6 네트워크 계층2

IP fragmentation, assembly

지나가는 길마다 최대로 보낼 수 있는 사이즈가 있음(maximum transport unit)

ex) 4000 byte → 20 byte의 IP header를 가짐

⇒ 3980 byte의 payload → 이 부분을 잘라서 보내야 함

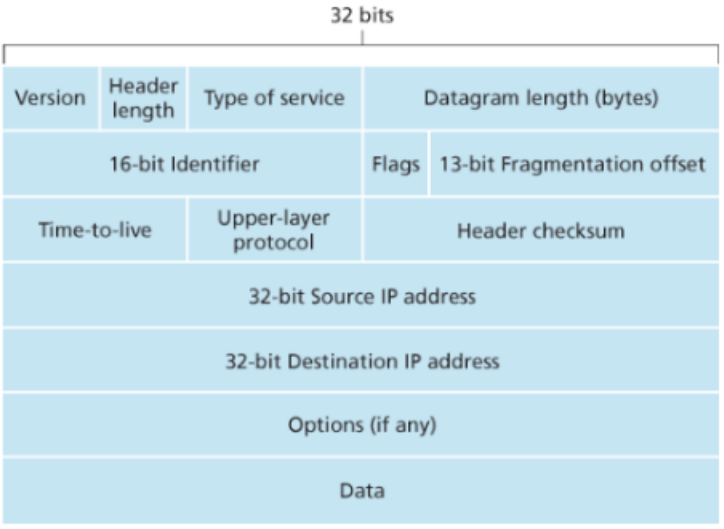
1500 byte가 최대라면 20 byte의 header를 붙여야 하니까 한 번에 1480 byte를 잘라낼 수 있음

IPv4 주소체계

IP주소 : 32 bit로 호스트와 라우터의 인터페이스에 할당

인터페이스 → 호스트와 라우터 사이, physical link 사이 연결 제공

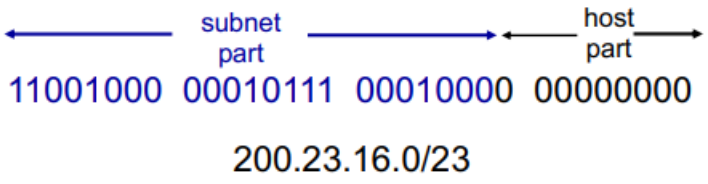
- 라우터는 여러 개의 인터페이스를 가짐
- 호스트는 주로 하나 아니면 두 개를 가지고 있음



IP 주소 상에서 24bit까지는 고정, 마지막 8 bit가 바뀌면서 identifier 역할을 함

~ 고정된 값 : subnet part / 변경되는 값 : host part / interface part

→ 어디까지가 subnet part 이고 host part인지는 IP 주소 설정할 때 정함 → subnet mask

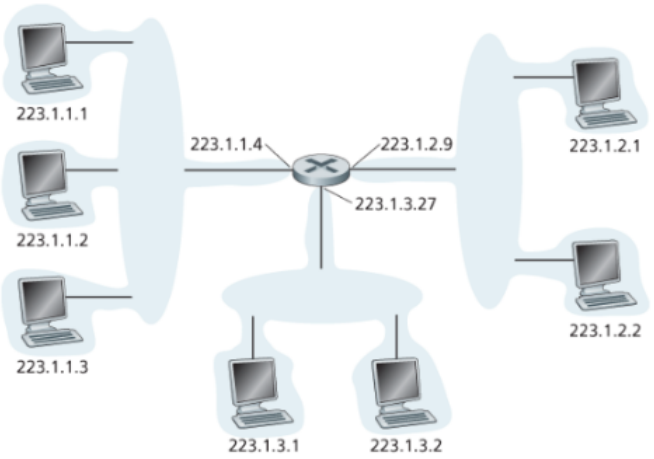


subnet

: 같은 subnet part를 갖는 interface들의 모음

→ subnet끼리는 router가 관여하지 않아도 물리적으로 도달 가능

→ subnet 결정 방법 : host에서 interface를 떼어낸 다음 isolation되는 하나하나가 subset이 됨



→ 3개의 subnet 존재

CIDR (Classless InterDomain Routing)

IP 주소에서 subnet portion의 길이가 정해져 있지 않음

a.b.c.d/x → x는 처음 x bit 만큼이 subnet part

IP주소 설정 방법

1. 고정된 IP주소 사용
2. 네트워크에서 제공하는 IP 주소 사용

DHCP (Dynamic Host Configuration Protocol)

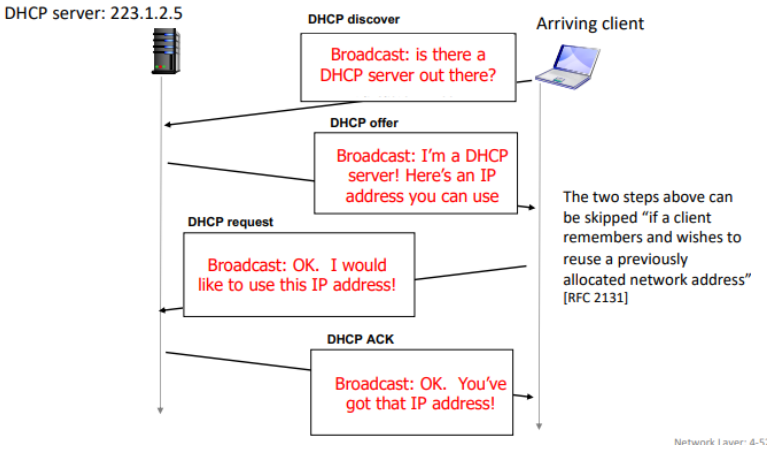
켜지면 IP 주소를 받고 꺼지면 IP 주소를 반납

→ 호스트가 네트워크 서버로부터 네트워크에 조인할 때 IP 주소를 서버로부터 받음

- 사용하고 있는 서버를 더 오랫동안 사용하겠다 할 수 있음
- IP 주소를 재사용

- 네트워크에 조인한 mobile user를 서포트하기도 함

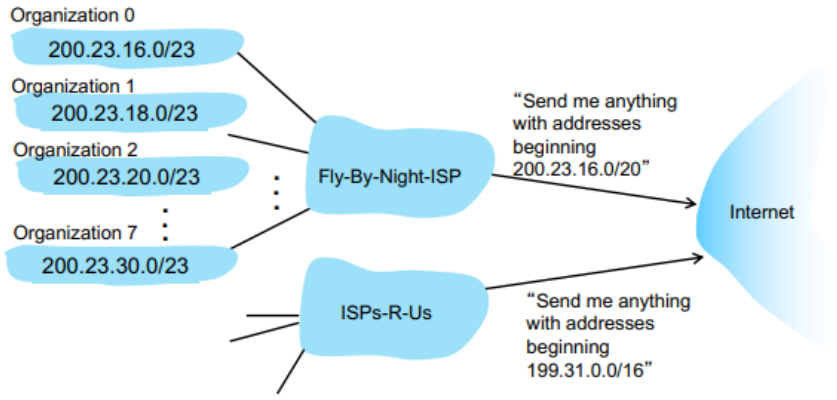
1. 호스트가 네트워크에 조인하면 "DHCP discover"메세지를 보냄
2. DHCP가 이 메세지를 보고 어떠한 IP 주소를 사용할 수 있다는 "DHCP offer"메세지를 보냄
3. 호스트가 해당 IP 주소를 사용하겠다는 "DHCP request" 메세지를 보냄
4. DHCP 서버가 "DHCP ack" 메세지를 보내면 해당 IP 주소를 사용할 수 있음



DHCP는 IP 주소 뿐만 아니라 수동으로 IP 주소 설정시 넣었던 값들도 받아옴

Hierarchical addressing : routing aggression

전체 address space가 존재 → ISP로 이동 → ISP에서 회사 1, 2, 3, ...에 나눠줌



ICANN (Internet Corporation for Assigned Names and Numbers)

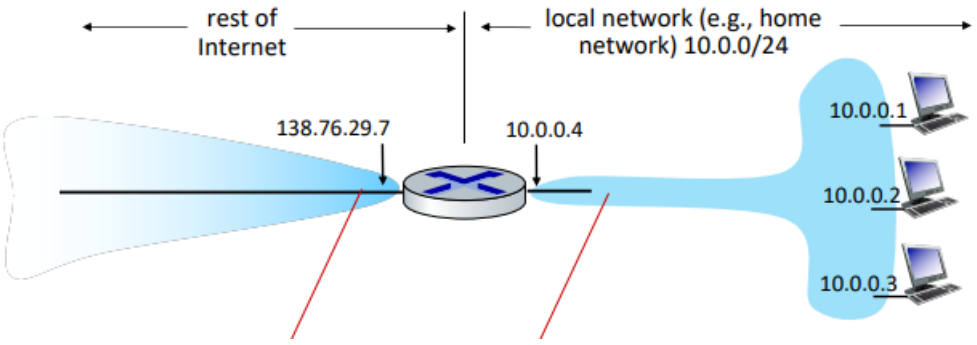
이 회사에서 IP 주소 할당, DNS 관리, 도메인 네임 관련 분쟁 해결 등

NAT (Network Address Translation)

유무선 공유기에 물려서 받는 IP주소는 그 네트워크 안에서만 유효함 → public IP address가 아님

→ public IP address 받으면 그 안에서 공유기에 여러 호스트가 물릴 수 있음

⇒ IP 주소를 효율적이게 사용하기 위해서 로컬 네트워크 안에서만 인식되는 IP 주소를 사용



오른쪽에 있는 10.0.0.1을 사용하지만 나갈 때는 왼쪽의 138.76.29.7의 주소를 사용

들어올 때 어떻게 forwarding하냐가 NAT

NAT router가 해야 할 일

- outgoing datagram에 대해서는 source IP 주소와 port 번호에 대해서 NAT IP 주소와 새로운 port 번호로 바꿔줘야 함
- NAT translation table을 기억하고 있어야 함
- ingoing datagram에 대해서 NAT table을 보고 forwarding 해줌

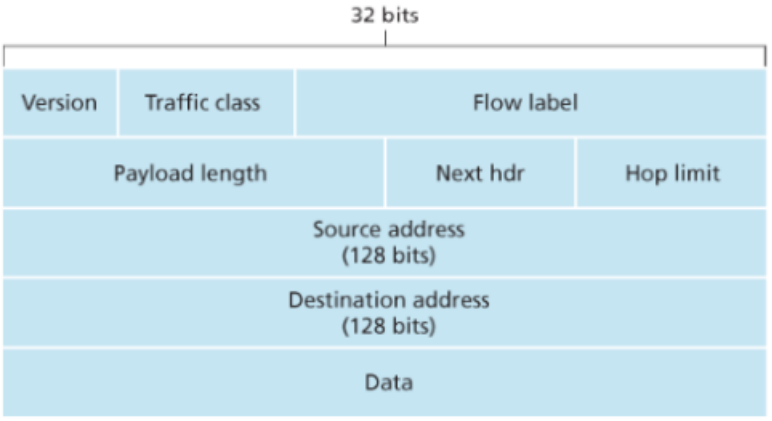
16bit port number field가 존재 → 6만개의 connection을 열 수 있음

NAT router가 port number까지 까서 보기 때문에 layer 3까지 봐야한다는 철학에 어긋남

IPv6

IPv4의 32 bit 주소 공간이 부족함

IPv6의 데이터그램의 포맷은 40 byte header로 고정되어 있음



- priority : 데이터그램들 중에 priority를 identify할 수 있는 것
- flow label : 같은 flow에 있는지 identify
- next header : upper layer protocol identify → TCP segment를 data field에 가지고 있으면 TCP가 적힘 → IPv4의 upper layer와 마찬가지

⇒ IPv4보다 간결해짐

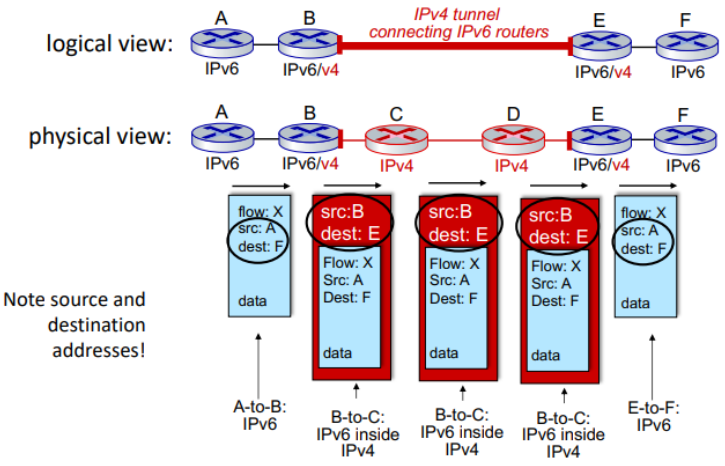
IP주소 128 bit로 늘림

IPv4와 달라진 점

- checksum이 없어짐
- options → header 바깥에 있음
- ICMPv6

IPv4를 서포트하는 router와 IPv6 서포트하는 router가 섞여있으면 **tunneling**

→ IPv6 데이터그램이 IPv4에 payload로 들어감

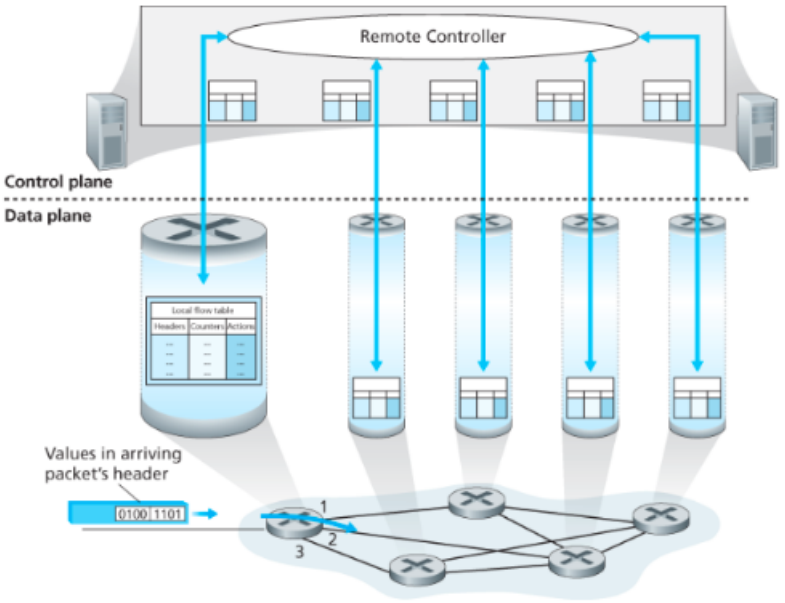


Generalized forwarding and SDN

→ 여러 개의 포트로 forwarding할 수도 있음

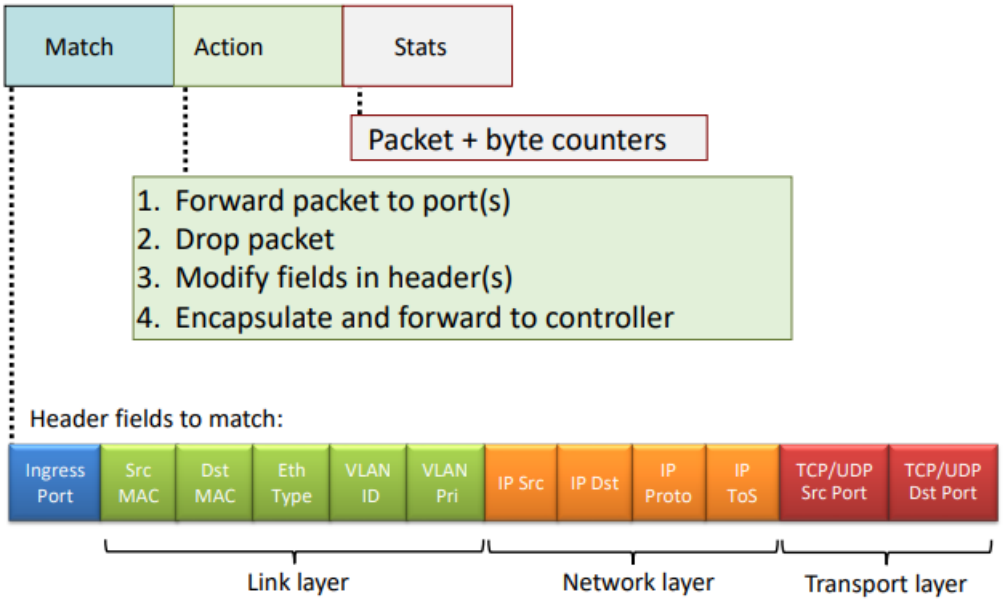
각각의 router가 flow table을 가지고 있음

→ centralized routing controller가 계산하고 분배한 flow table



- flow : header field에 의해서 정의됨
- generalized forwarding : simple packet-handling rules
  - pattern : packet header field에 있는 값들을 매칭해서 액션을 취함
  - action : matched packet을 controller로 보내라 등...
  - priority : overlapping pattern에 대해서 명확하게 함
  - counters

flow table이 router의 match + rule을 결정, 정의함



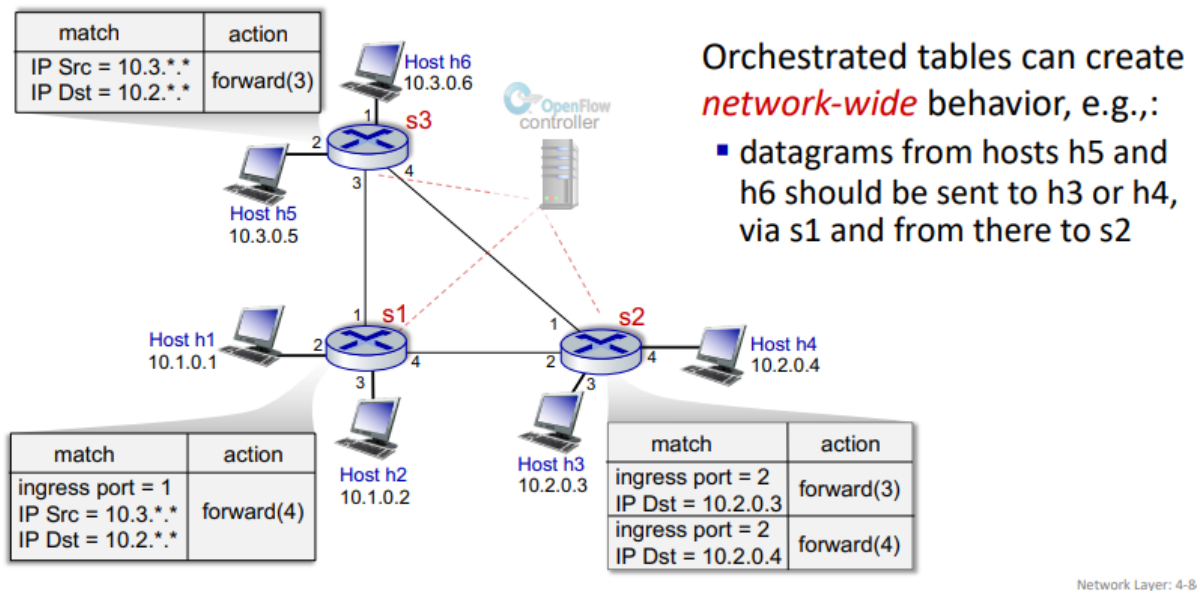
→ rule의 field들을 보고 matching을 함 → 특정 액션을 취함

⇒ TCP/UDP도 보고 해당 layer만 보는 layering concept에 위배됨

overflow abstraction

match + action 아래 다 포함

- Router
  - match : longest destination IP prefix
  - action : forward out a link
- Switch
  - match : destination MAC address
  - action : forward or flood
- Firewall
  - match : IP address and TCP/UDP port numbers
  - action : permit or deny
- NAT
  - match : IP address and port
  - action : rewrite address and port



network layer function

- data-plane (forwarding)

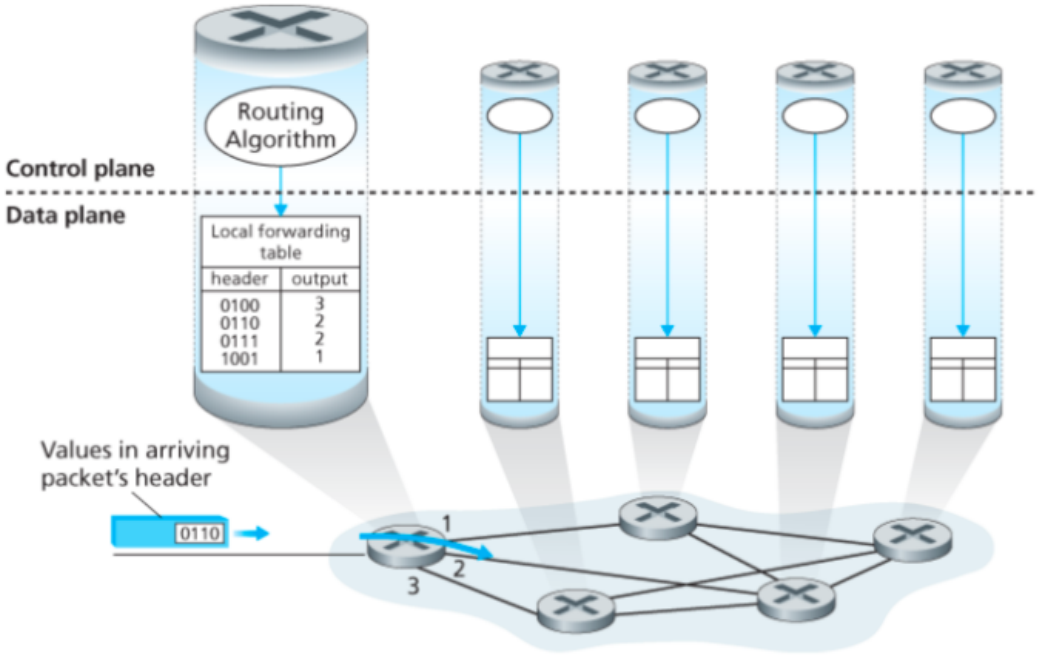
  → 하드웨어쪽 ~ 각각의 router에서 어느 port로 나가야할지 결정

- control-plane (routing)

  → 하드웨어를 어떻게 동작시킬지 ~ packet이 source address에서 destination address로 갈 때 어떻게 갈지

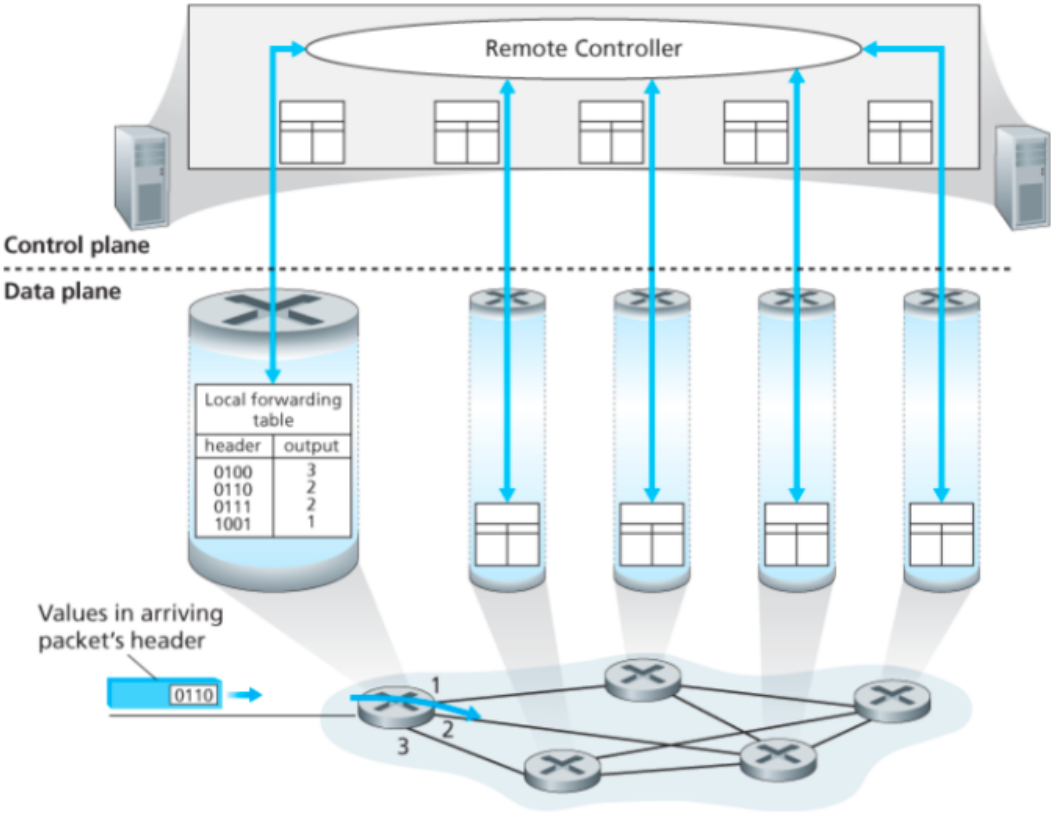
  - per-router control (전통적인 방법)

    각각의 router에 control 정보들이 있고 그 정보를 router들끼리 interact해서 주고받으며 router의 정보를 업데이트



- logically centralized control (SDN)

  centralized controller가 있어서 거기서 계산한 routing 정보 같은 것이 각각 router에 전달됨



routing protocol

source에서 destination까지 좋은 경로로 가는 길 찾기가 목표

→ 다익스트라 / 벨만포드 등이 쓰임

Graph abstraction : costs

routing algorithm → 어떤 노드에서 다른 노드로 가는 cost가 최소가 되는 경로를 찾는 것

routing algorithm classification

- global information

  모든 router가 전체 network topology, 그래프를 가지고 있는 것
- decentralized information

  전체가 아닌 단순히 이웃 노드 정보만을 가지는 방법
- static

  경로가 천천히 변함
- dynamic

  경로가 빨리 변함

  → time scale을 말함

routing algorithm

1. **link state**

다익스트라 알고리즘

: 모든 노드에게 network topology, link cost가 알려져 있음, 모든 노드가 동일한 정보 가지고 있음

→ 자기 노드에서 모든 노드로 가는 경로를 구함 ⇒ forwarding table을 만듦

c(x, y) : x에서 y로 가는 link cost

D(v) : 현재까지 계산된 노드 v까지 계산된 경로 cost

P(v) : v로 가는 경로 상에서 v 바로 이전 노드

N′ : 현재까지 알려진 최단 경로 destination

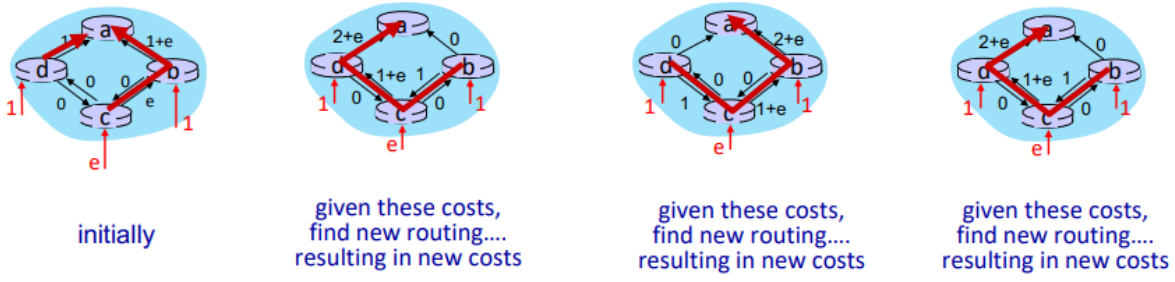
```
1 Initialization:
2   N' = {u}                    /* compute least cost path from u to all other nodes */
3   for all nodes v
4     if v adjacent to u        /* u initially knows direct-path-cost only to  direct neighbors  */
5       then D(v) = c_{u,v}     /* but may not be minimum cost!                                  */
6     else D(v) = ∞
7
8  Loop   네트워크에 있는 node의 개수
9     find w not in N' such that D(w) is a minimum
10    add w to N'
11    update D(v) for all v adjacent to w and not in N' :
12       D(v) = min ( D(v),  D(w) + c_{w,v} )
13    /* new least-path-cost to v is either old least-cost-path to v or known
14    least-cost-path to w plus direct-cost from w to v */
15 until all nodes in N'
```

loop → 노드 개수만큼 돔

algorithm complexity → $n(n+1)/2$번 비교 ⇒ $O(n^2)$

~ 더 효율적으로 계산하면 ⇒ $O(nlogn)$

oscillations possible



2. **distance vector**

<u>벨만포드 알고리즘</u>

$d_x y$: x로부터 y까지로 가는 최단 경로의 cost

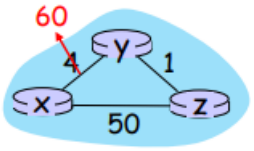
= $min_v(c(x,v) + d_v y)$

$D_x y$: x에서 y로 가는 최소 cost의 값 (현재까지 계산된)

distance vector $D_x = [D_x y : y \in N]$

- iterative, asynchronous → local link cost가 바뀜, 이웃으로 업데이트 정보가 퍼지고 전달함
- distributed

각각의 노드는 이웃으로부터 cost나 msg을 받으면 자신의 distance vector를 다시 계산, 변화있으면 이웃 알림



link cost가 바뀌면 자기 자신의 table을 업데이트 → 이웃들에게 보냄

loop이 생기는 문제가 생김 ⇒ count infinity problem

→ 이 문제를 해결하기 위해 poisoned reverse 방법 사용

⇒ z가 y를 거쳐서 x를 갈 때 z는 y에게 x로 가는게 ∞라고 알려줌

둘의 비교

- message complexity
  - LS : $O(nE)$
  - DV : 이웃들 간 교환이 끝임, convergence time이 상황에 따라 달라질 수 있음
- speed of convergence
  - LS : $O(n^2)$
  - DV : convergence time의 변화

만약 router가 오작동하면?

- LS : 자신의 테이블 값 바꿈
- DV : 이웃에게 계속 잘못 전달되니까 network 전체가 propagate됨