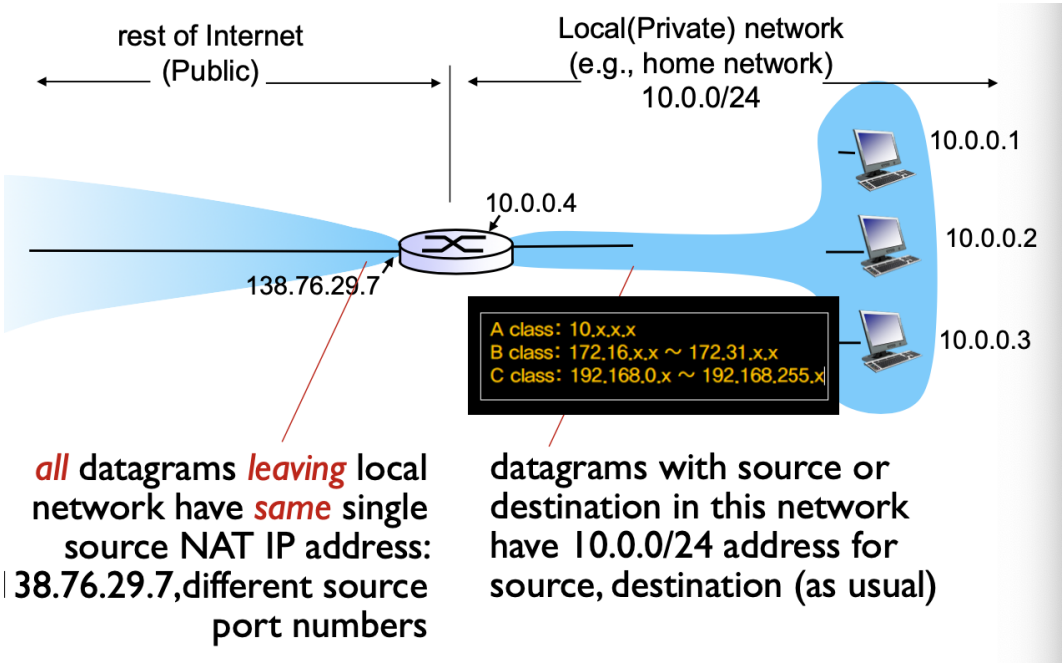
7일차

Network Layer

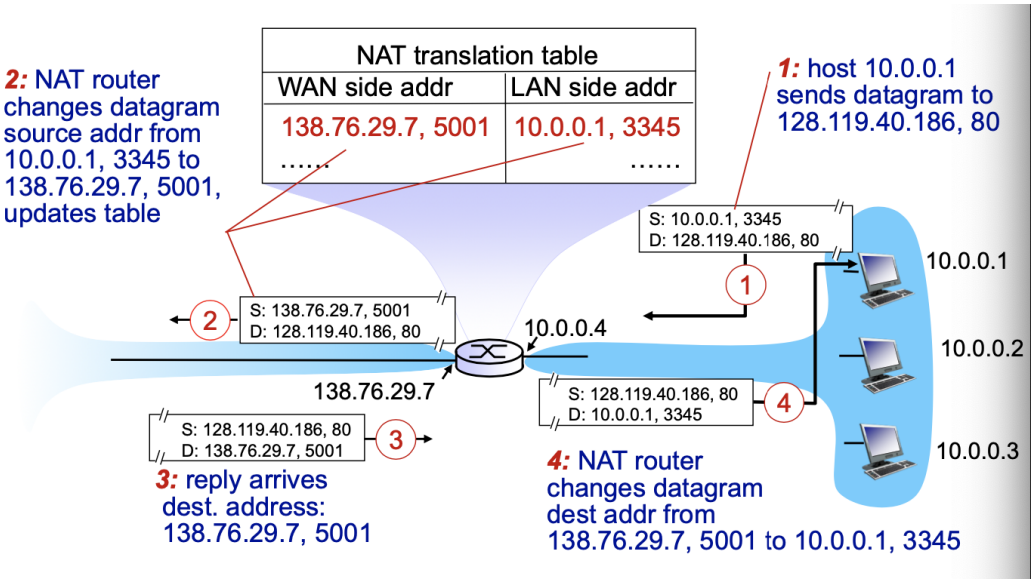
1. NAT : 여러개의 로컬네트워크가 외부와 연결할때 하나의 IP로 네트워크 트래픽을 주고 받는 기술
    a. 장점
        i. subnet 내부의 모든 device를 위해서 ISP가 단 하나의 IP 만 제공해도 된다.
        ii. outside world에 알리지 않고도 local network 내부의 host address를 변경할 수 있음
        iii. local network 내부의 device 주소 변경 없이 ISP 만 바꿀 수 있다
        iv. 외부에서 볼 때는 local address 를 확인 할 수 없기 때문에 보안적인 측면에서 좋다.

    → IPv4 의 IP 갯수 한계 때문에 이를 효과적으로 쓰기 위해서 사용

    b. 구조



        i. 같은 IP 를 사용하고 host 마다 다른 port 를 이용함



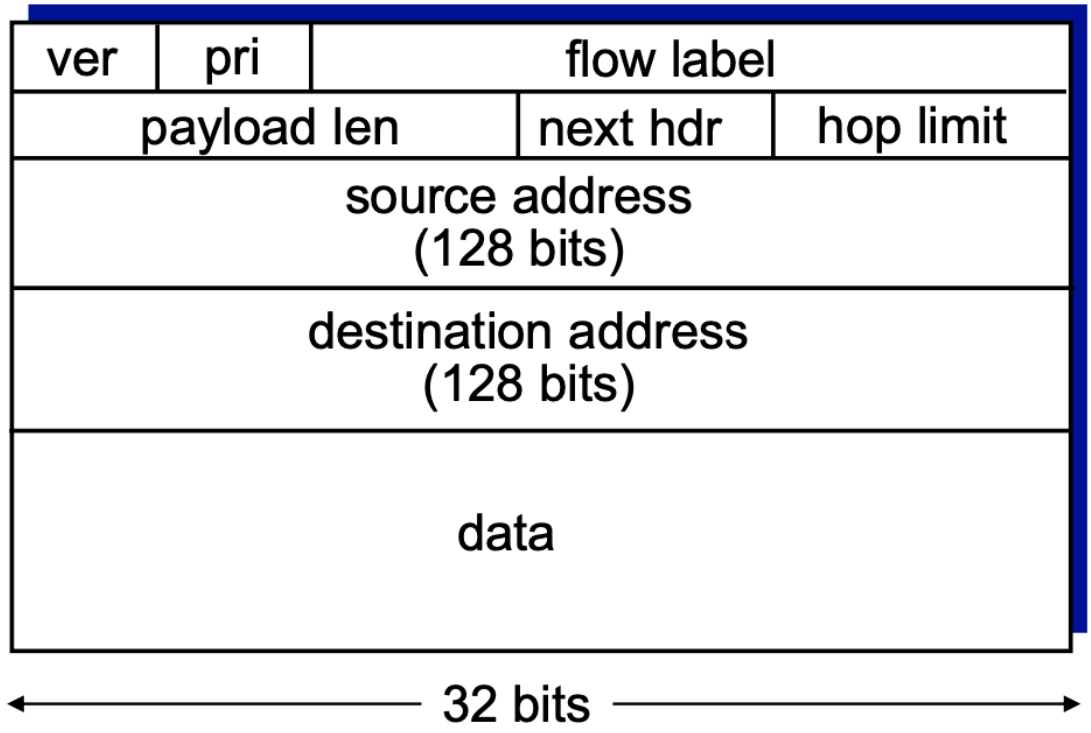
ii. local 3345 포트 → NAT 5001 포트 이런식으로 포트마다 길을 만들어줌

iii. NAT table 을 이용하기 때문에 외부에서 내부로 먼저 접근 불가!

IPv6 / IPv4

1. 기존에는 IPv4를 사용하였는데 32비트 주소가 곧 고갈 날 것이다 → IPv4 대신 IPv6 사용하지만 아직 활성화가 많이 되지는 않았음

2. IPv6 Datagram format



i. priority : datagram의 우선순위

ii. flow Label : datagram이 길어서 쪼갰을때 어떤 flow 에 속해 있는 건지 표시

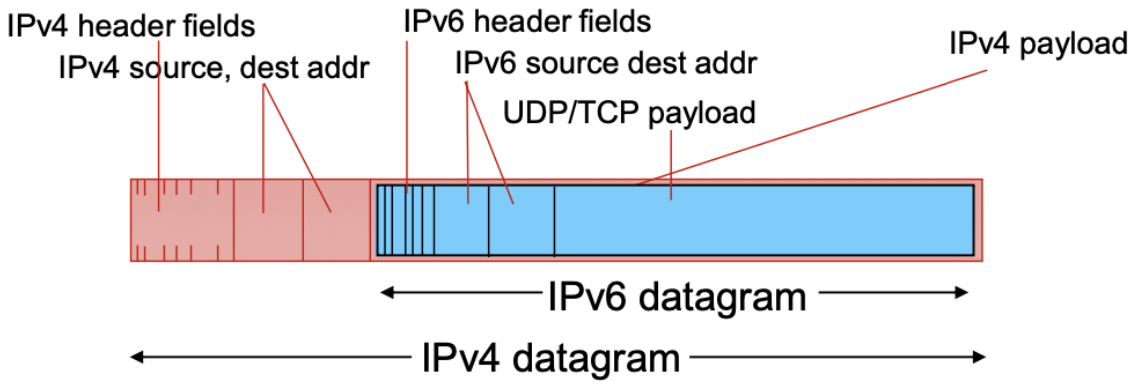
iii. next header : 상위계층이 프로토콜이 무엇인지 식별 (TCP? UDP?)

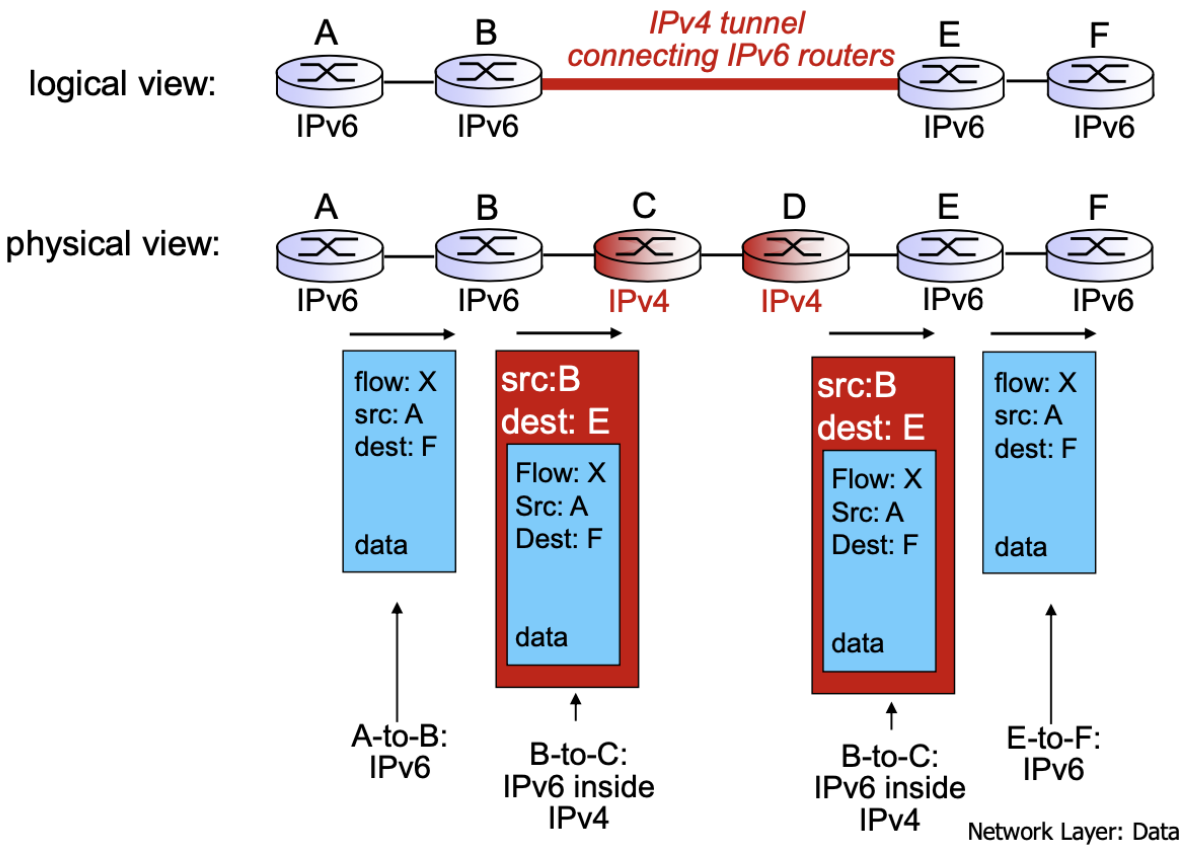
iv. address 가 32bit(IPv4) 에서 128 bit 로 늘어남. (더 많은 IP 주소 표현 가능)

v. IPv4와 다른점 : checksum, options, fragmentation 삭제

3. Transition from IPv4 to IPv6

a. tunneling : 기존 IPv4 를 IPv6로 바꾸려면 변환하는 것이 필요한데 (모든 router를 한번에 바꾸기는 불가능하기 때문에)
IPv6 라우터가 IPv4를 만나면 IPv4 header 를 추가해주고 IPv4가 IPv6를 만나면 header 를 떼어주는 식의 방법이다.





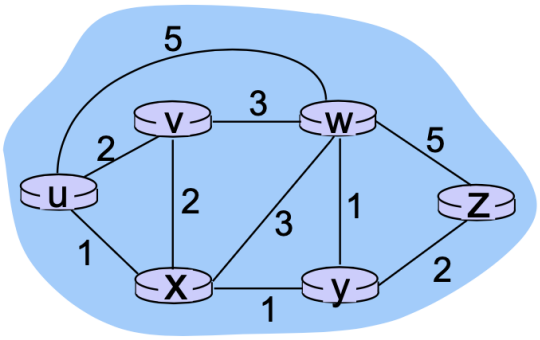
SDN

나중에 나옴 skip!

쭉 skip!

Routing protocols

1. Graph abstraction of the network



graph: G = (N,E)

N = set of routers = { u, v, w, x, y, z }

E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

그래프 형태로 표현 할 수 있다.

a. Graph abstraction : costs
    i. 네비게이션을 생각 할 때 시간을 빨리가거나 통행료를 적게쓰는 길로 가거나 등 → costs의 관점은 다 다르다.
    ii. network 의 costs 관점 → distance, error(차 사고 났으면 돌아가야지), state(환경을 봐야된다), 비용
b. Routing algorithm classification (우리가 배우는건 둘다 static 이 아닌 dynamic)
    a. global : 전체를 보고 vs decentralized : 단순 거리만을 보고
    → link state algorithms(Dijkstra's algorithm) vs distance vector algorithms
c. A link-state routing algorithm
    a. Dijkstra's algorithm + ppt notation 도 이해하자
    c(x,y) : x에서 y로 갈 때 cost, 없으면 무한대
    D(v) : source에서 destination 까지 현재 cost value
    p(v) : 이전 node 가 뭔지
    N' : 가장 최소 cost 로 갈때 node를 모두 쓴 것

notation:

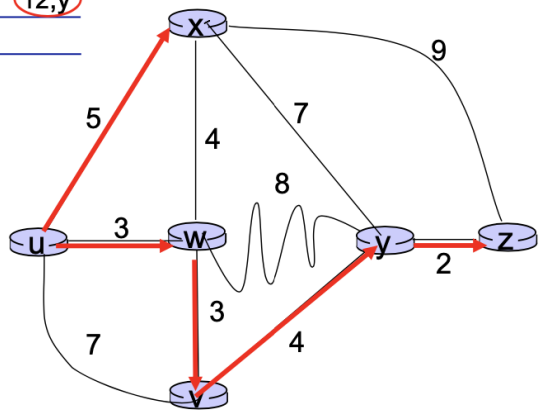
- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

i. 시작노드에서 모든 노드들에 대한 비용을 구한다.

Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

notes:

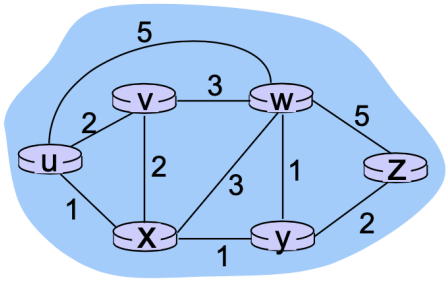
- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)



ii. oscillations possible : 반복 상황 발생 가능

i. C에서 A로 가려는데 갈때마다 상황이 바뀌어서 C → B - > C - > D → C 등의 반복되는 상황이 있을 수 있음.

iii. Distance vector algorithm = Bellman-Ford equation = dynamic programming

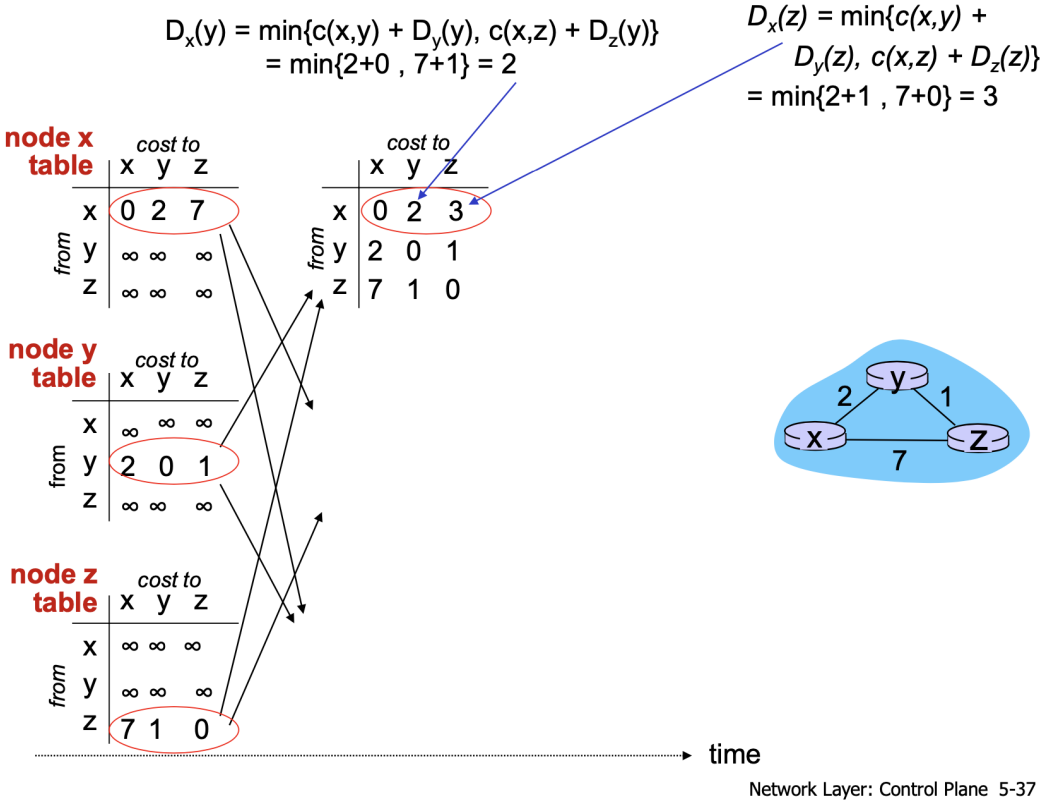


clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z), \; c(u,x) + d_x(z), \; c(u,w) + d_w(z) \}$$
$$= \min \{2 + 5, \; 1 + 3, \; 5 + 3\} = 4$$

1. (이웃노드로 가는 비용 + 이웃에서 목적지로 가는 비용) 에서 최소값
2. u 기준 x,v,w 로 갈 수 있는데 그 거리 + 각각 x,v,w 에서 z 까지 의 거리를 더해서 그것들 중 최솟값을 구한다.



Network Layer: Control Plane 5-37

node table 을 그려서 계산 할 수 도 있음.