지구 규모의 네트워크 만드는 법

A Bottom-Up Approach

고양우(yangwooko@gmail.com)

초판: 2013년 7월 26일
수정: 2014년 1월 5일

오픈넷(opennet.or.kr)의 시민학교 강의 자료로 작성된 문서이며 아직 한참 미완의 문서입니다. 오류 지적을 포함하여 모든 종류의 의견을 환영합니다. 메일로 보내주시면 감사히 받겠습니다.

주의: *이 글에 삽입된 그림 자료 중 URL이 표기된 것은 인터넷 검색을 통하여 가져온 자료로서 저작권 문제가 있을 수 있습니다.*

이 글의 목적: 어떻게 하면 전 지구 모든 컴퓨터를 연결한 거대한 네트워크를 만들 수 있을지 생각해보자. 거기에서 생겨나는 기술적인 어려움을 생각하고 (천재들이 이미 발견한) 해결책을 인터넷의 사례를 통해 알아보자.

1 배경 – 물리 계층 이야기

물리 계층(Physical layer)은 맨 밑에 있는 계층이라는 의미에서 1계층(Layer 1, L1)이라고도 부른다. 이 장은 이 글 전체의 시작이기도 하므로 물리 계층 이외의 얘기도 살짝 다루고 있다.

1.1 실 전화기 - 연결되어야 통신이 된다.

초등학교 때 종이컵으로 실 전화기를 만들어본 사람들이 많을 것이다. 두 사람이 서로 대화를 하려면 두 사람을 연결하는 한 줄기 실이 있어야 한다. 즉, 어떤 두 개체가 서로 통신을 한다는 것은 어떻게든 연결이 되어 있다는 것이 전제가 되어야 한다. 물론 연결은 직접 되어 있을 수도 있고 다른 개체를 통하여 한 다리 두 다리 건너서 되어 있을 수도 있지만 어쨌든 연결이 되어 있어야 한다. 실 전화기로 생각을 해본다면 중간에 있는 사람은 실 전화기 두 개를 하나는 왼손에 하나는 오른손에 들고 왼손의 전화기에서 들은 얘기는 오른손 전화기로 말해주고 그 반대도 마찬가

지로 함으로써 멀리 있는 사람들끼리도 전화가 되게 할 수 있을 것이다. (진짜 전화기도 이렇게 동작한다.)

그림 1 실 전화기[1]

여기서 연결은 물론 유선으로 될 수도 있고 무선으로 될 수도 있다. 선이 있느냐 없느냐 하는 것은 연결에서 사용하는 매체가 손으로 만질 수 있고 볼 수 있는 구리 전선이나 유리섬유를 사용하는 것이냐 아니면 전파 즉, 전자기파를 매체로 사용하느냐 일뿐 어떤 매체를 통하여 연결되어야 한다는 점에는 변함이 없다.

이때 실제로 통신을 하려는 두 개체를 통신의 당사자라는 의미에서 피어(peer)라고 하기도 하고 종단(end, endpoint)라고 하기도 한다. 그래서 이들 양 당사자간의 연결을 E2E 연결(종단간 연결, end-to-end 연결)이라 한다. 한편, 여러 다리를 건너 가는 경우 한 칸 건너는 것을 홉(hop)이라고 하며 이 한 칸 한 칸의 연결을 HBH 연결(hop-by-hop 연결)이라고 한다. 즉, 하나의 E2E 연결은 대개 여러 HBH 연결의 연쇄로 구성된다. 또한, 각각의 HBH 연결은 유선일 수도 무선 일 수도 있다.

1.2 코드

실 전화기에서 내가 전하려는 말은 그 자체로가 아니라 말소리(음성)로 표현되어 공기의 울림 또는 종이컵이나 실의 떨림으로 바뀌어 전달된다. 왜냐하면 말 그 자체는 공기나 종이컵이나 실이 잘(은 커녕 전혀) 전달할 수 없지만 이를 소리(즉, 물질이 특정한 주파수로 떠는 것)으로 바꾸면 그 주파수로 떨어주는 행위로 전달이 가능하기 때문이다. 이렇듯 내가 표현하려는 메시지를 (전달하기 편한) 다른 형태로 바꾼 것을 코드(부호)라고 하며 그 행위를 코딩(부호화)이라고 한다. "사랑"이라는 말을 소리 내었을 때 그 소리와 글로 적었을 때 그 글자는 같은 정보를 다르게 코딩한 것이다. 코드의 가장 중요한 원칙은 서로 약속이 되어 있어서 같은 뜻으로 해석되어야 한다는 것이다. 그러지 않으면 사랑한다고 했다가 뺨 맞는다. (합의가 되어 있어도 맞는 경우는 있다.)

설명을 간단히 하기 위해서 앞에서는 생략했지만 굳이 따지자면 말도 코드라고 할 수 있다. 즉,

---

[1] 그림 출처. http://cfile10.uf.tistory.com/image/124312364FED5F5E046201

내 맘속에 있는 사랑한다는 감정을 "사랑"이라는 두 음절의 말로 바꾸는 것도 코딩이고 코딩의 결과인 "사랑"을 두 음절의 소리나 두 글자로 표현하는 것도 코딩이라고 할 수 있다. 즉, 우리의 메시지는 실은 코딩된 것을 또 코딩하고 또 코딩하는 과정을 여러 차례 거쳐서 전달되는 것이다.

개념/정보/데이터를 코딩하기 위해서 컴퓨터에서는 코드 테이블을 쓰는데 그 중 가장 흔히 쓰이는 것이 아스키(ASCII, American Standard Code for Information Interchange)나 KS X 1001 이며 특히 글자의 경우에는 유니코드(Unicode)가 널리 쓰인다. 예를 들어, 아스키 코드 체계를 따르자면 알파벳 대문자 'A'에 해당하는 코드는 이진수로 나타내면 1000001 이다. 그럼 이 이진수를 어떻게 구리 선을 통해 전달할까? 간단히 생각해보면 전기를 흐르고 안 흐르게 하는 것이다. 전기가 흐르면 1 아니면 0을 나타내는 것으로 약속하는 것이다. 그럼 보내는 쪽에서 전기를 "흐안안안안안흐" 르게 하면 상태가 "아! A" 라고 알아 듣는 식이다. (물론 이 방식에는 문제가 많다. "안~~~~" 하고 있을 때 이게 0이 몇 개인지 정확하게 세려면 보내는 쪽과 받는 쪽이 단위 신호의 길이에 대하여 아주 정확하게 약속을 해야 한다. 또한, 무슨 이유에서건 중간에 연결이 끊기고 나면 어디서 끊어 읽어야 할 지 문제가 될 것이다. 잘못 끊어 읽으면 아버지 가방에 들어가시고 예수 까마귀를 쫓는 사태가 발생한다.

이때까지 살펴본 바와 같이 어떤 매체를 통해 전달하려고 할 때 그 매체에서 전달하기 좋게 코드로 표현하는 방식을 다루는 규격을 물리적 계층 또는 L1(layer 1)이라고 부른다. 즉, 물리적 계층은 하나의 홉에서 양 쪽이 서로 어떤 식으로 신호를 흘려 보낼까를 책임지는 계층이라고 보면 되겠다.

1.3 회선 vs. 패킷

앞서 실 전화기 이야기를 하면서 결국 전화기도 마찬가지로 두 전화기 사이에 선이 연결되어 통화가 된다고 설명하였다. 그럼 여기서 의문. 그런 식으로 선이 연결되어 있다면 한 쌍의 전화기는 통화가 되겠지만 또 다른 집하고는 어떻게 통화하나? 그래서 (요즘 어린이들은 들어본 적이 없겠지만) 전화교환양이[2] 필요하다. 전화교환양은 전화국에 앉아서 일한다. 각 가정의 전화기는 전화교환양의 전화기와 (실 전화기처럼) 곧바로 연결되어 있다. 내가 집에서 전화기를 들면 전화교환양과 바로 연결된다. 그 상태에서 "순돌네랑 연결해주세요"라고 하면 전화교환양은 내 집 전화기 선과 순돌네 전화기 선을 전선으로 연결해준다. 그럼 내 전화기와 순돌네 전화기가 직접 연결된 상태가 되고 통화가 가능한 것이다.

[2] 이 표현에 대하여 성차별 또는 특정 직업을 비하한다는 지적도 있으나 실제 이 직업이 있었을 당시의 표현을 그대로 쓴 것임을 이해해주기 바란다.

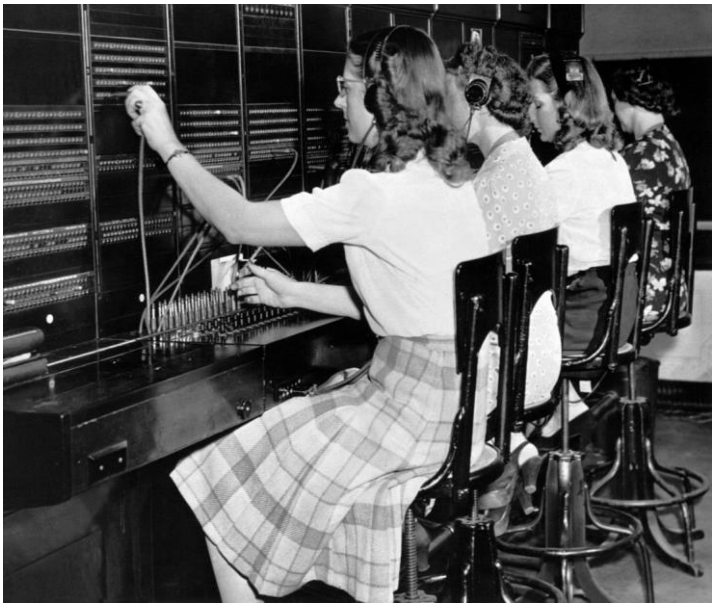


그림 2 딴나라 전화교환양 사진. 앞에 있는 패널(패치 패널이라고 부름)에 있는 구멍 각각이 가정의 전화기와 연결되었다고 보면 된다. 통화하려는 두 전화기 구멍을 손으로 잡고 있는 전선(패치 코드라고 부름)으로 서로 연결하면 두 전화기는 서로 연결된 상태가 된다.[3]

딴 집과 통화하고 싶다면 (전화교환양에게 부탁해서) 또 딴 집으로 선의 연결 상태를 바꾸면 된다. 이러한 방식을 회선을 바꾸는 연결하는 것이므로 ***회선*** 교환 방식이라고 부른다. 물론 요즘은 말로 상대방을 얘기하는 대신 숫자판을 누르고 전화국에는 전화교환양 대신 숫자판 눌리는 소리를 듣고 회선을 교환해주는 장치 즉, 전자교환기가 있을 뿐 원리상으로는 똑같다.

여기서 주목할 점은 내가 순돌네랑 통화하고 있을 때 내 전화기 ↔ 전화국 ↔ 순돌네 전화기 사이에는 하나의 회선처럼 연결된 상태이며 이 회선을 통해서는 "오직 내 통화만" 전달된다는 것이다. 하지만 조춘자-장소팔 커플이(이 분들을 모르는 젊은 독자를 위해서라면 속사포 래퍼 아웃사이더 같은 사람 둘이) 통화를 하고 있는 것이 아니라면 그 회선은 많은 시간을 놀고 있을 것이다. 앞의 예에서는 설명을 간단히 하기 위해서 나와 순돌네가 같은 전화국에 연결되어 있다고 (즉, 한 동네에 산다고) 가정했지만 만약 멀리 떨어져 있다면 전화국과 전화국 사이의 회선도 우리의 통화를 위해 "전용으로" 할당되어 있어야 한다.

이런 낭비를 피해서 하나의 회선을 공유할 수는 없을까? 그게 바로 ***패킷*** 교환 개념이다. 패킷 교환 방식에서는 보내려는 데이터를 적절한 크기로 조각 내서 각각의 조각을 네트워크를 통하여 전송한다. 시간을 정해놓고 그 시간 동안은 나만 미끄럼을 타는 것이 회선 교환 방식이라면 착한 아이들처럼 번갈아 가면서 미끄럼을 타는 것이 패킷 교환 방식이다. 그래서 미끄럼틀 하나만 있어도 여러 아이가 모두 행복할 수 있다. 하지만 내가 딱 타고 싶을 때 탈 수 있다는 보장이 없다는 것이 패킷 교환 방식의 단점이다. 예를 들어, 인터넷에서 파일을 내려 받을 때 지금 받아도 되고 1/10초 뒤에 받아도 되지만 말을 그런 식으로 지연시켜서 전송한다면 아마 짜증 나서 통화가

[3] 그림 출처. http://tpmdc.talkingpointsmemo.com/2011/07/voters-jam-capitol-phone-lines.php

안될 것이다. 따라서, 패킷 교환 방식은 굳이 실시간으로 데이터를 보낼 필요가 없는 컴퓨터 통신에 적합하고 회선 교환 방식은 실시간으로 보내야 하는 음성 통화에 적당한 기술이라고 할 수 있다. (물론 기술의 발전으로 이러한 차이는 지금으로선 큰 의미가 없다.)

패킷 교환 방식의 가장 큰 장점은 하나의 회선을 공유하여 회선의 활용도가 높아진다는 것이지만 또 한가지 주목할 점은 통신을 하고 있는 양 당사자를 단 하나의 회선으로 연결할 필요가 없다는 점이다. 그림 3에서처럼 어떤 구간을 연결하는 회선이 둘이 있다면 이 둘을 다 이용할 수 있으므로 일부는 이쪽으로 일부는 저쪽으로 가도 되고 심지어 둘 중에 한 회선이 망가지면 평소에 안 가던 길로 돌아가도 된다.

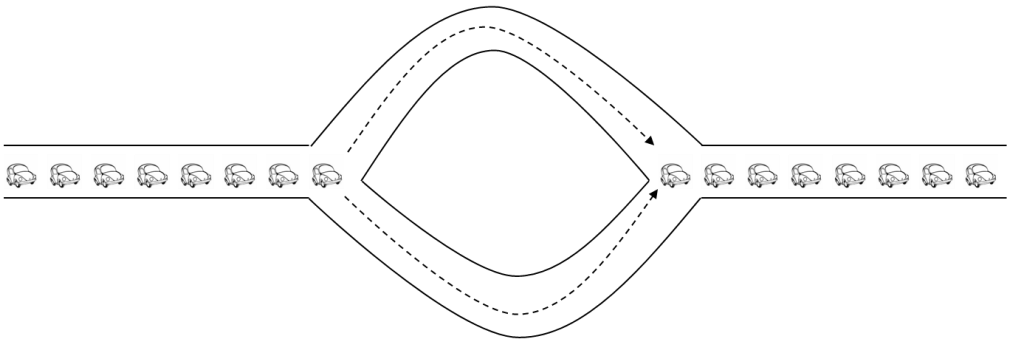


그림 3 길을 골라서 갈 수 있는 자유

즉, 패킷 교환 방식은 회선의 활용도를 높일 뿐만 아니라 여러 회선을 뒤섞어서 이용할 수 있으므로 일부 회선이 망가지더라도 돌아서라도 가는 길이 있으면 연결될 수 있다는 장점이 있어서 현재의 컴퓨터 네트워크는 패킷 교환 방식을 쓰고 있다.(고 대략 생각해도 크게 틀리는 않다.)

그럼 패킷 교환 방식은 누가 발명했을까? MIT(메사추세츠 공과대학)의 레너드 클라인락은 패킷 교환 방식을 다룬 최초의 논문을 1961년에 발표하였다. 한편, 같은 시기에 영국의 NPL(국립물리연구소, National Physical Laboratory)의 도널드 데이비스와 로저 스캔틀베리는 물론이고 RAND(랜드) 연구소의 폴 바란도 같은 생각을 갖고 연구를 하고 있었다는 점이다. 즉, 세 군데 연구소에서 같은 연구를 우연히 하고 있었다. 심지어 폴 바란은 미국 공군의 요청으로 생존 통신 기술 즉, 일부 회선이 끊어지는 등의 상황이 벌어져도 계속 통신할 수 있는 기술을 연구한 결과로서 패킷 통신 기술을 1961년 여름에 미공군에 제공한 바 있다. 우리가 현재까지 쓰고 있는 패킷이라는 용어는 영국 국립물리연구소에서 쓰던 용어를 따온 것이다.

2 MAC – 데이터 링크 계층 이야기

데이터 링크 계층(Date link layer)는 물리계층(L1) 바로 위의 계층이라는 의미에서 2계층(Layer 2, L2)라고도 부른다. 데이터 링크 계층에는 MAC 말고 다른 것도 많이 있지만 여기에서는 가장 대표적인 것으로 MAC만 다루기로 한다.

2.1 혼선을 피하는 법

무전기를 써보지 않은 사람이라도 이 말은 알고 있을 것이다. "오바". 할 말을 다 했으면 끝에 꼭

"오바"를 붙여줘야 된다. 그럼 이쪽에서 할 말을 다 했다는 것을 상대가 알고 자기가 할 말을 시작한다. 왜 이런 절차가 필요한가?

앞에서 설명한 시스템으로 생각해보자. 한 쪽에서 데이터를 보낼 것이 있어서 "안흐흐안"이라고 했는데 반대편도 보낼 데이터가 있다고 동시에 "흐안안흐"르게 해버리면 결과적으로는 "흐흐흐흐"르게 되어 완전 다른 얘기가 되어 버린다. 즉, 신호의 혼선이 발생하는 것이다. 따라서, 코드를 전송하는 회선이 공유되는 상황에서는 혼선을 막기 위해서 한쪽이 다 보냈다는 것이 확인할 수 있도록 끝에 "오바"를 붙이게 하는 새로운 약속이 필요한 것이다. 그렇다면 이 새로운 약속으로 문제는 해결되는가? 그렇지 않다.

그림 4 눈치껏 빨리 번호 붙이기 놀이[4]

이름은 모르겠는데 아이들의 놀이 중에서 번호를 차례로 붙이며 일어서는 놀이가 있다. 누군가 "1" 하고 일어섰다 앉으면 눈치보고 있다가 "2" 하고 일어섰다가 않는 식이다. 여기서 눈치를 보는 이유는 두 사람이상이 동시에 번호를 붙이며 일어나면 그 사람들 모두가 벌칙을 받는다. 동시에 번호를 붙이는 일 없이 끝까지 가면 아직 번호를 외치지 못하고 남은 한 사람이 벌칙을 받는다.

이 놀이에서 알 수 있듯이 상대방 말의 끝을 알 수 있어도 여전히 혼선은 생긴다. (심지어 둘이 대화해도 마찬가지다. "오바"라고 했는데 한참 기다려도 상대방이 아무 말도 안 해서 내가 얘기를 시작하는 순간 상대가 마침 얘기하면 또 혼선이 된다.) 그럼 혼선을 막으려면 어떻게 하면 되나? 추가의 약속과 희생(다르게 표현하자면 낭비)이 필요하다.

예를 들어, 순번을 정해서 보내면 된다. 매시 0분, 2분, 4분,... 에는 내가 보내고 1분, 3분, 5분, ... 에는 네가 보낸다. 그럼 혼선 생길 일은 없지만 대신 한쪽이 다른 쪽에 비하여 수다스러운 경우 (예를 들어, 인터넷에서 자료를 내려 받는 다면 서버 쪽에서 내 컴퓨터 쪽으로 데이터를 보낼 일

[4] http://www.old-picture.com/american-legacy/001/pictures/Children-Sitting-Circle.jpg

은 많아도 내 쪽에서 보낼 일은 없으므로 전체 통신 가능한 용량의 대략 절반은 허비하게 될 것이다.) 물론 사용자 수가 늘어난다면 다 쪼개면 된다. 사람 수가 늘어나도 쉽게 나눌 수 있고 혼선을 확실히 피할 수 있지만 허비되는 영역이 많다는 단점이 있다. 이러한 분배 방식을 시분할 방식(TDMA, time division multiple access)에서 사용한다.[5]

그런데 시분할 방식을 하려면 정확하게 몇 명이 동시에 쓸 것인지를 알아야 하고 어떤 식으로 배분되는지를 모두가 정확하게 알아야 한다. 따라서, 사용자수가 들쭉 날쭉하면 적용하기가 곤란할 뿐만 아니라 모든 사용자가 골고루 사용하는 경우가 아니라면 낭비가 심하다는 단점이 있다. 그렇다면 이렇게 해보면 어떨까? "아무도 안 보내고 있으면 일단 보내고 혼선이 생긴 것 같으면 기다렸다 또 보낸다" 이 기막힌 아이디어로 하버드 대학에서 1973년에 박사학위를 받은 사람이 메트칼프의 법칙으로도 유명한 메트칼프다.[6] 그 발명이자 제품인 이더넷(Ethernet)에서는 CSMA/CD라는 기법을 사용하는데 그 의미는 다음과 같다.

- CS: carrier sense 남이 보내나 안 보내나 느껴 보고 안 보내면 내가 보낸다.
- MA: multiple access 이런 짓을 여러 놈이 동시에 할 수 있다. 즉, 하나의 매체를 여러 놈이 공유한다.
- CD: collision detection 충돌(혼선)이 났는지 확인해서 충돌 나면 전송을 중지했다가 나중에 보낸다.

이 방식은 실은 우리가 일상 생활에서 많이 보는 방식이기도 하다. 예를 들어, 회사에서 화장실을 쓸 때 일단 누군가 쓰고 있나 본다. (CS) 쓰고 있으면 자리에 갔다가 좀 있다 다시 가본다. (이때 다시 가볼 때까지 지연 시간을 일정하게 하면 많은 사람이 계속 같은 시간에 화장실에 갔다가 허탕치는 것을 반복할 수 있으므로 지연 시간은 랜덤으로 해야 된다. 이더넷에서도 랜덤을 되어 있다.) 아무도 안 쓰는 것 같아서 들어가려고 하는데 누군가 동시에 들어가려고 하면 한쪽이 양보한다. (CD)

---

[5] 비슷한 개념으로서 우리나라의 휴대폰에 쓰이는 CDMA 방식이 있다. 여기서 C 는 code 로서 시간을 나누는 대신 코딩 방식을 다르게 하는 것이다. 즉, 각 사용자가 동시에 신호를 보내되 각기 다른 (하지만 혼선을 일으키지 않는) 코딩 방식을 쓰게 하는 것이다. 혼선을 일으키지 않게 하는 코딩 방식 중 우리나라의 휴대폰에서 쓰는 방식이 스프레드 스펙트럼 방식이다. CDMA는 아래의 CSMA/CD와 다른 것이므로 혼동하지 말 것.

[6] 현재 가장 널리 쓰이는 랜(LAN, local area network, 근거리 네트워크) 기술인 이더넷(Ethernet)을 발명한 사람. "PC와 LAN이 널리 보급되면서 인터넷 발전의 기반이 되는데 그 중에 특히 1973년 제록스 파크(Xerox PARC, Palo Alto Research Center)의 메트칼프가 만든 이더넷 기술이 결정적이다. 이로 인해 이전의 적절한 수의 시분할 컴퓨터가 연결되는 아르파넷 모델에서 많은 네트워크가 연결되는 인터넷 모델로 바뀌게 된 것이다." (그래서 등장한 개념이 A/B/C 클래스 개념) [1] 또한 랜 전문기업인 3Com을 창업했다. 주력 상품은 XNS 기반의 이더시리즈(EtherSeries). 참고로, 메트칼프의 법칙은 메트칼프가 만든 것이 아니다.

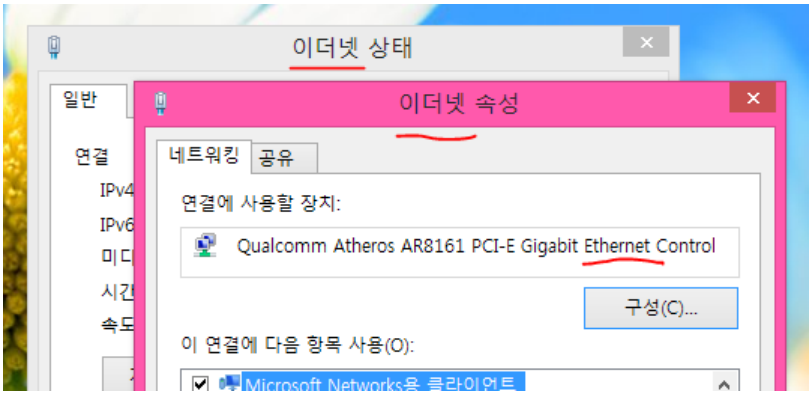


그림 5 이더넷이 대세입니다.

이 방식은 같은 매체를 공유하는 컴퓨터의 수가 많아지면 거의 항상 충돌과 대기가 일어나게 되어 통신이 마비가 될 수 있는 단점이 있지만 그런 특별한 경우를 제외한다면 아주 효율적인 방식이라고 할 수 있다. 컴퓨터 네트워크의 역사에 초기에는 전송률을 보장해줄 수 있는 TDMA 방식이 더 널리 쓰였지만 점차 효율도 높고 설치하기도 편리한 CSMA 가 대세가 되어 지금은 시중에서 랜이라고 하면 다 이더넷 방식을 쓰고 있다고 봐도 무방하다.[7] CSMA나 TDMA처럼 매체를 공유하는데 나타나는 문제를 해결하기 위한 규격을 싸잡아서 MAC(medium access control, 매체 접근 조정) 계층 규격 또는 L2(layer 2, 2계층) 규격이라고 하며 이 계층에서 데이터 전송은 당연히 패킷 단위로 이뤄지며 각각의 패킷은 MAC 패킷이라고 부른다.

2.2 왜 MA를 해야 되나?

앞서 하나의 매체를 여러 컴퓨터가 공유할 때 나타나는 문제와 해결 방법을 살펴보았다. 그렇다면 이렇게 복잡하게 할 것 없이 매체를 공유하지 않으면 될 것 아닌가? 맞다. 공유하지 말고 전용 직통 회선을[8] 뚫어버리면 내가 보내고 싶을 때 막 보내도 충돌이 날 것 없다. (그림 6에서 fully connected가 그런 경우다.) 그렇게 안 하는 이유는? 땅이 좁아서. 진짜? 계산을 해보자.

---

[7] 랜의 초창기에 많이 쓰였던 토큰 링 방식과 이더넷 방식을 비교하면 딴 것은 차치하고서라도 회선의 구성과 일부 회선에 문제가 생겼을 때 해결의 편의성 면에서 비교가 안될 정도이다. 하지만 이는 이 글에서 다루기에는 너무 긴 얘기라 생략.

[8] 남북한 당국자간에 맨날 끊었다 이었다 하는 직통회선(또는 핫 라인)이 이런 방식이다. 회선을 공유하지 않고 두 전화기를 전용으로 직통으로 잇기 때문에 전화를 걸 필요도 없이 들기만 하면 통화가 된다(고 생각한다).

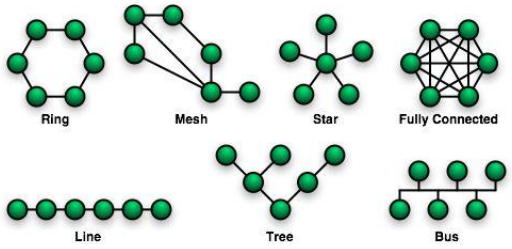
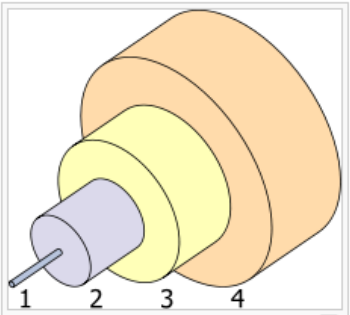


그림 6 네트워크를 연결하는 여러 방법 – 전문 용어로 토폴로지라고 한다.[9]

전 세계에 백 억 대의 컴퓨터가 있다고 하자. 내 컴퓨터에서 백억에게 연결하려면 내 컴퓨터로 백 억 개의 회선이 들어와야 한다. 백 억 개의 회선을 가로 세로 100 마이크론(즉, 0.1 밀리미터)를[10] 차지하는 회선으로 만들면 가로 세로 10미터가 된다. 이런 선을 지구 둘레만큼 돌려서 깔아야 되니 그 가로 세로 10미터에 길이가 4만 킬로미터짜리 케이블이 컴퓨터 한대 마다 필요하다. 그런데 정작 문제는 그래 봤자 나 혼자 쓸 수 있을 뿐이다. 그런 컴퓨터가 다시 백 억 대가 있으니 케이블의 단면적은 (높이는 세로 즉, 10미터로 가정하면)

4만 킬로미터 x 10 x 10 미터 x 100 억 = 4,000,000,000,000 제곱 킬로미터

가 되고 지구의 표면적이 500,000,000 제곱 킬로미터쯤 되므로 전 지구에 펼쳐 놓으면 높이가 대략 8만 킬로 미터가 된다. (땅을 아무 파도 묻을 곳이 없다.) 그래서 현재는 그림 6의 여러 방식 중 버스(bus) 방식 즉 하나의 매체(전선)을 모든 컴퓨터가 공유하는 방식이 일반적으로 쓰이고 있다.



The structure of a typical single-mode fiber,
1, Core: 8 µm diameter
2, Cladding: 125 µm dia,
3, Buffer: 250 µm dia,
4, Jacket: 400 µm dia,

그림 7 광 케이블의 구성 [11]

---

[9] 그림 출처. http://wiki.laptop.org/images/2/2b/Network_topologies.jpg

[10] 광 케이블의 굵기가 머리카락과 비슷한 굵기로서 대략 125 마이크론 쯤 된다. 겉에 껍데기 씌운 부분 등을 빼고.

[11] 그림 출처. https://en.wikipedia.org/wiki/Optical_fiber

2.3 MAC 주소는 뭐임?

그래서 혼선이 나는 것을 감수하고서라도 하나의 매체(대개는 전선)을 수 십 대의 컴퓨터가 공유해서 네트워크를 쓰는 것이다. 그런데 하나의 매체를 여러 컴퓨터가 공유하면 혼선 문제만 생기는 것이 아니다. 내가 보내려는 데이터를 누구에게 보내는 것인지 표시를 해야 된다. 그러지 않는다면 듣는 쪽에서 들어야 할 얘기와 듣지 않아도 될 얘기가 뒤죽박죽 되어 무슨 얘긴지 알 수 없게 되어버릴 것이다. 그래서 데이터를 보낼 때는 꼭 그 데이터를 받을 상대 컴퓨터가 누군지를 표시해야 한다. 이 때 데이터를 받아야 할 각 컴퓨터(더 정확히는 그 컴퓨터의 특정 네트워크 카드)의 고유한 이름을 MAC 계층에서 사용한다고 하여 MAC 주소라고 한다. MAC 주소는 네트워크 카드마다 고유하며 네트워크 카드를 생산할 때 공장에서 고유한 번호를 펌웨어에[12] 구워 넣는다. (MAC 주소는 펌웨어에 들어 있으므로 실제로는 언제든지 변경 가능하며 특별한 경우에 이를 수정하는 경우도 있다.) 그럼 MAC 주소를 이용하여 어떻게 데이터가 뒤죽박죽 되는 것을 피할 수 있나?

컴퓨터가 데이터를 보낼 때는 그 데이터만 보내는 것이 아니라 여기에 MAC 주소를 앞에 붙여서 보낸다. 흡사 편지를 보낼 때 편지를 봉투에 담고 봉투에는 받을 상대의 주소(와 보내는 사람의 주소)를 적는 것과 같은 원리이다. 컴퓨터 네트워크에서는 봉투에 해당하는 부분은 헤더(header, 머리)라고 하고 편지에 해당하는 부분을 페이로드(payload, 짐)라고 한다. 헤더에는 MAC 주소 외에 다른 정보도 들어가지만 이 글의 범위를 벗어나므로 생략한다. 헤더와 페이로드를 한 덩어리로 전송하는데 이것이 앞서 설명한 MAC 패킷이다. 그림 8에서 볼 수 있듯이 실제 보내려는 데이터(DATA)의 앞 부분에 헤더가 붙어 있다. 그리고 맨 뒤에는 4바이트의 확인용 정보가 붙어 있어서 전송 중에 패킷이 깨졌는지 따위를 알 수 있다.

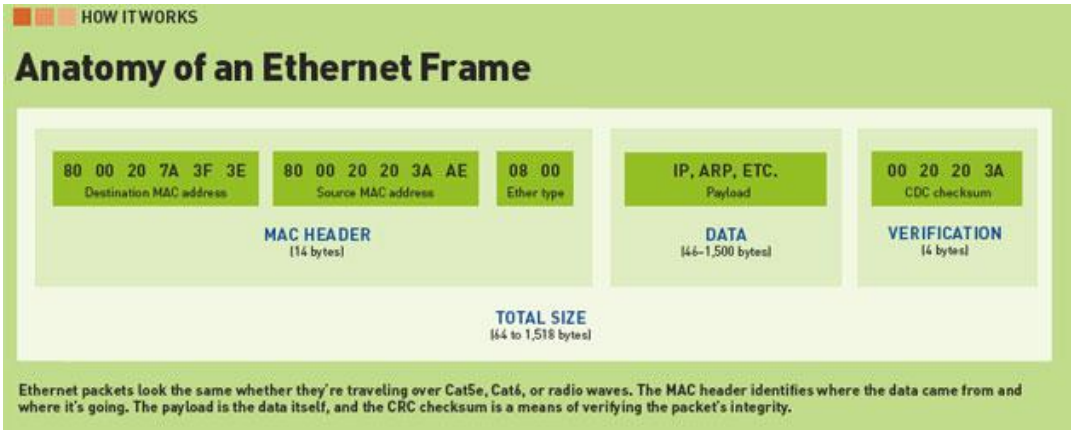


그림 8 이더넷에서 MAC 패킷의 모습[13]

데이터는 누구에게 보내려는 것인지 헤더를 달고 오므로 받는 쪽에서는 헤더를 보고 자기 것이면

---

[12] 비휘발성 메모리에 저장된 프로그램과 데이터. 특별한 장치와 프로그램을 쓰면 소프트웨어처럼 변경 가능하지만 일반 사용자 입장에서는 하드웨어처럼 변경이 불가능하다고 대개는 인식한다.

[13] 그림 출처. http://www.tech-juice.org/wp-content/uploads/2011/06/ethernetframe.png

받고 아니면 그냥 버리면 된다. 헉? 잠깐.

받지 않아야 할 데이터를 거르는 것은 받는 쪽의 책임이다. 즉, 굳이 버리지 않는다면 모든 데이터를 다 받을 수 있다는 얘기다. 따라서, 현재의 컴퓨터 네트워크에서는 내게 보낸 데이터가 아닌 것을 버리는 기능을 꺼버리면[14] 남들이 주고 받는 모든 데이터를 다 볼 수 있다. 별도의 도청 기능이 필요한 것이 아니다. 말하자면 광장 같은 데서 큰 소리로 서로 얘기하는 것과 같다고 생각하면 된다. 남의 얘기를 들을까 말까 하는 것은 듣는 사람의 재량이다. 다시 한번 말하지만 컴퓨터 네트워크에서 당신이 보내는 모든 것은 남들이 다 보고 있다. (그럼 비밀스럽게 통신하려면 어떻게 해야 되나? 이는 제4.5장에서 다룬다.)

3 서브 네트워크와 인터넷 프로토콜 – 인터넷 계층 이야기

이 장에서 다루는 내용은 인터넷의 중심이 되는 인터넷 프로토콜(Internet Protocol, IP)이다. 이 프로토콜이 담당하는 계층을 인터넷 계층 또는 네트워크 계층이라고 부르며 2계층(L2) 바로 위의 계층이라는 의미에서 3계층(Layer 3, L3)이라고도 부른다.

3.1 왜 네트워크를 쪼개야 하나?

여기까지 배운 기술 즉, 버스와 MAC 규격을 이용해서 전 지구를 연결하는 네트워크를 만들었다고 하자. 즉, 그림 9에 나온 것처럼 하나의 엄청나게 긴 전선을 깔고 모든 컴퓨터를 이 전선에 연결했다고 하자. 그럼 어떻게 될까? 통신이 거의 불가능하다. 왜? 너무 많은 컴퓨터가 다들 데이터를 보내고 싶어 하니 서로 눈치고 줄 서고 기다리다가 날새는 것이다. 명절 연휴 마지막 날 휴게소 화장실을 생각해보라.

---

[14] 이렇게 꺼버리고 다 받는 것을 프로미스큐어스 모드(promiscuous mode, 난잡 모드 ^^)라고 한다. 이 글의 범위를 훌쩍 벗어나는 얘기지만 비싼 네트워크 스위치에서는 매체를 공유하지 않게 함으로써 속도를 높인다. (대신 스위치 내부 구성이 복잡해지고 그래서 가격이 비싸다.) 따라서, 프로미스큐어스 모드로 하더라도 모든 데이터를 다 볼 수 없게 된다. 보안 관제를 위하여 대개 이러한 스위치에서는 프로미스큐어스 포트를 따로 제공해서 여기에 컴퓨터를 연결하면 모든 데이터를 볼 수 있게 된다.

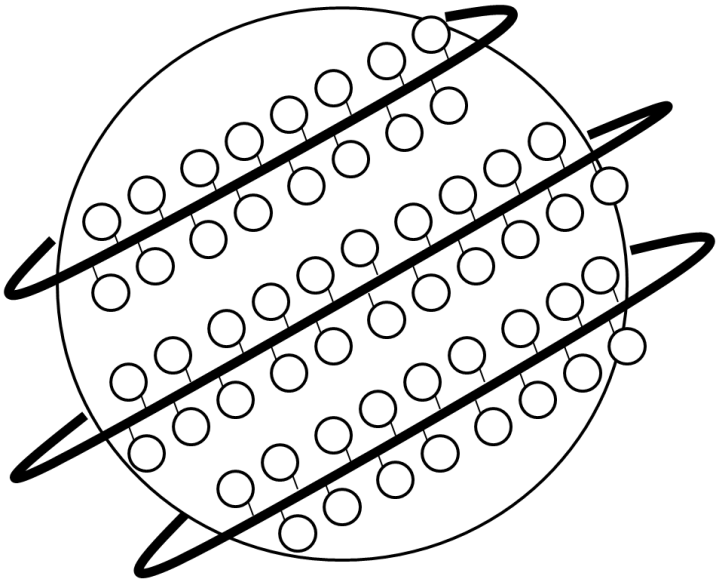


그림 9 전 지구 단일 회선 네트워크 (그리느라 고생 했음)

만약 컴퓨터들이 데이터를 보낼 일이 거의 없어서 줄을 설 일은 거의 없다고 하면 될까? 그래도 안 된다. CSMA/CD 방식의 첫 단계는 CS 즉, 누가 데이터를 보내고 있는지 아닌지 확인하는 단계인데 이게 너무 오래 걸린다. 왜? ***빛이 너무 느리기 때문이다***. 대부분의 사람들에게 생소하겠지만 컴퓨터를 하는 사람들에게 빛은 너무 느리다. 물리학의 이론에 따르면 빛 보다 빠를 수는 없다는데 앞날이 참으로 암담하다. 얼마나 느린지 보자.

어렸을 때 많이들 들어본 이야기. "빛은 1초에 지구를 일곱 바퀴 반이나 돈다." 빛의 속도가 대략 초속 30만 킬로미터이고 지구의 둘레가 대략 4만 킬로미터이기 때문에 저 산수가 맞다. 그렇다면 한 바퀴 도는 데는 1/7.5 초(= 0.1333초)가 걸린다는 얘기고 내가 네트워크 버스에 쏜 신호가 전기 또는 빛의 형태로 지구 반대편 누군가에게 전달되는데 (직선으로 반듯이 간다고 해도) 대략 0.1초는 걸린다는 얘기다. 즉, 누군가 데이터를 보내고 있는지 아닌지 확실히 확인하려면 대략 0.1초 이상은 기다려봐야 된다는 얘기다. 요즘 네트워크 카드가 대개는 초당 몇 기가비트를 전송할 수 있고 좀 성능이 떨어지는 것도 초당 100메가비트의 데이터를 보낼 수 있으므로 0.1초면 대략 1메가바이트의 파일을[15] 전송할 시간인데 그런 정도의 시간을 그냥 속절없이 기다린다? 속 터질 노릇이다. 마찬가지로 CD 단계에서 충돌이 났는지 아닌지 확인하려면 대략 0.2~0.3 초 정도를 기다려야 한다.

이러한 비효율을 다 참을 수 있다고 하더라도 만약 그 선이 어디선가 끊어진다면 지구 전체의 네트워크가 한꺼번에 중단된다. 이는 도저히 용서할 수 없는 사태다. 그 외에도 아직 설명하지 않은 (아래에서 설명할) 여러 이유 때문에 버스 형태의 공유 매체는 너무 길면 안 된다.

3.2 일단 쪼개보자

[15] MP3로 된 짧은 노래 한 곡쯤 되는 분량.

로마의 카이사르나 프랑스의 나폴레옹이 즐겨 사용했다는 "쪼개서 정복한다(divide and conquer)" 라는 작전은 실로 다양한 곳에 적용된다. 지구 규모의 네트워크도 당연히 쪼개서 풀어야 한다.

그럼 네트워클 적절한 크기로 다 쪼겠다고 하자. 즉, 공유 되는 버스를 짧게 만들고 가까이 있는 컴퓨터끼리 여기에 다 연결했다고 하자. 그럼 그 버스에 연결된 컴퓨터끼리는 통신이 잘 될 것이다. 하지만, 이렇게 하면 서로 다른 버스에 연결된 컴퓨터끼리는 통신이 될 수 없다.

그럼 어떻게 하나? 떨어진 네트워크를 서로 연결해주는 녀석이 필요하다. 그런 녀석을 우리 네트워크와 다른 네트워크 사이의 관문 역할을 한다고 하여 게이트웨이(gateway)라고 부른다. 직접 네트워크 설정을 해 본 사람들은 다들 들어본 말일 것이다. 그럼 게이트웨이가 뭘 할까?

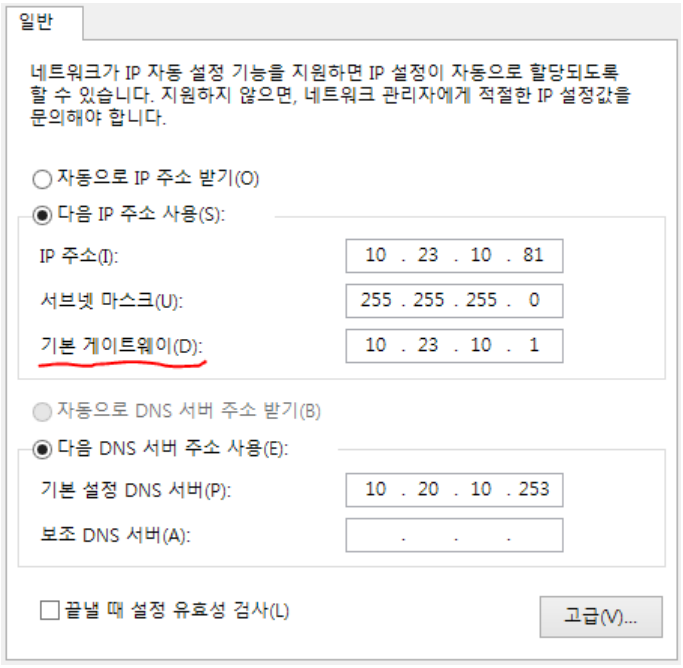


그림 10 마이크로소프트 윈도8의 네트워크 설정 화면에서 게이트웨이 지정 부분

떨어진 두 네트워크를 연결해야 되니 한 네트워크 당 하나씩 네트워크 카드가 달린 즉, 네트워크 카드가 두 개 달린 컴퓨터라고 생각하면 된다. 그림 11에서 게이트웨이는 한 쪽의 네트워크 카드로는 네트워크1에 또 한 쪽으로는 네트워크2에 ***속해 있음으로써*** 두 네트워크를 연결한다. 뭐 그렇다고 덜렁 이렇게 연결만 하면 된다고 하면 너무 쉽지? 게이트웨이는 뭘 해줘야 할까?

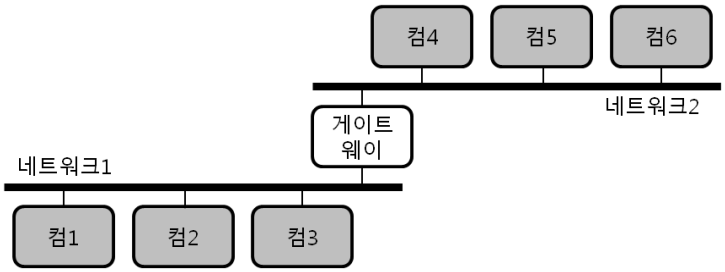


그림 11 게이트웨이는 두 네트워크를 연결할 수 있다

당연히 게이트웨이는 네트워크1에서 전송된 패킷을 받아서 네트워크2로 보내고 네트워크2에서 전송된 패킷을 받아서 네트워크1로 보내야 할 것이다. 그럼 되나? 처음에는 된다. 하지만, 규모가 커지면 안 된다. 달랑 두 네트워크만 서로 연결하면 되는 것이라면 그렇게 해도 된다. (이렇게 무작정 연결하는 것을 "브릿지"라고 부른다. 집으로 들어온 하나의 전화 회선에 여러 대의 전화기를 연결해서 쓰는 경우가 있(었)는데 그것도 브릿지다.)

예를 들어, 한 네트워크에서 평균 초당 10개의 패킷이 전송되고 대략 100 개가 전송될 때까지는 문제가 없고 그 이상이 되면 서로 충돌이 나서 쓸 수 없다고 하자. 두 네트워크를 브릿지하게 되면 이웃 네트워크의 패킷까지 나 우리 네트워크에 돌아다니니 초당 20개의 패킷이 보일 것이다. 이런 식으로 한 네트워크를 추가로 브릿지할 때 마다 돌아다니는 패킷의 수가 늘어나서 결국 네트워크를 10개쯤 붙이면 쓸 수 없게 될 것이다.

이 장면에서 문제의 원인을 분명히 해보자. 원인은 우리 네트워크에 꼭 전송되지 않아도 될 패킷까지 브릿지를 통하여 넘어온다는데 있다. (만약, 꼭 전송되어야 할 패킷의 총량이 우리 네트워크가 감당할 수 없을 정도로 많다면 다른 기술을 쓰던지 사용자들을 잡아서 족쳐야지 네트워크가 어떻게 해줄 수 있는 문제는 아니다.) 그럼 우리 네트워크에 꼭 전송되어야 할 패킷만 받으면 된다. 즉, 우리 네트워크에 연결된 컴퓨터의 목록을 알고 있다면 그 컴퓨터로 가야 할 패킷만 다른 네트워크에서 보내주면 된다.

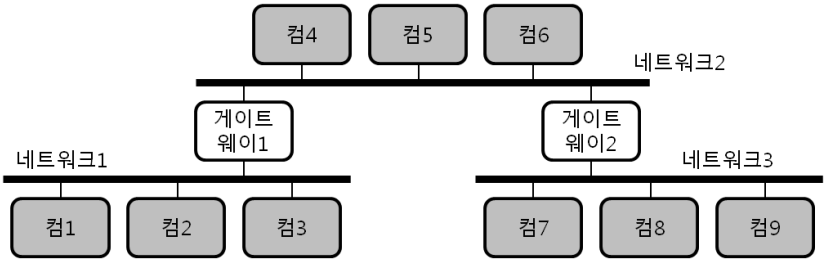


그림 12 네트워크가 셋으로 늘어나면

그림 12를 예로 든다면, 게이트웨이는 넷1에는 컴1,2,3, 넷2에는 컴4,5,6, 그리고 넷3에는 컴7,8,9가 연결되어 있다는 것을 ***어떻게든*** 안다고 생각해보자. 즉, 각 게이트웨이가 다음과 같은 표를 각자 갖고 있다고 생각해보자.

컴퓨터	소속 네트워크
컴1	넷1
컴2	넷1
컴3	넷1
컴4	넷2
컴5	넷2
컴6	넷2
컴7	넷3
……	…

만약 컴1이 컴5에게 보낼 데이터가 있어서 패킷을 쏘면 게이트웨이1이 컴5는 넷2에 있다는 것을 아니까 받아서 넷2에 다시 쏜다. 그럼 컴5가 받게 된다. 거기까지는 좋다. 그럼 이번에 컴2가 컴7에게 보낼 데이터가 있어서 패킷을 쐈다고 하자. 게이트웨이1은 이 패킷의 수신자가 넷1에 있지 않기 때문에 자기가 대신 받아서 넷3으로 전달해줘야 한다는 것까지는 알고 내가 넷1,2에 연결된 것 까지는 아는데 문제는 넷2에 그 패킷을 쏴도 넷3으로 전달된다는 것은 모른다는 점이다. 따라서, 이런 표를 추가로 갖고 있어야 한다.[16]

게이트웨이	연결 네트워크
게이트웨이1	넷1 -- 넷2
게이트웨이2	넷2 -- 넷3
……	…

그럼 이제 네트워크를 계속 연결하더라도 어떻게든 통신은 가능하다. 하지만, 실제로는 이렇게 만들면 거의 동작 안 한다. 여기서의 함정은 위에서 설명한 표를 만들 수 있다는 가정에 있다. 이렇게 해보자. 컴퓨터 한 대를 네트워크에 연결하면 그 사실을 세상의 모든 게이트웨이에게 알려주고 각 게이트웨이는 이 내용을 표에 기록한다. 마찬가지로 게이트웨이가 추가될 때에는 그 정보를 모두에게 알려야 한다.

지구 전체를 통틀어 본다면 새로운 컴퓨터를 네트워크에 추가하는 빈도가 얼마나 될까? 게이트웨이를 추가하거나 연결 상태를 바꾸는 일이 얼마나 될까? 어마어마하게 많을 것이다. 더군다나 요즘처럼 컴퓨터(라고 쓰고 스마트폰이라고 읽는다)가 이동이 가능한 상황에서 이들은 이동하면서 계속 다른 네트워크에 붙었다 떨어졌다 한다. 이 모든 정보를 어떻게 세상의 모든 게이트웨이에 알려줄 것인가? 어차피 이 정보는 전세계 모든 게이트웨이가 다 알아야 하므로 전세계 네트워크

[16] 네트워크가 단순한 선형 연결이 아니라 임의의 그래프가 되면 최적의 경로를 찾아내기 위한 알고리즘이 필요한데 이 설명은 생략한다.

전체에 다 보내야 한다.[17] 만약, 세상에 백 억대의 컴퓨터가 있고 한 네트워크에 대략 백 대가 연결된다면 네트워크랑 게이트웨이도 대략 1억 개쯤 있을 것이다. 따라서, 컴퓨터 하나가 추가되면 그 사실을 알리기 위해 최소 1억 개의 패킷이 전송되어야 한다. 이런 식으로 가다가는 컴퓨터 네트워크가 데이터를 전송하는 것이 아니라 네트워크 연결 상태 정보를 공유하는데 대부분의 시간을 보내게 될 판이다.[18] 이 문제는 어떻게 풀 수 있을까? 새로운 개념이 필요하다. 이를 위해 주소라는 개념을 도입해보자.

3.3 딴 이야기

주소 개념과 그 유용성을 이해하기 위해서 진짜 주소를 다루는 사례를 잠시 살펴보자.

요즘은 손 편지를 보낼 일이 많지 않아 택배 배달 아저씨로 변신하였지만 우편 집배원 아저씨들의 일상을 살펴보자. "집배원"이라는 단어에서 "집"은 모은다는 뜻이고 "배"는 나눈다(배달한다)는 뜻이다. 즉, 편지를 모아와서 배달해주는 사람들이다. 오산에서 부친 편지가 어떻게 마포의 어느 사무실로 가는지 과정을 보자.

편지를 오산시내 어딘가의 우체통에 넣으면 오산시 우체국의 집배원들이 와서 수거해간다. 수거하는 과정에서는 별다른 절차가 없다. 그냥 다 수거해서 오산시 우체국으로 가져간다. 오산시 우체국은 별다른 분류절차 필요 없이 자기가 속한 큰 우체국인 수원 우편집중국으로 보낸다. 수원 우편집중국에서는 주소에 따라(실은 주소를 처리하기 좋게 코드로 표현한 우편번호에 따라) 어느 우편집중국으로 보낼 것인지 분류하여 해당 우편집중국으로 보낸다. 마포의 경우에는 고양 우편집중국으로 보낸다. 고양 우편집중국은 한번 더 분류하여 마포로 가야 할 다른 편지와 함께 마포 우체국으로 보낸다. 마포 우체국에서는 동네 별로 분류하여 분류함에 넣어두면 각 동네를 담당하는 집배원들이 자기 동네 것을 가져다가 주소를 보면서 배달한다. 이 전달 방식이 잘 동작하는 이유는 첫째, 수거 과정에서는 일체의 정보가 필요 없다. (오산의 어느 우체통부터 수원 우편집중국까지는 일체의 판단 절차 없이 무조건 보낸다.) 둘째, 어디로 가야 하는 지 분류하고 전달하는 것은 소수의 전문적인 우체국(즉, 우편집중국에서 이뤄진다)에서 자동화된 방법으로 이뤄진다. 셋째, 최종 전달은 그 동네 사정을 잘 아는 사람이 한다.

사례를 하나 더 보자. 1960년대에 예일대학을 다녔던 프레드릭 스미스(Frederick W. Smith)는 경제학 수업 시간에 정보 기술을 이용한 당일 배송 시스템을 리포트로 낸 적이 있지만 C학점에 그친 바 있다. (현실성이 없었다나 어쨌다나……) 나중에 이 개념을 적용하여 배송 회사를 세운 것이 그

---

[17] 이런 식으로 네트워크 전체에 패킷을 다 보내는 것을 홍수(flood)라고 하며 네트워크의 효율을 떨어뜨리므로 특별한 이유가 없는 한 피해야 할 방법이다.

[18] 이 문제는 아니지만 다른 경우에도 이런 문제가 있어서 대부분의 네트워크 알고리즘(algorithm, 문제 처리 절차 또는 방법)은 전달할 정보가 발생하자마자 보내기 보다는 모아 두었다가 일정 주기가 되면 보내는 방식으로 하여 총 전송 패킷 수를 줄이는 수법을 쓴다. 물론, 이런 방식을 쓰게 되면 정보의 정확도는 희생될 수 밖에 없다. 정확도와 부담 사이의 균형이 (네트워크) 알고리즘 설계의 핵심이라 할 수 있다.

가 회장으로 있는 FedEx다. 이 개념의 핵심은 허브와 바퀴살이다. 자전거 바퀴를 유심히 보면 가운데 허브(hub)라는 원통이 있고 여기에 가느다란 바퀴살이 꿰어 방사형으로 뻗어 나간다. 즉, 택배 시스템도 이런 형태로 전달하는 것이 좋다는 것이 그의 생각 이었다. 그림 13에서 보는 바와 같이 각 도시에는 그 도시의 화물을 모아오거나 배달하는 기능을 두고 다른 도시로의 전달은 일단 허브인 덴버나 LA로 보낸 뒤 그곳에서 다른 도시로 전달하게 한다.

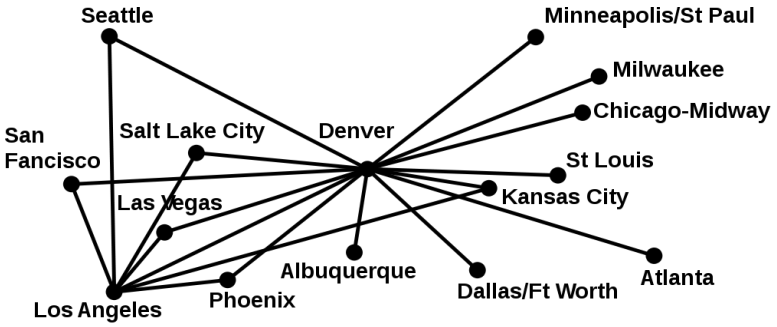


그림 13 공항의 연결 방식도 전형적으로 바퀴살 모양이다. 이 그림에서는 덴버와 LA가 허브 공항이고 여기에서 다른 모든 공항으로 연결된다. [19]

왜 이 방법이 더 좋은가? 첫째 모든 도시를 다 연결하려면 도시 수의 제곱에[20] 비례하는 연결편이 필요하다. 그 모든 연결편을 관리하는 것도 비용이 발생하지만 각 연결편이 모두 비행기 또는 트럭이 꽉 찬다는 보장이 없다. 둘째 화물의 분류, 요금 처리나 정확한 배송을 위한 주소확인 등 까다로운 처리는 허브에서만 하면 되고 다른 곳에서는 아무 생각 없이 일단 허브로 보내면 그만이다. 셋째, 새로운 도시에 바퀴살을 추가하는 것은 그 바퀴살이 연결되는 허브에만 영향을 미칠 뿐 다른 모든 도시에서는 알 필요가 없다. 이러한 장점 덕분에 FedEx는 단시간 내에 압도적인 지위를 차지하게 된다.

3.4 IP 주소의 탄생

이들 사례를 앞의 우리 문제와 연결해보자. 우리 시스템에서 게이트웨이는 집배원과 비슷한 역할을 해야 하는데 우리의 경우에는 이들 집배원이 전세계 지리 정보를 다 알고 있어야 한다는 것이 엄청난 부담이었다. 왜 이런 차이가 발생하는가? 주소라는 개념이 없기 때문이다.

우리 문제에서도 각 컴퓨터가 자기를 식별하기 위해서 MAC 주소를 쓴다. 하지만, 이 주소는 말이 주소지 그 녀석이 어디에 있는지는 알려주지 않는다. 따라서, MAC 주소는 주소라기 보다는 이름 또는 식별자(ID, identifier)라고 보는 것이 맞다. 다시 말해서 우리의 문제를 우편 배달 문제로 표현하자면 받을 사람 이름(또는 주민등록번호)가 달랑 적혀 있는 편지 봉투를 달고 배달을 해야 하는 것과 같다. 따라서, 모든 배달부가 모든 사람들의 이름과 있는 곳을 알아야 배달 가능한 것이었다. 따라서, MAC 주소 대신에 집 주소 또는 우편 번호처럼 어디로 보내야 될 지 ***힌트를 주는***

---

[19] 그림 출처. http://en.wikipedia.org/wiki/Spoke-hub_distribution_paradigm

[20] 쓸데 없이 정확하게 표현하자면 n(n-1) / 2 단, n 은 도시의 수.

주소를 써야 한다. 그런 주소를 IP 주소라고 한다.

IP 주소에서 IP란 인터넷 프로토콜(internet protocol)의[21] 줄임말이다. 인터넷은 inter – network 라는 뜻으로 여러 개의 네트워크를 연결한다는 뜻이다. 즉, MAC은 하나의 네트워크에서 쓰는 기술이고 IP 는 여러 네트워크를 연결할 때 쓰는 기술이다. 물론 실제 전달은 이 둘이 협력을 해야 전달이 되는 것이고.

그럼 IP 주소는 어떻게 생겼고 어떻게 붙이는가? IP 는 여러 번 개정되었고 현재 가장 널리 쓰는 것은 넷째(IPv4, IP version 4)와 여섯째(IPv6, IP version 6)다. 설명의 편의를 위해서 IPv4를 기준으로 설명하겠다. IPv4 주소는 4 바이트 숫자로 구성되며 대개는 사이에 점을 찍어서 표시한다. 192.168.0.1 이런 식으로 말이다. 한 바이트로는 0부터 255까지의 숫자를 표현할 수 있으므로 IPv4 주소는 0.0.0.0부터 255.255.255.255까지 총 43억 개가 있는 셈이다.[22] 이들 중 일부는 특수한 용도로 쓰므로 실제 컴퓨터에 나눠줄 수 있는 수는 좀 적다.

컴퓨터 한 대마다 IP 주소를 하나씩 나눠줘야 하는데 아무거나 막 주면 되나? 앞서 MAC 주소의 경우 공장에서 아예 박혀서 나오고 서로 고유한 번호로서 충돌만 나지 않으면 그만이었지만 IP 주소는 도입 목적이 있기 때문에 그 목적을 달성할 수 있게 나눠줘야 한다. 그 목적이란 무엇이었는가? 새 컴퓨터에게 새 IP 주소를 주더라도 전 지구적으로 이 정보를 알릴 필요 없어야 하며 그럼에도 불구하고 그 컴퓨터에게 패킷을 전달해야 할 때 어디로 보내야 하는지 알 수 있어야 한다. 그러려면 어떻게 해야 할까?

이 장면에서 우체국의 사례를 좀 더 살펴보자. 수원 우편집중국 산하에는 수원/화성/오산/평택 등이 속해있다. 고양 우편집중국 산하에는 서대문/마포/은평/고양 등이 속해있다. 앞의 예에서, 오산 우체국의 우편물이 수원 우편집중국을 거쳐 고양 우편집중국을 거쳐 마포우체국을 거쳐 마포의 어느 사무실로 전달되기 위해서 알아야 할 것은 다음과 같다.

- 오산 우체국

---

[21] 프로토콜이라고 하면 통신 프로토콜을 줄여서 하는 말이다. 통신 프로토콜이란 컴퓨터 통신에서 서로 어떤 순서로 어떤 내용을 어떤 형태로 주고 받을 지를 정해 놓은 것을 말한다. 원래 프로토콜은 (많은 서양 말이 다 그렇듯이)그리스에서 온 말이다. 고대 그리스에서는 파피루스 두루마리에 기록을 하였는데 파피루스 두루마리는 자꾸 펼쳐보면 문서가 쉽게 망가지므로 두루마리 겉에 종이를 붙이고 여기에 문서 날짜 등을 적어 두었다. 그래서 굳이 펴보지 않아도 무엇인지 알 수 있게 하였던 것이다. (책을 철한 쪽에 책 제목을 인쇄해서 책꽂이에 쭉 꽂힌 상태에서 굳이 펼쳐보지 않아도 책을 찾을 수 있는 것과 같은 원리다.) 그러던 것이 외교 문서에서 조약의 초안을 뜻하는 것으로 바뀌었다가 정식 절차를 진행하기 위해 미리 정한 절차(흔히 의전이라고 부르는 것)의 의미로 현재는 쓰이고 있다.

[22] 실제로 연결해야 할 컴퓨터의 수는 이것보다 훨씬 많기 때문에 IPv4로는 감당이 안되고 그래서 IPv6가 제안되었다. 하지만, 주소 부족을 극복하는(또는 회피하는) 방안도 없는 것이 아니라 지금 당장 고갈되었다고 말할 수는 없다.

    - 내가 속한 수원 우편집중국으로 가려면 어떻게 간다.
- 수원 우편집중국
    - 마포는 고양 우편집중국 산하에 있다.
    - 고양 우편집중국으로 가려면 어떻게 간다.
- 고양 우편집중국
    - 마포 우체국으로 가려면 어떻게 간다
- 마포 우체국
    - 그 사무실로 가려면 어떻게 간다.

이 과정을 요약해 보자면;

(1) 아래에서 위로 갈 때는 외길이므로 그 길만 알면 된다.
(2) 위에서 아래로 갈 때는 자기 산하에 속한 곳으로 가는 길만 알면 된다.[23]
(3) 더 이상 위로 넘기지 못하는 우편집중국은 다른 우편집중국으로 가는 길을 알아야 한다.
(4) 우편집중국은 다른 우편집중국 산하에 누가 있는지 알아야 한다.

그럼 이 방식을 IP 주소에도 적용해보자. IPv4의 주소가 네 칸으로 이뤄진다고 하니 첫째 칸은 우편집중국을 나타내고 둘째 칸은 우체국을 나타내고 셋째 칸은 한 우체국 관내의 동네를 나타내고 넷째 칸은 한 동네의 각각의 컴퓨터를 나타낸다고 해보자. 예를 들어, 수원 인근의 모든 컴퓨터는 주소 첫째 자리가 1로 시작하는 한편 고양 인근의 모든 컴퓨터는 2로 시작하는 것으로 하자. 또한 수원에 속하는 수원/화성/오산/평택 등은 각각 1.1./1.2/1.3/1.4...로 나타내고 고양에 속하는 서대문/마포/은평/고양 등은 각각 2.1/2.2/2.3/2.4/...로 나타내기로 하자. 그리고 마포에 있는 각 동네는 2.1.1/2.1.2/2.1.3/... 등으로 나타내고 마포의 첫째 동네의 첫째 컴퓨터에는 2.1.1.1의 주소를 줬다고 해보자. 그리고 우체국이나 우편집중국은 번호 0을 붙여보자.

	IP 주소 네 자리			
수원 우편집중국	1	0	0	0
수원 우체국	1	1	0	0
화성 우체국	1	2	0	0
오산 우체국	1	3	0	0
오산 첫째 동네 담당 집배원	1	3	1	0
...				
고양 우편집중국	2	0	0	0
서대문 우체국	2	1	0	0
마포 우체국	2	2	0	0
마포 첫째 동네 담당 집배원	2	2	1	0

[23] 수원 우편집중국은 마포 우체국 가는 길을 몰라도 된다.

마포 첫째 동네 첫째 컴퓨터	2	2	1	1
마포 첫째 동네 둘째 컴퓨터	2	2	1	2
…				

누군가 마포 첫째 동네 첫째 컴퓨터(2.2.1.1)에게 패킷을 보냈는데 오산 첫째 동네 담당 집배원이 이 패킷을 보고 "우리 동네로 가는 것이면 1.3.1로 시작해야 되는데 그게 아닌 걸 보내 딴 동네로 구나"라고 판단하여 곧바로 자신의 속한 우체국 즉 1.3.0.0으로 전송한다. (이 집배원은 자기 동네 것이 아니면 무조건 1.3.0.0으로 보내면 된다는 것을 알고 있다.) 오산 우체국은 이 패킷을 보고 "1.3으로 시작하지 않는 것을 보니 우리 우체국 관내가 아니구나"라고 판단하여 자신이 속한 우편집중국 즉, 1.0.0.0으로 보낸다. 수원 우편집중국에서는 "2로 시작하는 것을 보니 고양 우편집중국으로 보내야 되는구나"라고 판단하여 2.0.0.0으로 전송한다. 고양 우편집중국은 "우리 우체국 중 2.2로 시작하는 것은 마포 우체국"이라고 알고 있으므로 2.2.0.0으로 전송한다. 마포 우체국에서는 "2.2.1로 시작하는 것은 모두 첫째 동네다"라고 판단하여 첫째 동네 담당 집배원인 2.2.1.0에게 전달한다. 2.2.1.0은 이를 자기 동네 네트워크에 전달하고 해당 컴퓨터가 받게 된다.

얼른 생각하기에 이렇게 하면 될 것 같지만 실은 살짝 더 복잡하다. 위의 전달 과정을 실제 컴퓨터와 게이트웨이 그리고 라우터로[24] 표현해보면 그림 14과 같다. 이 그림에서 실제 컴퓨터는 연한 색칠이 된 1.1.1.1 등의 박스이고 그 외의 것들은 모두 네트워크의 상호 연결을 위한 게이트웨이 또는 라우터다. 각 박스에는 IP 주소가 씌어 있는데 컴퓨터와 달리 게이트웨이나 라우터는 IP 주소가 두 개씩 붙어 있다. 그 이유는 이들이 두 네트워크를 연결하기 위해서는 각각의 네트워크에 속하는 주소를 하나씩 갖고 있어야 하기 때문이다. (박쥐의 운명이다!) 만약 하나의 네트워크 장비가 세 개 이상의 네트워크를 서로 연결한다면 네트워크 장비에 그 개수만큼 네트워크 카드가 붙어야 하고 그 만큼의 주소를 할당해야 한다. 그런 경우가 왜 생기는지는 3.8장에서 다룬다.

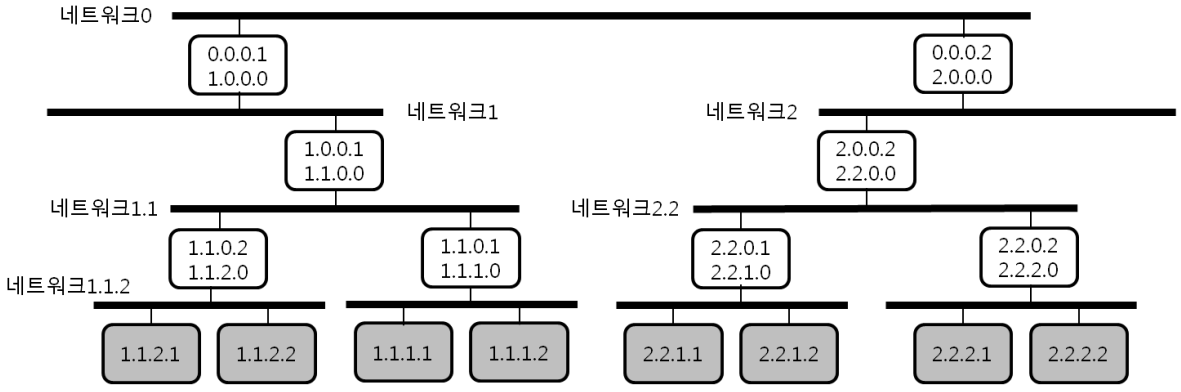


그림 14 무척 간단한 네트워크 구성 (흰 박스는 네트워크 장비)

[24] 라우터(router)는 게이트웨이보다 살짝 더 복잡한 장비라고 일단 생각해보자.

3.5 중간 결산 – 문제는 해결되었나?

이렇게 긴 이야기를 한 이유는 새로운 컴퓨터가 네트워크에 추가되었을 때 이 사실을 전세계에 다 알리지 않아도 되는 방법을 찾기 위함이었다. 그럼 그 목적은 달성되었는가? 우선, 어떤 컴퓨터(1.1.1.1)가 다른 컴퓨터(2.2.2.2)에게 패킷을 쏘았을 때 이를 제대로 전달하기 위해서 각 네트워크 장비가 알고 있어야 할 정보가 무엇인지 한번 더 정리해보자. (설명의 편의를 위하여 상당부분을 생략했다.)

1.1.1.0 장비: "1.1.1로 시작하는 것이 아니면 1.1.0.0에게 보낸다"
1.1.0.0 장비: "1.1로 시작하는 것이 아니면 1.0.0.0에게 보낸다"
1.0.0.0 장비: "2로 시작하면 0.0.0.2에게 보낸다"
2.0.0.0 장비: "2.2로 시작하면 2.0.0.2에게 보낸다"
2.2.0.0 장비: "2.2.2로 시작하면 2.2.0.2에게 보낸다"

이 상황에서 2.2.2로 시작하는 네트워크에 새 컴퓨터(2.2.2.3)이 추가되었다. 이때 이 사실을 아무에게도 알릴 필요가 없다. 여전히 위의 정보만 가지고도 1.1.1.1이 2.2.2.3에게 패킷을 보낼 수 있다. 즉, 새로운 컴퓨터를 추가한다고 해서 네트워크 장비가 이 사실은 알 필요가 없다. 왜냐하면 네트워크 장비가 알고 있는 정보는 개별 주소가 아니라 주소 블록(예를 들자면, 2.2.2으로 시작하는 모든 주소) 단위로 관리하므로 개별 컴퓨터의 추가는 신경 쓸 필요가 없다. 다만, 새로운 주소 블록이 추가되면 이를 알고 있어야 한다.

예를 들어, 위의 구조에서 3으로 시작하는 네트워크가 추가되었다면 1.0.0.0과 2.0.0.0 이 두 장비는 3으로 시작하는 주소를 전달하는 방법을 ***추가로*** 알아야 한다. 반면 그 아래의 네트워크 장비들 예컨대, 1.1.0.0은 이런 사실을 알 필요가 없다. 어차피 자기 위의 1.0.0.0이 알아서 할 거니까. 마찬가지로 만약, 1.1 네트워크 아래에 1.1.3으로 시작하는 네트워크를 추가했다고 했을 때 이를 알아야 할 장비는 1.1.0.0뿐이다. 이 하나의 장비를 제외하고는 전세계 어느 장비도 1.1.3으로 시작하는 네트워크가 추가되었다는 사실을 몰라도 패킷 전송에 아무런 문제가 없다.

그래서 애초의 문제는 해결되었다. 단, IP 주소 블록 할당을 체계적으로 잘 했다는 전제하에. 주소 블록 할당에 대해서는 다음 장에서 살펴보기로 하고 잠시 실용적인 얘기를 살펴보자. 요즘은 대부분의 환경에서 IP 주소 할당을 동적으로 하기 때문에 사용하는 사람들이 드물긴 하지만 정적으로 주소를 할당하는 경우에는 네트워크 설정을 위하여 알아야 할 정보로 (그림 10에서 보듯이) IP 주소와 서브넷 마스크, 그리고 게이트웨이 주소가 필수이다. (물론 DNS 서버 주소도 사실상 필수지만 이는 5.1장에서 더 알아보기로 하자. 또한 주소의 동적 할당과 정적 할당에 대해서는 3.6에서 다룬다.) 이제 이들의 의미를 분명히 이해할 수 있게 되었다.

우선 하나의 단위 네트워크(이를 전체 네트워크의 일부라는 의미에서 서브 네트워크라고 부른다)에 IP 주소 블록이 할당된다. 예를 들어, 1.1.1로 시작하는 모든 주소를 하나의 블록으로 그 서브 네트워크에 할당했다면 1.1.1.0~1.1.1.255 사이의 아무 주소나 쓸 수 있는 권한이 생겼다는 뜻이다.

그 중에서 ***아무거나*** 예를 들어, 1.1.1.10을 꺼내서 어떤 컴퓨터에[25] 할당했다고 하면 그것이 곧 그 컴퓨터의 IP 주소가 된다. 이 서브 네트워크의 모든 컴퓨터는 다 1.1.1으로 시작하므로 1.1.1로 시작하지 않는 모든 주소는 이 서브 네트워크의 ***밖에*** 있다. 예를 들어, 1.1.0.10은 비록 앞의 두 자리까지는 같아서 우리 서브 네트워크 소속이 아니다. 따라서, 몇 자리까지 같아야 우리 서브 네트워크인지를 나타내는 것이 서브넷 마스크다. 이 예에서처럼 앞의 세 자리까지 같으면 우리 네트워크라는 것을 나타내려면 255.255.255.0 이라고 쓰면 된다. 그리고 우리 네트워크 밖에 있는 컴퓨터에게 보낼 때에는 게이트웨이를 통해서 보내야 하므로 그 게이트웨이의 주소를 알고 있어야 한다. (어떻게 게이트웨이를 통해서 외부로 중계되는지에 대하여는 3.8장에서 설명한다.) 따라서, IP 주소, 서브넷 마스크, 게이트웨이 주소 이 세가지가 IP 네트워크 설정의 기본 중의 기본인 것이다.

3.6 IP 주소 블록의 할당은 어떻게 하나?

앞의 예에서 우리에게 1.1.1 블록을 할당해주는 것은 누굴까? 거의 대부분의 경우에 인터넷 서비스 사업자(ISP, Internet service provider)에게서 받는다. 그럼 ISP는 어디서 블록을 받아오나? 작은 ISP는 더 큰 ISP에게 받아오고 큰 ISP는 권역 별 주소 블록 할당을 담당하는 기구(RIR)에게서[26] 받아온다. 그럼 RIR은 어디서 받아오나? 궁극적으로는 IANA(Internet Assigned Numbers Authority)에게서[27] 받아온다. 즉, 0.0.0.0~255.255.255.255 사이의 주소를 적당한 크기를 블록으로 쪼개서 IANA → RIR → ISP → 실 사용자에게로 나눠주는 것이다. 물론 아주 초창기에는 이러한 체계 없이 사용자(초창기의 사용자라면 대개는 큰 대학들이나 연구소들)가 직접 IANA를 담당하는 존 포스텔에게 연락해서 받아가는 식이었겠지만 나중에는 이를 기구화하였으며 특히, IPv4 주소의 고갈 문제가 제기되면서부터는 나눠준 것을 다 썼다는 것이 확인되어야 추가로 할당하는 등 점차 짜게 나눠주고 있다. IANA를 기준으로 보면 2011년 2월 3일에 마지막으로 주소 블록을 할당함으로써 고갈되었다. 하지만 아직은 RIR에 할당되어 있는 것들이 고갈된 것이 아니므로 실제 IPv4 주소의 완전한 고갈은 아니다.

3.7 IP 패킷과 MAC 패킷 그리고 ARP

이 장면에서 헷갈리지 말아야 할 것. 전 지구적인 네트워크를 구축하고 운용하기 위해서 MAC 주소 대신 IP 주소를 도입했으니 그럼 MAC 주소는 안 쓰고 IP 주소만 쓰는 건가? 아니다. 둘 다 쓴다. ***어떻게*** 그리고 ***왜*** 둘 다 쓰는지 생각해보자. 먼저 뒤의 질문부터.

---

[25] 실은 더 정확히는 하나의 네트워크 카드에.

[26] 이러한 기구를 RIR(regional internet registry)이라고 부르며 아시아-태평양 권역의 경우에는 APNIC이 RIR의 역할을 하고 있다

[27] 인터넷의 전신인 아르파넷 시절부터 인터넷 주소와 포트 번호를 배분하는 역할은 주요 개발자 중 한 사람인 존 포스텔(Jon Postel)이 맡고 있었으며 그 역할을 IANA라고 불렀다. 처음에는 존 포스텔이 혼자 다 했을 것이고 그 이후로는 몇 명의 스태프를 두고 관리자의 역할을 했다. 1998년 급작스럽게 죽을 때까지 그는 IANA의 관리자인 동시에 (인터넷 표준 문서인) RFC 편집자의 역할을 계속하였다.

우선 실용적인 면에서 IP 주소 없이 동작해야 하는 프로토콜이 있으므로 IP 주소로 다 처리할 수 가없다. 예를 들어, DHCP(dynamic host configuration protocol)처럼 IP 주소를 할당하는 프로토콜이 동작하려면 IP 주소 없이도 특정 컴퓨터를 지정해서 통신할 수 있어야 하므로 MAC 주소는 필요하다.

하지만 더욱 근본적인 이유는 ***IP는 이 세상의 유일한 네트워크 프로토콜이 아니라는 점***이다. 컴퓨터 네트워크 기술이 처음으로 상업적으로 퍼져나가던 1980년대 초반 큰 컴퓨터 공급업체(예를 들어, IBM이나 DEC)는 물론이고 네트워크 전문업체(예를 들어, 3Com 이나 노벨)가 각기 자기 나름의 네트워크 기술 다시 말해서 서로 호환성 없는 네트워크 프로토콜을 사용하였다. 컴퓨터 공급자들이 제공했던 컴퓨터 네트워크 기술로 대표적인 것은 제록스의 XNS(Xerox Network Systems), DEC(Digital Equipment Corporation)사의 DECnet, IBM의 SNA(Systems Network Architecture) 등이었다. 당시의 컴퓨터 네트워크는 우리가 갖고 있는 컴퓨터가 어느 회사의 제품이냐에 따라 어느 네트워크 기술을 쓸 것이냐가 결정되는 상황이었다. IBM PC가 많이 팔리면서 이들을 시장으로 삼아 노벨사(Novell, Inc.)의 넷웨어(Netware)와 사이텍사(Sytek Inc.)의 넷바이오스(NetBIOS)가 1883년에 시장에 등장하였다. 그런 상황에서 네트워크 제품을 공급하는 회사들 관점에서 ***IP는 잘 봐줘야 성가신 부가기능 정도***였다.[28]

이렇게 경쟁하는 환경에서 당신이 새로운 네트워크 기술을 개발했다고 하자. 그리고 우리가 그 네트워크 기술을 판매하려는 시장에서는 이미 기존의 네트워크 기술이 있고 그 기술을 이용하기 위하여 회선이 다 설치되어 있다. 만약 당신의 기술이 기존의 회선에서 돌아가지 않는다면 이미 돈 들여서 설치한 회선을 다 들어내고 새로 회선을 설치해야 하는 부담이 있어서 시장에서 인기를 끌기 어려울 것이다. 따라서 새로운 네트워크 기술은 기존의 회선을 그대로 둔 채로 동작하도록 만들게 된다. 심지어, 처음 컴퓨터 네트워크 기술이 나왔을 때에도 새로운 회선을 설치한 것이 아니라 다른 네트워크(아마도 전화 네트워크)의 회선에 빌붙어서 시작했을 것이다. 이를 그림으로 표현해보자면 다음과 같다.

[28] http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet

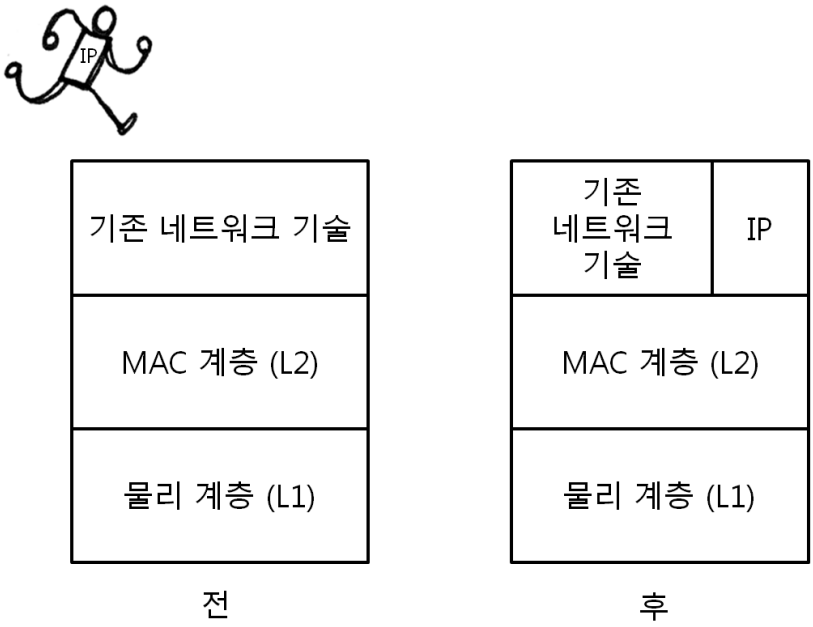


그림 15 네트워크 기술의 공존[29]

두 지점을 구리 선으로 연결할 지 유리 섬유를 쓸 지 심지어는 선을 안 쓰고 전파로 할 지를 고민하고 설치하는 문제(L1의 문제)와 이렇게 설치된 회선을 잘 활용하는 방법(L2의 문제)와 이들을 이용하여 전 세계에 흩어진 컴퓨터를 큰 부담 없이 찾아가게 만드는 방법(L3 또는 네트워크 계층 즉, L3의 문제)을 따로 해결함으로써 하나의 기술이 다른 기술을 제약하는 것도 피할 수 있고 각각 독립적으로 발전을 할 수 있는 것이다. 앞서 전 지구 규모의 네트워크를 작은 여러 개의 네트워크로 나눠서 네트워크의 네트워크 즉 인터넷(internet, inter-network)으로 쪼개 풀자는 아이디어가 ***수평적인 분할 정복***이라면 네트워크를 구성하기 위한 기술을 여러 다른 측면의 문제 즉, 여러 계층으로 나눠서 푸는 것은 ***수직적인 분할 정복***이라고 할 수 있다.

또한 이를 네트워크 산업이나 정책의 측면에서 본다면 각 계층별로 투자 주체가 분리될 수 있는 장점이 (또는 단점이) 있다. 예를 들어, 회선은 전화 회사가 설치하고 인터넷 서비스는 ISP가 전화 회사의 회선을 이용해서 할 수 있는 것이 이러한 수직적으로 분할된 기술 덕분이다.

이것으로 왜 MAC 주소와 IP 주소가 따로 있어야 하는지 설명이 되었다 치고 그럼 IP 주소와 MAC 주소가 어떻게 협력하는지 살펴보자. 앞서 MAC 주소를 설명하면서 데이터를 전송할 때에는 겉봉투에 주소를 쓰고 안에 편지를 넣듯이 앞에 MAC 주소를 쓰고 그 위에 보내려는 데이터를 달아서 보냄으로써 앞 부분을 보고 (즉, 겉봉투를 보고) 누구에게 전달되는 것인지 알 수 있다고 했다. IP 주소도 마찬가지다. 특정 IP 주소를 가진 상대에게 데이터를 전송하기 위해서는 맨 앞부분에 상대의 IP 주소를 적고 그 뒤에 보내려는 데이터를 달아야 한다. 그림으로 나타내면 다음과 같다.

---

[29] 달리는 사람 그림의 출처 http://dribbble.com/shots/58816-Running-Man

그림 16 데이터를 어디에 담아야 하나? MAC 패킷? IP 패킷?

우리는 IP 주소를 써서 데이터를 보낼 상대를 표현하고 찾아가기로 했으니 IP 패킷의 형태로 만들어서 네트워크에 쏴야 되는데 이걸 그대로 네트워크에 쏘면 어떻게 될까? 앞서 설명하였듯이 모든 네트워크 카드는 자기가 연결된 회선에 지나가는 모든 패킷 중에서 헤더 부분을 보고 MAC 주소가 자기 자신과 같은 것을 받고 그 외에는 버린다. 따라서 그림에서의 가상의 IP 패킷을 쏘면 그 누구도 받지 않는다. 따라서, 하는 수 없이 우리는 IP 패킷을 MAC 패킷으로 변신시켜서 보내야 한다. 그럼 MAC 주소에는 뭐라고 쓰나? 이제 우리는 상대 컴퓨터를 IP 주소로 표현하기 때문에 통신 하려는 상대의 IP 주소는 알지만 상대의 MAC 주소는 모른다. 따라서, IP 주소로부터 MAC 주소를 알아내는 방법이 필요하다. 이때 사용하는 프로토콜이 ARP(address resolution protocol)다.

상대의 MAC 주소를 모르기 때문에 ARP는 브로드캐스트(broadcast)라는 특별한 방법을 사용한다. 브로드캐스트는 영어 단어 그대로 방송이다. 아무나 다 듣는다는 얘기다. MAC 주소 중 FF:FF:FF:FF:FF:FF는 특별한 주소로서 상대방의 주소를 이것으로 쓰면 같은 서브 네트워크에 연결된 모든 컴퓨터가 받는다.[30] 상대방의 MAC 주소를 알아내려는 컴퓨터는 ARP 요청 패킷을 보내는데 그 패킷의 MAC 주소는 브로드캐스트 주소이며 데이터에는 이렇게 쓴다 "IP 주소 X.X.X.X는 누구에요?". 그럼 그 IP 주소를 가진 컴퓨터가 요청을 보냈던 컴퓨터에게 "IP 주소 X.X.X.X의 MAC 주소는 YY:YY:YY:YY:YY:YY 입니다"라고 답변을 보낸다. 요청과 달린 답변을 보낼 때는 브로드캐스트 주소를 사용하지 않는다. 왜냐하면 요청을 보낼 때 실은 요청하는 컴퓨터의 MAC 주소도 따라가기 때문에 답변을 보내는 컴퓨터는 요청한 컴퓨터의 MAC 주소를 알 수 있기 때문이다. 이렇게 ***브로드캐스트를 이용한 요청 → 유니캐스트를 이용한 답변*** 패턴은 컴퓨터 네트워크 기술에서 아주 흔히 사용되고 있다.

[30] 미안하다. 앞에서 MAC 주소가 자기 자신인 경우에만 받는다고 한 것은 설명의 편의를 위한 것이었다. 그리고, 앞에서 프로미스큐어스 모드로 패킷을 수신하는 것은 이것과는 다른 얘기다. 일반적으로 MAC 주소가 자기 자신이거나 브로드캐스트인 경우에만 수신하는데 프로미스큐어스 모드가 되면 그렇지 않는 경우에도 받는다는 뜻이다.

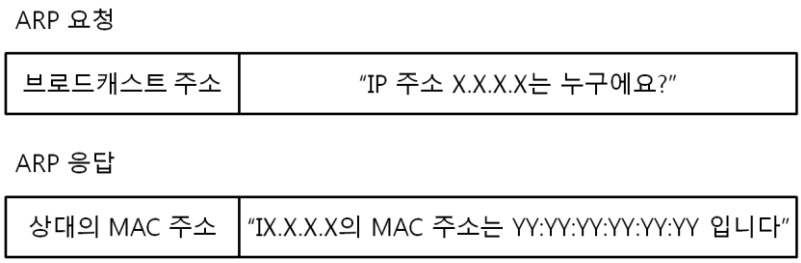


그림 17 (지나치게 간략하게 그려본) ARP 요청과 응답

그런데 앞서 설명한대로 홍수를 피하기 위해서 ***브로드캐스트는 전달되지 않는다***. ARP는 같은 네트워크에 붙어 있는 컴퓨터끼리 MAC 주소를 알아내는 것에 그친다. (하긴, 다른 네트워크에 붙어 있는 컴퓨터의 MAC 주소는 알 필요가 없다.) 그럼 다른 네트워크에 붙어 있는 상대에게 패킷을 보내려면 MAC 주소에는 뭐라고 써야 하나?

이 장면에서 다른 예를 들어 답을 찾아보자. 큰 회사에서는 우편물이 아주 많이 온다. 예를 들어, 회사가 본사와 여러 지사로 나뉘어 있다고 하자. 그럼 지사로 가야 할 상당수의 우편물이 본사에 온다. 그럼 그 우편물을 지사의 실제 수취인에게 보내야 하는데 이럴 때는 따로 보내는 대신 큰 자루에 담아서 보낸다. 그 자루에 겉에는 'XX 지사 우편 담당자 앞'라고 씌어 있다. 자루를 받은 우편 담당자는 실제 수취인에게 전달한다. (지사마저 무척 크다면 다시 부서별로 분류하여 하나로 묶은 뒤 부서별 담당자에게 묶음째로 전달할 것이다.) 요약하자면, 내가 보내려는 편지를 담은 봉투에는 실제 수취인의 주소가 적히지만 이런 편지를 담은 ***자루에는 중계해주는 사람들(즉, '담당자들')의 주소가 적힌다***.

이를 우리 상황에 대입해보자면 게이트웨이의 MAC 주소를 적어주면 된다는 얘기다. 물론, IP 네트워크 설정에서는 게이트웨이의 IP 주소만 입력하므로 ARP를 이용해서 MAC 주소를 알아내야 한다. 결국, MAC 패킷은 다음과 같은 모양이 된다.

그림 18 패킷 속의 패킷

그림에서 "상대의 MAC 주소" 부분은 같은 네트워크 내에서는 데이터를 실제로 받을 상대의 MAC 주소이지만 다른 네트워크로 보내야 하는 경우에는 이 자리에 게이트웨이의 MAC 주소가 들어간다. 게이트웨이가 이런 패킷을 받으면 어떤 일이 벌어질까?

3.8 IP 라우팅

게이트웨이가 자신의 MAC 주소로 보낸 패킷을 받아서 턱 까보니 그 안에는 IP 패킷이 들어 있다. 앞에서 살펴본 대로 게이트웨이는 목적지 IP 주소를 보면 그것을 어느 네트워크로 보내야 하는지

알고 있다. 앞의 그림 14에서 1.1.1.1이 2.2.2.2에게 보내려고 한다면 같은 네트워크가 아니므로 게이트웨이인 1.1.1.0의 MAC 주소를 알아내서 보냈을 것이다. 그럼 1.1.1.0과 1.1.0.1 두 개의 네트워크 카드를 가진 게이트웨이는 "내가 모르는 것은 위로 보낸다"라는 원칙에 따라 자기 위의 네트워크 즉, 네트워크 1.1의 게이트웨이인 1.1.0.0에게 자기가 받은 패킷을 ***전달***한다. 전달할 때는 다음 그림처럼 내용 부분은 복사하고 패킷의 헤더부분만 바꿔서 보낸다. 한 가지 더 알고 넘어가야 할 부분은 그 게이트웨이가 패킷을 받은 네트워크 카드(IP 주소 1.1.1.0)와 전달하기 위해서 내보내는 네트워크 카드(IP 주소 1.1.0.1)는 다르다. 말하자면 왼쪽에서 받아서 오른쪽으로 넘기고 오른쪽에서 받아서는 왼쪽으로 넘기고 하는 식으로 패킷은 네트워크에 흩어진 각종 장비를 통과하여 목적지로 날아간다.

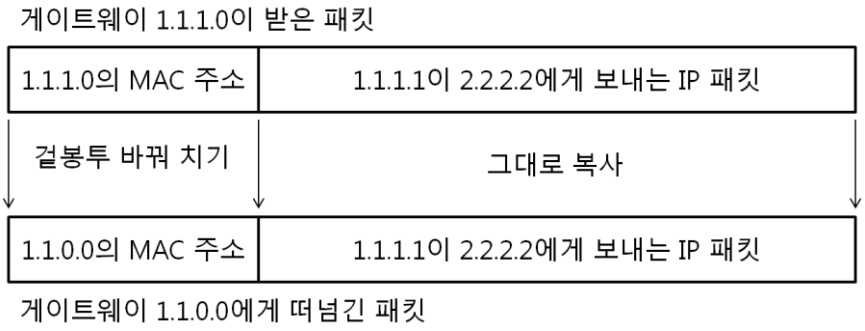


그림 19 패킷의 전달

이 과정을 기업이나 군대 같은 조직의 의사 전달 체계와 비교해서 설명해보자. 다른 부서의 누군가에게 할 말이 있으면 상관에게 보고를 하고 이 보고는 사장에게 전달된다. 사장은 전달 받을 부서장을 불러서 얘기를 하고 부서장은 해당 부서원에게 그 내용을 전달한다. 이 방법의 장점은 전달 체계가 단순하여 전달을 하기 위해서 ***알아야 할 정보가 최소화*** 된다는 점이다. 다만 단점이 있다면 의사 소통에 시간이 오래 걸리고 사장님은 귀가 아프고 입이 부르틀 것이며 혹시라도 외근을 나가는 날이면 부서간 의사 소통은 불가능해진다. 이렇게 기이한 전달 체계를 가진 곳은 거의 없을 것이며 실은 많은 정보가 실무자 선에서 직통으로 교환된다. "*혹시 저쪽 부서에 입사 동기나 아는 사람 없어?*" 뭐 이런 식으로 말이다. IP 네트워크도 마찬가지다. 큰 골격은 계층 구조를 따르지만 사이 사이에 지름길이 뚫려있다.

아래의 그림은 인터넷의 계층적 구성을 보여주고 있다. 가정이나 기업의 네트워크는 전화망, 케이블 TV망, 광케이블 등을 통하여 동네 ISP(Tier 3)에 연결되고 이들 ISP는 전국 규모의 ISP(Tier 2)에 연결되며[31] 이들은 다시 인터넷의 핵심부를 이루고 있는 최종 보스 ISP(Tier 1)로 연결된다. 어떻게 이 구조가 만들어지고 유지되는지를 이해하는 한 방법은 돈의 흐름을 보는 것이다.

---

[31] 우리 나라는 땅이 좁기도 하거니와 대기업들이 골목 상권 침투하기를 즐겨서 그런지 동네 ISP가 따로 없이 대부분 전국 규모의 ISP에 바로 가입되어 있다. 골목 상권을 보호하기 위해서 대기업의 동네 ISP 진출을 막으면 좋을라나?

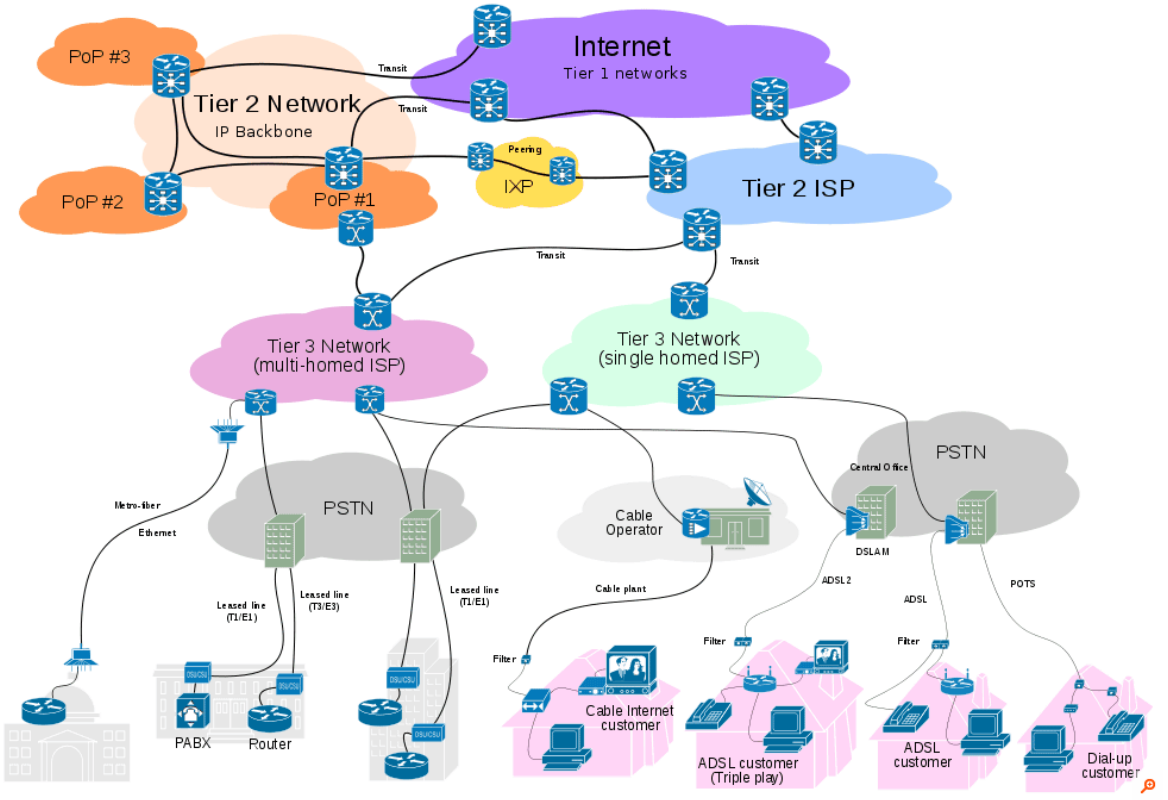


그림 20 인터넷의 연결 모습[32]

인터넷을 접속하기 위해서 돈을 내는 사람들은 가정이나 기업의 이용자들이다. 이들에게 돈을 걷는 곳은 동네 ISP들이다. 그런데 이들이 제공하는 인터넷 서비스가 제대로 동작하기 위해서는 다른 동네 ISP에 연결된 컴퓨터와 통신이 되어야 한다. 따라서, 다른 동네 ISP와 연결하기 위해서 큰 ISP에 가입한다. 그래서 큰 ISP는 동네 ISP에서 보내오는 패킷을 서로 전달해주는 서비스를 해주는 대신 돈을 받는다. 하지만 큰 ISP도 전 세계 모든 곳으로 다 연결되지는 않으니 더 큰 ISP에 가입한다. 이런 식으로 쭉 올라가다가 더 이상 올라갈 수 없는 곳까지 올라가면 거기에는 최종 보스 ISP인 Tier 1 ISP가 있으며 이들은 서로 돈을 주고 받지 않는다. (위/아래 개념이 없으니까.) 따라서, Tier 1 ISP를 제외하고는 모두 상위 ISP에게 돈을 내고 인터넷 서비스를 받고 있는 것이다. 그런데 만약 어떤 큰 ISP끼리 주고 받을 패킷이 아주 많다면 둘 사이에 회선을 직접 연결함으로써 상위 ISP에 주는 요금을 아낄 수 있을 것이다. 이렇게 ISP끼리 상호 연결하는 것을 IX(internet exchange 또는 IXP, internet exchange point)라고 부르며 앞의 그림에서도 Tier 2 ISP는 IXP를 통하여 서로 연결되어 있어서 굳이 Tier 1을 거치지 않아도 서로 패킷을 교환할 수 있다.

꼭 ISP 간에만 이렇게 연결되는 것은 아니고 예를 들어, 한 ISP 내부에도 많은 네트워크가 있을 것이고 이들 중 일부는 계층적으로 일부는 옆으로 뚫려 있는 구조가 될 것이다. 왜냐하면 계층적

---

32 그림 출처. http://arstechnica.com/security/2013/04/can-a-ddos-break-the-internet-sure-just-not-all-of-it/

으로만 구성한 경우 앞서 귀 아프고 입 부르튼 사장님의 경우처럼 일부 상층부의 네트워크 장비가 큰 부담을 지게 되고 데이터 전송도 더 많이 지연될 수 있기 때문이다. 그런데 이렇게 옆으로 샛길을 뚫기 시작하면 전에는 없던 문제가 생긴다.

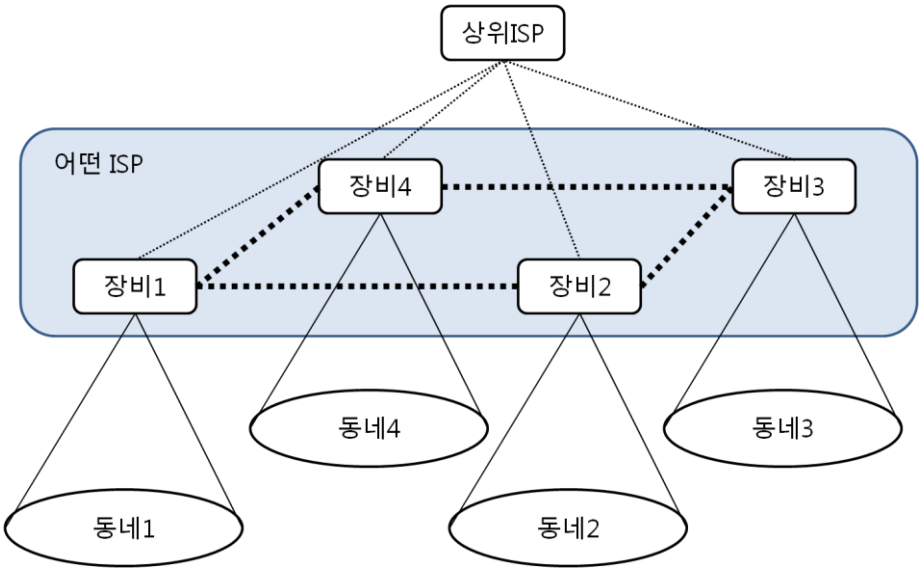


그림 21 네트워크 장비끼리 수평으로 연결한 모습

예전 같으면 동네1에서 동네2로 보낼 것이 있으면 그냥 장비1이 받아서 상위 ISP로 쏘면 상위 ISP에서 장비2로 보내줘서 해결이 되었는데 이제는 그 대신 장비1에서 상위 ISP를 거치지 않고 장비2로 바로 보낼 수 있다는 것을 ***추가로*** 알고 있어야 한다. 그리고 바로 보낼 때 더 좋은 경로가 [장비1→장비2] 인지 아니면 [장비1→장비4→장비3→장비2] 인지 알아야 한다. (심지어는 [장비1→장비4→상위 ISP→장비2] 가 더 좋은 경로일 수도?) 얼른 보기에는 여러 칸 건너는 것 보다 한 칸 건너는 것이 더 좋아 보이겠지만 만약 그 한 칸이 인공위성을 경유해서 연결된 것이고 다른 길은 그냥 짧은 광 섬유로 연결되어 있다면 당연히 다른 길이 더 좋은 길이다. 여기에 가끔 이들 연결 중 일부가 끊기기도 해서 우회해야 한다거나 각 칸을 지날 때 발생하는 비용(특히 상위 ISP 연결하는 비용)이 다 다르고 심지어는 시시 각각 가격이 달라진다면 어느 길이 더 좋으냐는 갑자기 몹시 어려운 문제가 되어버린다.

그렇다고 답이 없는 것은 아니고 다 방법이 있다. 여러 조건을 고려하여 최적의 경로를 찾아내서 패킷을 전송하는 것을 라우팅(routing)이라고 부르며 라우팅을 위해 각종 네트워크 장비가 서로 정보(예를 들어, 연결 상태나 연결 비용 등)를 교환하는 프로토콜을 라우팅 프로토콜이라고 한다. 또한 이렇게 서로 연결된 네트워크에서 적절한 경로를 찾아서 패킷을 전송해주는 네트워크 장비를 라우터(router)라고 부른다.

이상으로서 인터넷은 완성되었다. 이제 우리는 지구 규모의 네트워크를 구성할 수 있는 기술을 만들었다. 하지만 막상 써보니 생각지 않았던 문제가 줄줄이 생기기 시작했다. 다음 장부터 다루는 기술은 단지 연결을 하는 것을 너머 연결을 통하여 각종 서비스가 잘 동작하기 위해서 필요한 것들이다.

4 TCP와 UDP – 전송 계층 이야기

전송 계층(Transport layer)는 앞 장에서 다른 인터넷 계층(L3)을 이용하여 실제 데이터를 보내는 방법을 다루며 L3 바로 위의 계층이라는 의미에서 4계층(Layer 4, L4)라고도 부른다.

4.1 굵고 짧게 vs. 가늘고 길게

다음 그림은 어떤 이동통신 회사의 광고 화면이다. 물놀이 공원에서 미끄럼틀이 하나가 아니라 여럿이라 동시에 타니 빠른 것에 빗대어 ***속도가 빠른*** 데이터 통신을 표현하였다.

그림 22 길이 여럿이면 (넓어지면) 빠름 빠름 빠름[33]

이 광고는 컴퓨터 네트워크의 아주 중요한 특징을 잘 나타내고 있다. 하나의 회선을 통해 전송되는 패킷의 ***속도는 늘 똑같다***. 우리가 알고 있는 우주에서 빛보다 빠른 것은 없다. 따라서, 회선을 통해 전송되는 패킷 속도의 최댓값은 빛의 속도지만 회선의 특성에 따라 느려져서 대개의 회선에서는 빛의 속도의 60% 정도가 된다. (이 글에서는 그냥 생각하기 편하게 빛의 속도라고 가정한다.) 다시 말해 패킷을 특별히 더 빨리 보내는 방법 따위는 없다.

그러므로 길을 넓혀서 한번에 보내는 양을 늘리는 수 밖에 없다. 즉, 1초에 한 사람씩 미끄럼을 탈 수 있다면 미끄럼이 셋 있으면 1초에 세 사람씩 출발할 수 있으니 더 빠르게 느껴지는 것이다. 즉 회선의 굵기를 더 굵게 하여 동시에 지날 수 있는 양을 늘리는 것이다.[34] 하나의 패킷이 서로 다른 굵기의 회선을 지나는 것을 우리가 이해하기 쉽게 딱딱한 물체의 형상으로 그려본다면 다음

---

[33] 그림 출처. http://blog.naver.com/PostView.nhn?blogId=702story&logNo=100160513178

[34] 여기서 굵게 한다는 것은 무선 통신의 경우 할당된 주파수 대역을 넓게 한다든지 또는 같은 넓이의 주파수 대역, 구리선 또는 광 케이블이라고 해도 정보를 코드화 하는 방식을 개선(즉, 더 복잡하게)해서 더 많이 보낼 수 있게 하는 등을 말한다. 즉, 물리적으로 넓히는 것과 논리적으로 넓히는 것을 모두 포함한다.

과 같다. 그림에서 큰 상자 안에 작은 상자가 있는데 작은 상자는 크기 비교의 편의를 위한 것이고 큰 상자가 전송된는 패킷이라고 생각하자. (실제 패킷은 전자기파의 형태로 전송되므로 이렇게 생긴 네모 상자가 날아다니는 것은 아니다.)

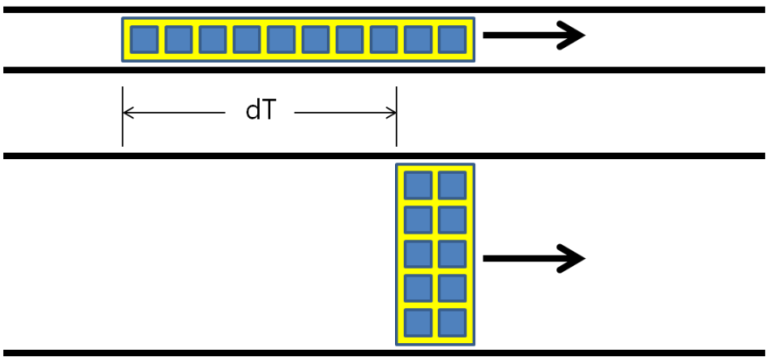


그림 23 다른 너비의 회선에서는 패킷의 길이가 달라진다

그림에서 보듯이 패킷의 맨 앞부분이 ***상대에게 도착하는 시점은 똑같다***. 단지 달라지는 것은 패킷을 다 받을 때까지 기다려야 하는 시간뿐이다. (그 차이를 그림에서는 **dT**라고 표시하였다.) 그럼 그 차이는 얼마나 될까? 예를 들어, 100Mbps(mega bit per second)의 회선을 생각해보자. 그럼 초당 1억(=100,000,000) 비트가 지날 수 있다는 것이다. 즉, 1억 비트 길이의 패킷을 이 회선을 통하여 전송하면 첫째 비트가 도착하고 1초 후에 마지막 비트가 도착한다. 이를 위의 그림과 같이 회선에서 차지하는 공간을 시간 단위로 표현하자면 가로 길이가 1초인 상자가 통과한다고 생각하면 된다. 만약 한 패킷이 1000 비트라면 길이가 1/100,000 초가 된다. 만약 이 보다 10배 빠른 (10배 더 넓은) 회선이 있다고 하면 패킷이 10배로 뚱뚱해지면서 길이는 1/10로 줄어들어서 차지하는 공간은 1/1,000,000 초가 될 것이다. 그럼 이 경우에 dT는 9/1,000,000 초가 된다. 물론 컴퓨터에게는 무척 긴 시간이겠지만 사람은 지금 바로 온 데이터와 백만분의 9초 뒤에 온 데이터는 구분할 수도 없다.

그럼 그렇게 차이가 없는데 왜 돈을 더 내고 더 빠른 네트워크를 사는가? 그건 보내려는 데이터가 많기 때문이다. 예를 들어, 토렌트에서 800 메가바이트의 영화 한편을 받는다고 하면 1000 비트 짜리 패킷을 아주 많이 줄줄이 보낼 것이고 각각의 패킷에서 발생한 dT가 누적이 되는 것이다. 그래서, 3G에서 7분 30초에 다운로드 받을 것을 LTE에서는 1분 30초에 받을 수 있게 되는 것이다.[35] 자 그럼 실제 큰 파일을 인터넷으로 전송해보자. 어떤 일이 벌어질까?

4.2 병목에서는 무슨 일이?

서로 멀리 떨어진 두 컴퓨터가 인터넷을 통하여 파일을 주고 받는다면 그 사이에는 여러 네트워크 회선을 거치게 된다. 그 회선 중에는 넓은 회선도 있고 좁은 회선도 있을 것이다. 또는 넓지만 붐벼서 사실상 좁은 회선처럼 통과하기 힘든 구간도 있을 것이다. 그럼 패킷이 네트워크를 날아가다가 이런 병목을 만나면 어떤 일이 생기는 걸까? 최대한 간단히 그림으로 그려보면 다음과 같

[35] 물론 이론적인 수치가 그렇다는 것이고 실제로는 여러 가지 다른 요인이 작용한다.

다. 네트워크A에서 병목 구간인 네트워크B로 들어갈 때 오던 속도 그대로 밀어 넣을 수 없으므로 앞 패킷이 다 들어갈 때까지 뒤 패킷은 ***버퍼에서 대기***한다. (고속도로에서 정체 구간 지나는 것과 완전히 같은 현상) 이때, 네트워크 B를 통과하는 모습을 보면 패킷이 꽉 차서 통과하는 상태가 된다. 병목 구간을 지나면 네트워크C는 빨리 통과시키고 싶겠지만 앞 패킷의 꼬리가 다 올 때까지 기다렸다가 보내고 또 뒤 패킷의 꼬리가 다 올 때까지 기렸다 보내는 식이 된다. 따라서 이 장면에서 패킷 사이의 간격이 벌어지게 된다. 그래서 아래의 그림처럼 네트워크C에서는 패킷 사이에 "최소한" 병목구간(네트워크 B)에서 한 패킷이 차지한 길이만큼의 빈 공간이 생기게 된다.

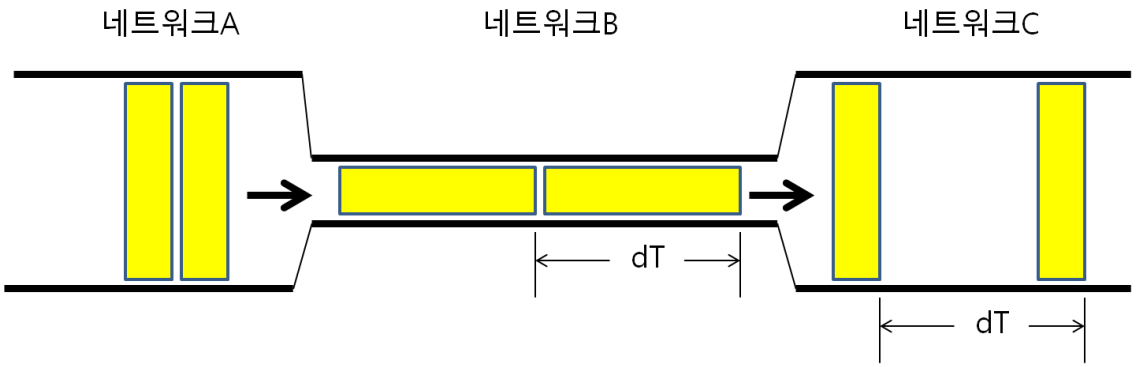


그림 24 병목을 지나면 패킷의 간격이 넓어진다

따라서 네트워크 회선이 다 좋고 하나가 병목이면 그 병목에 의하여 전송 시간이 결정된다. 흡사 가장 약한 고리의 강도가 체인의 강도를 결정하는 것과 같은 원리다. 하지만 병목이 끼치는 영향이 단지 시간 지연뿐이라면 차라리 다행이다. 더 심각한 것이 있다. 즉, 패킷 손실이 발생한다. 앞의 설명에서 병목 구간에 이르면 통과시킬 수 있는 속도보다 더 빨리 패킷이 도달하기 때문에 버퍼에 쌓아둔다고 했다. (그래서 장사가 안 되는 미용실에도 꼭 손님은 한 두 사람 대기하기 마련이다.) 만약, 쉴 틈 없이 연속으로 계속 패킷이 도착한다면 버퍼에 쌓이는 패킷도 늘어날 것이고 언젠가는 버퍼가 넘치게 될 것이다.[36] 버퍼가 넘치기 시작하면 당연히 그 이후에 도착하는 패킷은 ***버려진다***. 하나의 파일을 여러 조각으로 나눠 패킷에 담아서 보냈는데 그 중 한 둘이 없어진다면 받는 쪽에서는 원본과 다른 파일을 받게 되므로 데이터 통신으로서는 빵점이다. 그럼 어떡하나? 패킷을 제대로 받았음을 확인하는 추가 절차를 만들어야 한다. IP 통신 즉, Layer 3이 두 컴퓨터 사이의 연결을 해주었다면 그 위에 한 계층을 더 올려서 Layer 4에서는 제대로 전송되었음을 확인하는 기능을 하게 해보자.

4.3 ARQ

일반 사회에서는 잘 안 하고 군대에서만 볼 수 있는 것으로 복명복창이라는 것이 있다. 윗사람이 지시한 내용을 알아들었다는 것을 확인하기 위해서 지시 내용을 그대로 따라 해서 말하게 한다.

---

[36] 설령 넘치지 않게 버퍼를 무한대로 길게 해도 문제는 해결되지 않는다. 지금 접속한 웹 페이지가 "비록 한 달이 걸리건 두 달이 걸리건 지연은 있겠지만 언젠가는 화면에 뜹니다"라는 게 무슨 위안이 되겠는가?

같은 방법을 데이터 네트워크에 적용해보자. 하나의 파일을 전송하기 좋게 100 조각으로 나눴다고 하자. 그리고 각 조각을 봉투에 담고 겉에 1, 2, … 이런 식으로 번호를 붙인다. 1번 봉투를 IP 패킷으로 만들어 전송한다. 한편, 받는 쪽에서는 패킷이 도착하면 겉봉투에 쓰인 번호(1번)를 조그만 패킷에 담아서 답장으로 보낸다. 말하자면 "옜다 1번 패킷" 하고 보내면 "1번 받았음" 이라고 답을 보내는 것이다. 이렇게 전송한 것을 받았다고 통지하는 것을 접수통지(ACK, acknowledgement)라고 부르며 그런 답장 패킷을 ACK 패킷이라고 한다. 한편, 보낼 데이터를 담은 패킷은 DATA 패킷이라고 부른다. 1번 DATA 패킷에 대해 ACK이 오면 보내는 쪽에서는 2번 DATA 패킷을 보내고 2번에 답이 오고… 이런 식으로 쭉 보내면 된다. 일정 시간을 기다려도 답이 안 오면 패킷이 손실되었다고 보고 다시 보내면 된다.

이 과정을 다르게 설명해보자면 ACK은 "다음 DATA 패킷을 보내달라"는 요청이라고 생각할 수도 있다. 따라서, 이러한 방식을 자동반복요청(ARQ, automatic repeat request)라고 부르며 ARQ 중 가장 간단한 방법은 위에서 설명한 것처럼 답이 올 때까지 기다렸다 보내는 *멈춰 기다리기 ARQ*(stop-and-wait ARQ)다. 그런데 이 방법은 치명적인 단점이 있다. 너무 느리다. 간단히 예를 들어 보자.

칠레 산티아고의 노친에게 한국에 유학 온 아들이 사진을 보내려고 한다고 하자. 앞서 얘기한대로 빛은 너무 느리고 게다가 패킷의 전송 속도는 이 보다 더 느리므로 한국에서 보낸 패킷이 칠레에 도착하는 데는 짧게 잡아도 대략 100 밀리초(100 msec = 0.1초)가 걸린다. 이 시간은 초고속 광랜을 쓰건 뭘 쓰건 더 단축이 안 된다. 자연의 법칙이니까. 사진을 100 개의 패킷으로 쪼갰다고 하면 1번 DATA 패킷을 보내는데 0.1초 1번 ACK 패킷을 받는데 또 0.1초 2번 패킷을 보내는데 0.1, ACK을 받는데 또 0.1, … 이걸 100번 반복하면 20초가 걸린다. 물론 그나마도 패킷 손실이 한 번도 없을 때 얘기다.

패킷 손실이 있다면 다시 보내야 하는데 패킷 손실이 되었다는 것을 얼마나 빨리 알 수 있느냐가 문제다. 데이터를 보내는데 0.1초 답이 오는데 0.1초 이므로 대략 0.2초 후에 답이 오겠지만 실제 네트워크에서는 전송 지연이 들쭉날쭉해서 0.2보다 더 걸려서 답이 오는 경우도 있을 것이다. 그래서 0.2초 보다 더 길게 기다려줘야 된다. (안 그러고 성급하게 판단하여 같은 패킷을 또 보낸다면 인터넷은 아마 쓸모 없이 중복된 패킷으로 난장판이 될 것이다.) 즉, 네트워크 환경에서는 패킷의 손실이 생겼을 때 이를 알아내는데 긴 시간이 걸린다. 따라서, 전체의 전송 시간은 패킷 손실이 얼마나 발생하느냐에 따라 급격히 늘어나게 된다. 따라서, 20초보다 훨씬 더 길어질 것이다.

만약, 패킷 손실도 없고 ACK을 기다리지 않아도 되어서 그냥 줄줄이 보내서 성공한다면 시간이 얼마나 걸릴까? 만약 (병목)회선의 전송 속도가 10Mbps라고 해보면 1 메가바이트 사진 한잔 받는데 1초밖에 안 걸린다. 즉, ACK을 써서 확인하려는 순간 수십 배로 전송 시간이 길어져 버리는 것이다. 따라서, 빠르게 파일을 잘 보내려면 다음과 같은 점에 주의해야 할 것이다.

- ✓ 요구사항 1: 어떻게든 줄줄이 보내야 한다
- ✓ 요구사항 2: 패킷 손실은 최대한 회피해야 한다

위와 같은 주의 사항을 잘 고려하여 ARQ를 개선해서 만든 것이 TCP다.

4.4 TCP

줄줄이 보내려면 궁금함을 참아야 한다. 내가 보낸 패킷이 잘 갔는지 확인하지 말고 그 다음 것을 보낸다. 계속 보내다 보면 언젠가 아까 보낸 것에 대한 응답으로 ACK이 올 것이다. 만약 줄줄이 보내는데 아무리 기다려도 ACK이 안온다면 회선이 끊어졌거나 상대방 컴퓨터가 꺼졌거나 무슨 일이 있는 것이니 그만 보내야 한다. 그럼 얼마나 기다릴까? 그건 뭐 ***상황에 따라 잘 정하면*** 될 일이다. 예를 들어, 패킷 5개까지는 상대가 ACK을 안 보내도 보내고 그 다음부터는 ACK이 와야 보낸다고 하면 다음과 같이 패킷 교환이 이뤄진다. 참고로, 이때 참고 기다리는 구간을 윈도(window)라고 하며 5를 윈도의 크기라 부른다. 다음 그림에서 까만 화살표는 DATA 패킷의 전송을 빨간 화살표는 ACK 패킷의 전송을 나타낸다. 보내는 쪽에서는 윈도의 크기만큼은 일단 연속으로 보낸 다음 1번 패킷에 대한 ACK이 오면 그제서야 다음 DATA 패킷(6번 DATA 패킷)을 보내게 된다.

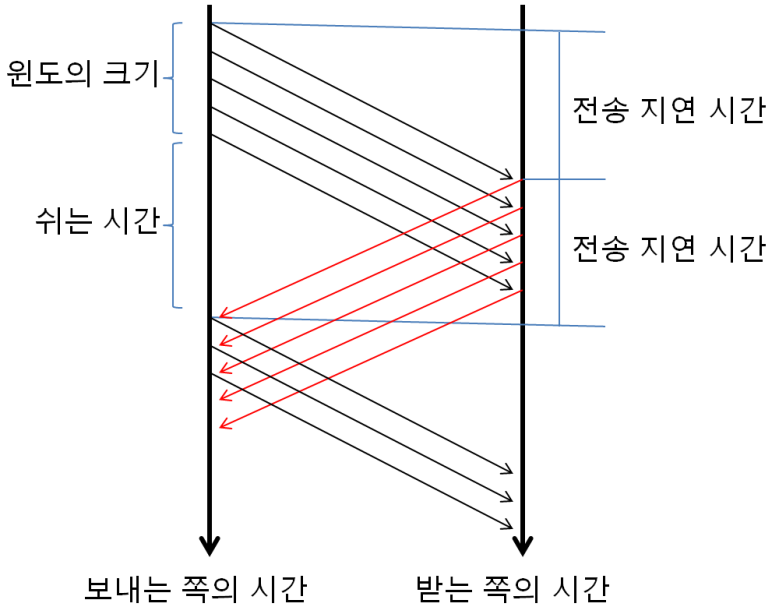


그림 25 윈도를 사용할 때 패킷의 송수신 모습

윈도 크기를 작게 하면 앞의 멈춰 기다리기 ARQ(stop-and-wait ARQ)가 갖고 있는 문제가 다 나타날 것이고 크게 하면 병목 구간에서 패킷 손실이 생기게 될 것이다. 따라서, 윈도 크기를 잘 조절하는 것이 중요하다. 잘 조절하면 패킷의 손실 없이 주어진 회선에서 보낼 수 있는 최대한의 속도로 보낼 수 있게 될 것이다. 그러므로 이상적인 전송 모습은 다음 그림과 같다. (여기서는 쉽게 그리기 위해서 두 컴퓨터를 연결한 회선이 병목 구간 없이 넓이가 같다고 가정하자.)

그림 26 회선이 감당할 수 있는 최대 용량

즉, 윈도의 크기를 잘 조절하여 **회선의 너비(= 대역폭) x 회선의 길이(= 지연 시간)**만큼의 패킷이 회선에 흘러가게 하면 우리는 100% 회선을 활용하고 있는 것이다. 만약, (대역폭 x 지연 시간)이 작다면 즉, 두 컴퓨터가 충분히 가까이 있거나 회선의 폭이 좁다면 윈도의 크기를 어떻게 하든 큰 낭비가 없겠지만 이 곱이 아주 큰 값이라면[37] 윈도 크기를 잘 조절하지 않으면 낭비가 심해지는 것이다. 기술이 발달함에 따라 대역폭은 점점 커지고 전 지구적으로 소통이 늘어남에 따라 지연 시간도 점점 커지고 있어서 윈도 크기 조절은 점점 더 중요한 문제가 되고 있는 셈이다.

그렇다면 최적의 윈도 크기를 어떻게 찾아낼 것인가? 솔직히 정답이 없다. 아마 이 주제로 석사나 박사 학위 받은 사람도 수백 명에 이를 것이다. 그래도 딱 떨어지는 답은 없다. 이 문제가 어려운 이유는 여러 가지지만 딱 하나만 꼽자면 **움직이는 목표물**(moving target) 즉, 늘 값이 바뀌기 때문이다. 앞에서 병목 구간을 설명하면서 병목이란 회선자체의 대역폭이 좁은 것일 수도 있고 너무 많은 패킷이 몰려서 혼잡한 곳일 수 있다고 했는데 전자의 경우에는 알아낼 수 있는 대략의 방법이 있지만 후자의 경우에는 알기가 어렵다. 명절이라 휴가철에 길이 막히는 정보를 알려면 방송을 참고하는데 교통방송의 통신원들도 별 수 없다. 막히는 구간에 실제로 가서 눈으로 확인하고 알려주는 경우가 대부분이다.

컴퓨터 네트워크에서도 마찬가지로 **실제로 가보는 방법**을 쓴다. 윈도 크기를 막 늘린 다음 패킷의 손실이 발생하면 다시 줄이고 괜찮다 싶으면 또 늘리고 손실이 발생하면 또 줄이고... 이 짓을 반복한다. 패킷의 손실이 발생하기 직전이 가장 이상적인 윈도 크기라고 할 수 있겠지만 그 값으로 고정해놓고 쓸 수 없다. 그 이유는 첫째 패킷 손실이 발생하기 직전이라는 시점을 정확히 알 수 없다. 왜? 패킷 손실을 알게 되는 것은 수신자 쪽에서 해당 패킷에 대한 ACK이 오지 안았을 경우인데 ACK이 오는데도 지연이 있을 뿐만 아니라 오지 않았다는 것을 확인하는데도 추가로 시간이 걸리기 때문이다. 둘째 이유는 앞서 얘기한 대로 혼잡 상황은 계속 바뀌므로 설령 최적의 윈도 크기를 알았다고 해도 1초 뒤에는 틀린 값이기 때문이다. 따라서, 끊임없이 시도 하고 조절하고 시도 하고 조절할 수 밖에 없다.

그럼 윈도 크기를 늘릴 때는 얼마씩 늘리고 줄일 때는 얼마씩 줄일까? 만약, 너무 과감하게 늘린다면 최적의 윈도 크기를 빨리 찾아갈 수 있다는 장점은 있지만 병목이 가득 차는 순간 이미 보내는 쪽에서는 회선에 엄청난 패킷을 보낸 상태이며 이들 대부분은 버려진다는 문제가 있다. 역으로 너무 소극적으로 키워나가면 최적의 크기를 아는데 시간이 많이 걸릴 것이다. 10초 걸려서 겨우 알아냈는데 1초 후에 달라져 버린다면 맥 빠지는 일 아니겠는가? 마찬가지로 윈도 크기를 줄일 때도 너무 급격히 줄이면 회선을 덜 활용하게 되고 너무 천천히 줄이면 이미 패킷 손실이 생기는데도 계속 패킷을 보내는 낭비가 발생한다.

현재의 인터넷에서는 L4 프로토콜로 TCP(transmission control protocol)가 가장 널리 사용되는데 여기서는 AIMD(additive increase/multiplicative decrease) 즉 윈도를 키울 때는 적절한 값을 더해나가고 줄일 때는 일정 비율로 확 줄여버리는 방식을 사용한다. 다시 말해, 키울 때는 소극적으로 줄일 때는 적극적으로 줄이는 방식이다. 그 외에도 적극적으로 키우고 적극적으로 줄이기(MIMD),

---

[37] 이 곱이 큰 회선을 **길고 뚱뚱한 네트워크**(LFN, long fat network)라고 부른다.

소극적으로 키우고 소극적으로 줄이기(AIAD), 적극적으로 키우고 소극적으로 줄이기(MIAD) 등 네 가지 다른 조합이 있을 수 있는데 AIMD 방식이 최적의 회선 상태로 수렴하는 경향이 있다는 것이 증명되어 있다.[38] 다음 그림은 AIMD 방식을 썼을 때 윈도 크기의 변화를 보여주는 예이다. 실제 TCP에서는 AIMD를 변형해서 쓰는데 맨 처음 윈도 크기를 늘려갈 때 좀 더 공격적으로 늘리므로 맨 처음에는 직선이 아니라 그보다 더 급격히 올라감을 볼 수 있다. 그러다가 패킷의 손실이 생길 때마다 절반 크기로 확 줄어들게 된다. 이런 패턴이 반복되므로 이 그림을 계속 그리면 톱날 모양이 나온다.

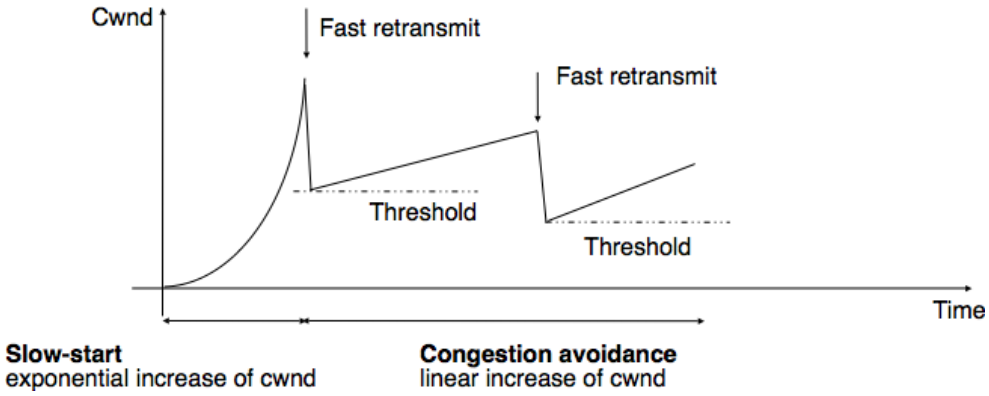


그림 27 TCP를 사용했을 때 윈도 크기의 변화(혼잡이 거의 없는 경우)[39]

4.5 UDP

인터넷의 근간을 이루고 있는 프로토콜이 TCP/IP인데 이걸 만든 사람은 빈튼 서프와 로버트 칸이다. (이 글에서는 편의를 위해서 IP를 먼저 다 설명하고 이를 기반으로 삼아서 TCP 를 구현하는 것으로 설명하였지만) 실제로 이 사람들이 처음 프로토콜을 제안하고 설계하고 구현할 때에는 TCP와 IP의 구분이 명확하지 않았다. 그래서 여러 네트워크가 연결된 환경에서 패킷을 전송하면서 파일을 보낸 다거나 다른 컴퓨터에 접속해서 원격 작업을 할 수 있도록 하기 위해서 TCP/IP를 만들었다.

그런데 이것을 써보니 어떤 종류의 응용 프로그램에서는 TCP 가 제대로 동작하지 않는다는 것을 깨달았다. 예를 들어, TCP에서는 ACK이 오지 않으면 패킷을 잃어버렸다고 생각해서 다시 보내는데 음성 통화와 같은 응용 프로그램에서는 이것을 차라리 안 하는 것이 더 낫다. (2초 전에 못 듣고 지나간 소리를 지금 듣고 있는 얘기 사이에 들려준다면 무슨 소용?) 그래서 TCP와 IP를 분리하고 TCP의 기능 즉, 전송 중 손실을 복구해주는 기능을 쓰지 않고 싶은 응용 프로그램을 위해서 UDP라는 것을 추가로 만들었다. 따라서, 응용 프로그램은 성격에 따라 TCP 또는 UDP를 선택해서 쓰면 된다. (둘 다 섞어서 써도 된다.)

현재 인터넷에서 가장 널리 쓰이는 것이 웹인데 웹의 경우 화면을 제대로 표시하기 위해서는 데

---

[38] 실제 TCP를 만든 사람들이 이런 성질을 알고 AIMD를 선택한 것인지는 분명하지 않다.

[39] 그림 출처. http://cnp3book.info.ucl.ac.be/transport/tcp/

이터가 깨지면 안되므로 당연히 TCP가 사용된다. 다음 그림은 Ineternet2에서[40] 지나가는 트래픽을 TCP와 UDP 로 구분하여 측정한 결과인데 80% 이상이 TCP임을 알 수 있다. 다른 네트워크에서의 데이터를 보면 TCP의 비율은 대개 이것보다 더 높다.

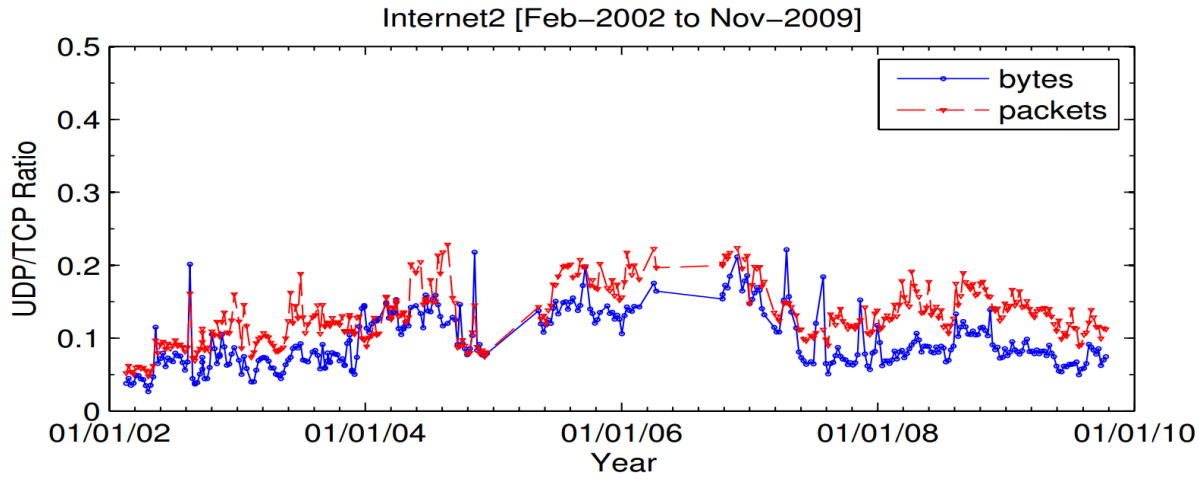


그림 28 TCP 대비 UDP 트래픽의 비율[41]

5 응용 계층 이야기

앞서 살펴본 모든 것은 결국 응용 프로그램이 인터넷 기반으로 동작하게 만들기 위한 도구에 지나지 않는다. 정말 인터넷을 꽃피우는 것은 다양하고 기발한 인터넷 응용 프로그램이 세상 도처에서 쏟아져 나오기 때문이다. 하지만 개별 응용 프로그램에서 다루기에는 까다로운 기능들이나 많은 응용 프로그램에 공통된 부분은 별도의 프로토콜이나 규격으로 미리 정해져 있다. 이러한 프로토콜이나 규격이 응용 계층(Application layer)을 구성하고 있다. 네트워크 계층의 참조 모델로 많이 사용하는 ISO의 OSI 모델에서는 세션 계층(Session layer, Layer 5), 표현 계층(Presentation layer, Layer 6), 응용 계층(Application layer, Layer 7)으로 구분하지만 인터넷에서는 대개 이렇게 구분하지 않고 응용 계층이라고 뭉뚱그려 얘기한다.

응용 계층에는 워낙 많은 기술이 포함되어 있으므로 대표적인 기술 몇 가지만을 소개한다. 우선, 사용의 편의를 위해서 발명된 도메인 이름 시스템(Domain Name System, DNS)과 동적 호스트 설정 프로토콜(Dynamic Host Configuration Protocol, DHCP)를 소개한다.

어지간한 응용 프로그램은 TCP와 UDP를 가지고 다 만들 수 있다. 하지만 특별한 종류의 프로그램은 추가의 전송 기능이 필요한데 그 중에서 대표적으로 실시간 데이터 전송이 필요한 응용을 위한 실시간 전송 프로토콜(RTP, real-time transport protocol)과 안전한 데이터 송수신이 필요한 응용을 위한 전송 계층 보안(TLS, transport layer security)을 소개한다.

---

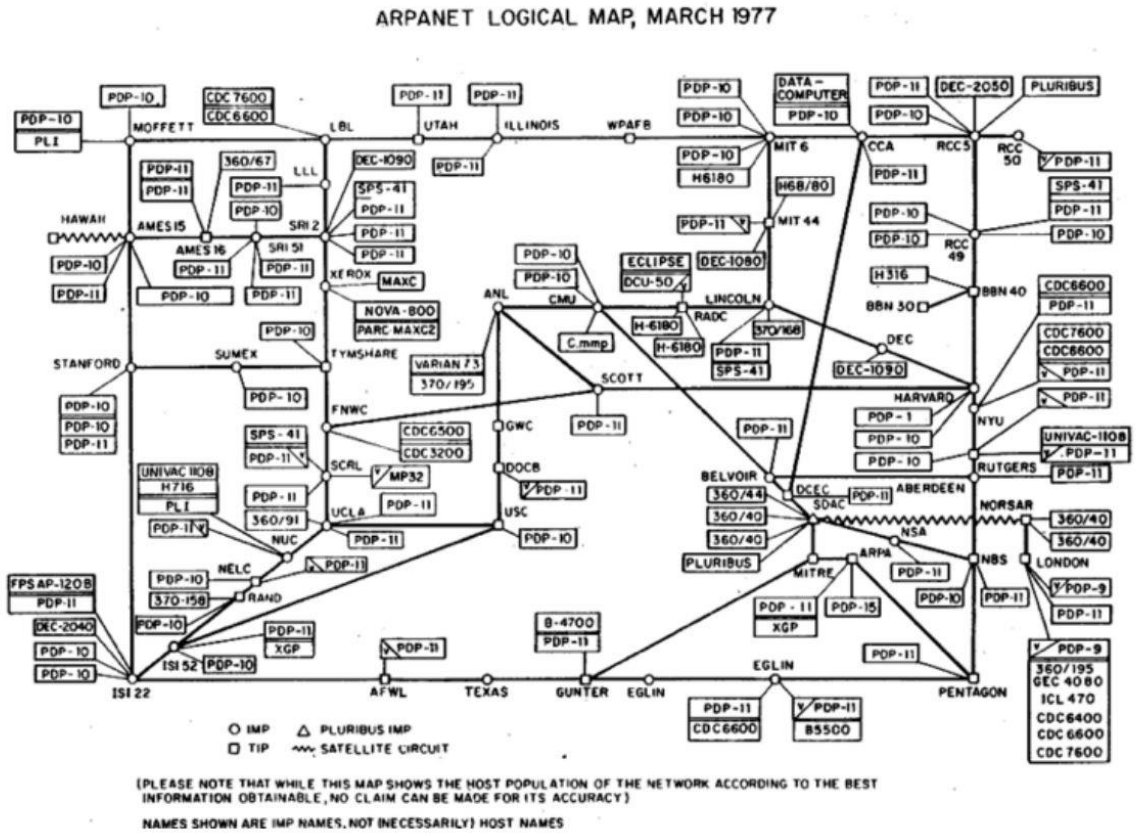
[40] Internet2는 인터넷 연구자들을 위한 연구 네트워크 중의 하나다.

[41] 그림 출처. https://www.cs.auckland.ac.nz/~brian/media-UDP-TCP.pdf

그리고 이 모든 것이 결국 사용자에게 서비스 특히 컨텐츠를 제공하기 위한 것이므로 컨텐츠를 잘 전달하기 위한 기술을 간단히 소개한다.

5.1 DNS – IP 주소를 어떻게 아냐고?

뭐든지 그렇지만 인터넷의 초창기에는 연결된 컴퓨터의 수가 많지 않아서 대부분의 관리 작업도 수작업으로 해도 큰 문제가 없었다. 아래의 그림은 1977년 인터넷의 모습이다. 연결된 모든 컴퓨터를 다 적어 놓았다. 얼른 세어봐서 수십 남짓이다.



Source: ARPANET Completion Report, January 4, 1978.

그림 29 1977년 3월 인터넷의 구성

이렇게 아직 연결된 컴퓨터의 수가 많지 않을 때에도 상대방 컴퓨터로 연결할 때 IP주소로 연결하려면 숫자 외기가 쉽지 않아 영 불편하였다. 그래서 고안된 것이 컴퓨터에 이름을 붙이고 그 이름을 IP 주소로 변환하는 기능이다. 예를 들어, 당시 인터넷 연결에 가장 널리 사용되었던 유닉스(Unix) 시스템에서는 **/etc/hosts** 라는 파일에 **101.202.103.204 순돌이컴** 이라고 써넣고 그 다음부터는 순돌이컴이라고 쓰면 101.202.103.204이라는 IP주소로 이해하게 만들었다. 그래서 인터넷의 모든 컴퓨터의 이름과 IP주소를 수집하여 **hosts** 파일을 만들어두면 누구든지 이 파일을 받아서 적절한 폴더에 넣어주기만 하면 다른 컴퓨터의 IP주소를 외지 않고 이름만 알아도 접속할 수 있게 되는 것이다. 이러한 인터넷 정보의 수집과 배포 역할을 맡은 기관을 네트워크 정보 센

터(NIC, network information center)라고 불렀다.[42] 지금도 hosts 파일을 이용하여 IP주소로 변환하는 기능은 거의 모든 환경에 그대로 남아 있다.

그런데 인터넷에 연결된 컴퓨터의 수가 수억, 수십억, 수천억으로 늘어나면 어떤 문제가 있을까? 당연히 hosts 파일을 수정해서 배포하는 일도 엄청나게 부담스러운 일이 될 것이고 hosts 파일 자체의 크기도 계속 커질 것이다. 이렇게 큰 파일을 인터넷에 연결된 모든 컴퓨터가(심지어는 스마트 폰까지) 갖고 있어야 한다는 것은 뭔가 문제가 있다. 이런 문제를 해결하기 위해 등장하는 것이 도메인 이름 시스템(Domain Name System, DNS)이다.

DNS에서는 우선 이름 공간을 계층적으로 쪼갰다. 맨 위층에는 KR, JP, CN 등과 같이 나라를 대표하는 이름이나 COM, NET 등과 기관의 분류를 대표하는 이름을 둔다. 이들 각각이 하나의 영역 즉, 도메인이 된다. 그리고 각 영역마다 그 영역을 담당하는 서버(**이름 서버**, name server)가 있다. 한 영역에는 그 영역에 속하는 **컴퓨터를 등록**할 수도 있고 **영역을 등록**할 수도 있다. 예를 들어, 어떤 컴퓨터의 이름을 **어떤컴퓨터.KR** 이라고 짓고 그 IP주소를 KR영역을 담당하는 이름 서버에 등록해둘 수 있다.[43] 또는, **CO.KR** 이라는 영역을 만들어 이름 서버에 등록할 수도 있다. 영역을 등록할 때에는 그 **영역을 담당하는 이름 서버의 IP 주소도 같이 기록**해둔다. CO.KR은 그 자체가 또 하나의 영역이므로 그 아래로 컴퓨터나 영역을 등록할 수 있다. 그런 식으로 영역과 컴퓨터를 계층적으로 등록해나가면 다음 그림과 같이 된다. (아래의 그림은 3단계로 되어 있는데 그 아래 단계로 계속 내려 갈 수 있다.)

---

[42] 전세계로 인터넷이 뻗어간 이후에는 각 국가별로 NIC이 생겨났는데 현재 한국인터넷진흥원 산하의 한국인터넷정보센터(KRNIC)은 한국에서 이 역할을 담당하던(담당하는???) 기관이다.

[43] 실제로는 KR 바로 아래에 컴퓨터를 등록하지 않는데 이는 정책적인 것이므로 DNS 기술의 관점에서 보자면 등록하지 못할 이유는 없다.

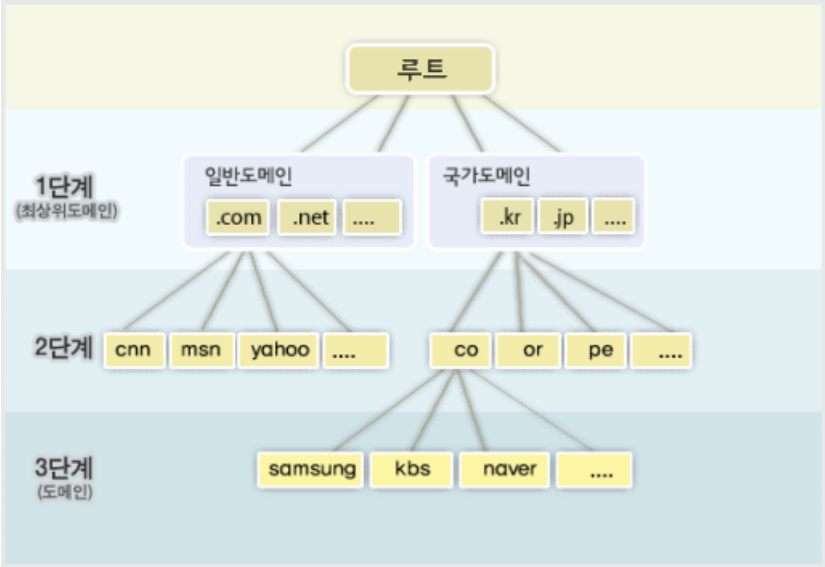


그림 30 DNS는 계층적인 이름 공간을 만들게 된다[44]

그림에서 보듯 이름 공간의 맨 위는 루트로 시작하는데 이는 최상위의 영역(COM 이나 KR)의 등록을 담당하는 영역이다.[45] 요약하자면 DNS에서는 이름 공간을 영역의 계층 형태로 만들고 각 영역에는 그 영역에서의 등록을 담당하는 이름 서버가 있다.

이렇게 해서 컴퓨터의 이름은 등록했다고 하자. 그럼 그 이름(예를 들어, www.wikipedia.org)으로부터 IP주소는 어떻게 알아내는가? 앞서 3장에서 살펴본 바와 같이 각 컴퓨터의 네트워크 설정에 보면 DNS 서버의 IP 주소를 지정하는 칸이 있어서 여기에 적절한 주소가 꼭 들어 있어야 한다. (나는 그런 설정 한 적 없이도 되던데? 스마트 폰은 그런 거 안 해도 되잖아? 라는 의문은 다음 장에서 해결된다. 지금은 곤란하다. 조금만 기다려 달라.) 그래서 내가 www.wikipedia.org을 접속하려고 하면 내 컴퓨터는 네트워크 설정에 들어 있는 DNS 서버에게 "www.wikipedia.org 주소 알려줘"라고 질의를 보내게 된다. 그럼 (아래에서 설명하는 절차를 거쳐) DNS 서버는 "google.com 주소는 208.80.154.224 입니다"라고 응답한다.

그럼 DNS 서버는 어떻게 www.wikipedia.org의 주소를 알게 되는가? www.wikipedia.org의 주소를 확실히 알고 있는 컴퓨터가 있다. 누구? wikipedia.org 영역을 담당하는 이름 서버다. 왜냐하면 www.wikipedia.org이라는 이름을 쓰기 위해서 www라는 컴퓨터 이름과 IP 주소를 wikipedia.org 영역에 등록했을 테니까. 그럼 wikipedia.org 영역을 담당하는 이름 서버에게 물어보면 되는데 이 서버의 주소는 어떻게 알 수 있는가? 마찬가지다. wikipedia.org 영역은 org 영역에 등록되어 있으니 org 영역의 이름 서버에게 물어보면 된다. 그럼 org 영역의 이름 서버 주소는 어떻게 알 수

---

[44] 그림 출처. http://blog.naver.com/PostList.nhn?blogId=enumdns

[45] 만약 루트 영역에서 COM을 빼버린다면 사람들은 (IP주소를 따로 외고 있지 않는 한) 더 이상 아마존(amazon.com)이나 네이버(naver.com)를 접속할 수 없게 된다. 따라서 루트 영역의 관리는 무지하게 중요한 문제로서 이 글의 범위를 넘어서는 이슈이므로 생략한다.

있나? org 영역은 루트에 등록된 영역이므로 루트를 담당하는 이름 서버에게 물어보면 org 영역을 담당하는 이름 서버의 주소를 알 것이다. 그럼 루트를 담당하는 이름 서버의 주소는 어떻게 알 수 있나? 그건 다른 방법이 없다. 그냥 사람이 입력해준다고 생각하자. 어쨌든 루트 이름 서버에게 org 이름 서버의 주소를 물어보니 204.74.112.1 라고 해서 거기에 접속해서 wikipedia.org 이름 서버의 주소를 물어보니 207.142.131.234 라고 해서 거기에 접속해서 www.wikipedia.org 의 주소를 물어보아 답을 얻게 되는 것이다. 다음의 그림이 그 과정을 표현한 것이다.

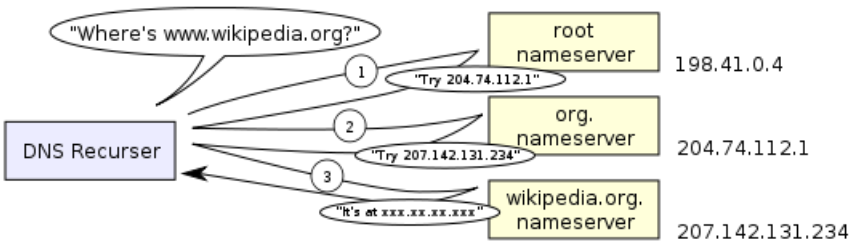


그림 31 DNS를 이용하여 이름을 IP 주소로 바꾸는 과정[46]

물론 이렇게 하면 루트 이름 서버의 부담이 너무 커지므로 실제로 대다수의 이름 서버는 이름을 IP 주소로 바꾸는 과정에서 알게 된 정보(예를 들어, org 이름 서버의 주소)를 기억해 두었다가 다음에 쓸 일이 있으면 다시 묻지 않고 알고 있는 정보를 사용한다.[47]

5.2 DHCP – 네트워크 관리자를 악몽에서 구출하라

컴퓨터나 스마트 폰을 사용하는 사람들 중에 상당수는 IP 주소 설정이나 DNS 서버 설정을 해본 적이 없다. 하지만 이러한 설정이 없이는 인터넷으로 데이터를 보내거나 받을 수 없다. 그렇다면 어떻게 된 노릇인가? 답은 간단하다. 누군가 **인터넷 설정을 자동으로** 대신해준다. 이 때 사용하는 프로토콜이 동적 호스트 설정 프로토콜(Dynamic Host Configuration Protocol, DHCP)이다.

집에 유무선 공유기를 사서 연결하고 내 스마트 폰에서 Wi-Fi를 찾아보면 내 공유기의 무선 접속 포인트 이름 (예, iptime)이 보이고 클릭만 하면 연결되고 인터넷이 된다. 왜 되지? 마찬가지로 출장을 가서 호텔 방에서 인터넷 케이블을 내 노트북에 꽂기만 했는데 인터넷이 된다. 왜 되지? 이런 경우에 네트워크 설정을 들어가보면 다음 그림과 같이 되어 있다.

---

[46] 그림 출처. http://en.wikipedia.org/wiki/Domain_Name_System

[47] 그렇다고 오래된 정보를 계속 쓰면 안되므로 DNS의 모든 정보는 적절한 수명 정보를 갖고 있어서 앞으로 얼마 동안 거의 바뀌지 않을지 힌트를 준다. 따라서, 그 이상 지난 정보는 나중에 필요할 때 다시 물어보게 된다.

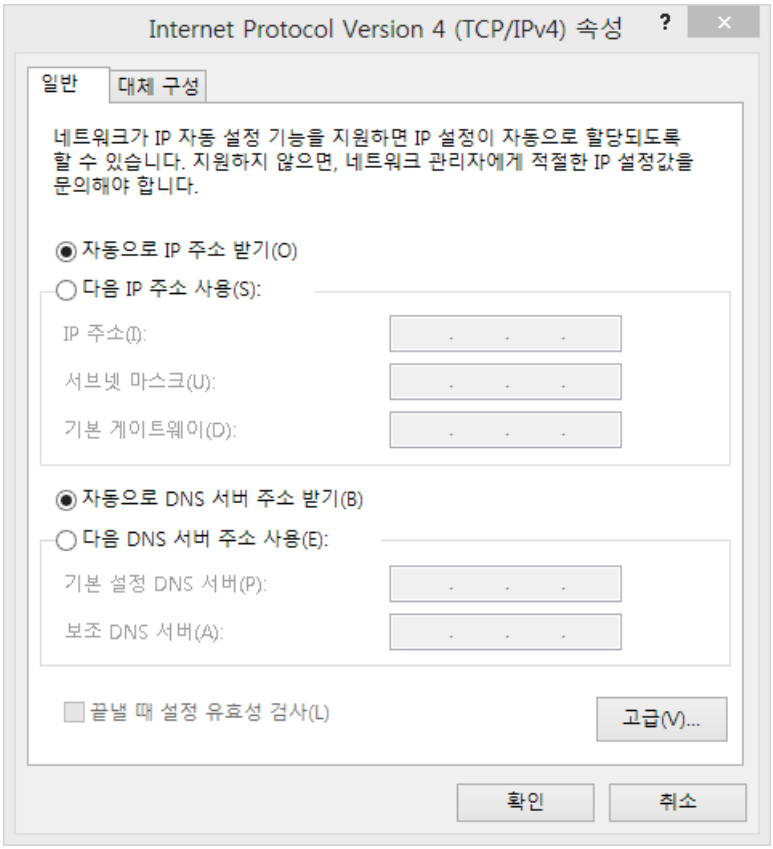


그림 32 DHCP를 사용하는 네트워크 설정 방법

그렇다. IP 주소, 서브넷 마스크, 게이트웨이 주소, DNS 서버 주소까지 몽땅 **자동으로** 받게 되어 있다. 그런데 네트워크가 안 되는데 어떻게 자동으로 받을 수 있나? 앞에서도 비슷한 경우가 있었다. MAC 주소를 알아야 통신을 할 수 있는데 IP 주소만 알고 상대방의 MAC 주소를 모를 때 ARP 프로토콜을 사용하는 과정을 3.7장에서 설명한 바 있는데 그 때와 같은 방법을 쓰면 된다. 즉, 브로드캐스트를 써서 요청하는 것이다.

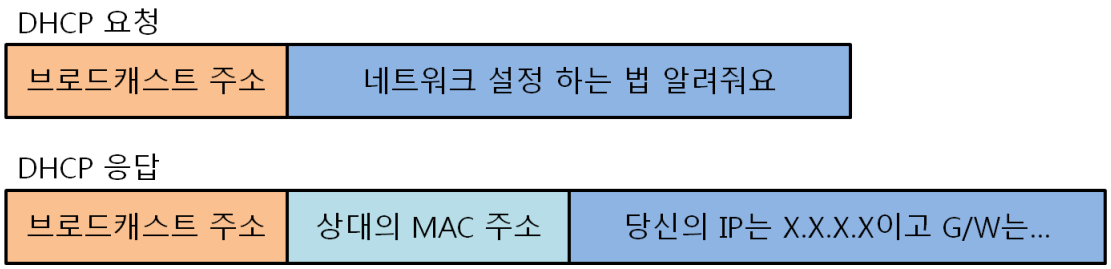


그림 33 DHCP 메시지의 모양

유선 이건 무선 이건 새로운 네트워크에 연결이 되었는데 설정을 자동으로 하도록 되어 있으면 **DHCP 요청**을 보낸다. (누구한테 물어봐야 되는지 모르니 브로드캐스트 한다.) 그럼 그 네트워크에는 DHCP를 담당하는 서버가 있어서 그 요청을 듣고 적절한 IP 주소를 할당하고 그 외 필요한 정보를 다 넣어서 **DHCP 응답**을 보내준다. (물론 DHCP 서버 설정은 제대로 되어 있어야 하고 이

건 사람이 해야 한다.[48])

DHCP 서버 설정만 한 번 잘 해두면 그 네트워크에 컴퓨터를 꽂았다 뺐다 하더라도 관리자가 해야 할 일은 거의 없다. 특히, 스마트 폰이나 가전 제품처럼 네트워크 설정을 하기가 곤란한 장치가 늘어가고 추세라서 (지금도 이미 그렇지만) 앞으로는 대부분의 네트워크가 특별한 이유가 없다면 DHCP를 쓴다고 생각하면 된다.

5.3 RTP – 인터넷 전화를 위한 기반

인터넷 전화 기능을 구현하려고 한다고 하자. 그럼 일단 TCP는 앞서 얘기한 것처럼 과도하게 복구를 시도하기 때문에 적절하지 않고 UDP를 기반으로 만들면 될 것이다. 한 사람이 얘기한 것을 디지털 데이터(즉, 비트)로 바꾼 뒤 적절한 크기로 나눠 데이터 패킷에 담아서 UDP를 이용하여 상대방에게 보냈다고 하자. 그럼 상대방은 패킷에 들어 있는 데이터를 소리로 다시 변환하여 스피커로 들려줄 것이다. 이렇게 하면 아무 문제 없을까?

앞서 1.3장에서 설명한 것처럼 회선 교환 방식에서는 두 단말기가 하나의 경로로 연결되어 모든 데이터가 그 경로를 따라서 가므로 (혹시 데이터의 분실을 있을지 몰라도) 모든 데이터는 차례로 도착한다. 하지만, 패킷 교환 방식에서는 상황에 따라 데이터 패킷이 서로 다른 경로를 거쳐갈 수 있으므로 상대방에 도착하였을 때에는 차례가 맞지 않게 된다. 따라서, 패킷을 **보낸 순서에 맞춰 정렬**하여야 한다. 또한, 패킷의 도착 간격이 보낼 때와는 달리 들쭉날쭉 할 수 있으니 **간격을 일정하게** 맞춰주어야 한다. 이 두 가지 기능을 하기 위해서 만들어진 것이 RTP이며 대개는 UDP와 결합하여 사용한다. (규격상으로는 TCP와 결합하여 사용할 수도 있게 되어있다.) 요즘 많이 쓰는 대다수의 인터넷 전화가 이 RTP를 이용하고 있다.

---

[48] 공인 IP 주소(public IP address)를 써야 한다면 설정이 좀 복잡하고 사설 IP 주소(private IP address)를 쓴다면 DHCP 서버의 설정도 거의 손 댈 일이 없다. 그래서 대부부의 가정에서 쓰는 유무선 공유기는 사설 IP 주소를 기본으로 사용하도록 되어 있고 아무나 쉽게 설치해서 쓸 수 있다.

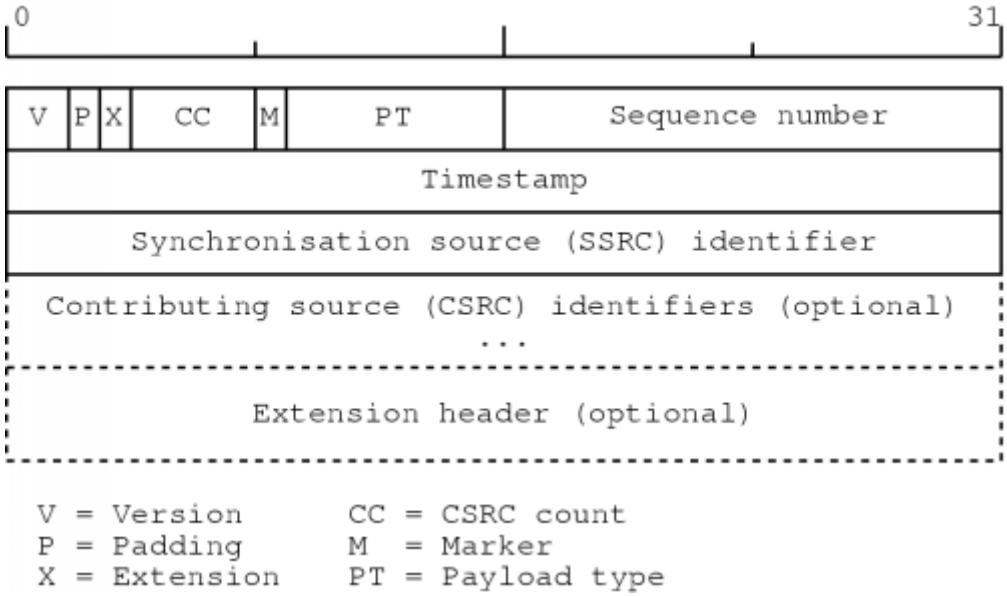


그림 34 RTP 패킷 헤더[49]

위의 그림은 RTP 패킷의 앞부분에 붙는 헤더를 나타내고 있다. 보낸 순서에 따라 정렬하기 위하여 순서를 나타내는 번호(sequence number)가 있고 시간 간격을 조절하기 위하여 시간을 기록하는 칸(timestamp)이 있다. 또 하나 특이한 것은 PT(payload type)인데 여기에는 이 RTP가 담고 있는 데이터가 어떤 종류인지를 담고 있다. 예를 들어, 이 값이 34라면 이 패킷에는 90KHz H.263 형식의 영상 데이터를 담고 있다는 뜻이다.

5.4 TLS/SSL – 안전한 인터넷의 출발

은행 거래나 비밀스런 문서의 교환 따위를 인터넷을 통해서 하고 싶다면 어떻게 해야 할까? 앞서 살펴본 바와 같이 인터넷에서는 당연히 모든 패킷의 내용을 (안 보려고 애쓰지 않는 한) 다 볼 수 있다. 그런 상황에서도 통신상 보안을 지켜야 한다면 과연 어떤 방법이 필요할까?

인터넷 쇼핑을 예로 생각해보자. 내가 ***믿을_만한_가게.com***에 접속하여 무엇인가 구매한다고 할 때 맨 먼저 보장되어야 할 것은 내가 접속한 웹 사이트가 흉악한 놈들이 비슷하게 꾸민 웹 사이트가 아니라 내가 정말 접속하려고 했던 그 웹 사이트인지 확인을 해야 한다. 이럴 때 사용하는 것이 인증서(certificate)다. 웹 사이트에 접속하면 웹 사이트는 내 웹 브라우저로 인증서를 보내준다. 이 인증서에 있는 웹 사이트 이름이 내가 입력한 주소(***믿을_만한_가게.com***)와 같은지 확인한다. **그리고** 이 인증서를 발급한 기관이 어딘지, 그 기관에서 발급한 것이 맞는지, 그리고 그 발급 기관이 믿을만한 곳인지를 **모두 확인**하여 문제가 없다면 최소한 내가 접속하려는 웹 사이트에 접속한 것은 맞다. (이들 조건 중 하나라도 만족이 되지 않으면 믿으면 안 된다.)

---

[49] 그림 출처. https://access.redhat.com/site/documentation/en-US/JBoss_Communications_Platform/

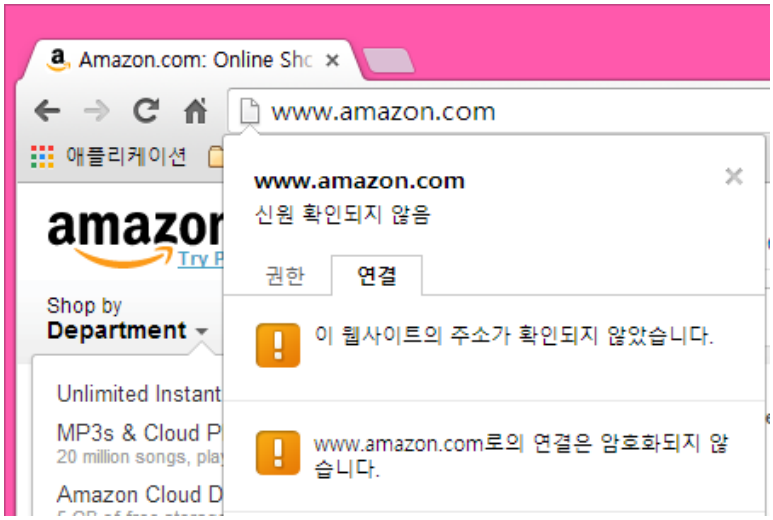


그림 35 보통의 웹 연결 상태

위의 그림은 어떤 인터넷 쇼핑 사이트에 그냥 접속했을 때 웹 브라우저의 모습이다. 웹 브라우저마다 보는 방법은 다르지만 어쨌든 현재 웹 사이트 연결의 보안 상태를 보여주는 기능은 다 있다. 이 그림에서 볼 수 있듯이 내가 접속한 사이트가 무엇인지는 알지만 그 신원은 확인되지 않은 상태이다. 이 상태에서 좀 중요한 작업을 하는 화면(예를 들어, 고객 정보 수정 화면)으로 가면 어떻게 바뀌는지 살펴보자.

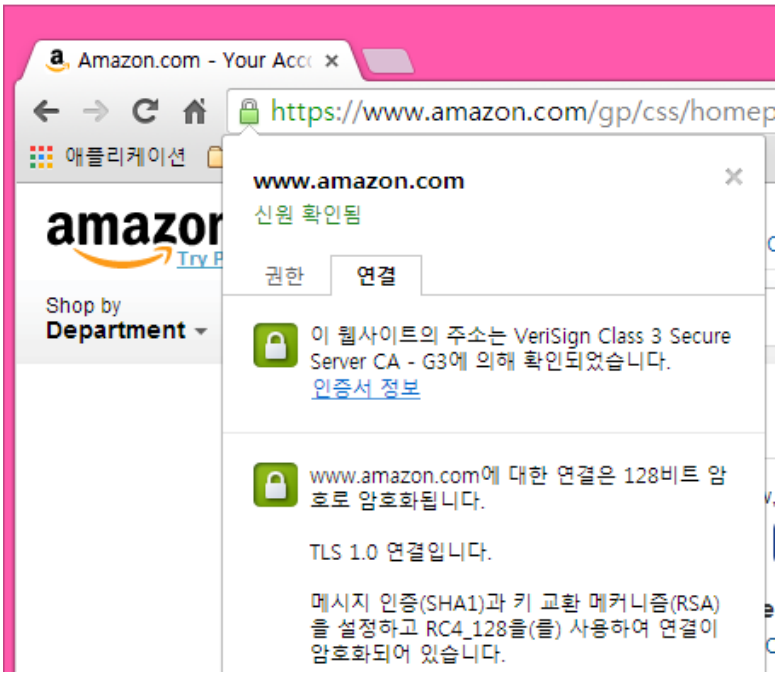


그림 36 웹 사이트를 보안 연결한 상태

그림에서 자물통 표시는 이 웹 사이트가 내가 접속하려는 사이트라는 것이 확인되었다는 뜻이다. 그리고 이 확인 작업을 위해서 인증서를 확인했는데 그 인증서는 베리사인(VeriSign, 인증서를 발급하는 세계적인 회사 중 하나)이 발급했으며 현재의 연결은 지금 설명하려는 TLS 즉, 전송 계층

보안 1.0을 이용하고 있다는 것을 보여주고 있다. 웹 브라우저가 알아서 확인했다고는 하지만 인증서에 어떤 내용이 들어 있는지 살펴보자.

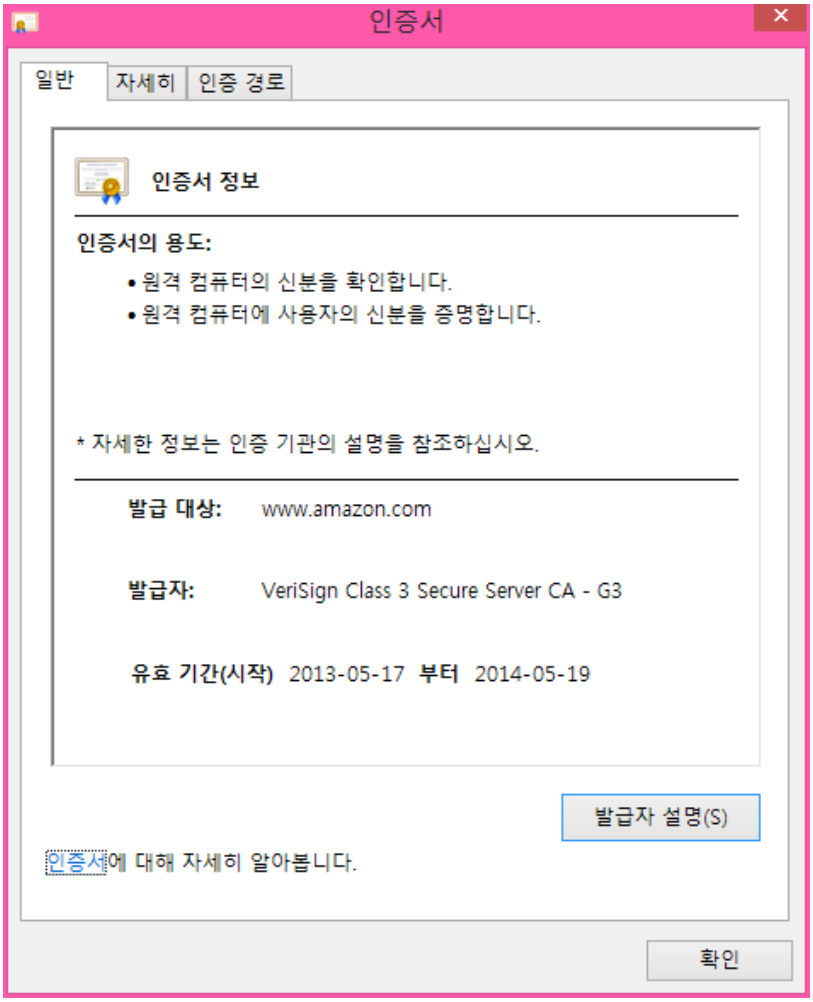


그림 37 어떤 인터넷 쇼핑 사이트의 인증서 정보(일반)

인증서에는 이 인증서가 누구 것인지, 누가 발급했는지 그리고 유효기간은 언제인지 등의 정보가 들어 있다. 당연히 발급 대상이 우리가 접속한 사이트와 같은지 확인해야 하고 유효기간이 만료되지 않았는지도 보아야 한다. 그리고 발급자도 믿을 수 있는 기관인지 보아야 한다. 이 인증서의 경우 발급자가 "VeriSign Class 3 ..." 라는 곳이다. 베리사인은 유명한 회사가 맞는데 혹시 이 회사는 베리사인을 사칭한 곳은 아닐까? 그렇다면 그 **발급자의 인증서**를 확인해봐야 한다. 발급자의 인증서가 제대로 구성되어 있고 **발급자의 인증서를 발급한 곳**이 믿을 수 있다면 안심이 되는 것이다. 이런 식으로 인증서를 검증하는 단계는 **믿을 수 있는 발급자**를 만날 때까지 줄줄이 연결된다. 그 과정을 보여주는 것이 다음의 그림이다.

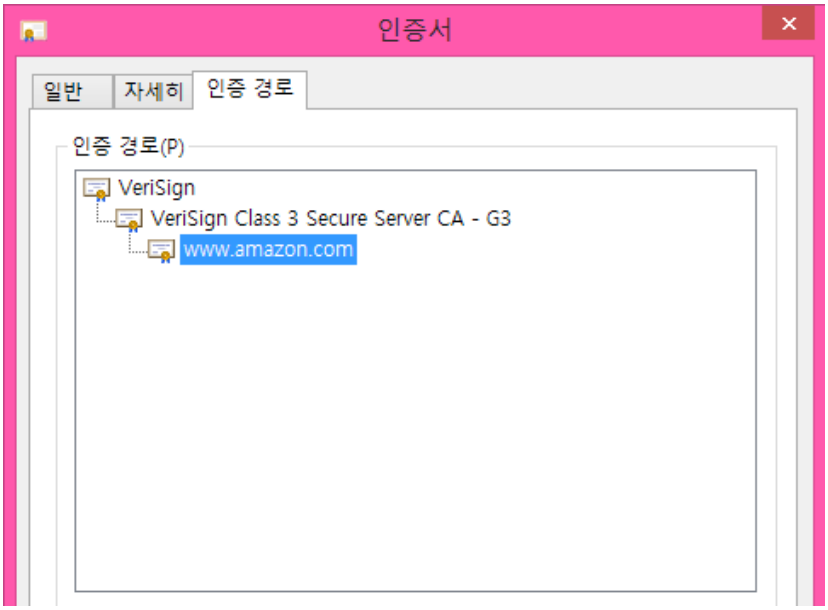


그림 38 어떤 인터넷 쇼핑 사이트의 인증서 정보(인증 경로)

이 그림에서 보듯 "VeriSign Class 3 ..."라는 곳의 인증서는 베리사인이 발급했고 우리가 베리사인을 믿는다면 우리는 이들 인증서를 모두 믿게 되는 것이다. 그렇다면 베리사인의 인증서는 누가 발급할까? 자기 자신이 발급한다. 말하자면 자기의 보증인이 자기 자신인 것이다. 따라서, 그 사람 자체를 믿느냐 마느냐를 정해야 한다. 세상에는 몇몇 믿을 수 있다고 대개 인정하는 발급자들이 있으며 이들의 목록은 다음의 그림처럼 웹 브라우저에 내장되어 있다. 따라서, 설령 베리사인의 인증서를 베리사인 스스로 발급했음에도 우리는 믿는 것이다.

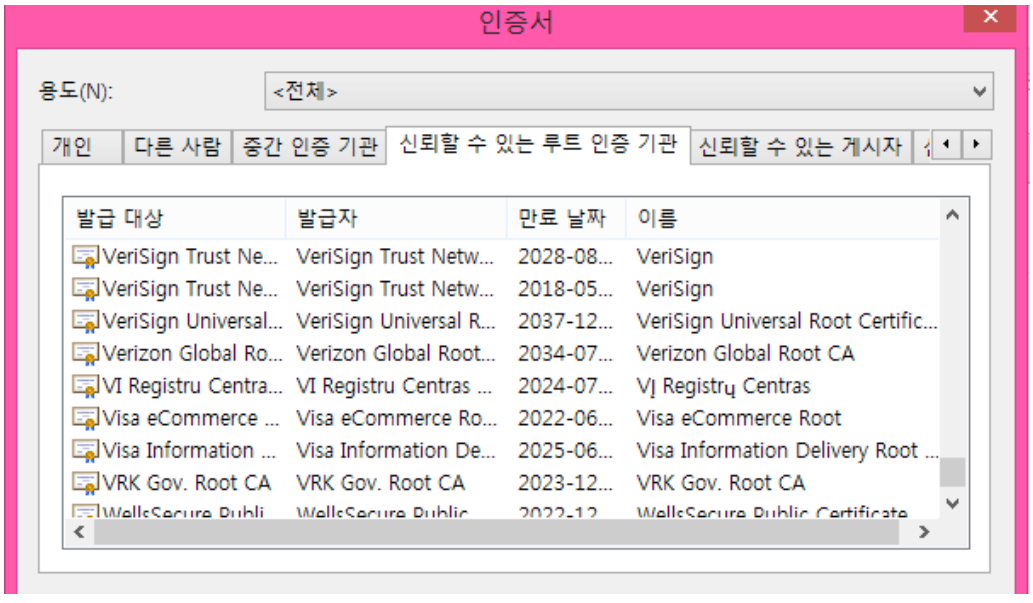


그림 39 웹 브라우저에 내장된 믿을 수 있는 인증 기관의 목록

설명이 좀 길었지만 어쨌든 이 긴 과정을 거쳐서 우리는 우리가 접속한 웹 사이트가 우리가 접속하려 했던 그 사이트라는 것을 확인했다. 그럼 이제 믿고 **중요한** 정보(예, 결제 정보)를 보내려는

데 남들이 정보를 엿들을 수 있으므로 암호화를 해야 한다. 대량의 데이터를 빠르게 암호화하면서 주고 받기 위해서는 **대칭 암호화** 알고리즘을 쓰면 된다.[50] 즉, 암호 키 하나를 정하고 내가 웹 사이트로 보낼 때 정보를 그 암호 키로 암호화 해서 보내면 웹 사이트에서는 같은 암호 키로 풀어서 내용을 보고 답을 줄 때도 같은 키를 써서 암호화해서 보내면 된다.

문제는 암호 키를 남들은 몰라야 하고 나와 웹 사이트만 알아야 한다는 점이다. 만약 암호 키를 주고 받는 과정을 누군가 엿듣고 있으면 암호 키가 노출 되므로 아무런 소용이 없다. 그럴 때 구원 투수로 등장하는 것인 **비대칭 암호화** 알고리즘이다.[51] 예를 들어, 한쪽에서 암호 키(이를 **공유된 비밀** - shared secret - 이라 부른다)를 만들고 이를 상대의 공개 키로 암호화해서 보내면 이를 풀 수 있는 것은 짝이 되는 비밀 키를 가진 상대방뿐이므로 누군가 엿듣더라도 풀 수가 없어서 나와 그 상대만 암호 키를 공유하게 되는 것이다.

이러한 과정을 통하여 남들이 다 들을 수 있는 *안전하지 못한* 인터넷을 통하여 신원이 확인된 상대와 비밀스럽게 대화할 수 있는 채널은 열어주는 것이 전송 계층 보안(TLS, transport layer security)의 역할이다. 이 기술은 보안 소켓 계층(SSL, secure socket layer)의 뒤를 잇는 기술이므로 TLS와 SSL라는 용어를 혼용하기도 한다.

TLS에서는 주고 받는 메시지를 암호화 하는 공유된 비밀 키를 **세션 키**라고 부른다. 즉, 이번 연결 동안에만 사용하는 키라는 의미를 담고 있다. 또한 TLS에서는 다음 그림에서 알 수 있듯이 세션 키로 메시지를 암호화 하기 전에 메시지에 해시함수를[52] 적용하여 메시지 인증 코드(MAC, message authentication code)를 생성하고 메시지 뒤에 인증 코드를 덧붙여서 암호화 한다. 상대방은 패킷을 받은 후 암호를 풀고 메시지에 해시함수를 적용하여 나온 값이 메시지에 딸려온 메시지 인증 코드와 같은지 확인한다. 만약 무슨 이유에서든지 메시지의 일부가 변조가 된다면 이 과정에서 변조되었음을 알 수 있다.

---

[50] 하나의 암호 키로 암호화 되지 않은 문장(평문)을 암호문으로 바꾸고 역으로 암호문을 평문으로 되돌리는 기능이 되는 암호화 기법.

[51] 암호 키가 공개 키(public key)와 비밀 키(private key)의 짝으로 되어 있어서 공개 키로 암호화 하면 비밀 키로만 풀 수 있고 반대로 비밀 키로 암호화 하면 공개 키로 풀 수 있다. 일반적으로 (이름에서 알 수 있듯이) 공개 키는 공개하고 비밀 키는 숨겨둔다.

[52] 임의의 데이터를 고정된 길이의 데이터로 바꾸는 함수. 해시함수의 결과 값을 해시 값이라고 부른다. 비유적으로 얘기하자면 사람의 지문과 같다고 보면 된다. 사람마다 지문이 다르므로 어딘가 있는 지문이 누군가의 지문과 같다면 그 사람이 범인일 가능성이 매우 높아지는 것이다. 입력으로 들어가는 데이터가 조금이라도 다르면 해시 값도 다른 것이 이상적이다. 물론, 지문과 마찬가지로 해시 값이 같다고 원문 데이터가 같다는 보장은 없다.

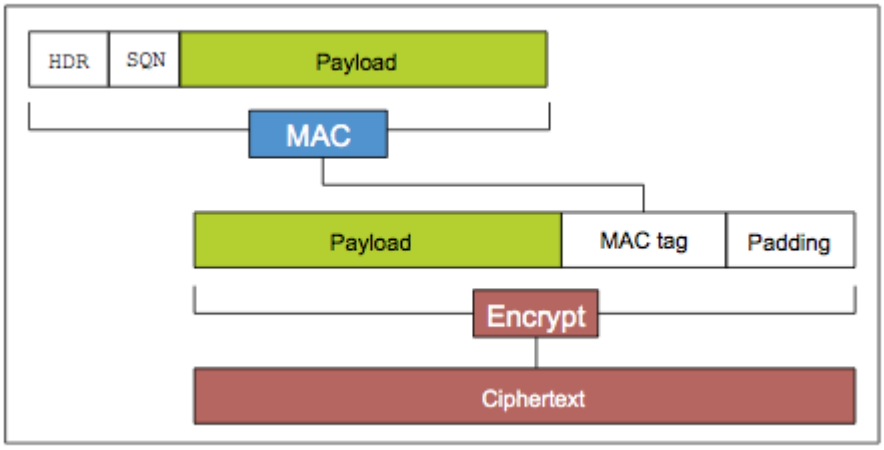


그림 40 TLS에서 패킷의 암호화[53]

TLS를 가장 활발히 활용하는 것은 앞서 예시에서 본 것처럼 웹 서비스를 안전하게 할 수 있도록 웹 프로토콜인 HTTP와 TLS를 결합한 HTTPS(HTTP Secure)이다.

5.5 온갖 컨텐트를 전송하기 위한 기술

인터넷을 써본 사람이라면 누구나 아래 그림과 같이 도저히 알아볼 수 없는 글씨로 가득 찬 메일이나 웹 페이지를 본 적이 있을 것이다. 왜 이런 일이 생기는지 생각을 해보자.

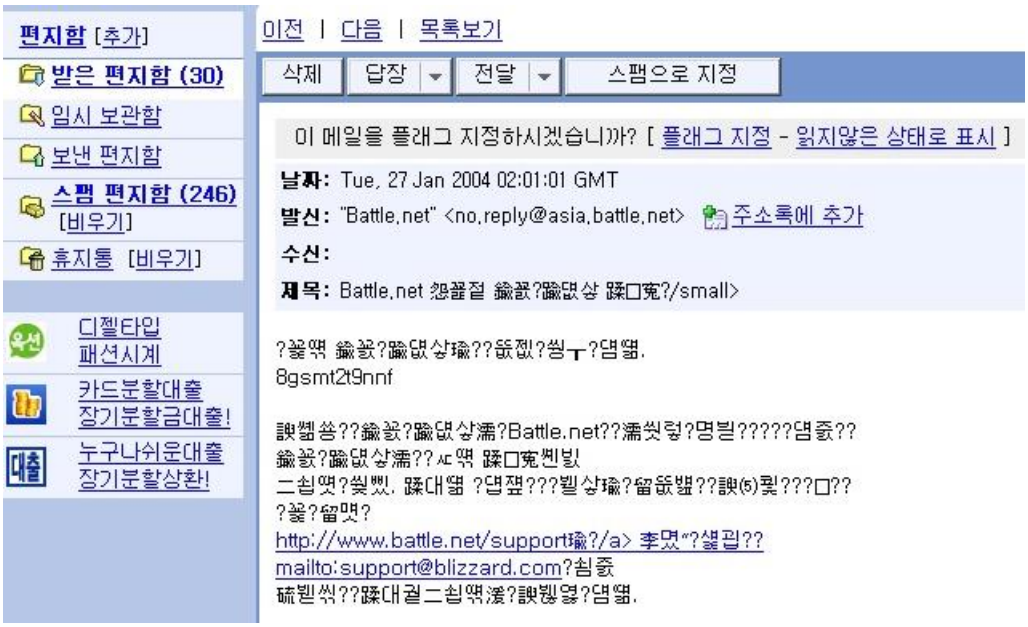


그림 41 인코딩이 깨진 한글 메일의 예시[54]

---

[53] 그림 출처. http://www.h-online.com/imgs/43/9/7/9/4/6/6/lucky13-58163e1f4ee55b70.png

[54] 그림 출처. http://support.hanbitsoft.co.kr/FAQ/DIA2/IMAGES/GetNewPw_err_1.jpg

앞서 1.2장에서 살펴본 것과 같이 선을 타고 전달되는 것은 0 또는 1의 비트일 뿐이므로 우리가 보내려는 글자나 그림이나 소리를 보내려면 보내려는 데이터(글자/그림/소리)를 비트로 바꾸고 비트로 표현된 것을 어떻게 해석할 것인지 **약속**이 필요하며 그 약속을 코드라고 부른다고 하였다. 그리고, 데이터를 약속된 코드로 바꾸는 행위를 코드화(인코딩, encoding)라고 부른다.

예를 들어, 미국에서 (그리고 세계적으로 가장 널리) 쓰는 코드인 아스키 코드에서 영어 대문자 A는 65 (= 1000001_2), B는 66 (= 1000010_2), C는 67 (= 1000011_2), … 이렇게 쭉 나간다.[55] 만약 내가 상대에게 "ABCD"라는 문자열을 보내고 싶다면 아래 그림과 같이 비트로 바꿔서 쭉 보내면 상대는 그것을 미리 약속된 표에 따라 해석하는 것이다. (그림에서는 숫자 표시의 편의를 위해서 오른쪽에서 왼쪽으로 보내는 것으로 표현했으니 헷갈리지 말 것)

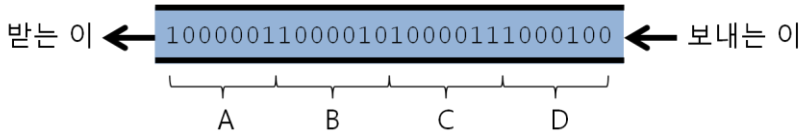


그림 42 "ABCD"를 보내면

위와 같이 메시지를 코드화하여 교환할 때 꼭 보장되어야 할 것은 상대와 내가 사용하는 코드 표가 같아야 한다는 점이다. 만약 다르다면 어떤 일이 벌어질까? 만약에 받는 쪽이 한글 완성형 코드 표를 사용하고 있고 저 데이터가 한글이라고 착각하면 "ABCD"가 아니라 "좌쳐"라는 문자열로 해석된다.[56] 그래서 멀쩡한 메시지를 보냈는데도 앞의 사례처럼 알아볼 수 없게 다 깨져 보이는 사태가 벌어지는 것이다. 이러한 문제를 해결하기 위하여 상당수의 인터넷 프로토콜에서는 보내려는 데이터가 어떤 문자표를 이용하는지를 표시해준다.

```
<!DOCTYPE html>
<html lang="ko" dir="ltr" class="client-nojs">
<head>
<meta charset="UTF-8" /><title>MIME - 위키백과, 우
<meta name="generator" content="MediaWiki 1.23wmf8
<link rel="alternate" type="application/x-wiki" ti
<link rel="edit" title="편집" href="/w/index.php?t
<link rel="apple-touch-icon" href="//bits.wikimedi
<link rel="shortcut icon" href="//bits.wikimedia.o
<link rel="search" type="application/opensearchdes
<link rel="EditURI" type="application/rsd+xml" hre
```

그림 43 어떤 웹 사이트의 원문 일부

위의 그림은 위키 백과의 한 페이지를 (우리가 보통 보는 모습이 아니라) 원문(source)으로 본 것이다. 웹 브라우저는 실제 통신을 통하여 저렇게 생긴 원문을 받아서 우리가 보기에 편한 모양으

---

[55] 숫자 옆의 아래 첨자 표시는 진법을 나타낸다. 진법 표시가 없는 경우는 대부분 10진법이다.

[56] 실제로 이런 일이 벌어지려면 데이터가 8비트가 아니라 7비트로 전송되어 각 바이트의 최상위 비트(8번째 비트)가 손실되는 환경이라는 가정 등이 성립해야 하는데 여기서는 설명이 너무 복잡하므로 깨지기 위한 가정이 모두 *운 나쁘게도* 성립했다고 가정한다. 그리고 글자가 깨지는 조건이 다양하므로 어떤 조건이 어떻게 결합되느냐에 따라 깨진 결과도 달라진다.

로 가공하여 보여주는 역할을 한다. 넷째 줄을 보면, **`charset="UTF-8"`** 이라고 되어 있는 부분이 있는데 이것이 어떤 문자 코드 표를 쓸 것인지를 지정하는 것이다.[57] 여기서는 전세계 거의 대부분의 글자를 다 표현할 수 있는 유니코드(Unicode)를 코드 표로 사용하겠다는 뜻이다. 요즘은 컨텐트도 세계화되고 있어서 유니코드가 가장 흔히 쓰인다. 만약 저기에 UTF-8 대신 ASCII를 써주게 되면 웹 페이지 내용 중 영어(라틴 글자)와 숫자가 아닌 글자는 대부분 깨져서 나오게 될 것이다.

코드 표만 맞춰주면 문제가 없을까? 대부분의 경우에는 그렇다. 특히, 요즘의 컴퓨터 환경에서는 특히나 더 그렇다. 하지만 인터넷이 만들어지던 초창기에는 컴퓨터 기술도 역시나 초창기였기 때문에 한계가 많았다. 예를 들어, 대다수의 컴퓨터는 한 바이트의 데이터 중 맨 위의 비트는 버리고 7비트만 처리하였다. 따라서, 이런 환경에서 맨 위의 비트까지 써야 하는 메시지(예를 들어, 한글처럼 7비트로 표현되지 않는 글자나 그림이나 음악 파일처럼 임의의 비트 덩어리인 경우)를 전송하려면 메시지를 7비트로 표현 가능한 형태로 바꾸어야 했다. 이렇게 전송을 위한 부호화를 컨텐트 전송 부호화(C-T-E, Content-Transfer-Encoding)이라고 부른다. 예를 들어, **한글**이라는 두 글자를 대표적인 C-T-E인 base64 방식으로 부호화하면 **`JiM1NDYyMDsmIzQ0NTQ0Ow==`** 가 된다.

마임(MIME, Multipurpose Internet Mail Extensions)은 전자 우편을 보낼 때 편지의 내용을 표현하는 형식이다.[58] 전자 우편에서 내용을 잘 담으려면 어떤 기술이 필요할까? 우선 앞에서 얘기한 charset 이나 C-T-E 는 당연히 있어야 할 것이고 그 외에도 첨부 파일을 처리할 수 있어야 한다. 첨부 파일을 처리하기 위해서는 본문과 첨부 파일을 구분할 수 있어야 하고 첨부 파일이 여러 개여 되더라도 각각의 구분할 수 있어야 한다. 따라서, 하나의 메일 메시지에서 **각 부분을 구분하는 기능**이 필요하다.

또한 요즘은 메일의 본문에서 글자에 **강조 표시**를 하거나 **색깔**을 넣는 경우가 있는데 어떤 메일 프로그램은 이러한 풍부한 표현을 처리하지 못할 수도 있으므로 그냥 알맹이 문자로만 된 메일과 (웹에서 사용하는 HTML 형식으로 된) 풍부한 표현의 메일을 각기 별개의 부분으로 만들어 하나의 메일에 묶어서 보낸다. 여러 부분으로 구성된 하나의 메일 메시지에서 어떤 부분은 메일 본문이고 어떤 부분은 첨부 파일이고 또 첨부 파일인 경우에는 **파일의 형식**이나 **이름** 등을 담는 기능 등을 MIME이 제공한다.

아래의 그림은 아주 짧은 본문과 아주 작은 그림 파일 하나를 첨부한 메일의 예시다. 첫 줄을 보면 **`multipart`** 라고 나오는데 이는 이 메일 메시지가 여러 부분으로 구성되어 있다는 의미다. 그 줄의 맨 뒤를 보면 각 부분의 구분을 나타내는 표시가 **`047…3a3`** 라는 것을 알려준다. 이로부터

---

[57] 사실 UTF-8은 코드 표로 유니코드를 사용한다는 것 외에도 어떻게 부호화할 것인지도 포함하고 있지만 여기서는 설명을 편의를 위해 생략한다. 어쨌든 같은 유니코드 표를 쓰더라도 여러 가지 다른 부호화 방식(예를 들어, UTF-16, UTF-32 등)을 쓸 수 있다는 정도는 기억해두자. 물론 그 중에서 UTF-8이 현재로서는 대세로 자리잡았다.

[58] MIME에 해당하는 적절한 한국어 번역은 찾지 못하였다. 직역해보면 **다목적 인터넷 우편 확장**이 되는데 뭔가 자연스럽지 못하다. 그냥 MIME이라고 쓰고 **마임**이라고 읽자.

(빈 줄은 빼고) 다음 줄부터 그림의 2/3 지점까지 그리고 거기부터 맨 아랫줄까지 두 부분으로 구성되어 있음을 알 수 있다.

첫째 부분을 보면 다시 **multipart** 라고 되어 있어서 여러 개의 부분으로 구성되어 있음을 알 수 있는데 그 중 첫째 부분은 컨텐트의 형식(Content-Type)이 보통 문자(text/plain)이고 둘째 부분은 HTML(text/html)이다. 앞서 설명한 것처럼 이 둘은 모두 메일 본문이며 하나는 그냥 알맹이 텍스트만 들어 있고 나머지 하나는 HTML 형식으로 같은 내용이 들어 있다. 그리고 이들 메시지 본문은 유니코드를 사용하며(UTF-8) 컨텐트 전송을 위한 부호화 방식은 **base64**를 사용하고 있다.

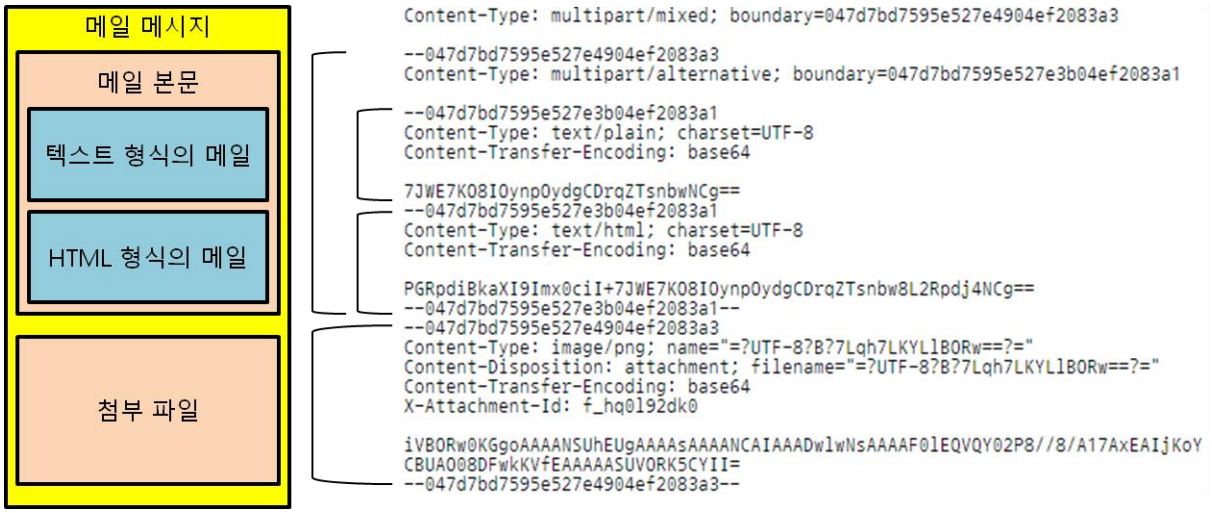


그림 44 아주 작은 첨부파일을 가진 아주 짧은 메일의 원문

첨부 파일은 둘째 부분에 들어 있는데 컨텐트의 형식은 PNG형식의 그림 파일이며(image/png) 파일 이름은 **캡처.PNG**다. (파일 이름도 UTF-8 + base64 로 부호화 되어서 **=?UTF-8?B?7Lqh7LKYLlBORw==?** 라고 표현되어 있다.)

일반 사용자는 평생 볼 일이 없는 이런 다소 너저분하고 복잡해 보이는 방식을 사용함으로써 다양한 메일을 다양한 환경에서 자연스럽게 보내고 받을 수 있게 되는 것이다. MIME은 애초에 전자 메일을 위해서 개발되었지만 그 외에도 임의의 복합적인 데이터를 전송해야 하는 환경에서 활발히 사용되고 있다.