터그램의 순서를 바꾸거나, 데이터그램의 비트를 손상시킬수도 있다. 이러한 문제들은 트랜스포트 계층의 Segment 에 많은 영향을 준다. TCP 는 이러한 문제를 해결하는 RDT 를 제공한다.

1. 손실 Segment를 복구 - TCP는 보통 하나의 통합된 타이머를 사용한다. 이 타이머를 기준으로 ACK를 받지 못하고 타임아웃이 일어나면 재전송을 요구해 Segment를 복구한다.
2. ACK 값 수신 - ACK가 수신되면, TCP는 변수 $SendBase$ 와 ACK 값 y 를 비교한다. $SendBase$ 는 수신 확인응답이 확인되지 않은 가장 오래된 바이트의 순서번호이다($SendBase-1$ 은 정확하게 수신되었음을 증명하는 변수). 만약 $y > SendBase$ 면 ACK는 이전에 확인응답 안된 하나 이상의 Segment들을 확인해 준다.
3. Timeout을 복구(지수적 증가) - 위에서도 말했듯이 Timeout 값은 RTT보다 과도하게 커서도, 작아서도 안되고 적당한 값을 유지해야 한다.(커지면 과도한 전송 지연 / 작아지면 필요도 없는 재전송이 일어남) 보통 Timeout은 위치된 RTT에 $Margin(DevRTT)$ 값을 더한 값으로 정하고, 초기 Timeout 값은 1초를 권고한다. 또한 Timeout이 일어났을 때, timeout 값을 두 배로 하여 후속 timeout을 방지한다. 거기서도 Timeout이 일어나면 또 두배를 곱한다.(지수적 증가) 그러나 세그먼트가 수신되고 RTT가 수정되면 Timeout 또한 원래 값으로 돌아간다.
4. 빠른 재전송 - Timeout이 유발하는 재전송의 한 가지 문제는 타임아웃 주기가 때때로 매우 길다는 것이다. Segment를 잃었을 때, Timeout 주기가 너무 길면 잃어버린 패킷을 기다리는 시간이 길어 중단간 지연을 중단시킨다. 다행히도 송신자는 중복 ACK를 통해 빠른 재전송을 할 수 있다. 중복 ACK는 송신자가 이미 이전에 받은 확인응답에 대한 재확인 응답 Segment ACK이다.

- TCP에서의 흐름제어(Flow-control)

TCP 연결의 종단에서 호스트들은 연결에 대한 개별 수신 버퍼를 설정한다. TCP 연결이 순서대로 올바르게 바이트를 수신할 때 TCP는 데이터를 수신 버퍼에 저장한다. 하지만 애플리케이션이 데이터를 읽는 속도가 느리다면 송신자가 보내는 데이터가 수신 버퍼에 오버플로를 발생시킨다. 이를 방지하기 위해 TCP는 흐름제어 서비스(flow-control service)를 제공한다. 흐름제어 서비스는 송신자가 수신자의 버퍼를 overflow 시키는 것을 방지하기 위해 수신자와 송신자의 속도를 일치시키는 작업이다. 여기서 송신자를 제어하는 형태는 혼잡제어(Congestion control)로 알려져 있다. 흐름제어와 혼잡제어가 비슷한 동작으로 움직이지만 사실은 서로 다른 목적을 위해 수행된다. TCP에서는 송신자가 수신 윈도우(receive window)라는 변수를 유지해 흐름제어를 유지한다.(TCP는 전이중 방식(full-duplex)이므로 각 송신자는 별개의 수신 윈도우를 유지한다.) 수신 윈도우 중 하나인 $rwnd$ 는 버퍼의 여유 공간으로 설정된다. 시간에 따라 여유 공간은 변화하므로 $rwnd$ 는 동적이다. 호스트 A가 호스트 B에게 큰 파일을 전송한다고 할 때, 호스트 B는 호스트 A에게 전송하는 모든 Segment의 윈도우 필드에 $rwnd$ 값을 전달함으로써 얼마나 많은 값이 차있는지를 호스트 A에게 알려준다. 호스트 A는 확인응답 되지 않은 값을 $rwnd$ 값보다 작게 유지시킨다.



3-6. 혼잡제어(Congestion Control)와 TCP 혼잡제어

네트워크가 혼잡해질수록 라우터 버퍼들의 오버플로우는 필연적으로 발생하고, 그를 예방하기 위한 혼잡제어(Congestion Control)는 필연적으로 필요하였다. 네트워크 혼잡을 처리하기 위해서는 당연히 네트워크 혼잡의 원인을 알아야 한다.

- 혼잡의 원인과 비용 - 세 가지 시나리오

1. 두 개의 송신자와 무한 버퍼를 갖는 하나의 라우터

가장 간단한 혼잡제어 시나리오이다. 호스트 A로부터 호스트 B로 전송되는 패킷은 라우터를 통해서 전달되고, 총 용량이 R로 정의된 고용 출력 링크상으로 전달된다.



■ maximum per-connection throughput: $R/2$

❖ large delays as arrival rate, λ_{in} , approaches capacity

송신자 측에서 수신자로 전달되는 패킷은 유한한 지연으로 송신자에게 수신된다. 단 $R/2$ 까지만. 그 이상의, $R/2$ 를 넘는 패킷을 수신자에게 전달할 수 없다. 이는 호스트 A와 B가 전송률을 아무리 높여더라도 동일하다. 오른쪽 그래프를 보면 $R/2$ 에 가까워질수록 딜레이가 기하급수적으로 늘어나는 것을 볼 수 있다. 다시 말해 라우터가 무한하다고 해도 혼잡 네트워크의 비용이 발생할 수 있다는 것이다.

2. 두 개의 송신자와 유한 버퍼를 갖는 하나의 라우터

라우터 버퍼의 야이 유한하다면 라우터가 가득 찼을때 들어오는 패킷이 버려지게 되고 재전송을 요청할 것이다.

3. 네 개의 송신자와 유한 버퍼를 가지는 하나의 라우터, 그리고 멀티홉 경로

결과적으로 부하량을 증가하면 처리량이 감소하게 된다.

- 혼잡제어에 대한 접근법

네트워크 계층이 어떻게 혼잡제어를 위해 트랜스포트 계층에 도움을 제공하는지에 따라 혼잡제어 접근을 구분할 수 있다.

1. 종단간의 혼잡제어 : 네트워크 계층은 혼잡제어 목적을 위해 트랜스포트 계층에게 어떤 직접적인 지원도 제공하지 않는다. 종단 시스템은 네트워크 활동에 기초해 혼잡 정도를 추측해야 한다(예_ 패킷 손실 및 지연 등을 통해)

2. 네트워크 지원 혼잡제어 : 네트워크 계층 구성요소는 네트워크 안에서 혼잡 상태와 관련해 송신자에게 직접적인 피드백을 제공한다. 예를 들면 라우터가 각각의 출발자에게 처리하는 패킷의 패킷 헤더 안 어떻게 출발지의 전송률을 증가시키거나 감소시킬 것인지에 대한 정보를 넣어 보낸다.

- TCP에서의 혼잡제어

TCP의 중요한 특성 중 하나가 바로 혼잡제어 메커니즘이다. TCP에서는 네트워크 지원 혼잡제어보다는 종단간의 혼잡제어를 주로 사용한다. 그렇기 때문에 TCP는 네트워크 혼잡을 알아채서 송신률을 줄이거나 늘려야 할 것이다.

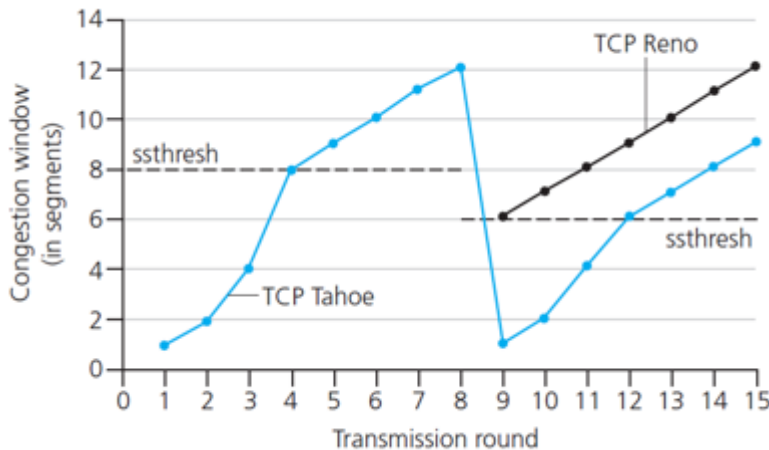
1. TCP가 연결로 트래픽 전송률을 제한하는 방법 : TCP 연결의 양 끝 각 호스트들은 수신 버퍼, 송신 버퍼, 그리고 몇 가지의 변수로 구성되었다. 그중에서도 $rwnd$ 로 표현되는 수신 윈도우(receive window)와 $cwnd$ 로 표시되는 혼잡 윈도우(congestion window)는 TCP 송신자가 네트워크로 트래픽을 전송할 수 있는 비율을 제한한다. 특히 송신자의 확인응답안된 데이터의 양은 $cwnd$ 와 $rwnd$ 의 최솟값을 초과하면 안된다. 여기서는 혼잡제어에 대해 다루기 때문에 $cwnd$ 에만 집중하도록 하자. $rwnd$ 를 무시하기 때문에, TCP는 매 왕복시간(RTT)마다 최대 $cwnd$ 만큼의 데이터를 보낼 수 있을 것이다. 그러므로 송신자의 송신율(rate)은 대략 $cwnd/RTT$ byte/s이다. $cwnd$ 의 값을 조정하여 송신자는 링크에 데이터를 전송하는 비율을 조절할 수 있다.

2. TCP가 자신과 목적지 사이 경로에 혼잡이 발생하는지 감지하는 방법 : 과도한 혼잡이 발생하면 자연스럽게 경로에 있는 하나 이상의 라우터 버퍼들이 오버플로되고, 그 결과 데이터그램이 버려지고 손실 이벤트가 발생하게 된다(ACK x). 이를 통해 혼잡이 일어났음을 알 수 있다. 정리하자면 TCP는 확인응답(ACK)을 통해 혼잡함을 감지하고 그 증가나 감소를 유발하는데, 이런 현상을 trigger 또는 clock이라 부르므로 TCP는 자체 클로킹(self-clocking)을 가지고 있다고 한다. 더 구체적으로 알아보자

- 손실된 Segment는 혼잡을 의미하며, 이에 따라 TCP의 $rwnd$ 는 한 세그먼트가 손실되어 ACK 값을 보내지 않을 때(손실 이벤트가 일어났을 때) 줄어야 한다.

- 이전에 확인응답되지 않은 Segment가 재전송되어 ACK가 전송되면, 송신자의 전송률은 다시 늘려야 한다. ACK의 도착은 묵시적으로 네트워크가 혼잡하지 않다는 표시로 받아들여진다.

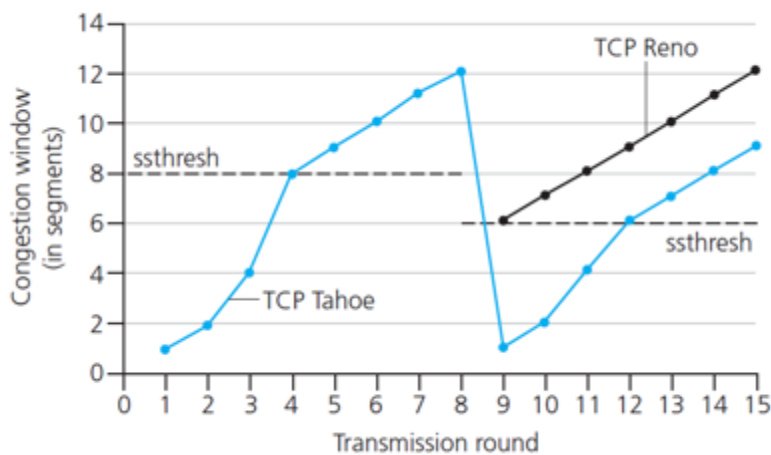
간단하게 말하면 짚끔 cwnd를 늘리다가 ACK 손실 이벤트가 발생하면 갑자기 훅 줄었다가 다시 짚끔짚끔 늘리는 것이다. 그래프로 보면 아래와 같다.



이 개념을 통해 만들어진 것이 바로 TCP 혼잡제어 알고리즘(TCP congestion control algorithm)이다.

TCP 혼잡제어 알고리즘은 다음의 중요한 세 구성요소를 갖는다

느린 시작(Slow Start), 혼잡 회피(Congestion avoidance), 빠른 회복(Fast recovery). 이름만 들어도 어떻게 진행되는지 직관적으로 알 듯 하다.



1. Slow Start

TCP 연결이 시작될 때, cwnd의 값은 일반적으로 1MSS로 초기화되고, 초기 전송률은 대략적으로 MSS/RTT 가 된다. TCP 송신자의 가용 폭은 일반적으로 $1MSS/RTT$ 보다 클 것이므로, TCP 송신자는 빠르게 가용 송신폭을 찾으려고 한다. Slow Start 상태에서는 확인 응답(ACK)을 받을 때마다 가용 폭을 1MSS부터 1MSS씩 증가시킨다. MSS가 늘어나면 그에 따라 확인 응답(ACK)도 늘어나는데, 그 늘어난 확인 응답이 MSS도 또다시 증가시키기 때문에, 결국에는 기하급수적으로 늘어나는 지수적 증가가 일어날 것이다.(slow start 인 이유)

이런 지수적 증가는 Timeout에 의한 손실 이벤트(혼잡)가 있을 경우에 끝나게 되는데, 가장 보수적으로 cwnd값을 다시 1로 줄이게 된다.

두 번째로 저장된 sssthresh(slow start의 임계치)값에 따라서도 $cwnd/2$ (혼잡이 검출되었을 때의 혼잡 윈도우값의 반)으로 정한다. cwnd가 sssthresh와 같거나 지나치면 cwnd를 $cwnd/2$ 로 재설정 후, slow start를 종료하고 TCP는 Congestion avoidance 모드로 변환된다.

마지막으로, 만약 3개의 중복 ACK들이 검출되면 slow start를 종료하고 Fast Recovery 단계로 들어간다.

2. Congestion avoidance

계속해서 늘어난 cwnd는 sssthresh값에 가까이 가거나 넘어서면 거의 혼잡을 목전에 두게 된다. 그러므로 여기서부터 계속해서 지수적으로 증가시키는 것은 현명하지 못한 일이다. 여기서 TCP는 좀 더 보수적인 접근법을 채택하여 매 RTT마다 cwnd값을 두 배로 하기보다 RTT마다 cwnd값을 1씩 올려나간다.

이런 방식은 증가 속도를 느리게 하는 효과는 있겠지만, 결국에는 Timeout 임계점에 다다르고 말 것이다. timeout이 발생하면 slow start의 경우와 같이 cwnd의 값을 1로 하고, sssthresh값은 손실 event값이 발생했을 때의 절반으로 저장한다.

3. Fast recovery

빠른 회복 상태에서 cwnd 값은 잃었던 segment에 대한 매 중복된 ACK를 수신할 때마다 1MSS 씩 증가한다. 빠른 회복은 일제치에 빨리 다가가기 위한 권고 사항이지만 필수 사항은 아니다. 딱히 안해도 상관이 없다.

4-1. 네트워크 계층 : 데이터 평면(Network layer: Data Plane)

네트워크 계층의 호스트를 이용한 통신은 프로세스간 통신을 제공한다.

네트워크 계층은 프로토콜 계층 중 가장 복잡한 계층이고, 데이터 평면(Data Plane)과 제어 평면(Control Plane)으로 나뉜다.

4 장에서는 데이터 평면에 대하여 공부한다.

만약 호스트 H1에서 H2로 데이터를 보낼 때, 네트워크 계층은 트랜스포트 계층으로부터 Segment(TCP or UDP)를 받아 각 Segment를 Datagram(네트워크 계층의 Packet)으로 캡슐화하고, 인접한 라우터 R1에게 데이터그램을 보낸다. 수신 호스트 H2의 네트워크 계층은, 트랜스포트 계층 segment를 추출하여 H2의 트랜스포트 계층으로 전달한다.

- 데이터 평면의 역할은 입력 링크에서 출력 링크로 데이터그램을 전달하는 것이다.
- 제어 평면은 데이터그램이 송신 호스트에서 목적지 호스트까지 잘 전달되게끔 로컬(local), 퍼 라우터(per-router)포워딩을 조정하는 것이다.
- 라우터는 잘려진 프로토콜 스택을 가진다. 즉 트랜스포트 계층과 애플리케이션 계층이 존재하지 않으므로 네트워크 계층의 상위 계층이 존재하지 않는다.

네트워크 계층의 근본 역할은 송신 호스트에서 수신 호스트로 패킷을 전달하는 것 뿐이다. 이를 위한 두 가지 역할은 다음과 같다

- 포워딩(fowarding) : 데이터를 적절한 출력 링크로 이동시키는 기능이다. 데이터 평면에 구현된 가장 중요한 기능이다. 포워딩은 실제로 데이터 단위가 이동하는 것이고, 하드웨어 단위에서 실행된다.
- 라우팅(routing) : 데이터가 갈 패킷의 경로를 결정하는 것이다. 이러한 경로를 계산하는 알고리즘을 라우팅 알고리즘(rouring algorithm)이라고 한다. 라우팅은 소프트웨어 단계에서 실행된다.

- 네트워크 서비스 모델

송신 호스트의 트랜스포트 계층이 네트워크 계층에 패킷을 전달할 때, 트랜스포트 계층은 네트워크 계층이 목적지까지 패킷을 전달한다는 것을 믿을 수 있을까? 네트워크 서비스 모델(network service model)은 송수신 호스트 간 패킷 전송 특성을 정의한다.

- 보장된 전달 : 이 서비스는 패킷이 소스 호스트부터 목적 호스트까지 도착하는 것을 보장한다
- 지연 제한 이내의 보장된 전달 : 이 서비스는 특정 지연 제한(ex:100ms)안에 목적 호스트까지 도착하는 것을 보장한다.
- 순서화(in-order) 패킷 전달 : 이 서비스는 패킷이 목적지에 송신된 순서대로 도착하는 것을 보장한다
- 최소 대역폭 보장 : 이 서비스는 송신과 수신 호스트 사이 특정 비트 속도의 전송 속도를 에를레이트한다. 송신 호스트가 비트들을 특정한 비트 속도 이하로 이동하는 한, 모든 패킷이 목적지 호스트까지 전달된다.
- 보안 서비스 : 네트워크 계층은 모든 데이터그램을 소스 호스트에서는 암호화, 목적지 호스트에는 해독을 할 수 있게 하여 Transport 계층에서의 모든 segment들에 대해 기밀성을 제공하여야 한다.

인터넷 네트워크 계층은 최선형 서비스(best-effort-service)를 제공한다. 최선형 서비스 이름과 달리 패킷 순서 보장, 전송 보장조차 해주질 않는다. 어떻게 보면 굉장히 허술해보이지만, 더 좋은 개념들이 나왔는데도 최선형 서비스는 간단하다는 장점 하나만으로 많은 애플리케이션에서 사용되고 있다.

4-2. 네트워크 계층, 라우터 내부의 구성 요소 - input port, switching fabric, output port

- 라우터란?

라우터는 네트워킹 계층에서 사용되는 패킷 스위치를 의미한다. 라우터는 입력 포트, 스위칭 구조, 출력 포트, 라우팅 프로세서로 구성된다. 여기서 라우팅 프로세서는 제어 평면, 나머지 세개는 데이터 평면으로 구분된다.

- 라우터의 구성 요소(Router architecture overview)

존재하지 않는 이미지입니다.

- 입력 포트(input port)와 출력 포트(output port) 여러개의 상자들로 이루어져 있다. 그 중 인풋 포트의 가장 왼쪽과 출력 포트의 가장 오른쪽 상자는 라우터로 들어오는 입력 링크의 물리계층 기능을 수행한다. 또한 입력 포트는 입력 링크의 반대편에 있는 링크 계층과 상호 운용하기 위해 필요한 링크 계층 기능을 수행한다. 또한 입력 포트의 가장 오른쪽 상자에서는 검색 기능을 수행한다.
- 스위칭 구조(switching fabric)는 라우터의 입력 포트와 출력 포트를 연결한다
- 출력 포트(output port)는 스위칭 구조에서 수신한 패킷을 저장하고 필요한 링크 및 물리적 계층 기능을 수행해 출력 링크로 패킷을 전송한다. 링크가 양방향일 때, 출력 포트는 일반적으로 동일한 링크의 입력 포트와 한 쌍을 이룬다.

- 라우팅 프로세서(routing processor) : 라우팅 프로세서는 5장에서 배울 제어 평면 기능을 수행한다.

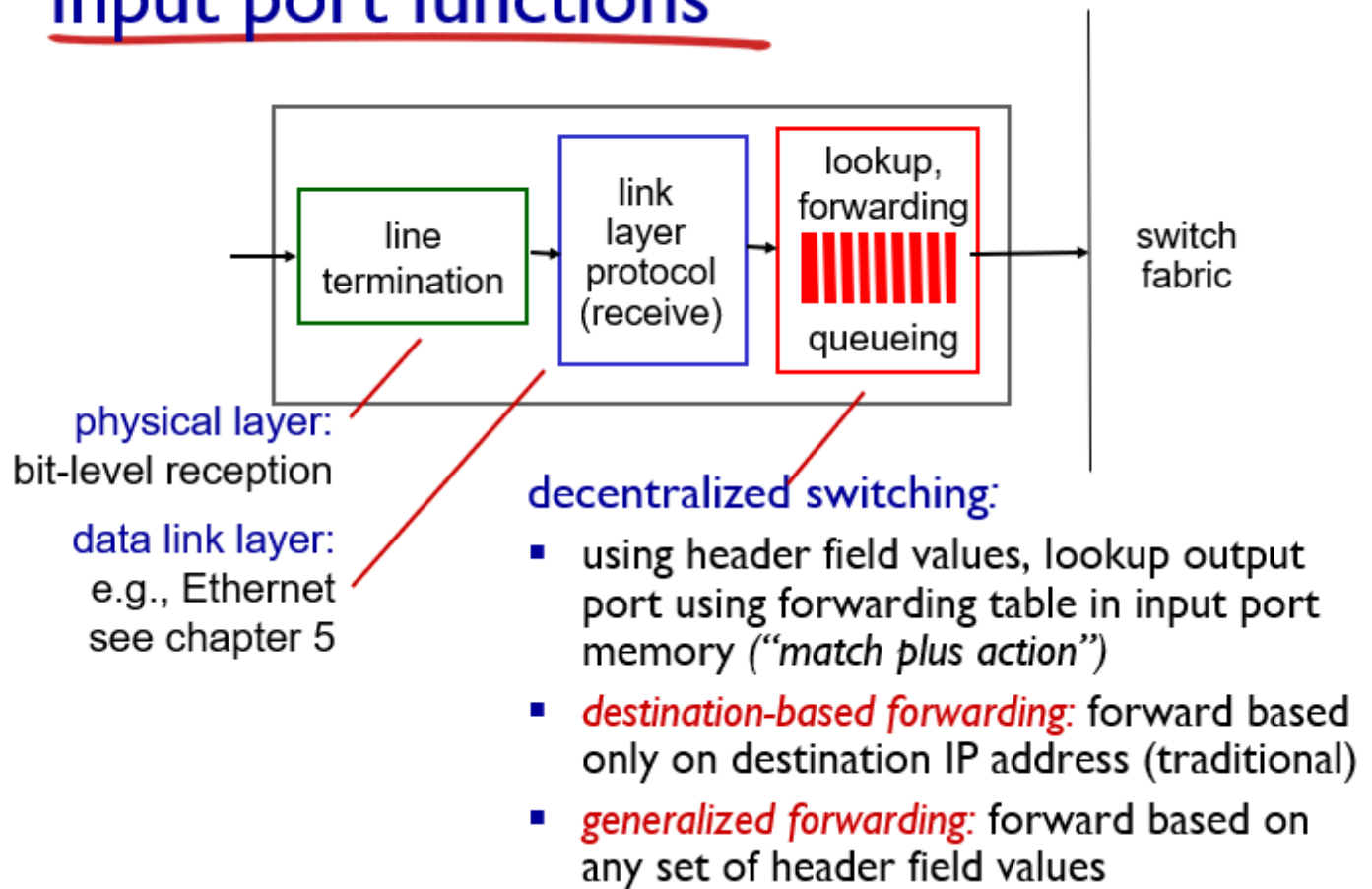
라우터의 입력 포트, 스위칭 구조, 출력 포트는 거의 하드웨어로 구현되고, 포워딩역할을 수행하며, 라우팅 프로세서는 소프트웨어로 구현되어 라우팅과 관리를 수행한다.

라우터를 이용한 전달은 크게 두가지로 나뉘어진다.

1. 목적지 기반 전달 : 라우터가 최종 목적지를 검색하고 최종 목적지로 연결되는 교차로 출구를 결정한 후 어떤 교차로 출구가 있는지를 알려준다
2. 일반적인 전달 : 목적지로 도달하는 과정에서 한산한 네트워크, 안전도와 신뢰도 등을 고려해서 이루어진다.

1. 입력 포트(Input port)

Input port functions



위 그림은 입력 포트의 간단한 입력 처리를 보여준다.

입력 포트의 link termination과 link layer protocol은 라우터의 개별 입력 링크와 관련된 물리계층/데이터링크를 구현한다. 그리고 오른쪽의 상자에서는 검색 기능을 수행하는데, 포워딩 테이블을 사용하여 검색한다.

- 목적지 기반 전달

포워딩 테이블을 사용해 포트를 처리하고 보낸다고 가정했을 때, 가장 쉽게 생각할 수 있는 방법은 각 목적지 ip 주소마다 하나의 엔트리를 배정하는 것이다. 하지만 32 비트 IP 주소의 경우, 40 개 이상의 가능한 옵션이 있어야 하므로 매우 비효율적이다. 이 문제를 어떻게 해결할 수 있을까?

Destination-based forwarding

forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Q: but what happens if ranges don't divide up so nicely?

위 그림은 라우터에 0에서 3까지 네 개의 링크가 있으며, 그 패킷을 다음과 같이 링크 인터페이스로 전달하는 예제이다. 간단하게 말해서 특정 값 범위끼리 범주화 시키는 것과 같다.

- 최장 프리픽스 매칭 규칙(Longest prefix matching)

Longest prefix matching

longest prefix matching

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
otherwise	3

examples:

DA: 11001000 00010111 00010110 10100001

which interface?

DA: 11001000 00010111 00011000 10101010

which interface?

최장 프리픽스 매칭 규칙은 IP 주소가 주어질 때, 각 포워딩 테이블에서 가장 긴 대응 엔트리를 찾고 여기에 연관된 링크 인터페이스로 패킷을 보낸다.

이런 검색을 통해 채킷의 출력 포트가 결정되면 패킷을 스위칭 구조로 보낼 수 있다.

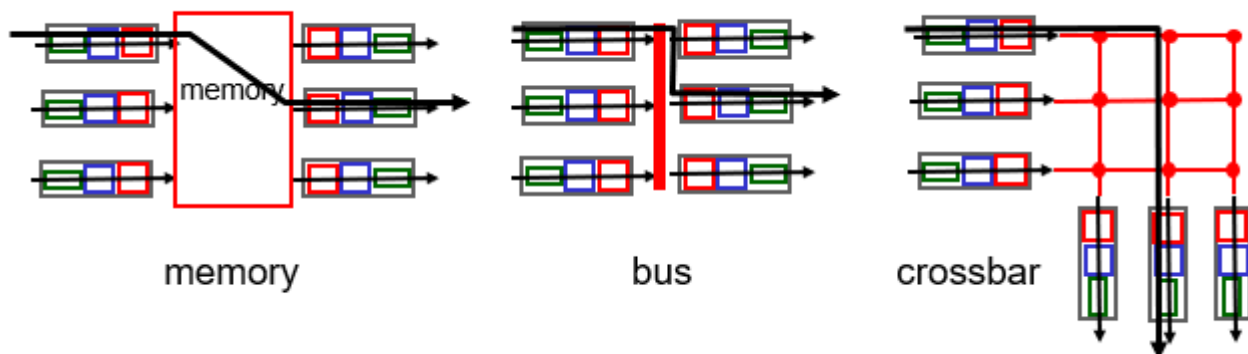
이제는 match 이후의 action 과정에 대해 살펴보자.

2. 변환기(Switching Fabrics)

스위칭 구조가 있어야만 패킷이 입력 포트에서 출력 포트로 전달될 수 있다.'

Switching rate : 얼마나 많은 패킷이 input 에서 output 으로 transfer 되는지

- Three types of switching fabrics



1. Memory : 가장 단순한 방법이다. 패킷이 도착하면 입력 포트는 라우팅 프로세서에게 인터럽트를 보내 패킷을 프로세서 메모리에 복사하고, 라우팅 프로세서는 헤더에서 대상 주소를 추출하고 포워딩 테이블에서 적절한 출력 포트를 찾은 다음 패킷을 출력 포트의 버퍼에 복사한다.

2. Bus : 여기서 입력 포트는 라우팅 프로세서가 없이 공유 버스만을 통해 직접 출력 포트에 패킷을 전송한다. 버스에 하나의 패킷만 통과할 수 있기 때문에 여러 포트가 버스에 패킷을 전송하면 하나가 통과하는 동안 나머지는 대기해야 한다. 라우터의 속도는 버스의 속도에 의해 제한된다.

3. Crossbar : 공유 버스의 대역폭 제한을 극복하기 위해, corssbar 를 이용한 interconnection network 방법이 제안되었다. 크로스바 스위치는 N 개의 입력 포트를 N 개의 출력 포트에 연결하는 2N 버스로 구성된 인터커넥션 네트워크이다. 패킷이 포트 A 에서 포트 Y 로 전달되어야 할때, 스위치 컨트롤러는 A 와 Y 버스와 포트 A 의 교차점에서 교차점을 닫고, 버스로 패킷을 전달한다. 이런 방식을 통해 Bus 의 가장 큰 문제점인 시간 문제점을 해결할 수 있게 되었다.

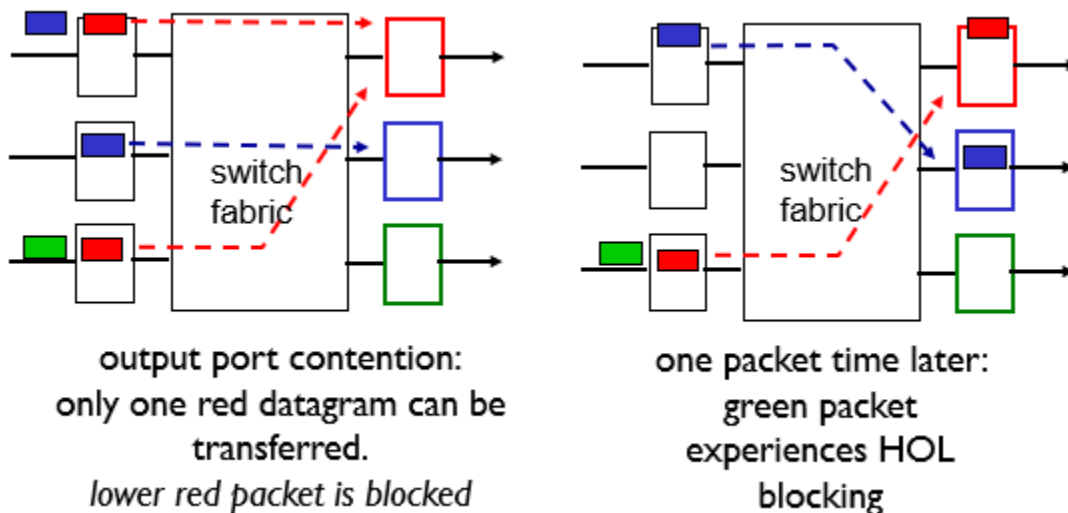
3. 출력 포트(output port)

출력 포트 프로세싱은 출력 포트의 메모리에 저장된 패킷을 가져와 출력 링크를 통해 전송한다. 여기에는 패킷 선택/대기열 제거/필요한 링크 계층 전송 기능을 수행하는 것이 포함된다.

- Queueing

큐잉은 라우터의 기본 구조이면서, 패킷 손실이 발생할 수밖에 없는 이유이기도 하다. 대기열이 커지면 커질수록 도착하는 패킷을 받을 메모리가 부족해지면서 패킷 손실이 일어난다.

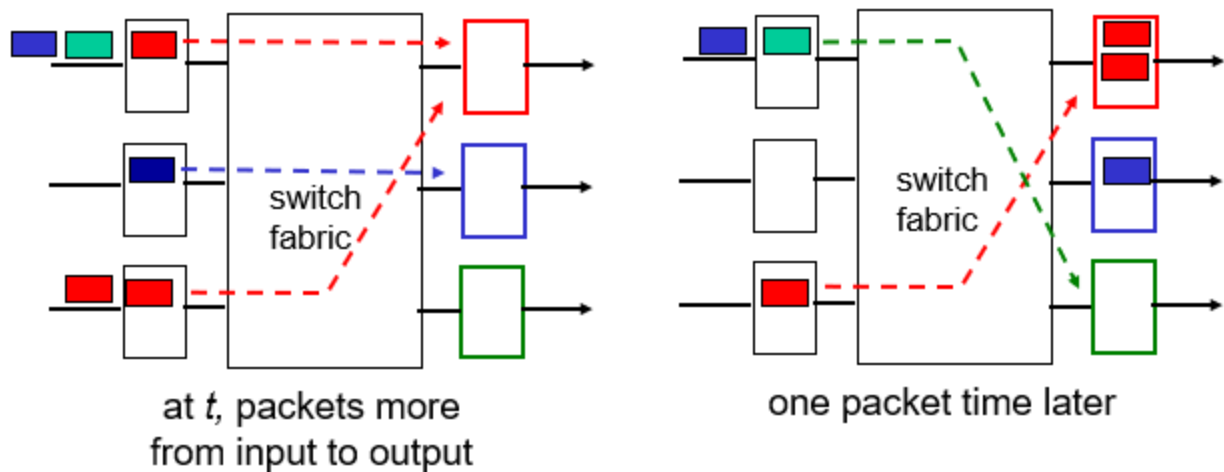
입력 큐잉(input queueing)



도착하는 패킷을 스위치 구조를 통해 전송하려는데, 스위치 구조가 충분히 빠르지 않으면 패킷이 출력 포트까지 가기 위해 기다려야 하는 지연 시간이 소모된다.

예를 들어서, 위쪽 그림의 왼쪽처럼 가장 위의 1 번 포트와 가장 아래의 3 번 포트가 동일한 빨간색 패킷을 스위치에 보내려고 할 때, lower red packet 은 가는 길이 막힌다. 이 때문에 뒤에 있는 초록 패킷(그리고 아마 뒤에 있을 수많은 패킷)까지 지연되어 버린다. 이를 HOL(head-of-the-line) 차단이라고 한다.

출력 큐잉(output queueing)



출력 포트 또한 큐잉 스피드가 들어오는 데이터를 감당하지 못했을 손실이 일어난다. 이를 막기 위해서는 패킷 스케줄링(packet scheduling)이 필수적이다.

4-3. 인터넷 프로토콜(IP) - IPv4, 주소 지정, CIDR, DHCP, NAT, IPv6

네트워크 계층 중에서 가장 널리 사용되는 것은 인터넷 네트워크 계층인 IP이다. 오늘날 사용 중인 IP는 크게 IPv4로 알려진 IP 프로토콜 버전 4와, IPv6로 알려진 IP 프로토콜 버전 6으로 만들어진다.

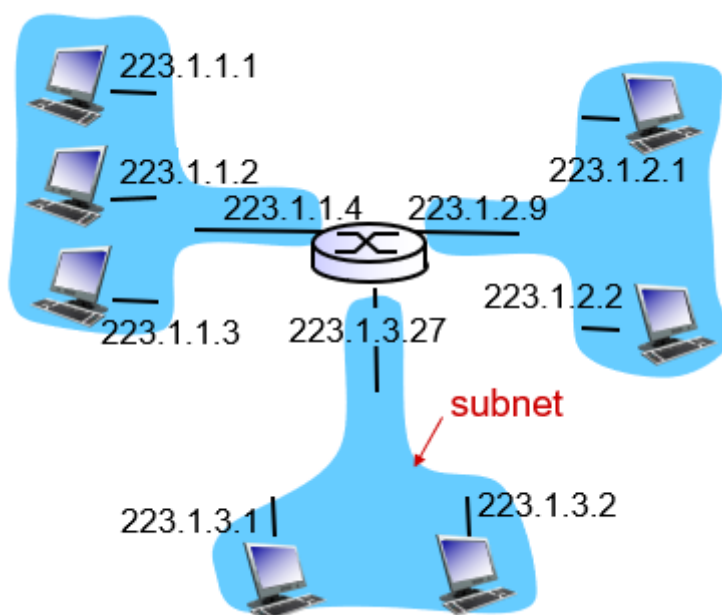
- IPv4 데이터그램 단편화

링크 계층 프레임에서 전달할 수 있는 데이터 크기에 한계가 있기 때문에(MTU라고 부른다) 각 데이터프레임은 fragment라고 불리는 파편으로 조각화하며(단편화) 목적지 전송 계층에 도착하기 전에 재결합한다.(TCP와 UDP는 재결합된 데이터를 받는 것만 기대하기 때문)

- IPv4 데이터그램 형식

데이터그램은 인터넷 네트워크 계층 패킷을 의미한다. 각 IP 주소는 2^{32} 개, 즉 4294967296 개(40억)의 자기만의 고유한 아이피 주소를 가질 수 있다.(물론 설정값으로 빠지는 IP 주소를 제외하면 가능한 숫자는 이보다 적어진다.) 이 IP 주소는 일반적으로 주소의 각 바이트를 십진수로 표현하고, 주소의 다른 바이트와 점(.)으로 구분하는 십진 표기법을 사용한다. 예를 들어 아래의 아이피를 이야기해보자. 193.32.216.9 → 193은 주소의 첫 번째 8비트와 같다. 32는 두 번째 8비트... 이런 식으로 전개된다. 이를 2진수로 표현하면 110000001 00100000 11011000 00001001 이런 식으로 표현할 수 있다. 이러한 아이피 주소는 사용자가 마음대로 가질 수 없고, IP 주소 일부는 연결된 서브넷이 결정한다.

서브넷 : 서브넷은 인터넷에서의 IP 네트워크를 의미한다.

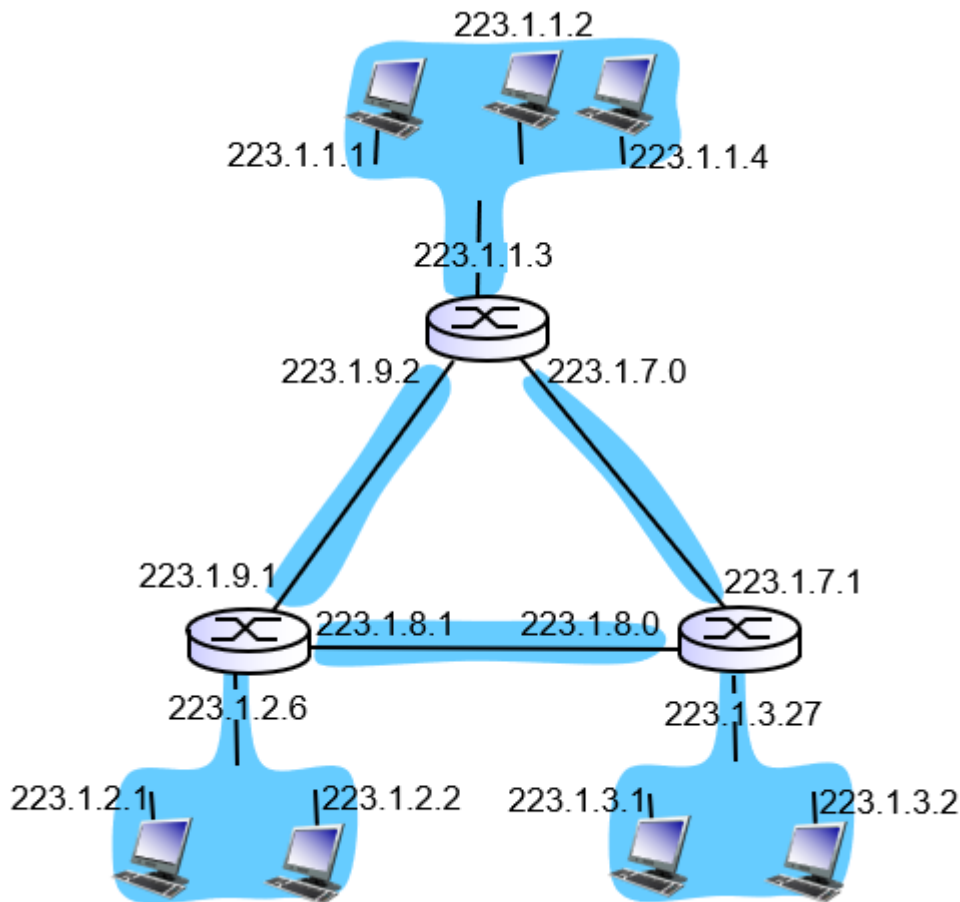


서브넷의 구조. 좌우는 wired Ethernet, 아래는 wireless Ethernet이다.

위 그림은 한 개의 라우터가 3 개의 인터페이스로 연결되어 7 개의 호스트를 연결한 모습을 보여준다. 여기서 왼쪽에 연결되어 있는 3 개의 호스트는 앞 세자리의 8 진수가 223.1.1 로 고정되어 있는 것을 알 수 있다. 또한 4 개의 인터페이스가 중계하는 라우터 없이 단 하나의 네트워크에 연결된다.

서브넷은 위 그림과 같이 몇개의 인터페이스와 하나의 라우터 인터페이스로 연결되어 있다. IP 주소체계는 이 서브넷에 223.1.1.0/24 라는 주소를 할당하는데, 이는 앞의 24 자리(8 진수 3 개)는 서브넷 주소라는 것을 가리킨다. 뒤의 /24 는 서브넷 마스크(subnet mask)라 불린다. 위의 예를 보자.

서브넷 마스크 223.1.1.0/24(왼쪽 서브넷)은 세 호스트 인터페이스(223.1.1.1, 223.1.1.2, 223.1.1.3)와 하나의 라우터 인터페이스(223.1.1.4) 이렇게 네 개를 구성한다. 만약 이 서브넷에 하나의 호스트를 더 추가할라면 아마 223.1.1.5 이런 식으로 붙을 것이다. 서브넷은 단순히 여러 호스트를 라우터 인터페이스에 연결하는 것 뿐만 아니라, 고립된 네트워크 각각을 모두 지칭하는 명칭이다.



/24 의 경우, 앞의 24 자리가 같으면 같은 서브넷, 아니면 다른 서브넷이다.

예를 들어, 위 그림은 3 개의 라우터에 연결된 이더넷 세그먼트를 잇는 3 개의 서브넷과, 라우터와 라우터를 연결하는 점대점 서브넷, 총 6 개의 서브넷을 가진다.

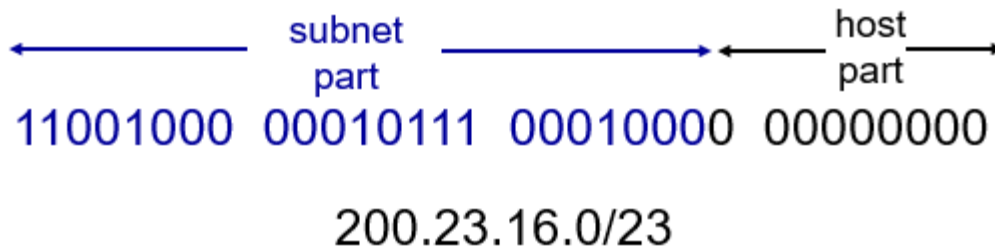
- CIDR(Classless Interdomain Routing)

CIDR 은 서브넷 주소체계를 일반화하기 위해 정해진 주소체계이다. 서브넷 주소체계는 32 비트 IP 를 가지고 있고, x 개와 32-x 개의 두 부분으로 나뉘어 점으로 된 십진수 형태의 a.b.c.d/x 구조를 가지게 된다. 앞의 x 개는 network prefix 라고 부르고, 공통 프리픽스를 공유한다. 여기서 외부 기관의 라우터는 이 앞의 프리픽스까지만 고려하고, 뒤는 고려하지 않는데, 프리픽스를 통해 들어온 것은 내부 라우터에게 위임해 포워딩 테이블의 크기를 줄일 수 있다.

주소의 나머지 32-x 비트들은 기관 내부에 같은 네트워크 프리픽스를 사용하는 모든 장비를 구분해준다.

CIDR: Classless InterDomain Routing

- subnet portion of address of arbitrary length
- address format: **a.b.c.d/x**, where x is # bits in subnet portion of address



예를 들어 위의 200.23.16.0/23 의 경우에는 앞의 23 자리가 prefix 이고 뒤의 9 자리는 기관 내부의 특정 호스트를 식별한다. 9 자리의 일정 비율을 특정 서브넷을 위해 할당하는 것도 가능하다.

CIDR 전 IP 주소의 네트워크 부분은 8,16,24 비트로 엄격히 제한했지만, 유연성이 부족하다는 문제가 생겨 CIDR 가 요즘에는 대세로 인정받고 있다.

- 주소 블록을 장비에 할당하는 방법

1. 주소 블록 획득

주소 블록을 획득하기 위해, 네트워크 관리자는 ISP 를 통해 주소를 받는다. 이 받은 주소는 특정 서브넷을 위해 더 작은 블록으로 나누어 임의적으로 할당할 수 있다.

예를 들어 ISP 의 블록이 200.23.16.0/20 일 경우

200.23.0001 까지가 prefix 이고 그 이후 12 자리는 네트워크 관리자가 임의로

200.23.00010000.00000000(200.23.16.0/24)

200.23.00010010.00000000(200.23.17.0/24)

.....

200.23.00011111.00000000(200.23.31.0/24)

이런 식으로 16 개를 만들 수도 있을 것이다. 여기서 만약에 x 가 24 가 아닌 23 이라면? 8 개의 하부 블록을 만들 수 있다. 이러한 주소 블록을 할당하는 최상위 국제기관은 ICANN 이 RFC7020 을 통해 관리한다고 한다.(TMI)

- 예를 들어보자. 서브넷 1, 서브넷 2, 서브넷 3 이 223.17.0/24 라우터에서 주소를 나눠가질 때, 서브넷 1 은 62 개, 서브넷 2 는

122 개, 서브넷 3 은 12 개의 인터페이스를 지원한다. 이러한 조건을 만족시키는 3 개의 네트워크 인터페이스를 각각 써보자.

서브넷 1 은 62 개의 서브넷이 필요하기 때문에 최소 $2^6 = 64$ 개의 서브넷 자리를 차지할 것이고, 서브넷 2 는 $2^7 = 128$, 서브넷 3 은 $2^4 = 16$ 개의 서브넷 자리를 각각 차지할 것이다. 각 서브넷의 크기를 8 비트의 자리에서 놓기 위해서는 서브넷 1 의 경우

223.17.0/26 을 차지해 00000000~00111111 까지 점유할 것이고

2 의 경우 223.17.65

2. 호스트 주소의 획득, DHCP

한 기관은 ISP로부터 주소 블록을 획득해, 개별 IP 주소를 기관 내부 네트워크와 라우터 인터페이스에 할당한다. 여기서 호스트에 IP 주소를 할당하는 것은 수동으로도 가능하지만 일반적으로 DHCP(Dynamic Host Configuration Protocol, 동적 호스트 구성 프로토콜)을 이용한다. 네트워크 관리자는 호스트가 네트워크에 접속하고자 할 때마다 고정된 IP 주소를 주거나 다른 임시 IP 주소를 할당하도록 DHCP 를 설정한다.

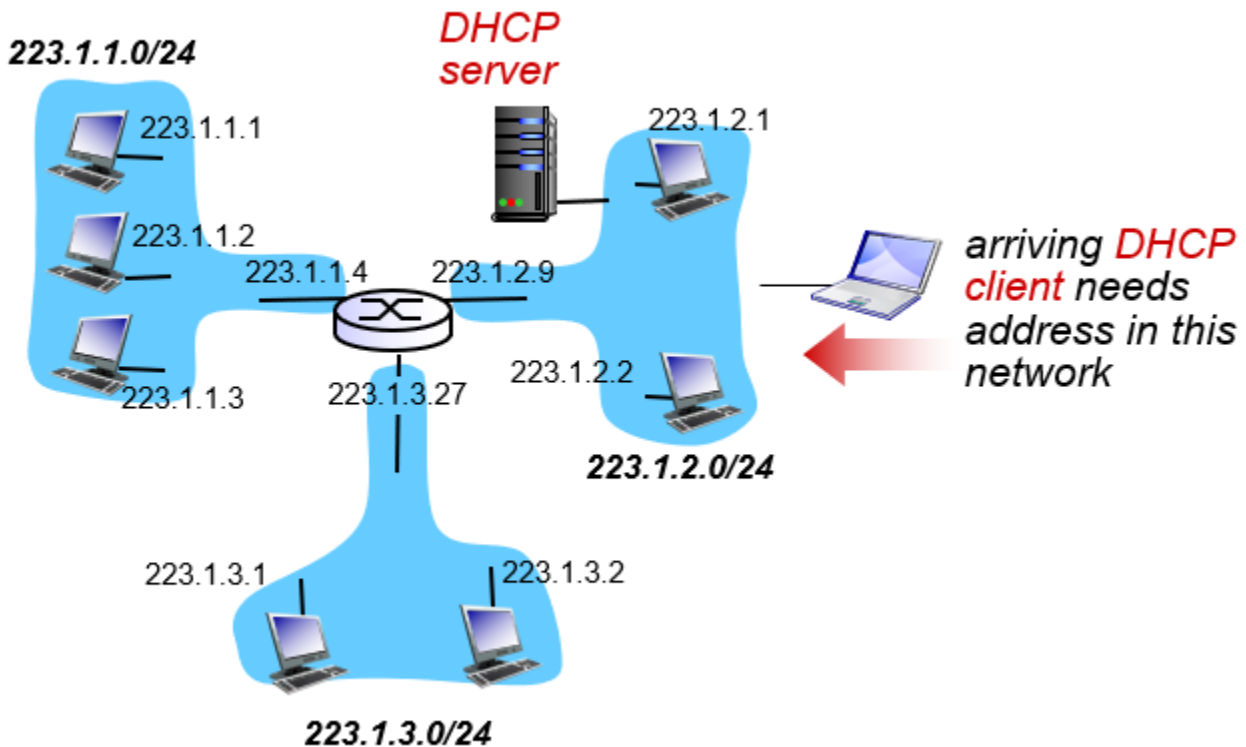
DHCP : 호스트에게 동적으로 IP 정보를 주기 위해 설정되는 프로토콜.

- 연결-후-작동 프로토콜(plug-and-play protocol)이라고도 불린다.

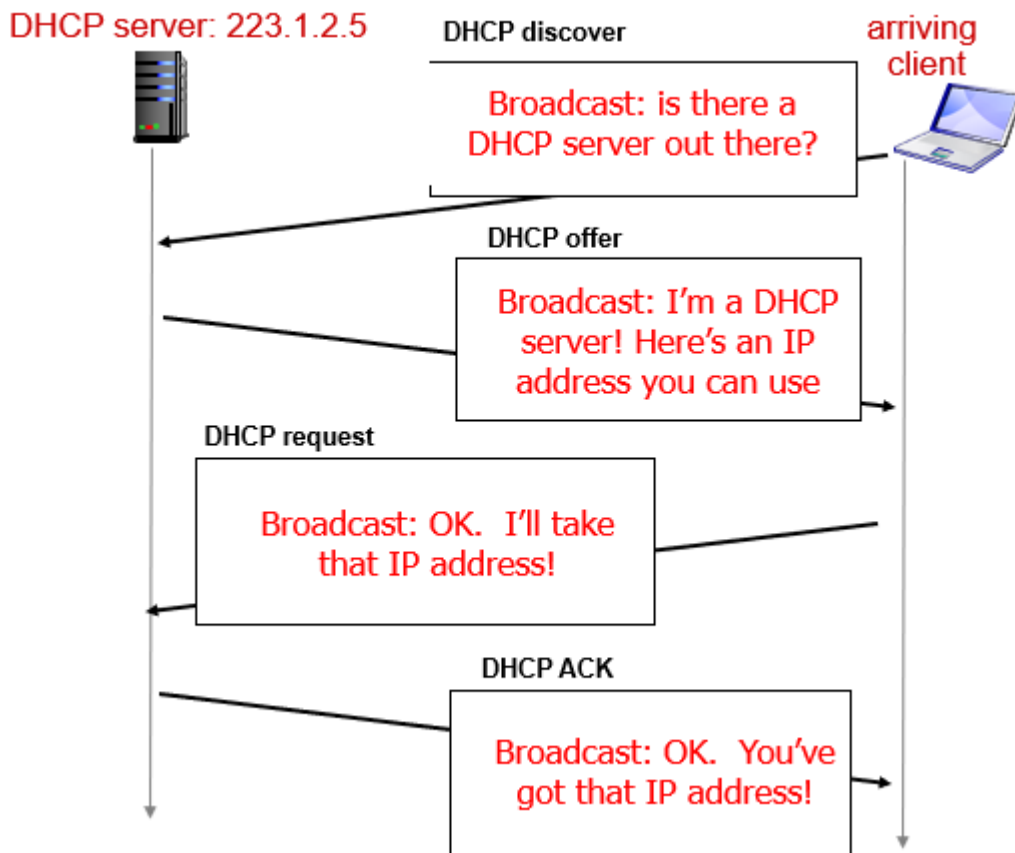
- 주소의 재사용을 가능하게 해준다.

- 호스트가 켜져있을 때만 주소를 할당해, 주소 공간을 절약할 수 있다.

- 네트워크에 참가하고 싶은 참여자가 많고, 이동이 잦고, 주소가 제한된 시간에만 필요할 경우 DHCP 기술은 매우 유용하게 사용된다.



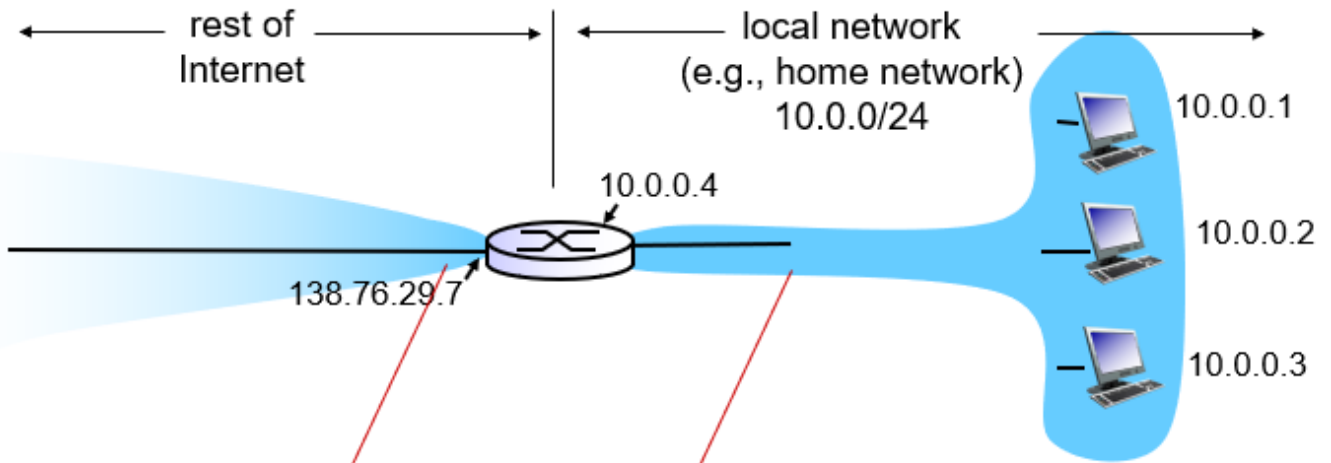
DHCP 는 그 자체로 클라이언트-서버 프로토콜이다. DHCP 는 아래의 과정으로 사용자에게 IP 를 지원해준다.



1. DHCP 서버 발견 2. DHCP 서버 제공 3. DHCP 요청 4. DHCP ACK

3. 네트워크 주소 변경, NAT

NAT 은 Network Address Translation 을 의미하며, 네트워크의 IP 주소를 관리하거나, IP 주소를 변경하고 싶을 이용한다. NAT 가능 라우터는 라우터 안 독립적인 주소를 가진다.



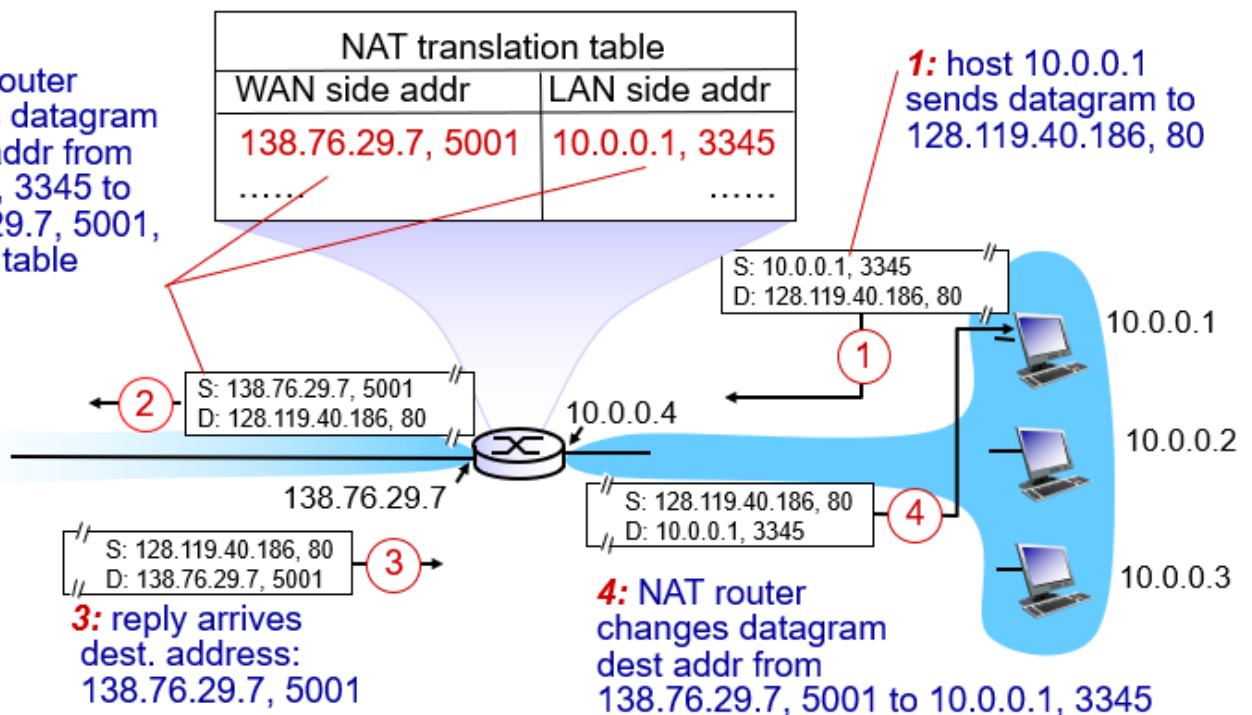
all datagrams **leaving** local network have **same** single source NAT IP address: 138.76.29.7, different source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

위 그림처럼, 138.76.29.7 안에 또다른 IP 인 10.0.0/24 사설 개인 주소 아이피를 가진다. NAT 을 쓴다면 이 세상에 수많은 10.0.0/24 아이피가 존재하겠지만, 이 개별 주소는 주어진 네트워크에만 의미가 있고, 밖에서는 138.76.29.7 로만 보인다. 이 안의 공간을 NAT 가능 공간이라고 한다. 밖에서 이 공간을 보면 단순히 하나의 IP(138.76.29.7)를 가지는 하나의 장비로 보일 뿐이다.

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80



위 그림에서 호스트 10.0.0.1의 가장 위 컴퓨터가 ip 주소 138.76.29.7 인 웹 서버(포트 80)에 웹 페이지를 요청한다고 생각해보자. 호스트 10.0.0.1은 임의의 포트번호 3345를 할당하고, LAN 에 데이터그램을 보낸다. NAT 라우터는 데이터그램을 받아 데이터그램에 대한 새로운 출발지 포트 5001을 생성하고, 새 출발지 포트번호 5001과 원래 출발지 포트번호 3345를 변경한다. 그리고 NAT translation table 에서 소스 ip 주소와 NAT ip 주소를 모두 저장한다음, 데이터가 오면 다시 들어오는 138.76.29.7,5001 포트를 저장된 호스트 10.0.0.1, 3345에 보낸다.

NAT 는 포트 번호가 호스트가 아닌 프로세스 주소 지정에 쓰여진다는 것 때문에 많은 논란을 낳았다. end-to-end argument 를 부순다는 점 때문에.

IPv6은 32비트로 이루어진 IPv4가 포화상태에 이르자 만들어진 새로운 접속체계이다. 여러 일을 벌이기 좋아하는 우리 개발자들은 이왕 IPv6으로 바꾸는 걸 여러가지 잡다한 일을 하기 시작하였다. IPv6의 특징은 다음과 같다.

- IPv6은 32비트 대신 128비트의 주소를 사용하면서 주소 제한에 대한 논란을 불식시켰다.(지구의 모든 모래알을 합친 숫자보다 크다 =_=)
- 헤더를 40바이트로 간소화시켜 라우터가 IP 데이터그램을 빨리 처리하게 하였다
- flow labeling을 새로 만들어 데이터 흐름에 대한 처리가 가능하게 하였다.
- fragmentation 기술을 삭제하고, 너무 큰 데이터는 거절 메시지(Packet is too big)를 송신자에게 보낸다. 송신자는 데이터를 IP 데이터그램 크기를 줄여 다시 보낸다.

4.4 - SDN, OpenFlow

SDN - Software Definition Networking. 복잡해지는 네트워크 계층 기능과 특정 링크 계층 기능을 제공하기 위해 통합된 정보 방식. 목적지 기반 포워딩 결정에서 일반 포워딩으로 확장하기 위해 OpenFlow라는 포워딩 통합 개념을 통해 정의되었다.

Openflow - SDN에서 명확한 개념화 및 컨트롤을 개척할 수 있게 한 표준. Openflow는 플로우 테이블로 알려진 검색 투가 작업을 통해 최단거리를 찾는다.

플로우 테이블은 다음과 같은 기능을 한다

- 플로우 테이블은 들어오는 패킷의 헤더 값들의 세팅을 매칭하여, 목적지로 신속하게 배달한다
- 패킷들에 의해 업데이트 되는 카운터 세팅은 플로우 테이블 요소들과 일치된다
- 패킷이 플로우 테이블 항목과 일치될 때 전달, 삭제, 복사 등의 여러 작업(action)이 가능하다.

플로우 테이블은 매칭-액션 작업을 주로 수행한다.

액션들은

1. 포워딩(Forwarding) : 들어오는 패킷을 특정 포트로 전달하거나, 모든 포트를 향해 브로드캐스트하거나, 선택된 여러개의 포트로 멀티캐스트한다. 패킷은 캡슐화되어 원격 컨트롤러로 전송된다.
2. 드로핑(Dropping) : 아무 동작이 없는 플로우 테이블 항목은 일치된 패킷을 삭제한다.
3. 수정필드(modify-field) : IP 프로토콜 필드를 제외한 모든 필드값을 다시 작성한다.

5-1. 네트워크 계층 - 제어 평면, 라우팅 그래프 알고리즘(Network Layer - Control Plane), Routing Graph algorithm

- 제어 평면

제어 평면은 데이터 평면과 더불어 네트워크 전체를 아우르는 구성요소 중 하나이다. 제어 평면은 데이터그램이 송신 호스트에서 목적지 호스트까지 라우터를 간 어떻게 전달되어야 하는지 뿐만 아니라 네트워크 계층 구성요소들과 서비스들을 어떻게 설정하고 관리할지도 제어한다. 여기서 최단 경로를 찾기 위해 그래프의 최단경로 알고리즘이 필요하다.

라우팅 알고리즘 : 라우팅 알고리즘의 목표는 송신자부터 수신자까지 네트워크를 통과하는 가장 좋은 경로를 찾는, 가장 대표적인 그래프 문제 중 하나이다.

일반적으로 좋은 경로란 최단 경로, 더 정확히 말해 최소 비용 경로를 말하지만, 금지된 경로 또는 시간이 오래 걸리는 경로를 생각하면서 데이터를 보내야 하기 때문에 상당히 복잡한 작업이 필요함을 알 수 있다. 라우팅 알고리즘은 다음과 같은 몇가지 분류로 나뉜다.

1. 중앙 집중형 라우팅 알고리즘(global routing algorithms)

여기서는 최단거리 정보를 찾을 때 네트워크 정보에 대한 완전한 정보를 가지고 출발지와 목적지 사이의 최소 비용 경로를 계산한다. 이러한 정보는 알고리즘이 실제 계산을 수행하기 전에 미리 알고 있어야 한다.

2. 분산 라우팅 알고리즘(decentralized routing algorithms)

여기서는 최소 비용 경로의 계산이 라우터들에 의해 반복적이고 분산된 방식으로 수행된다. 어떤 노드도 본인이 연결된 링크에 대한 정보만 알지, 모든 링크에 대한 완전한 정보를 가지고 있지 않다.

3. 정적 라우팅 알고리즘(static routing algorithms)

여기서는 한번 정해진 경로가 쉽게 바뀌지 않고, 바뀐다 해도 대부분 사람이 직접 개입해서 바꾸는 경우가 대부분이다.

4. 동적 라우팅 알고리즘(dynamic routing algorithms)

여기서는 네트워크 트래픽 부하(load)나 토폴로지 변화에 따라 라우팅 경로가 동적으로 바뀐다. 동적 알고리즘은 주기적으로, 또는 부하나 토폴로지 변화에 대응하는 식으로 수행된다.

5-2. 라우팅 기술 -AS, OSPF, BGP

- AS 내부 라우팅, OSPF

AS : AS 는 자율 시스템(Autonomous System)이라는 뜻이고, AS 는 동일한 관리 제어하에 있는 라우터의 그룹으로 구성된다. AS 는 AS 번호로 식별되고, IP 주소처럼 ICANN 의 지역 등록 기관에 의해 할당된다. IGP(interior gateway protocols)로도 불린다.

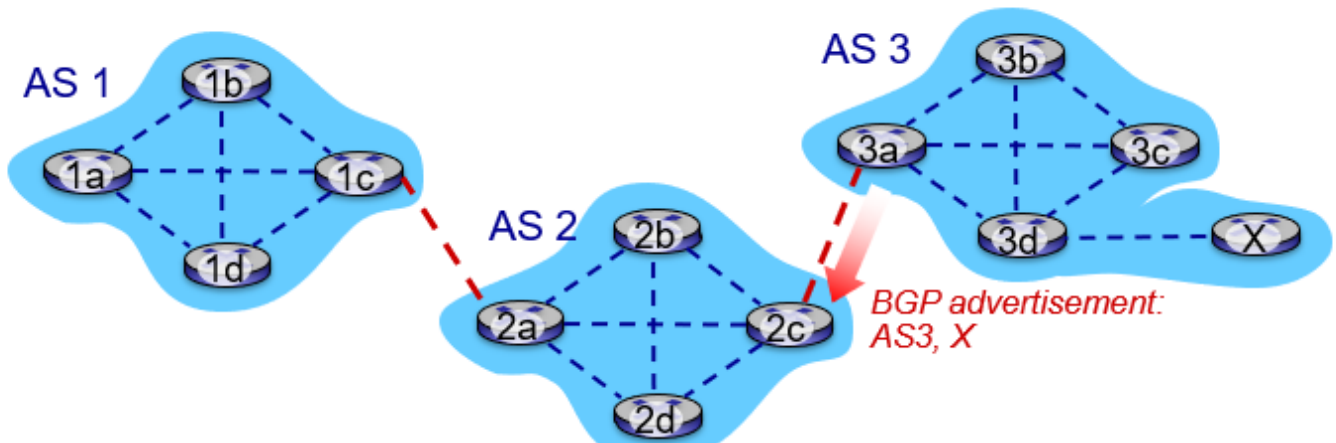
OSPF 프로토콜 : OSPF 라우팅은 인트라-AS 라우팅에 널리 사용되는 프로토콜이다. OSPF 의 뜻은 개방형 최단 경로 우선(Open Shortest Path First)이라는 뜻이다. Open 의 뜻은 라우팅 프로토콜에 대한 명세가 공개적으로 사용 가능함을 의미한다. OSPF 는 링크 상태 정보를 플러딩(flooding)하고, 다익스트라 알고리즘을 사용하는 링크 상태 알고리즘이다. OSPF 에 의해 각 라우터는 전체 AS 에 대한 완벽한 그래프를 얻고, 각 라우터는 자신을 루트 노드로 두고 모든 서브넷에 이르는 최단 경로 트리를 결정하기 위해 각각 다익스트라 최단거리 알고리즘을 설계한다. OSPF 는 다음과 같은 기능을 지원한다

- 보안 : OSPF 정보 교환 간 보안을 제공한다
- 복수 동일 비용 경로 : 하나의 목적지에 대해 동일한 비용을 가진 여러개의 경로가 존재할 때, OSPF 는 여러 개의 경로를 무작위로 사용할 수 있다.
- unicast 와 multicast 라우팅의 통합 지원
- 단일 AS 내에서의 계층 지원

- BGP

BGP 는 인터넷에 있는 수많은 ISP 들을 연결하는 프로토콜이며, 인터넷 프로토콜 중 가장 중요한 프로토콜 중 하나이다.

BGP 에서는 패킷이 특정한 목적지 주소가 아닌, CIDR 형식으로 표현된, 주소의 앞쪽 접두부(prefix)를 향해 포워딩된다. 각 접두부는 서브넷이나 서브넷 집합을 의미한다. BGP 에서는 목적지가 138.16.68/22 와 같은 주소로 표현되고, BGP 는 1. 이웃 AS 로부터 도달 가능한 서브넷 접두부 정보를 얻은 후, 2. 서브넷 접두부로의 가장 좋은 경로를 예측하고, 결정한다.



1. BGP 경로 정보 알리기

위 그림은 세 개의 자율 시스템 AS1, AS2, AS3 을 가지는 네트워크이다. AS1 과 AS2, AS3 을 잇는 빨간 선은 eBGP(external BGP) connect, 각 AS 내부에 잇는 파랑색 선은 iBGP(internal BGP) connect 이다. 가장 오른쪽의 X로부터 받은 정보를 AS3 이 AS2 와 AS1 에게 보낼 때, 게이트웨이 라우터 3a 는 2c 에게 AS3 x 라는, 2a 는 1c 에게 AS2 x 라는 메시지를 보낸다. 여기서 최적의 경로를 설정하는 것은 어떻게 진행되는 걸까?

2. BGP 의 최고 경로 설정인 뜨거운 감자 라우팅(Hot Potato Routing)

라우터가 BGP 연결을 통해 주소 접두부를 알릴 때, 속성(attribute) 안에 경로를 포함하는데, 그 안에 AS-PATH 와 NEXT-HOP 이라는 정보가 있다. AS-PATH 속성 안에는 알릴 메시지가 통과하는 AS 들의 리스트를 담는다.

뜨거운 감자 라우팅이라 불리는 라우팅 기술은 경로 각각의 시작점인 NEXT-HOP 라우터까지의 비용이 최소가 되는 경로가 선택된다. 뜨거운 감자라고 불리는 이유는 각각의 라우터가 자신 안에 있는 AS 만 신경쓰고 AS 바깥에 대한 비용은 신경쓰지 않은 채, 패킷을 최대한 재빠르게 자신이 머무는 AS 밖으로 넘기게 된다.

실제로는 단순히 움직이지 않고 좀 더 복잡한 경로 선택이 이뤄지게 된다. 특히 여러 개의 경로가 같은 값을 가지고 있을 때 다음의 규칙을 수행한다.

1. 라우터에 의해 설정된 지역 선호도(local preference)값이 가장 높은 경로가 우선적으로 선택된다.
2. 가장 높은 지역 선호값을 가진 경로가 여러 개 존재한다면, 이들 중에서 최단 AS_PATH 를 가진 경로가 선택된다.
3. 최단 AS_PATH 값까지 같은 경우 NEXT-HOP 라우터까지의 거리가 가장 가까운 경로가 선택된다.
4. 마지막까지 결정하지 못한 경로가 남아있다면 라우터는 BGP 식별자를 통해 최단경로를 결정한다

5-3. 제어 평면에서의 SDN과 ICMP

이번에 설명했던 SDN(Software Definition Networking) 또한 제어 평면의 로직을 필요로 한다. SDN 구조의 네 가지 특징은 다음과 같이 정리된다

1. Flow 기반 Forwarding : SDN 으로 제어되는 스위치들에서의 패킷 포워딩은 전송 계층, 링크, 네트워크, 계층 헤더의 어떤 값들을 기반으로도 이루어질 수 있다. SDN 에서는 네트워크 스위치들의 플로우 테이블 항목들을 모두 계산하고 관리, 설치하는 일들을 모두 SDN 제어 평면에서 수행한다.
2. 데이터 평면과 제어 평면의 분리 : 제어 평면은 플로우 테이블을 결정,관리하는 소프트웨어로 이루어지고 데이터 평면은 비교와 실행만 수행하는 네트워크 스위치들로 구성된다.
3. 네트워크 제어 기능이 데이터 평면 스위치 외부에 존재 : 전통적인 라우터와 달리, SDN 의 제어 소프트웨어는 네트워크 스위치에서부터 멀리 떨어진 별도의 서버에서 수행된다.
4. 프로그래밍이 가능한 네트워크 : SDN 이 제공하는 API 를 이용하여 네트워크를 프로그래밍할 수 있다.

ICMP - Internet Control Message Protocol. 호스트와 라우터가 서로 간 네트워크 계층 정보를 주고받기 위해 사용되는 것이다. 대부분 오류 보고를 위해 사용된다. ICMP 는 IP 의 메세지만 전달하는 한 부분으로 볼 수도 있다.

- ping : ping 프로그램은 타입 8, 코드 0 인 ICMP 메시지를 특정 호스트에 보내는 것이다. 대부분의 TCP/IP 구현은 ping 서버를 운영체제에서 직접 지원한다.

Type	Code	description
0	0	echo reply (ping)
3	0	dest network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	에코 요청
9	0	라우터 알림
10	0	라우터 발견
11	0	TTL 만료
12	0	IP 헤더 불량

위는 IPv4 기준 ICMP 메시지 타입이다.