

6주차_네트워크 계층 2

☀ 상태	진행 중
📌 강의	CS 스터디
📅 작성 일	@2024년 4월 22일
☰ 자료	https://github.com/IT-Book-Organization/Computer-Networking_A-Top-Down-Approach

4.3.2 IPv4 주소체계

호스트는 일반적으로 네트워크와 연결되는 하나의 링크를 갖는다.

호스트 IP가 데이터그램을 보낼 때 이 링크를 통해 데이터 링크를 보낸다.

IP 주소 : 32-bit로 호스트, 라우터, 인터페이스를 위한 identifier다.

→ Interface에 할당되는 것!!!

호스트와 물리적 링크 사이의 경계를 인터페이스(interface)라고 부른다.

라우터는 여러개의 링크와 연결되고, 링크와 라우터 사이도 인터페이스(interface)로 이루어져 있어 여러개의 인터페이스(interface)를 갖는다.

모든 호스트와 라우터는 IP 데이터그램을 송수신할 수 있으므로 IP는 각 호스트와 라우터 인터페이스가 IP 주소를 갖도록 요구한다.

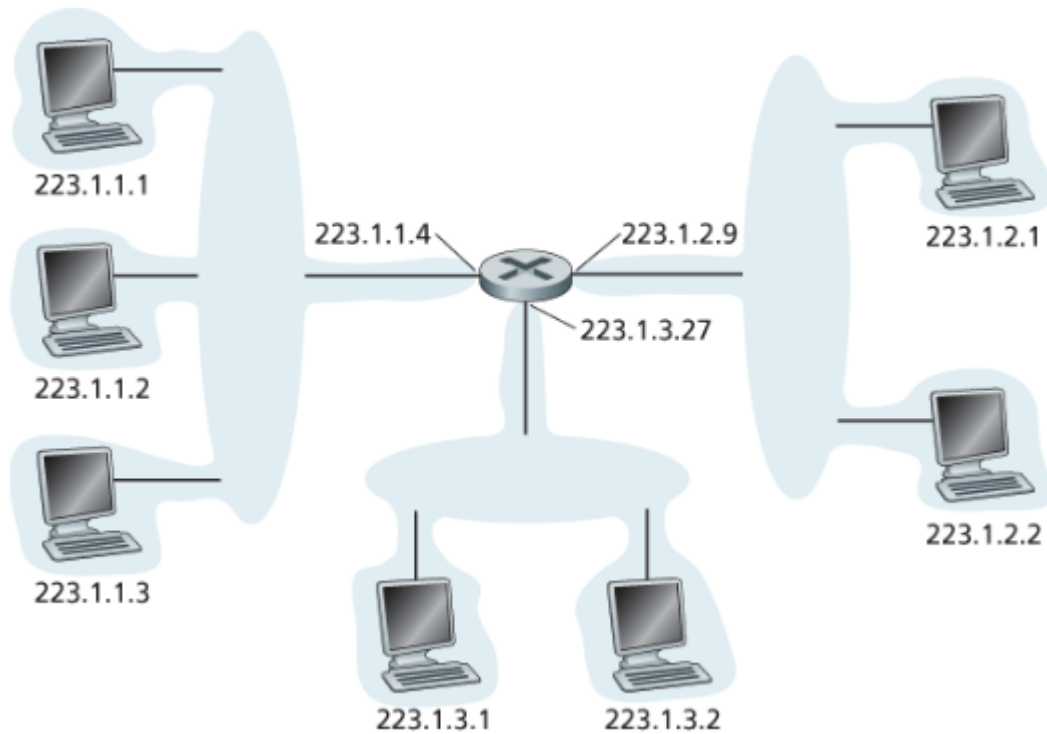
이러한 각 인터페이스는 고유한 IP 주소를 갖는다.

따라서 기술 면에서 IP주소는 인터페이스(interface)를 포함하는 호스트 라우터보다는 인터페이스(interface)와 관련이 있다.

서브넷과 IP 주소

각 IP 주소는 32비트 길이다. 따라서 2^{32} 개의 주소를 사용할 수 있다. 일반적으로 주소와 각 바이트를 십진수로 표현하고 주소의 다른 바이트와 점으로 구분하는 십진 표기법을 사용한다.

인터페이스의 IP는 마음대로 선택할 수 없다. IP 주소의 일부는 연결된 서브넷이 결정한다.



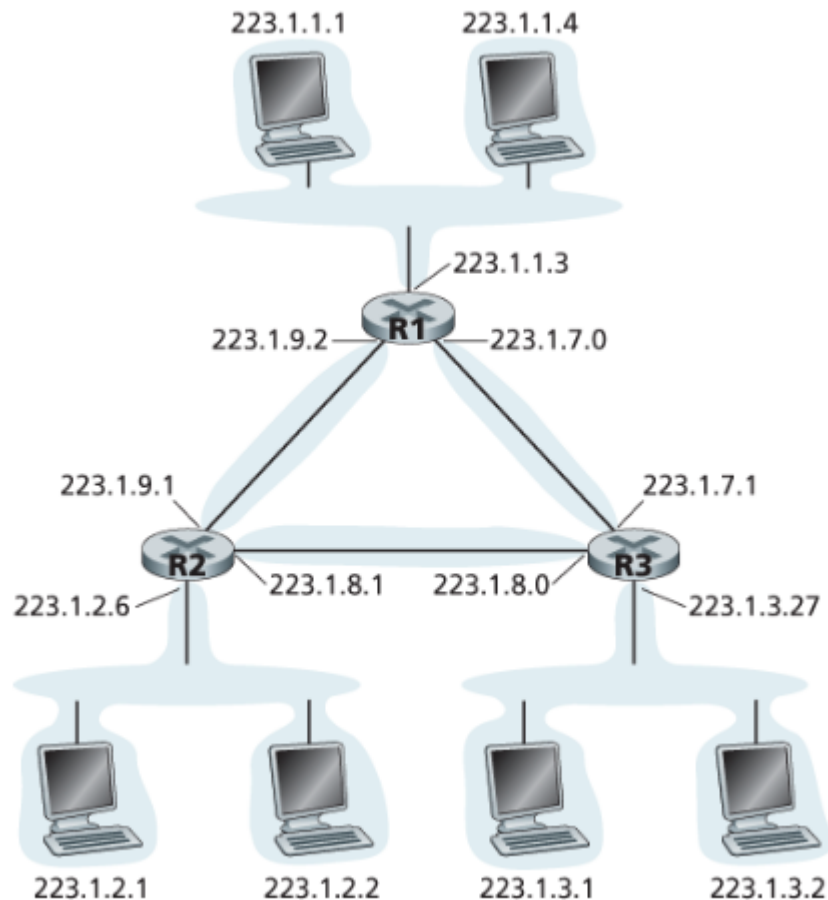
왼쪽 3개의 호스트와 라우터 인터페이스는 모두 223.1.1.xxx 형식의 IP 주소를 갖는다. 또, 4개의 인터페이스가 중계하는 라우터 없이 하나의 네트워크에 서로 연결되어 있다.

이 네트워크는 이더넷 LAN으로 상호 연결되고 이 경우 인터페이스는 이더넷 허브나 이더넷 스위치 또는 무선 AP로 상호 연결된다.

IP 용어로 세 호스트 들의 인터페이스들과 하나의 라우터 인터페이스로 연결된 네트워크는 서브넷(subnet)을 구성한다고 말한다.

IP 주소체계는 이 서브넷에 223.1.0/24 라는 주소를 할당하는데 여기서 /24는 서브넷 마스크라 부르는데, 왼쪽 24비트가 서브넷 주소라는 것을 가리킨다.

위 그림에서는 3개의 서브넷을 볼 수 있다.



서브넷 IP의 정의는 여러 호스트를 라우터 인터페이스에 연결하는 이더넷 세그먼트만을 의미하는 것은 아니다.

위 그림을 보면 3개의 라우터도 점대점으로 연결되어있는 것을 볼 수 있다.

여기서는 호스트와 라우터의 연결 뿐만 아니라 라우터 간의 연결에서도 서브넷을 볼 수 있다.

총 6개의 서브넷을 찾을 수 있다.

서브넷의 정의



서브넷을 결정하려면 먼저 호스트나 라우터에서 각 인터페이스를 분리하고 고립된 네트워크를 만든다. 이 고립된 네트워크의 종단점은 인터페이스의 끝이 된다. 이렇게 고립된 네트워크 각각을 서브넷이라고 부른다.

위에 의하면 다수의 이더넷 세그먼트와 종단 간의 링크를 갖는 기관은 한 서브넷에서 모든 장비가 같은 서브넷 주소를 갖는 그런 서브넷을 여러개 가질 수 있다.

서로 다른 서브넷은 다른 주소를 가져야 하지만 실제로 서브넷 주소는 같은 부분이 많다.

CIDR(Classless Inter Domain Routing)

인터넷 주소 할당 방식 중 하나

CIDR는 서브넷 주소 체계 표기를 일반화하고 있다.

a.b.c.d/x 형식의 주소에서 최상위 비트(Most significant bit)를 의미하는 x는 IP 주소의 네트워크 부분을 구성한다.

이를 해당 주소의 프리픽스 또는 네트워크 프리픽스 라고 부른다.

한 기관은 통상 연속적인 주소의 블록(공통 프리픽스를 갖는 주소 범위)을 할당받고 기관 장비들의 IP주소는 공통 프리픽스를 공유한다.

외부 기관의 라우터는 목적지 주소가 내부 기관인 데이터그램을 전달할 때, 단지 앞의 x 비트들만 고려한다.

a.b.c.d/x형태의 한 엔트리만으로 기관 목적지로 패킷을 전달하는 데 충분하므로, 이런 라우터들에서 포워딩 테이블의 크기를 상당히 줄여준다.

주소의 나머지 32-x 비트들은 기관 내부에 같은 네트워크 프리픽스를 갖는 모든 장비를 구별한다.

이 비트들은 기관 내부의 라우터에서 패킷을 전달할 때 사용되며 이 하위 비트들은 추가적으로 서브넷 구조를 가질 수도 있다.

클래스 주소체계

CIDR가 채택되지 전에는 IP 주소의 네트워크 부분을 8, 16, 24 비트로 제한해도 각각의 비트를 서브넷 주소로 갖는 서브넷을 각각 A, B, C 클래스 네트워크로 분류했기 때문에 이러한 주소체계는 클래스 주소체계라고 알려졌다.

그러나 서브넷 부분이 정확히 1, 2, 3 바이트여야 하는 요구사항은 중소형 크기의 네트워크로 급속히 증가하는 기관의 수를 지원하기엔 문제가 있었다.

예를 들어 클래스(/24) 서브넷은 254개의 호스트만을 제공하므로 많은 조직을 위해서는 턱없이 부족하고, 클래스(/16) 서브넷은 65534개의 호스트를 제공하여 너무 크다.

브로드캐스트 주소

IP의 또 다른 형태인 브로드캐스트 주소 255.255.255.255가 있다.

호스트가 목적지 주소가 255.255.255.255인 데이터그램을 보내면, 이 메시지는 같은 서브넷에 있는 모든 호스트에게 전달된다.

주소 블록 획득

기관의 서브넷에서 사용하기 위한 IP 주소 블록을 얻기 위해, 네트워크 관리자는 먼저 이미 할당받은 주소의 큰 블록에서 주소를 제공하는 ISP와 접촉해야 한다.

ISP's block:	200.23.16.0/20	<u>11001000 00010111 00010000</u>	00000000
Organization 0	200.23.16.0/23	<u>11001000 00010111 00010000</u>	00000000
Organization 1	200.23.18.0/23	<u>11001000 00010111 00010010</u>	00000000
Organization 2	200.23.20.0/23	<u>11001000 00010111 00010100</u>	00000000
Organization 7	200.23.30.0/23	<u>11001000 00010111 00011110</u>	00000000

ISP가 위와 같은 블록을 할당 받았다고 할 때, ISP는 블록을 다음처럼 같은 크기의 작은 주소 블록 8개로 나누고, 이것으로 8개 조직을 지원할 수 있다.

ISP는 비영리 단체인 **ICANN**으로 부터 주소 블록을 할당 받는다.

ICANN의 역할은 **IP 주소 할당**과 **DNS 루트 서버 관리**다.

호스트 주소 획득 : 동적 호스트 구성 프로토콜(DHCP:Dynamic Host Configuration Protocol)

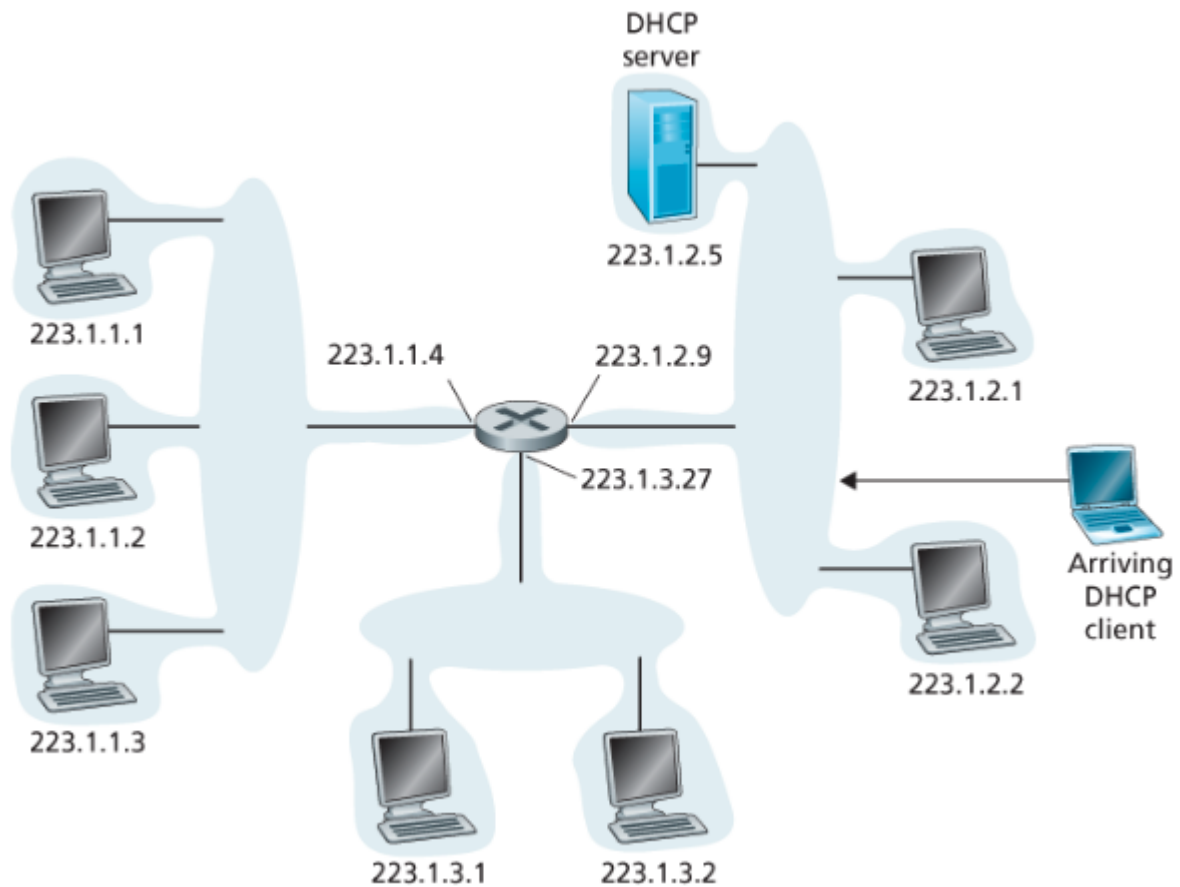
한 기관은 ISP로부터 주소 블록을 획득하여, 개별 IP 주소를 기관 내부의 호스트와 라우터 인터페이스에 할당한다.

라우터 인터페이스 주소에 대해, 시스템 관리자는 라우터 안에 IP 주소를 할당한다.

호스트에 IP 주소를 할당하는 것은 수동으로 구성이 가능하지만 일반적으로 동적 호스트 구성 프로토콜(DHCP)을 많이 사용한다.

DHCP는 호스트가 IP주소를 자동으로 얻을 수 있게 하고, 서브넷 마스크, 첫번째 홉 라우터 주소나 로컬 DNS 서버 주소 같은 추가 정보를 얻게 해준다.

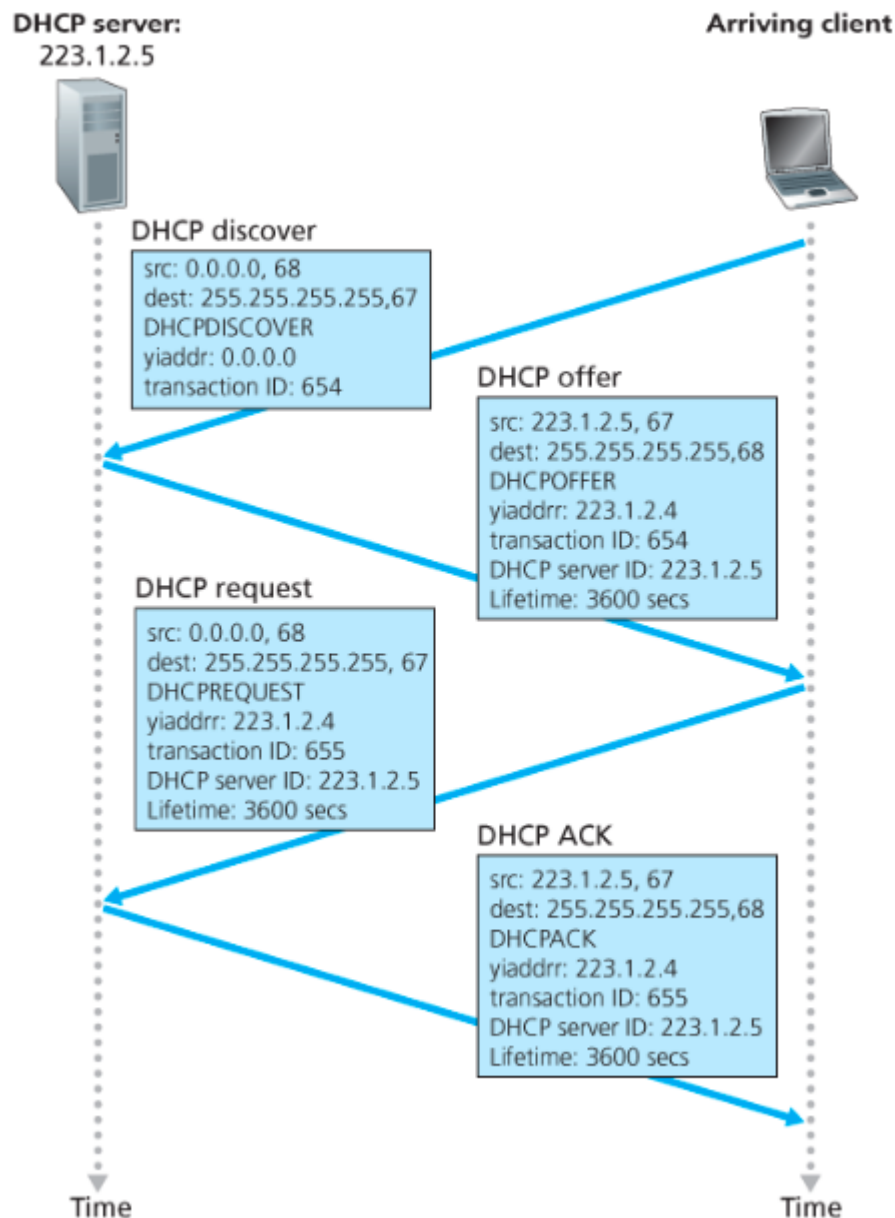
DHCP는 자동으로 호스트와 연결해주는 능력 때문에 플러그 앤 플레이 프로토콜 또는 제로 구성 프로토콜이라고도 한다.



DHCP는 클라이언트 - 서버 프로토콜이다.

클라이언트는 일반적으로 IP주소를 포함하여 네트워크 설정을 위한 정보를 얻고자 새롭게 도착한 호스트다.

각 서브넷은 DHCP 서버를 갖거나 없다면 해당 네트워크에 대한 DHCP 서버 주소를 알려줄 DHCP연결 에이전트(일반적으로 라우터)가 필요하다.



새로운 소스트가 도착할 경우, DHCP 프로토콜의 4단계 과정을 보여준다.

• DHCP 서버 발견

- 먼저 새롭게 도착한 호스트는 상호작용할 DHCP를 발견한다. 이것은 **DHCP 발견 메시지**를 사용하여 수행되며, 클라이언트는 포트 67번으로 UDP 패킷을 보낸다.
- DHCP 클라이언트는 **DHCP 발견 메시지**를 포함하는 IP 데이터그램을 생성하는데, 이 메시지 내의 목적지 IP 주소를 브로드캐스트 IP 주소 255.255.255.255로 설정하고 출발지 IP 주소는 0.0.0.0으로 설정한다.
- 링크 계층으로 IP 데이터그램을 보내며 이 프레임은 서브넷에 연결된 모든 노드로 브로드캐

- DHCP 서버 제공
 - DHCP 발견 메시지를 받은 DHCP 서버는 DHCP 제공 메시지를 클라이언트로 응답한다.
 - 이때에도 IP 브로드캐스트 주소를 사용하여 서브넷의 모든 노드로 이 메시지를 브로드캐스트한다.
 - 서브넷에는 여러 DHCP 서버가 존재하기 때문에, 클라이언트는 여러 DHCP 제공 메시지로부터 가장 최적의 위치에 있는 DHCP 서버를 선택한다.
 - 각각의 서버 제공 메시지는 수신된 발견 메시지의 트랜잭션 ID, 클라이언트에 제공된 IP 주소, 네트워크 마스크, **IP 주소 임대 기간**을 포함한다.
- DHCP 요청
 - 새롭게 도착한 클라이언트는 하나 또는 그 이상의 서버 제공자 중에서 선택할 것이고 선택된 제공자에게 파라미터 설정으로 되돌아오는 **DHCP 요청 메시지**로 응답한다.
- DHCP ACK
 - 서버는 DHCP 요청 메시지에 대해 요청된 파라미터를 확인하는 **DHCP ACK 메시지**로 응답한다.

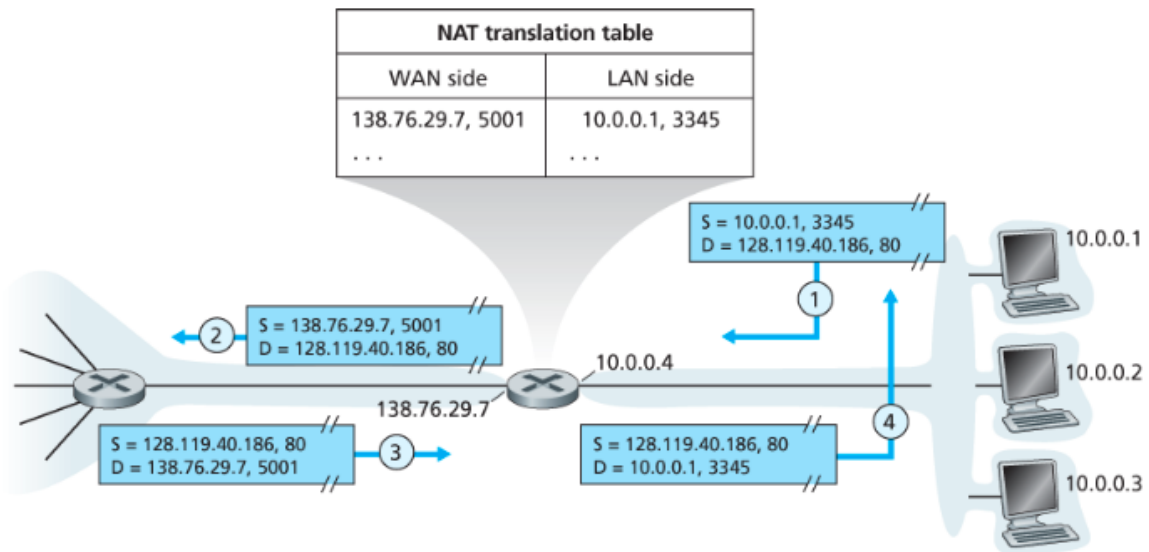
클라이언트가 DHCP ACK을 받으면 상호작용이 종료되고 클라이언트는 임대 기간동안 할당 IP 주소를 사용할 수 있다.

DHCP는 노드가 새로운 서브넷에 연결하고자 할 때마다 새로운 IP 주소를 DHCP로부터 얻기 때문에, 이동 노드가 서브넷 사이를 이동할 때 원격 애플리케이션에 대한 TCP 연결이 유지될 수 없다는 결점이 있다.

4.3.3 네트워크 주소 변환 (NAT)

모든 호스트가 서브넷으로부터 IP를 할당받고 할 때 네트워크가 현저하게 커지면 큰 주소 블록이 할당되어야 하난.

ISP가 이미 해당 주소 범위의 인접한 부분을 할당해버리는 등의 문제가 발생할 수 있는데 이런 경우에는 네트워크 주소 변환 (NAT)으로 주소를 할당할 수 있다.



NAT 기능 라우터는 위 그림의 오른쪽처럼 홈 네트워크의 일부인 인터페이스를 갖는다.
 홈 네트워크의 4개 인터페이스 모두 같은 네트워크 주소 10.0.0.0/24를 갖는다.
 주소 공간 10.0.0.0/8은 사설망 또는 그림의 홈 네트워크와 같은 사설 개인 주소를 갖는 권역(realm)을 위해 RFC에 예약된 IP 주소 공간 세 부분 중의 하나다.

💡 사설 개인 주소를 갖는 권역(realm)이란 네트워크 주소들이 그 네트워크 내부에 있는 장비에게만 의미가 있는 그런 네트워크를 의미한다.

사설 개인 주소를 갖는 권역은 홈 네트워크 내부에서만 의미가 있다. 즉 글로벌 인터넷과는 송수신할 수 없다.

NAT 가능 라우터는 외부 세계로는 하나의 IP 주소를 갖는 하나의 장비로 동작한다.

그림을 보면 홈 라우터를 떠나 인터넷으로 가는 트래픽의 출발지 IP 주소는 138.76.29.7 즉, 라우터의 출력 라우터 인터페이스의 IP 주소를 갖는다.

홈으로 들어오는 트래픽의 목적지 주소는 마찬가지로 138.76.29.7을 가져야한다.

본질적으로 NAT 가능 라우터는 외부에서 들어오는 홈 네트워크의 상세한 사항을 숨긴다.

WAN에서 같은 목적지 IP주소를 갖는 NAT 라우터에 모든 데이터그램이 도착하면, 라우터가 주어진 데이터그램을 전달하는 내부 호스트를 어떻게 알 수 있을까?

NAT 라우터에서 NAT 변환 테이블을 사용하고, 그 테이블에서 IP 주소와 포트번호를 포함하여 알 수 있다.

웹 서버는 내부 호스트를 모른채 WAN side의 라우터를 목적지 IP로 하여 응답하고, 라우터는 이 응답을 NAT 변환 테이블을 사용하여 알맞은 내부 호스트에 전달한다.

포트 번호는 호스트 주소 지정이 아닌 프로세스 주소 지정에 사용된다.

서버 프로세스는 잘 알려진 포트번호에서 요청이 올 때까지 기다리고 P2P프로토콜의 피어는 서버로서의 역할을 할 때 들어오는 연결을 수락해야 하기 때문에 홈 네트워크에서 실행되는 서버에 문제가 발생할 수 있다.

이 문제의 기술적인 해결책으로는 NAT 순회 도구가 있다.

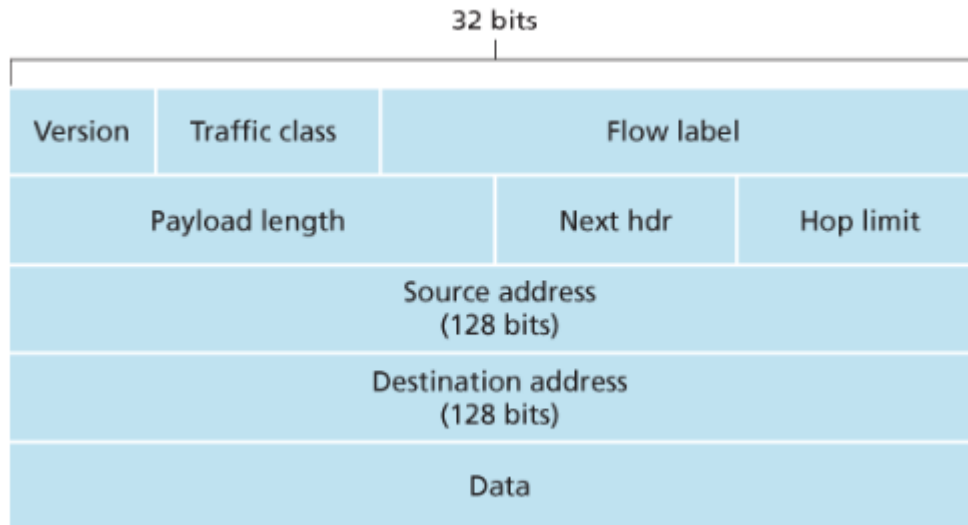
4.3.4 IP46

IPv4 주소 공간이 빠르게 고갈되어가면서 IPv6 주소체계가 개발되었다.

IPv6 데이터그램 포맷

IPv6 중요한 변화

- 확장된 주소 기능
 - IPv6는 IP 주소 크기를 32비트에서 128비트로 확장했으므로 IP 주소가 고갈되는 일은 발생하지 않을 것이다.
 - IPv6는 유니캐스트, 멀티캐스트 주소 뿐만 아니라 새로운 주소 형태인 애니캐스트 주소가 도입되었다. 애니캐스트 주소록 명시된 데이터그램은 호스트 그룹의 어떤에게든 전달될 수 있다.
- 간소화된 40바이트 헤더
 - 40바이트 고정 길이 헤더는 라우터가 IP 데이터그램을 더 빨리 처리하게 해준다.
 - 새로운 옵션 인코딩은 유연한 옵션 처리를 가능하게 한다.
- 흐름 레이블링
 - 정의하기 어려운 흐름을 갖고 있다. RFC는 “비 디폴트 품질 서비스나 실시간 서비스 같은 특별한 처리를 요청하는 송신자에 대해 특정 흐름에 속하는 패킷 레이블링”을 가능하게 한다고 설명한다. 아직 정확한 의미는 정의되지 않았지만, 언젠가 필요할 흐름 차별화를 예견하여 구현하였다.



IPv6 데이터그램 포맷은 그림과 같다.

- **버전**
 - 4비트 필드는 IP 버전 번호를 인식한다. IPv6라면 6이다.
- **트래픽 클래스**
 - IPv4의 TOS 필드와 비슷한 의미로 만든 8비트 필드는 흐름 내의 SMTP 이메일 같은 애플리케이션의 데이터그램보다 Voip 같은 특정 애플리케이션 데이터그램에 우선순위를 부여하는데 사용된다.
- **흐름 레이블**
 - 데이터그램의 흐름을 인식하는데 사용된다.
- **페이로드 길이**
 - 이 16비트 값은 IPv6 데이터그램에서 고정 길이 40바이트 패킷 헤더 뒤에 나오는 바이트 길이이며, 부호 없는 정수다.
- **다음 헤더**
 - 이 필드는 데이터그램의 내용이 전달될 프로토콜을 구분한다.(TCP, UDP)
- **홉 제한**
 - 라우터가 데이터그램을 전달할 때 마다 1씩 감소하고, 0이되면 데이터그램이 라우터에 의해 버려진다.
- **출발지와 목적지 주소**
 - 출발지와 목적지 주소를 담고 있다.
- **데이터**

- IPv6 데이터그램의 페이로드 부분이다. 데이터그램이 목적지에 도착하면 IP 데이터그램에서 페이로드를 제거한 후, 다음 헤더 필드에 명시한 프로토콜에 전달한다.

IPv4에는 있지만 IPv6에는 없는 필드

- 단편화/재결합
 - IPv6에서는 단편화와 재결합을 출발지와 목적지만이 수행한다.
 - 라우터가 받은 IPv6데이터그램이 너무 커서 링크로 전달할 수 없다면 라우터는 데이터그램을 폐기하고 너무 크다는 ICMP 오류 메시지를 송신자에게 보낸다.
 - 송신자는 데이터를 IP 데이터그램 크기를 줄여서 다시 보낸다.
 - 라우터에서 이 기능을 수행하는 것은 시간이 오래 걸리므로 이 기능을 삭제하여 IP 전달 속도를 증가시켰다.
- 헤더 체크섬
 - 드랜스포트 계층 프로토콜과 데이터링크 프로토콜은 체크섬을 수행하므로 IP 설계자는 네트워크 계층의 체크섬 기능이 반복되는 것으로 생략해도 될 것이라 생각하여 삭제했다.
- 옵션
 - IPv4에서도 잘 사용되지 않았던 필드가 사라지고 고정 헤더의 길이를 갖게 되었다.
 - 대신 옵션 필드는 IPv6헤더에서 다른 헤더 중 하나가 될 수 있다.

IPv4에서 IPv6로의 전환

IPv6는 IPv4 데이터그램을 보내고 라우팅하며 받을 수 있는 새 IPv6시스템이 있는 반면에, IPv4로 구축된 시스템은 IPv6데이터그램을 처리할 수 없다는 것에서 발생한다.

플래그 데이 선언

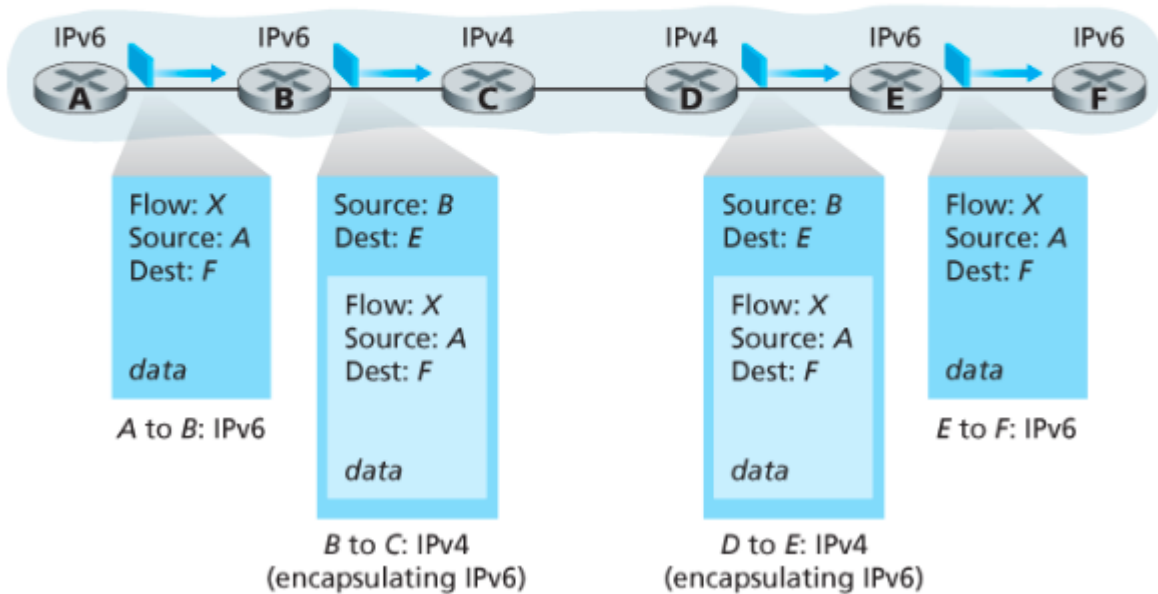
모든 인터넷 장비를 끄고 IPv4를 IPv6로 업그레이드 하는 시간과 날짜를 정하는 것으로 40년전에 실제로 NCP를 TCP로 전이하였다. 그러나 수억개의 장비가 관련된 플래그 데이는 오늘날에 절대 불가능

터널링

Logical view



Physical view



실제로 널리 사용하는 방법이다.

두 IPv6 노드(그림 속 B와 E)가 IPv6 데이터그램을 사용해서 작동한다고 가정해보자
물론 이들은 IPv4 라우터를 통해 연결되어있다. 이렇게 IPv6 노드 사이에 연결되어있는
IPv4라우터들을 터널(tunnel)이라고 한다.

<IPv6 송신 과정>

1. 터널의 송신 측에 있는 IPv6노드는 IPv6데이터그램을 받고 IPv4데이터그램의 데이터 필드에 이것을 넣는다.
2. IPv4데이터그램에 목적지 주소를 터널의 수신 측에 IPv6노드로 적어서 터널의 첫번째 노드에 보낸다.
3. 터널 내부에 있는 IPv4 라우터는 IPv4 데이터그램이 IPv6 데이터그램을 갖고 있다는 사실을 모른채 다른 데이터그램을 처리하는 방식으로 IPv4데이터그램을 처리한다.
4. 터널 수신 측에 있는 IPv6 노드는 IPv4 데이터그램을 받고 이 IPv4 데이터그램이 실제 IPv6 데이터그램임을 결정한다.
5. 다음 노드에 IPv6 데이터그램을 보낸다.

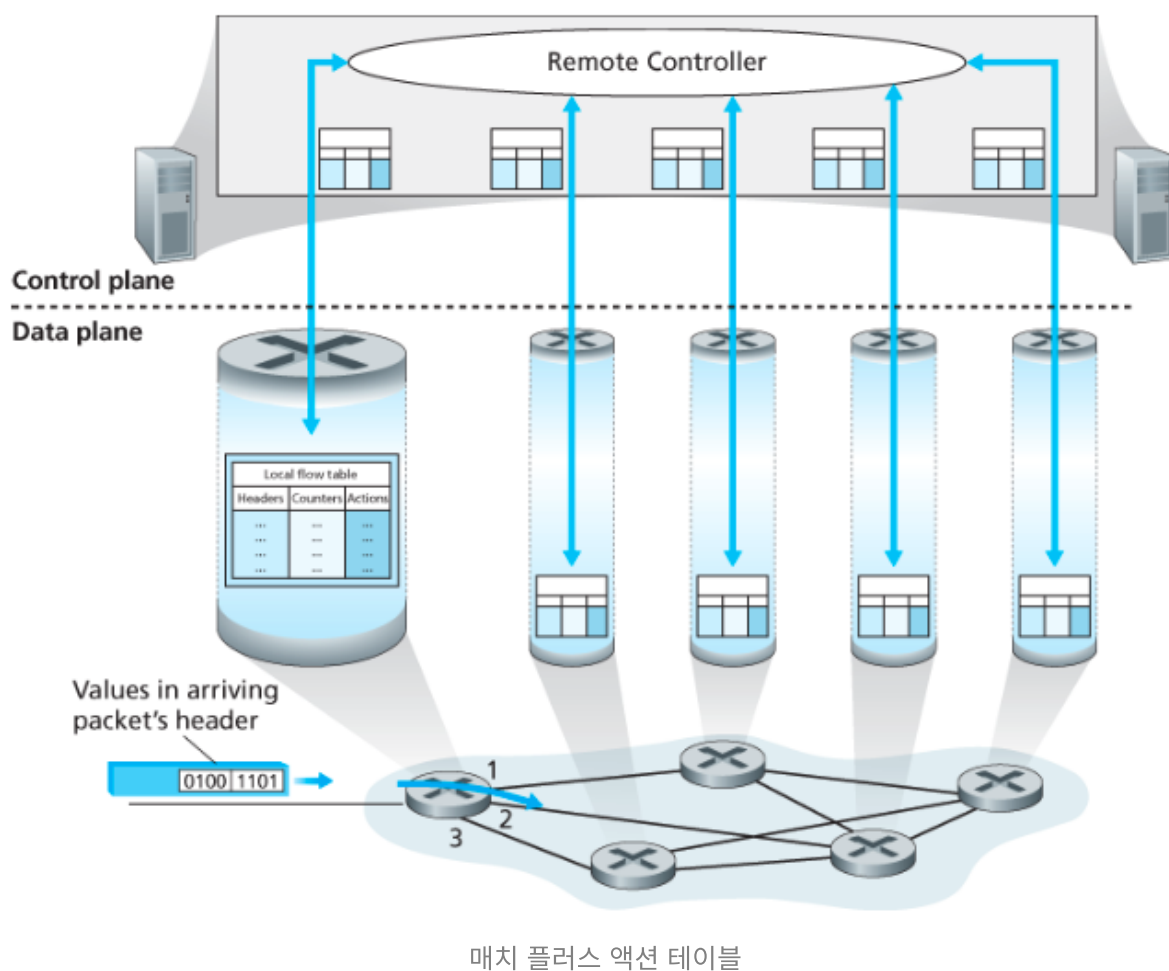
기초적인 IPv6수용은 이루어지고 있지만 최근에 이루어진 것은 없다.

4.4 일반화된 포워딩 및 소프트웨어 기반 네트워크 (SDN)

프로토콜 스택의 다른 계층에서 다른 프로토콜과 관련된 여러 헤더 필드에 대해 매치를 수행할 수 있는 일반적인 매치 플러스 액션 방법을 생각해보자

액션은 하나 이상의 출력포트로 패킷을 전달하고, 인터페이스에서 나가는 패킷을 로드 밸런싱(load balancing)하고 헤더 값을 다시 쓰고, 의도적으로 패킷을 차단/삭제 및 추가 처리 작업을 위해 특수 서버로 패킷을 보내는 등의 작업을 수행한다.

일반화된 포워딩에서는 각각의 패킷 스위치는 원격 컨트롤러에 의해 계산 및 분포된 매치 플러스 액션 테이블을 포함하고 있다.



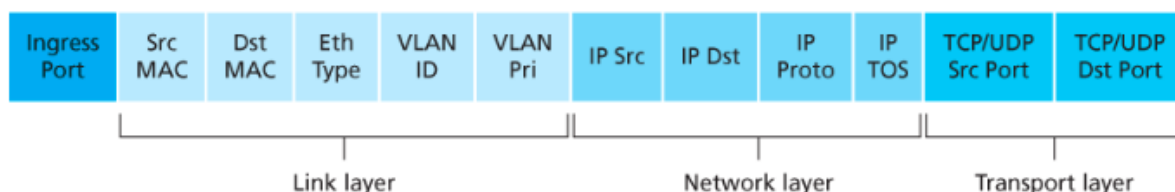
OpenFlow 1.0

명확하고 간결한 방식으로 SDN 개념 및 기능을 도입한 OpenFlow 1.0을 살펴보자

OpenFlow의 플로우 테이블로 알려진 매치 플러스 액션 포워딩 테이블의 각 엔트리는 다음을 포함한다.

- 들어오는 패킷에 대한 헤더 값들의 세트가 매치될 것이다. 하드웨어 기반 매치는 TCAM 메모리에서 가장 신속하게 수행되며, 백만 개가 넘는 목적지 주소를 동반한다. 플로우 테이블 엔트리와 매치되지 않는 패킷을 더 많은 처리를 위해 원격 컨트롤러로 전송될 수 있다.
- 패킷들에 의해 갱신되는 카운터 세트는 플로우 테이블 엔트리들과 매치된다. 이러한 카운터는 플로우 테이블 엔트리와 마지막으로 갱신된 테이블 엔트리 이후에 매치된 다수의 패킷을 포함하고 있다.
- 패킷이 플로우 테이블 엔트리와 매치될 때 여러가지 액션이 가능해진다. 이러한 액션은 패킷을 지정된 출력 포트로 전달하고, 패킷을 삭제하고, 패킷의 복사본을 만들어 여러 출력 포트로 보내거나 선택한 헤더 필드로 다시 쓰는 것일 수 있다.

4.4.1 매치



위 그림은 OpenFlow 1.0 매치 플러스 액션 규칙에서 매치될 수 있는 11개의 패킷 헤더 필드와 수신 포트 ID를 보여준다. 진입 포트는 패킷이 수신되는 패킷 스위치의 입력 포트를 나타낸다.

플로우 테이블 엔트리에는 와일드 카드도 있을 수 있다. 예를 들어, 플로우 테이블의 128.119.*.*의 주소는 128.119를 주소의 첫번째 16비트로 갖는 데이터그램의 해당 주소 필드와 매치된다.

또한 각 플로우 테이블 엔트리에는 우선순위가 있어 여러 플로우 테이블 엔트리와 매치되면, 선별된 매치 엔트리에 해당하는 패킷이 가장 높은 우선순위가 된다.

IP 헤더의 모든 필드가 매치될 수 있는 것은 아니다.

예를 들어 OpenFlow에서는 TTL필드 또는 데이터그램 길이 필드에 기반한 매치를 허용하지 않는다.

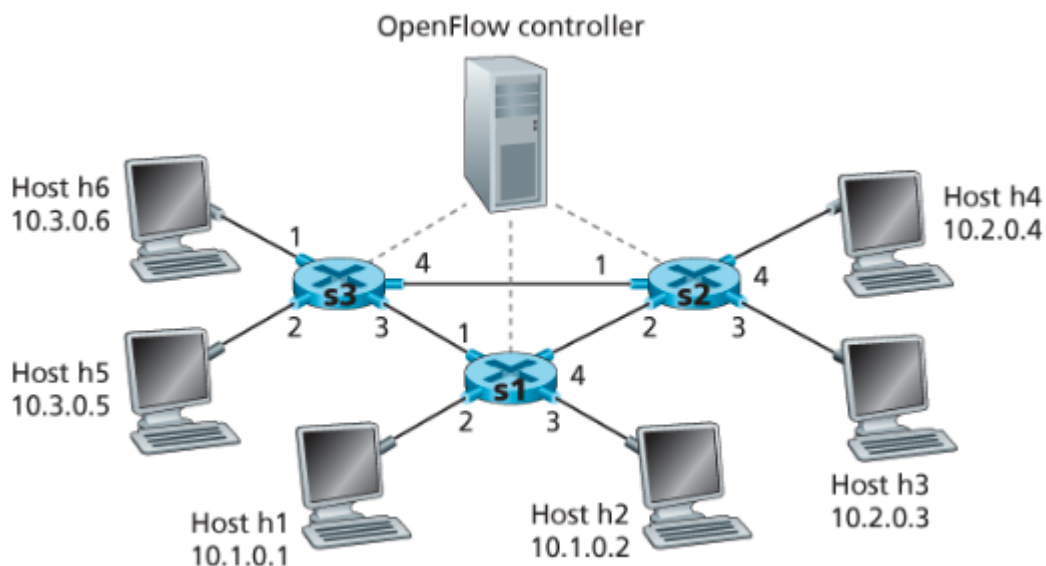
4.4.2 액션

플로우 테이블 엔트리는 플로우 테이블 엔트리와 매치되는 패킷 처리를 결정하는 0개 이상의 액션 목록을 갖고 있다. 여러 액션이 있는 경우 목록에 지정된 순서대로 수행된다.

가장 중요한 액션들은 다음과 같다.

- 포워딩
 - 들어오는 패킷은 특정 실제 출력 포트로 전달되거나 모든 포트를 통해 브로드캐스트되거나 선택된 포트 세트를 통해 멀티캐스트할 수 있다.
 - 패킷은 캡슐화되어 원격 컨트롤러로 전송될 수 있다.
 - 컨트롤러는 새 플로우 테이블 엔트리를 설치하고 해당 패킷에 대한 조치를 취하거나 갱신된 플로우 테이블 규칙에 따라 포워딩을 위해 패킷을 장치로 반환할 수 있다.
- 삭제
 - 아무 액션이 없는 플로우 테이블 엔트리는 매치된 패킷을 삭제해야함을 나타낸다.
- 필드 수정
 - 패킷이 선택된 출력 포트로 전달되기 전에 10개의 패킷 헤더 필드의 값을 다시 쓸 수 있다.

4.4.3 매치 플러스 액션 작업의 OpenFlow 예



첫번째 예 : 간단한 포워딩

아주 간단한 예로 포워딩 동작이 h3 또는 h4로 예정된 h5 또는 h6 패킷이 s3에서 s1으로 전달된 다음 s1에서 s2로 전달된다고 가정한다.

위 상황에서 s1의 플로우 테이블 엔트리는 다음과 같다.

s1 Flow Table (Example 1)	
Match	Action
Ingress Port = 1 ; IP Src = 10.3.*.* ; IP Dst = 10.2.*.*	Forward(4)
...	...

s3에 플로우 테이블 엔트리가 필요하므로 h5또는 h6에서 전송된 데이터그램은 인터페이스 3을 통해 s1으로 전달된다.

s3 Flow Table (Example 1)	
Match	Action
IP Src = 10.3.*.* ; IP Dst = 10.2.*.*	Forward(3)
...	...

마찬가지로, s1에 도착한 데이터그램 호스트 h3 또는 h4로 전달할 수 있도록 s2에 플로우 테이블 엔트리가 필요하다.

두번째 예 : 로드 밸런싱

두번째 예로 h3에서 10.1.*.*로 향하는 데이터그램이 s2와 s1사이의 링크를 통해 전달되는 반면, h4에서 10.1.*.*로의 데이터그램은 s2와 s3 사이의 링크를 통해 전달되는 로드 밸런싱 시나리오를 고려해보자

이 동작은 IP의 목적지 기반 포워딩으로 수행될 수 없다.

이 경우 s2의 포워딩 테이블을 다음과 같다.

s2 Flow Table (Example 2)	
Match	Action
Ingress port = 3; IP Dst = 10.1.*.*	Forward(2)
Ingress port = 4; IP Dst = 10.1.*.*	Forward(1)
...	...

s2에서 수신한 데이터그램을 h1 또는 h2로 전달하려면 s1에서 플로우테이블 엔트리가 필요하다.

인터페이스 4에서 수신한 데이터그램을 s3에서 인터페이스 3을 통해 s1으로 전달하려면 s3에서 플로우 테이블 엔트리가 필요하다. s1 및 s3에서 이러한 플로우 테이블 엔트리를 파악할 수 있는지 확인하자

세번째 예 : 방화벽

s2 Flow Table (Example 3)	
Match	Action
IP Src = 10.3.*.* IP Dst = 10.2.0.3	Forward(3)
IP Src = 10.3.*.* IP Dst = 10.2.0.4	Forward(4)
...	...

s2가 s3에 연결된 호스트에서 보낸 트래픽만 수신하려고 하는 방화벽 시나리오를 생각해 보자

s2 플로우 테이블에 다른 엔트리가 없으면 10.3.*.*의 트래픽만 s2에 연결된 호스트로 전달된다.

매치 플러스 액션 플로우 테이블은 제한된 형태의 프로그래밍 가능성이다.

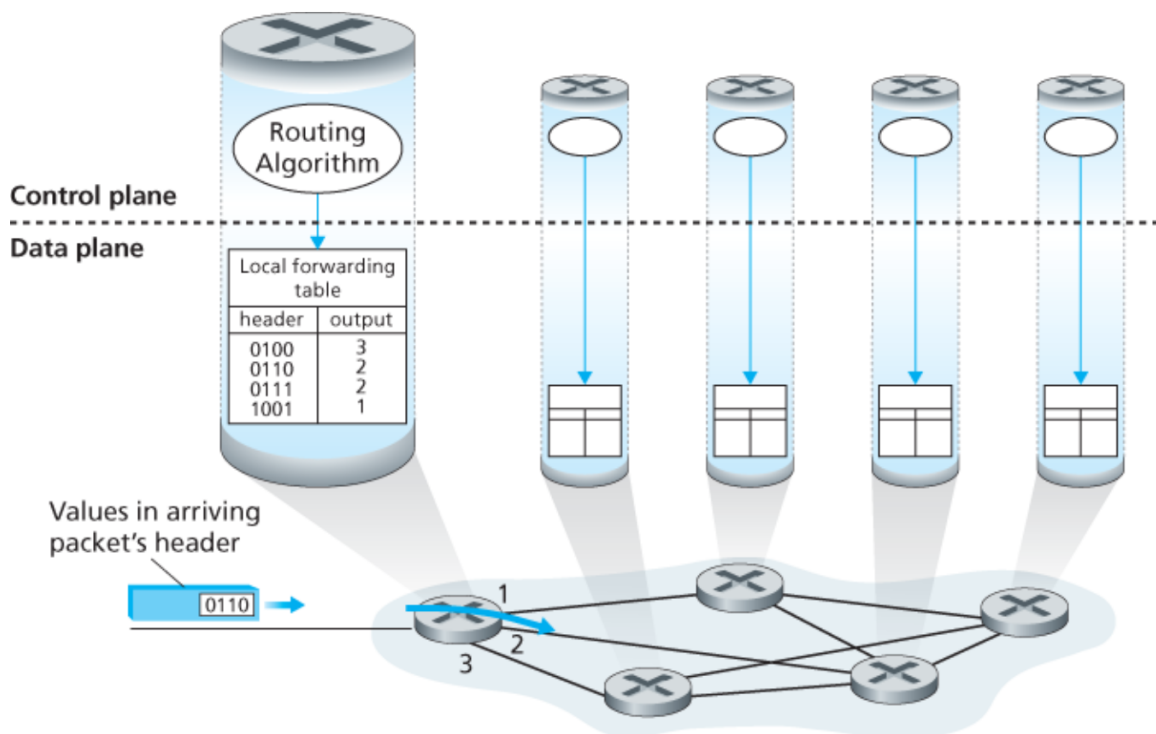
데이터그램의 헤더값과 매치 조건 사이의 매치를 기반으로 라우터가 데이터그램을 전달하고 조작하는 방법을 명시한다.

따라서 더 풍부한 형태의 프로그래밍 가능성을 상상할 수 있다.

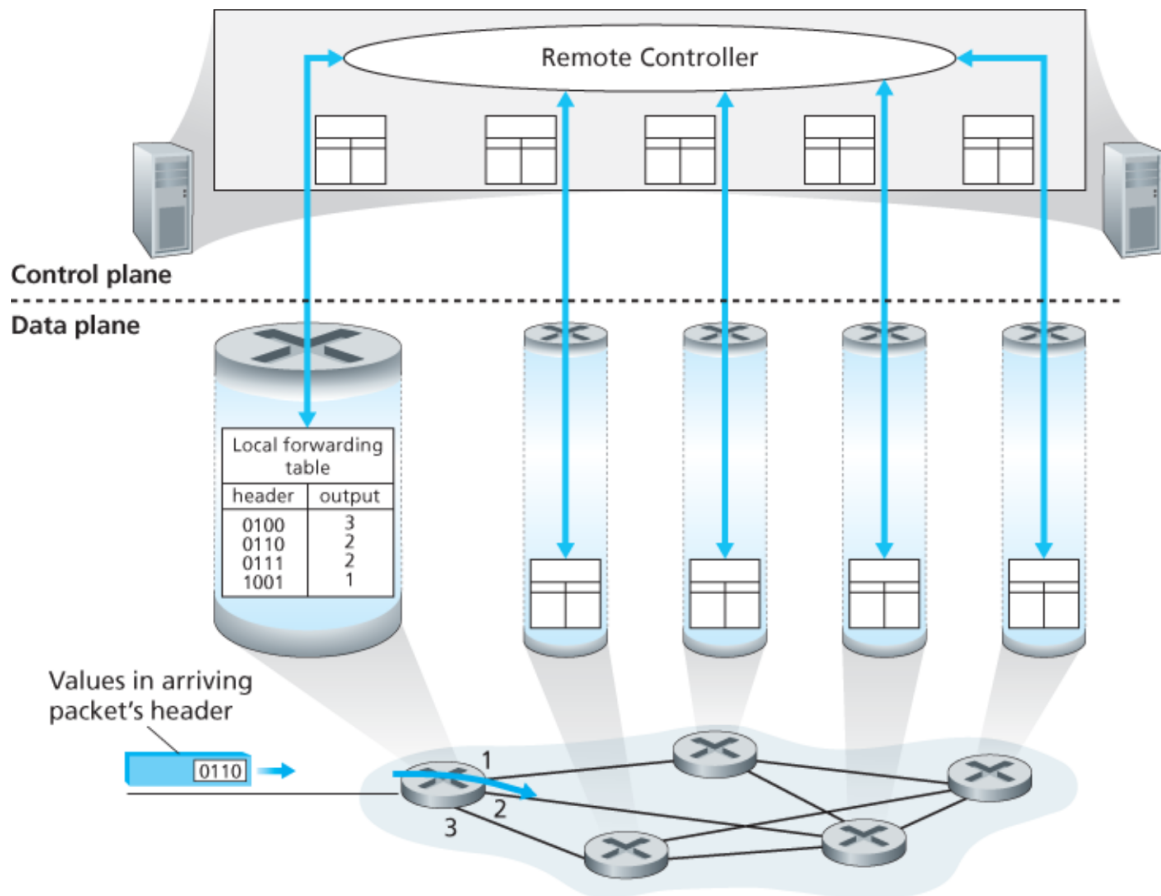
5. Network Control Plane

5.1 개요

포워딩 테이블(목적지 기반 포워딩의 경우)과 플로우 테이블(일반화된 포워딩의 경우)이 네트워크 계층의 데이터 평면과 제어 평면을 연결하는 주요 요소였는데, 이 테이블들이 라우터의 로컬데이터 평면에서의 포워딩을 지정했다.



전통적인 방식의 포워딩

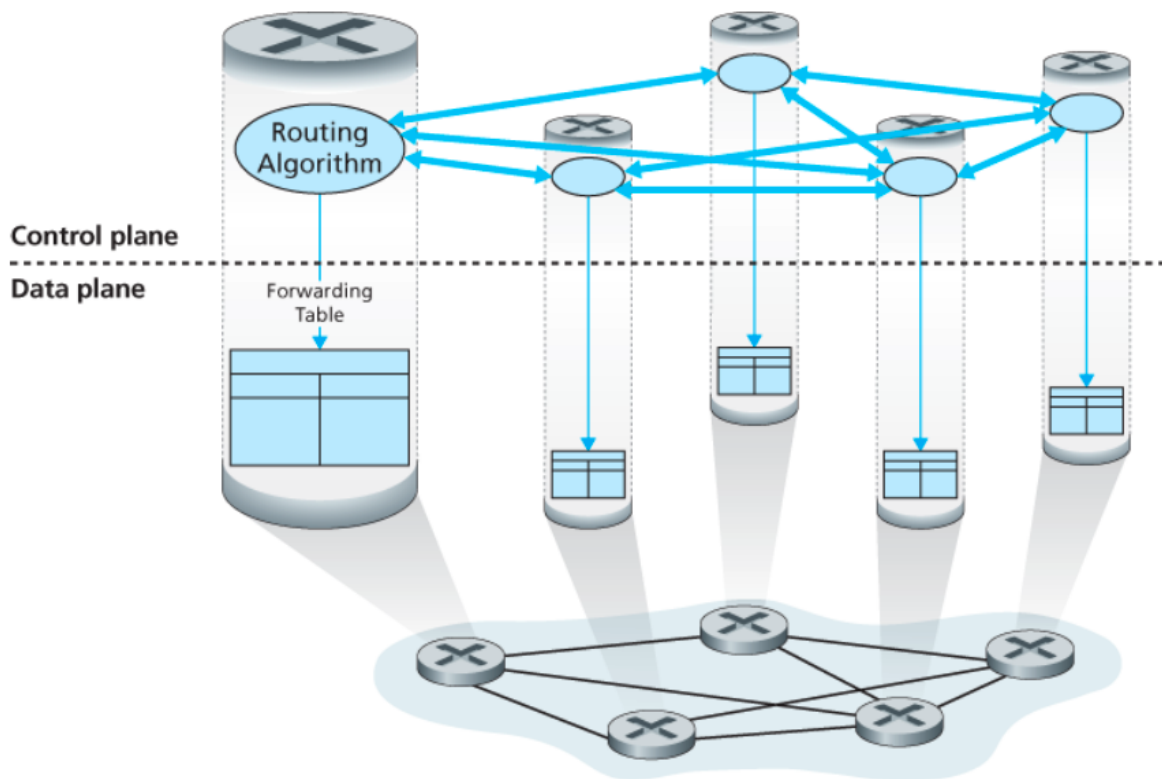


일반화된 포워딩의 경우에 라우터가 취하는 행동을 다양한 형태로 나타낼 수 있었다.

- 라우터의 출력 포트로 패킷을 전달
- 패킷을 버리거나 복제
- 2, 3, 4계층의 헤더 필드를 재작성

라우터별 제어

개별 라우팅 알고리즘들이 제어 평면에서 상호작용한다.

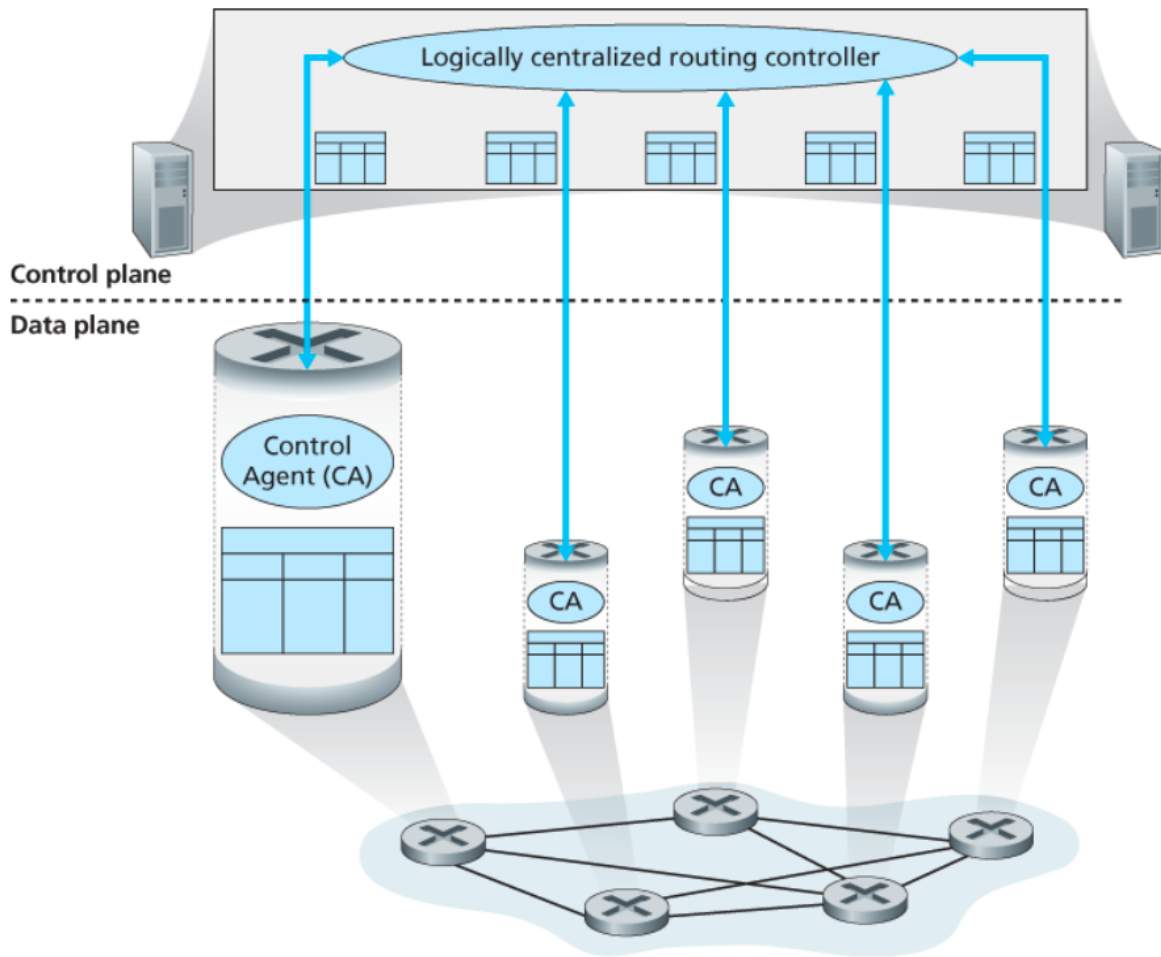


라우팅 알고리즘들이 모든 라우터 각각에서 동작하는 경우를 나타낸다.

- 포워딩과 라우팅 기능이 모두 개별 라우터에 포함되어있다.
- 각 라우터는 다른 라우터의 라우팅 구성요소와 통신하여 자신의 포워딩 테이블의 값을 계산하는 구성요소를 갖고 있다.
- OSPF, BGP 프로토콜이 이 라우터별 제어방식을 기반으로 한다.

논리적 중앙 집중형 제어

일반적으로 원격에 위치한 별개의 컨트롤러가 지역의 제어 에이전트(CA)와 상호작용한다.



논리적 중앙 집중형 컨트롤러가 포워딩 테이블을 작성하고, 이를 모든 개별 라우터가 사용할 수 있도록 배포한 경우

일반화된 '매치 플러스 액션(match plus action)' 추상화를 통해 라우터는 기존에는 별도로 장치로 구현되었던 다양한 기능(부하 분산, 방화벽, NAT) 뿐만 아니라 전통적인 IP 포워딩을 수행할 수 있다.

컨트롤러는 프로토콜을 통해 각 라우터의 제어 에이전트(control agent, CA)와 상호작용하여 라우터의 플로우 테이블을 구성 및 관리한다.

라우터별 제어방식과는 다르게, CA는 서로 직접 상호작용하지 않으며, 포워딩 테이블을 계산하는 데도 적극적으로 참여하지 않는다.

5.2 라우팅 알고리즘

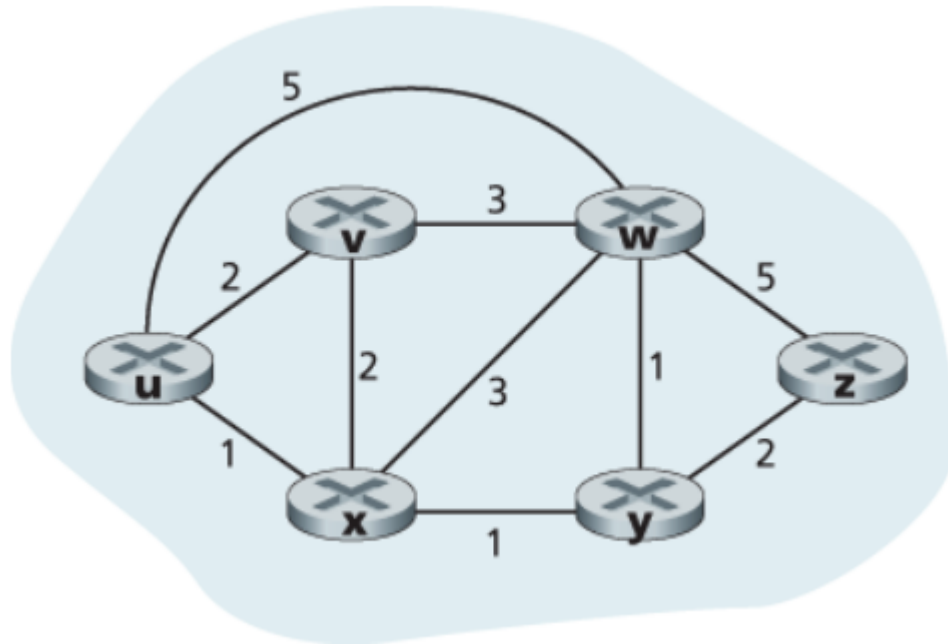
라우팅 알고리즘(routing-algorithm)의 목표 : 송신자부터 수신자까지 라우터의 네트워크를 통과하는 좋은 경로(루트)를 결정하는 것

일반적으로 '좋은' 경로란 최소비용 경로(least-cost path)를 말한다.

그러나 현실적으로는 네트워크 정책과 같은 실제 문제가 고려된다.

(e.g. Y 기관에 속해있는 라우터 x는 Z기관이 소유한 네트워크가 보낸 패킷을 전달해서는 안 됨)

그래프



그래프(graph), $G(N, E)$

- **N**
 - 노드(node)의 집합
 - 네트워크 계층 라우팅 상황에서 그래프상의 **노드**는 패킷 전달 결정이 이루어지는 지점인 **라우터**를 나타낸다.
- **E**
 - 에지(edge)의 집합
 - 네트워크 계층 라우팅 상황에서 그래프상의 **에지**는 **라우터들 간의 물리 링크**를 나타낸다.
 - 에지는 그 **비용**을 나타내는 값을 가진다.(일반적으로 해당 링크의 물리적인 거리, 링크 속도, 링크와 관련된 금전 비용 등을 반영)

- 집합 E에 포함된 어떤 에지 (x, y)에 대해 $c(x, y)$ 는 노드 x와 y 간의 비용을 의미한다.
- 에지 (x, y)가 집합 E에 속하면, 노드 y는 노드 x의 **이웃(neighbor)**이라고 한다.
- 하나의 에지는 집합 E에 속하는 한 쌍의 노드로 표시된다.

그래프 $G(N, E)$ 에서의 **경로(path)**는 노드의 연속($x_1, x_2, x_3, \dots, x_p$)이고,

노드 쌍 $(x_1, x_2), (x_2, x_3), \dots, (x_{p-1}, x_p)$ 는 집합 E에 속한 에지들이다.

경로 $(x_1, x_2, x_3, \dots, x_p)$ 의 비용은 경로상 모든 에지 비용의 단순 합이다.

$$c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$$

라우팅 알고리즘의 분류

라우팅 알고리즘을 분류하는 일반적인 방법 한 가지는 알고리즘이 중앙 집중형인지 분산형인지이다.

중앙 집중형 라우팅 알고리즘 (centralized routing algorithm)

네트워크 전체에 대한 완전한 정보를 가지고 출발지와 목적지 사이의 최소 비용 경로를 계산한다.

계산 자체는 한 장소에서 수행되거나 모든 라우터 각각의 라우팅 모듈로 복사될 수 있다.

전체 상태 정보를 갖는 알고리즘을 링크 상태(link-state, LS) 알고리즘이라고 하는데, 이는 이 알고리즘이 네트워크 내 각 링크의 비용을 알고 있어야 하기 때문이다. (다익스트라 알고리즘)

분산 라우팅 알고리즘(decentralized routing algorithm)

최소 비용 경로의 계산이 라우터들에 의해 반복적이고 분산된 방식으로 수행된다.



각 노드는 자신에게 직접 연결된 링크에 대한 비용 정보만을 가지고 시작한다.

이후 반복된 계산과 이웃 노드와의 정보 교환을 통해 노드는 점차적으로 목적지 또는 목적지 집합까지의 최소비용 경로를 계산한다.

분산 라우팅 알고리즘은 거리벡터(distance-vector, DV) 알고리즘이라고도 하는데, 이는 각 노드가 네트워크 내 다른 모든 노드까지 비용(거리)의 추정값을 벡터 형태로 유지하기 때문이다.

라우팅 알고리즘을 분류하는 일반적인 두번째 방식은 정적 알고리즘과 동적 알고리즘으로 분류하는 것이다.

정적 라우팅 알고리즘(static routing algorithm)

사람이 직접 링크 비용을 수정하는 경우와 같은 종종 사람이 개입하는 상황 때문에 정적 라우팅 알고리즘에서 경로는 아주 느리게 변한다.

동적 라우팅 알고리즘(dynamic routing algorithm)

네트워크 트래픽 부하(load)나 토폴로지 변화에 따라 라우팅 경로를 바꾼다.

동적 알고리즘은 주기적으로, 혹은 토폴로지나 링크 비용의 변경에 직접적으로 응답하는 방식으로 수행된다.

- 장점 : 네트워크 변화에 빠르게 대응한다.
- 단점 : 경로의 루프(loop)나 경로 진동(oscillation)같은 문제에 취약하다.

라우팅 알고리즘을 분류하는 세번째 방식은 라우팅 알고리즘이 부하에 민감한지 아닌지에 따른다.

부하에 민감한 알고리즘(load-sensitive algorithm)

링크 비용은 해당 링크의 현재 혼잡 수준을 나타내기 위해 동적으로 변한다.

현재 혼잡한 링크에 높은 비용을 부과한다면, 라우팅 알고리즘은 혼잡한 링크를 우회하는 경로를 태하는 경향을 보일 것이다.

초기 ARPAnet 라우팅 알고리즘이 부하에 민감해서 많은 어려움이 있었다.

부하에 민감하지 않은 알고리즘(load-insensitive algorithm)

오늘날 인터넷 라우팅 알고리즘(RIP, OSPF, BGP 등)은 링크 비용이 현재(또는 가장 최근)의 혼잡을 반영하지 않기 때문에 부하에 민감하지 않다.

5.2.1 링크상태(LS) 라우팅 알고리즘



링크 상태 알고리즘에서는 네트워크 토폴로지와 모든 비용이 알려져있어서 링크 상태 알고리즘의 입력값으로 사용될 수 있다.

이것은 각 노드가 자신과 직접 연결된 링크의 식별자와 비용 정보를 담은 링크 상태 패킷을 네트워크상의 다른 모든 노드로 브로드캐스트하게 함으로써 가능하며, 이는 종종 인터넷 OSPF 라우팅 프로토콜 같은 링크 상태 브로드캐스트(link-state broadcast) 알고리즘에 의해 수행된다.

다익스트라 알고리즘(Dijkstra's algorithm)



다익스트라 알고리즘은 하나의 노드(출발지, u 라고 지칭)에서 네트워크 내 다른 노드들의 최소 비용 경로를 계산한다.

알고리즘의 k 번째 반복 이후에는 k 개의 목적지 노드에 대해 최소비용 경로가 알려지며, 이들은 모든 목적지 노드들의 최소비용 경로 중에서 가장 낮은 비용을 갖는 k 개의 경로다.

기호 정의

- $D(v)$: 알고리즘의 현재 반복 시점에서 출발지 노드부터 목적지 v 까지의 최소비용 경로의 비용
- $p(v)$: 출발지에서 v 까지의 현재 최소비용 경로에서 v 의 직전 노드
- N' : 노드의 집합

출발지에서 v 까지의 최소 비용 경로가 명확히 알려져 있다면, v 는 N' 에 포함된다.

출발지 노드 u 를 위한 링크 상태(LS) 알고리즘

중앙 집중형 라우팅 알고리즘은 2단계로 구성된다.

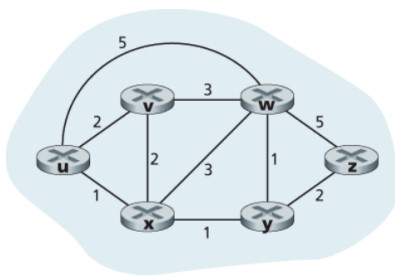
1. 초기화 단계(initialization)
2. 반복 부분(Loop) : 수행 횟수는 네트워크의 노드 수와 같다.

```

1  Initialization:
2   $N' = \{u\}$ 
3  for all nodes  $v$ 
4      if  $v$  is a neighbor of  $u$ 
5          then  $D(v) = c(u, v)$ 
6      else  $D(v) = \infty$ 
7
8  Loop
9  find  $w$  not in  $N'$  such that  $D(w)$  is a minimum
10 add  $w$  to  $N'$ 
11 update  $D(v)$  for each neighbor  $v$  of  $w$  and not in  $N'$ :
12      $D(v) = \min(D(v), D(w) + c(w, v))$ 
13 /* new cost to  $v$  is either old cost to  $v$  or known
14    least path cost to  $w$  plus cost from  $w$  to  $v$  */
15 until  $N' = N$ 

```

<링크 상태 알고리즘 수행 결과>



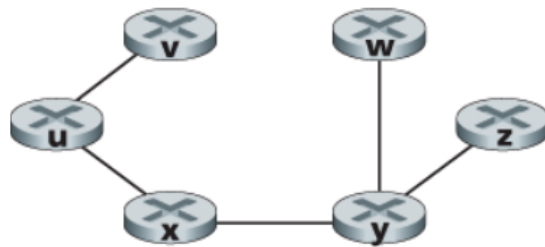
step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

포워딩 테이블

링크 상태 알고리즘이 종료된 후에 우리는 각 노드에 대해 출발지 노드로부터의 최소비용 경로상의 직전 노드를 알게 된다.

노드 u 의 포워딩 테이블은 각 목적지에 대해 / 노드 u 에서 그 목적지까지의 최소비용 경로상의 다음 홉 노드 정보를 저장하여 구성한다.

<최소비용 경로의 결과와 노드 u의 포워딩 테이블>



Destination	Link
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

계산 복잡도

n개의 노드(출발지 노드 제외)가 있다면 출발지에서 모든 목적지까지 최단 비용 경로를 찾기 위해 최악의 경우 얼마나 많은 계산이 필요한가?

첫 번째 반복에서 최소 비용이 이미 계산된 노드의 집합 N'에 포함되지 않은 노드 w를 결정하기 위해 모든 n개의 노드를 검사해야 하며,

두 번째 반복에서는 n-1개의 노드를, 세 번째 반복에서는 n-1개의 노드를 검사해야 한다.

따라서 찾아야 하는 노드의 총수는 $n(n+1)/2$ 가 되며,

링크 상태 알고리즘은 최악의 경우 $O(n^2)$ 의 복잡성을 갖는다.

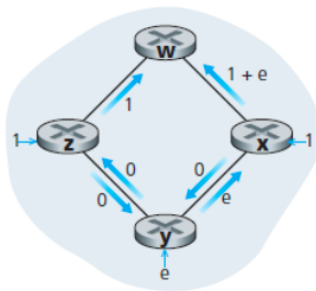
진동(Oscillation) 문제

진동 문제는 링크 상태 알고리즘뿐만 아니라 혼잡이나 지연 시간을 기반으로 링크 비용을 산출하는 모든 알고리즘에서 발생할 수 있다.

<초기의 라우팅>

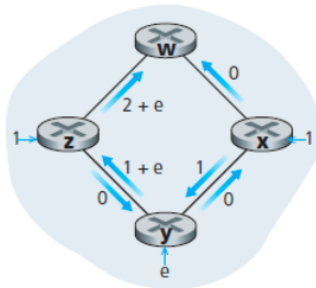
링크의 비용은 통과하는 트래픽 양에 따른다.

링크 상태 알고리즘이 다시 수행되면 노드 y는 w로 가는 시계 방향의 경로 비용이 1인 반면, 지금까지 사용해왔던 반시계 방향으로의 경로비용은 $1+e$ 임을 알게 된다.



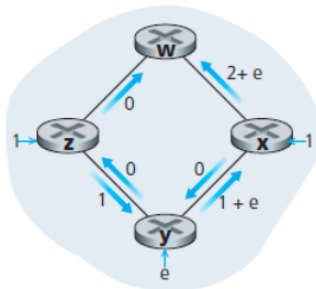
a. Initial routing

따라서 w로 가는 y의 최소 비용 경로는 시계 방향이며, x도 마찬가지로 w로 가는 시계방향 경로를 새로운 최소비용 경로로 결정한다.



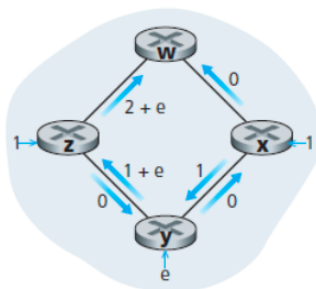
b. x, y detect better path to w, clockwise

링크 상태 알고리즘이 다시 한번 수행되면 노드 x, y, z 모두 w로 가는 반시계 방향의 경로 비용이 0임을 알게 되어 모든 트래픽을 반시계 방향 경로로 보낸다.



c. x, y, z detect better path to w, counterclockwise

다음번 링크 상태 알고리즘 수행 시에는 x, y, z 모두 시계 방향으로 트래픽을 전송한다.



d. x, y, z, detect better path to w, clockwise

이러한 진동 문제를 방지하기 위한 방법들 중 하나는 모든 라우터가 동시에 링크 상태 알고리즘을 실행하지 못하도록 하는 것이다.

라우터들이 동일한 주기 간격으로 링크 상태 알고리즘을 수행한다 하더라도 각 노드에서의 알고리즘의 실행 시각은 같지 않을 것이기 때문에 합리적인 방법이라고 생각된다.

하지만 연구자들은 라우터들이 알고리즘을 처음에는 각기 다른 시작 시각에, 그러나 같은 주기를 갖도록 해서 실행하더라도 점진적으로 결국엔 서로 동기화 된다는 것을 발견하였다.

이러한 자기 동기화는 각 노드가 링크상태 정보를 송신하는 시각을 임의로 결정하게 함으로써 회피할 수 있다.

5.2.2 거리 벡터(DV) 라우팅 알고리즘

오늘날 실제로 사용되는 알고리즘은 거리 벡터(distance-vector, DV) 라우팅 알고리즘이다.

링크 상태 알고리즘이 네트워크 전체 정보를 이용하는 알고리즘인 반면, 거리 벡터 알고리즘은 **반복적이고 비동기적이며 분산적**이다.

- **분산적(distributed)** : 각 노드는 하나 이상의 직접 연결된 이웃으로부터 정보를 받고, 계산을 수행하며, 계산된 결과를 다시 이웃들에게 배포한다.
- **반복적(iterative)** : 이웃끼리 더 이상 정보를 교환하지 않을 때까지 프로세스가 지속된다.
- **비동기적(asynchronous)** : 모든 노드가 서로 정확히 맞물려 동작할 필요가 없다.

벨만-포드(Bellman-Ford) 식

$$d_x(y) = \min_v \{c(x, v) + d_v(y)\}$$

노드 x부터 y까지 최소 비용 경로의 비용

- min.v는 x의 모든 이웃에 적용된다.
- x에서 v로 이동한 후, v에서 y까지의 최소 비용 경로를 택한다면, 경로 비용은 $c(x, v) + d_v(y)$ 일 것이다.

- 반드시 하나의 이웃 v 로 가는 것부터 시작해야 하므로,
- x 에서 y 까지의 최소 비용은 모든 이웃 노드 v 에 대해 계산된 $c(x, v) + d.v(y)$ 중 **최솟값**이 된다.

벨만-포드 식의 해답은 각 노드 포워딩 테이블의 엔트리를 제공한다.

- 위의 식을 최소로 만드는 이웃 노드를 v^* 라고 해보자.
- 만약 **노드 x 가 노드 y 에게** 최소 비용 경로로 패킷을 보내기 원한다면, 노드 x 는 패킷을 노드 v^* 로 전달해야 한다.

그러므로 **노드 x 의 포워딩 테이블**에는 최종 목적지 y 로 가기 위한 다음 홉 라우터로 v^* 가 지정되어 있어야 한다.

따라서 거리 벡터 라우팅 알고리즘의 기본 아이디어는 다음과 같다.

출발지 노드를 x 라고 가정하면, 노드 x 는 자신으로부터 집합 N 에 속한 다른 모든 노드 y 까지의 최소 비용 경로의 비용 $D.x(y)$ 를 추정한다.

$D.x$ 을 노드 x 에서부터 N 에 속한 모든 다른 노드 y 까지의 비용 추정값의 벡터라고 하자.

$$D_x = [D_x(y): y \text{ in } N]$$

DV 알고리즘으로 각 **노드 x** 는 다음과 같은 라우팅 정보를 유지한다.

- 각 이웃 노드 v 중에서 x 에 **직접 접속된 이웃 노드까지의 비용** $c(x, v)$
- **노드 x 의 거리 벡터**, 즉 x 로부터 N 에 있는 모든 목적지 y 로의 비용 예측값을 포함하는 벡터 $D.x$
- **이웃 노드들의 거리 벡터들**, 즉 v 가 x 의 이웃이라고 하면 $D.v = [D.v(y): y \text{ in } N]$

분산적이고 비동기적으로 동작하는 알고리즘에서는 때때로 각 노드가 자신의 거리 벡터를 이웃들에게 보낸다.

노드 x 가 이웃 w 에게서 새로운 거리 벡터를 수신하면,

x는 w의 거리 벡터를 저장하고 벨만-포드 식을 사용하여 다음처럼 자신의 거리 벡터를 갱신한다.

$$D_x(y) = \min_v \{c(x, v) + D_v(y)\}$$

(N에 속하는 각 노드 y에 대해)

만약 이 갱신으로 인해 노드 x의 거리 벡터가 변경된다면

1. 노드 x는 이 수정된 거리 벡터를 자신의 이웃들에게 보내고
2. 그에 따라 이웃들도 자신의 거리 벡터를 갱신한다.

모든 노드가 자신의 거리 벡터를 비동기적으로 교환하는 동작을 계속하다 보면,

비용 추정값 $D.x(y)$ 는 노드 x에서 노드 y까지의 실제 최소 비용 경로의 비용인 $d.x(y)$ 로 수렴하게 된다.

거리 벡터(DV) 알고리즘



```

1  Initialization:
2    for all destinations  $y$  in  $N$ :
3       $D_x(y) = c(x, y)$  /* if  $y$  is not a neighbor then  $c(x, y) = \infty$  */
4    for each neighbor  $w$ 
5       $D_w(y) = ?$  for all destinations  $y$  in  $N$ 
6    for each neighbor  $w$ 
7      send distance vector  $D_x = [D_x(y) : y \text{ in } N]$  to  $w$ 
8
9  loop
10   wait (until I see a link cost change to some neighbor  $w$  or
11         until I receive a distance vector from some neighbor  $w$ )
12
13   for each  $y$  in  $N$ :
14      $D_x(y) = \min_v \{c(x, v) + D_v(y)\}$ 
15
16   if  $D_x(y)$  changed for any destination  $y$ 
17     send distance vector  $D_x = [D_x(y) : y \text{ in } N]$  to all neighbors
18
19 forever

```

각 노드 x 에서, 특정 목적지 y 에 대한 자신의 포워딩 테이블을 갱신하기 위해 노드 x 가 알아야 하는 것은 **y 로의 최단 경로상의 다음 홉 라우터인 이웃 노드 $v^*(y)$** 다.

다음 홉 라우터 $v^*(y)$ 는 위 DV 알고리즘의 14번째 줄에서 최솟값을 갖게 하는 이웃 v 이기에, 13~14번째 줄에서 각 목적지 y 에 대해 노드 x 는 $v^*(y)$ 를 결정하고 목적지 y 에 대해 포워딩 테이블도 갱신한다.

 DV 알고리즘에서 하나의 노드가 갖는 정보는 단지 자신에게 직접 연결된 이웃으로의 링크 비용과 그 이웃들로부터 수신하는 정보뿐이다.

1. 각 노드는 이웃으로부터의 갱신을 기다리고 (10~11번째 줄)
2. 업데이트를 수신하면 새로 거리 벡터를 계산하고 (14번째 줄)
3. 이 새로운 거리 벡터를 이웃들에게 배포한다. (16~17번째 줄)
- 4.

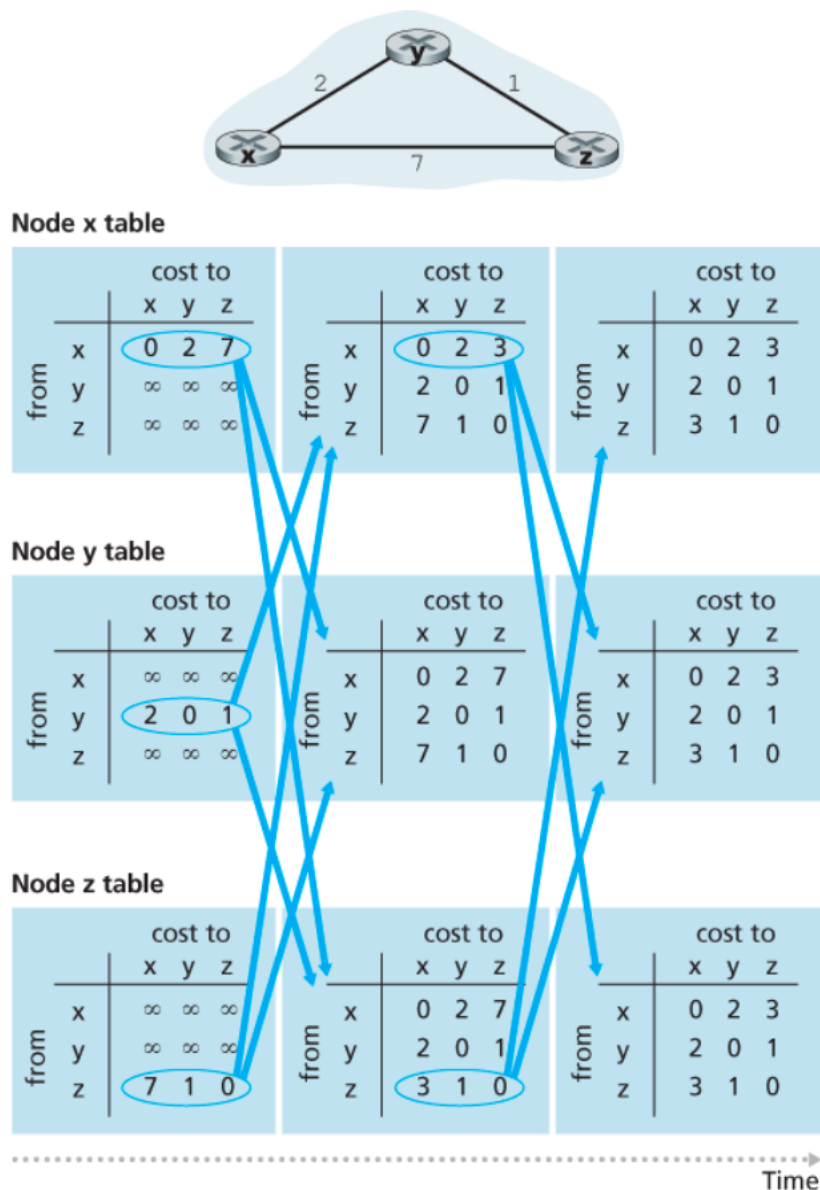
이 과정은 **더 이상의 갱신 메시지가 없을 때까지** 계속된다.

갱신 메시지가 더 이상 없으면 라우팅 테이블 계산도 더 이상 없고 알고리즘은 정지 상태가 된다. (10~11번째 줄 대기 명령을 수행)

이 알고리즘은 링크 비용이 변할 때까지 정지 상태로 있다.

아래 그림은 세 노드로 이루어진 단순한 네트워크에서의 거리 벡터 알고리즘의 동작을 보여 준다.

여기서는 모든 노드가 동기적 방식으로 동작하지만, 비동기적 방식으로도 알고리즘은 올바르게 동작한다.



거리 벡터(DV)알고리즘 : 링크 비용 변경과 링크 고장

거리 벡터 알고리즘을 수행하는 노드가

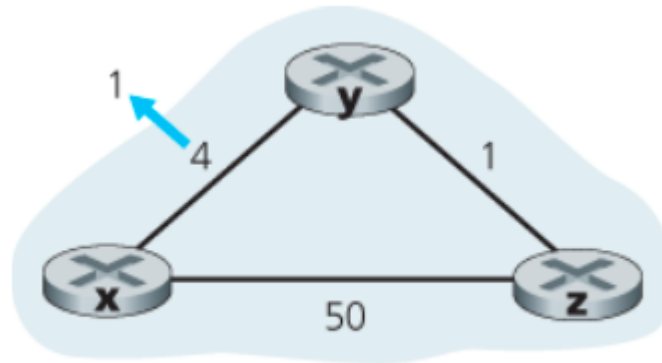
1. 자신과 이웃 사이 링크의 비용이 변경된 것을 알게 되면

2. 자신의 거리 벡터를 갱신한 후

3. **최소 비용 경로의 비용에 변화가 있는 경우에는** 이웃에게 새로운 거리 벡터를 보낸다.

이때 최소 비용 경로의 비용이 감소한 상황과 증가한 상황 두 가지를 전부 살펴보자.

a. 비용이 감소한 상황



a.

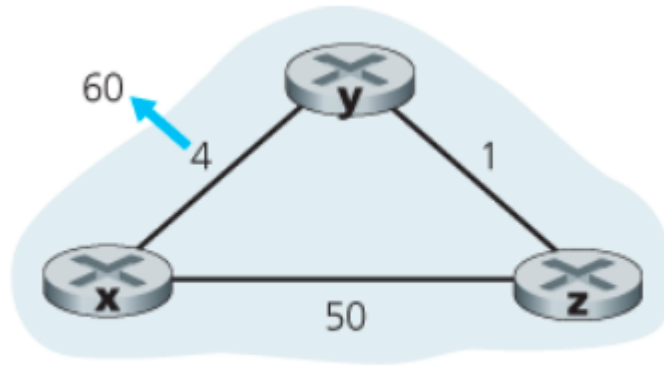
y에서 x로의 링크 비용이 4에서 1로 변한 상황

이 상황에서의 DV 알고리즘은 다음과 같은 일련의 사건을 발생시킨다.

- 시각 t_0 : **y**가 링크 비용의 변화를 감지하고, 자신의 거리 벡터를 갱신한 후 이 변경값을 이웃에게 알린다.
- 시각 t_1 : **z**는 y로부터 갱신 정보를 받고 자신의 테이블을 갱신한다.
 - z는 x까지의 새로운 최소 비용을 계산한다.
 - 이웃에게 자신의 새로운 거리 벡터를 전송한다.
- 시각 t_2 : **y**는 z로부터 갱신 정보를 받고 자신의 테이블을 갱신한다.
 - y의 최소 비용은 변화가 없으므로 y는 z에게 아무런 메시지를 보내지 않는다.**
 - 이에 알고리즘은 정지 상태가 된다.

따라서 거리 벡터 알고리즘은 정지 상태가 될 때까지 두 번만 반복하면 된다.

b. 비용이 증가한 상황



b.

y에서 x로의 링크 비용이 4에서 60로 변한 상황

이 상황에서의 DV 알고리즘은 다음과 같은 일련의 사건을 발생시킨다.

1. 시각 t0 : y가 링크 비용 변화를 감지하고 노드 x까지 다음의 비용을 갖는 새로운 최소 비용 경로를 계산한다.

$$D_y(x) = \min\{c(y, x) + D_x(x), c(y, z) + D_z(x)\} = \min\{60 + 0, 1 + 5\} = 6$$

- 이때 우리는 네트워크 전체를 한눈에 볼 수 있기 때문에 z를 경유하는 이 새로운 비용이 **잘못되었다**는 사실을 알 수 있지만, 노드 y의 입장에서는 아니다.

2. 시각 t1

- x로 가기 위해 y는 z로 경로 설정을 하고, z는 y로 경로 설정을 하는 라우팅 루프(routing loop)가 발생한다.

t1에 x를 목적지로 하는 패킷이 y나 z에 도착하면 포워딩 테이블이 변할 때까지 이 두 노드 사이에서 왔다 갔다 순환할 것이다.

- 노드 y는 x까지의 새로운 최소 비용을 계산했으므로 z에게 새로운 거리 벡터를 알린다.

3. 시각 t2 : z는 y로부터 갱신 정보를 받고 새로운 최소 비용을 계산한다.

- $D_z(x) = \min\{50+0, 1+6\} = 7$
- x까지의 z의 최소 비용이 증가했으므로, 새로운 거리 벡터를 y에 알린다.

4. 시각 t3 : y는 z로부터 새로운 거리 벡터를 수신하고 새로운 최소 비용을 계산한다.

- $D_y(x) = \min\{60+0, 1+7\} = 8$

- x까지의 y의 최소 비용이 증가했으므로, 새로운 거리 벡터를 z에 알린다.

5. ...

이렇게 계속 반복되는 문제를 무한 계수 문제(count-to-infinity)라고 한다.

거리 벡터 알고리즘: 포이즌 리버스 추가

방금 설명한 특정한 라우팅 루프 시나리오는 포이즌 리버스(poisoned reverse)라는 방법을 통해 방지할 수 있다.

즉, 만약 z가 y를 통해 목적지 x로 가는 경로 설정을 했다면, **z는 y에게 x까지의 거리가 무한대라고 알린다.**

z는 y를 통과해서 x로 가는 동안은 이러한 거짓말을 계속한다.

이에 y는 z에서 x로 가는 경로가 없다고 믿으므로, z가 계속해서 y를 통해 x로 가는 경로를 사용하는 동안은 y는 z를 통해 x로 가는 경로를 시도하지 않을 것이다.

하지만 포이즌 리버스는 모든 무한 계수 문제를 해결할 수는 없다.

단순히 직접 이웃한 2개의 노드가 아닌, 3개 이상의 노드를 포함한 루프는 포이즌 리버스로는 감지할 수 없다.

링크 상태 알고리즘과 거리 벡터 라우팅 알고리즘의 비교

경로 계산 방법

LS와 DV 알고리즘은 경로를 계산할 때 서로 대비되는 방법을 취한다.

LS 알고리즘

- 전체 정보를 필요로 한다.
- 각 노드는 다른 **모든** 노드와 (브로드캐스트를 통해) 통신한다.
- **오직** 자신에게 직접 연결된 링크의 비용만 알린다.

DV 알고리즘

- 각 노드는 **오직** 직접 연결된 이웃과만 메시지를 교환한다.
- 자신으로부터 네트워크 내 (자신이 알고 있는) **모든** 노드로의 최소 비용 추정값을 이웃들에게 제공한다.

메시지 복잡성

LS 알고리즘

- 각 노드는 네트워크 내 각 링크 비용을 알아야 하며, 이를 위해서는 $O(|N| |E|)$ 개의 메시지가 전송되어야 한다.
- 링크 비용이 변할 때마다 새로운 링크 비용이 모든 노드에게 전달되어야 한다.

DV 알고리즘

- 매번 반복마다 직접 연결된 이웃끼리 메시지를 교환한다.
- 알고리즘의 결과가 수렴하는 데 걸리는 시간은 많은 요소에 좌우된다.
- 링크 비용이 변하고, 이 새로운 링크 비용이 이 링크에 연결된 어떤 노드의 최소 비용 경로에 변화를 준 경우에만 DV 알고리즘은 수정된 링크 비용을 전파한다.

견고성

라우터가 고장나거나 오동작하거나 파손된다면 어떤 일이 발생할까?

LS 알고리즘

- 라우터는 연결된 링크에 대해 잘못된 비용 정보를 브로드캐스트할 수 있다.
- 노드는 링크 상태 브로드캐스트를 통해 받은 패킷을 변질시키거나 폐기할 수 있다.

그러나 **하나의 링크 상태 노드는 자신의 포워딩 테이블만 계산하기 때문에** 링크 상태 알고리즘에서 경로 계산은 어느 정도 분산되어 수행된다.

따라서 링크 상태 알고리즘은 어느 정도의 견고성을 제공한다.

DS 알고리즘

- 노드는 잘못된 최소 비용 경로를 일부 혹은 모든 목적지에 알릴 수 있다.

각 반복마다 한 노드의 거리 벡터 계산이 이웃에게 전달되고 다음 반복에서 이웃의 이웃에게 간접적으로 전달된다.

따라서 거리 벡터 알고리즘을 사용하는 네트워크에서 한 노드의 잘못된 계산은 **전체로 확산 될 수 있다.**

실제로 1997년에 작은 ISP에서 오작동한 라우터가 잘못된 라우팅 정보를 전국망의 백본 라우터에 제공한 적이 있었다.

이는 다른 라우터들이 오작동한 라우터에게 대규모 트래픽을 보내게 만들었고, 인터넷의 상당 부분이 여러 시간 동안 단절되었다고 한다.