Lecture 5.1

Network Layer: Routing

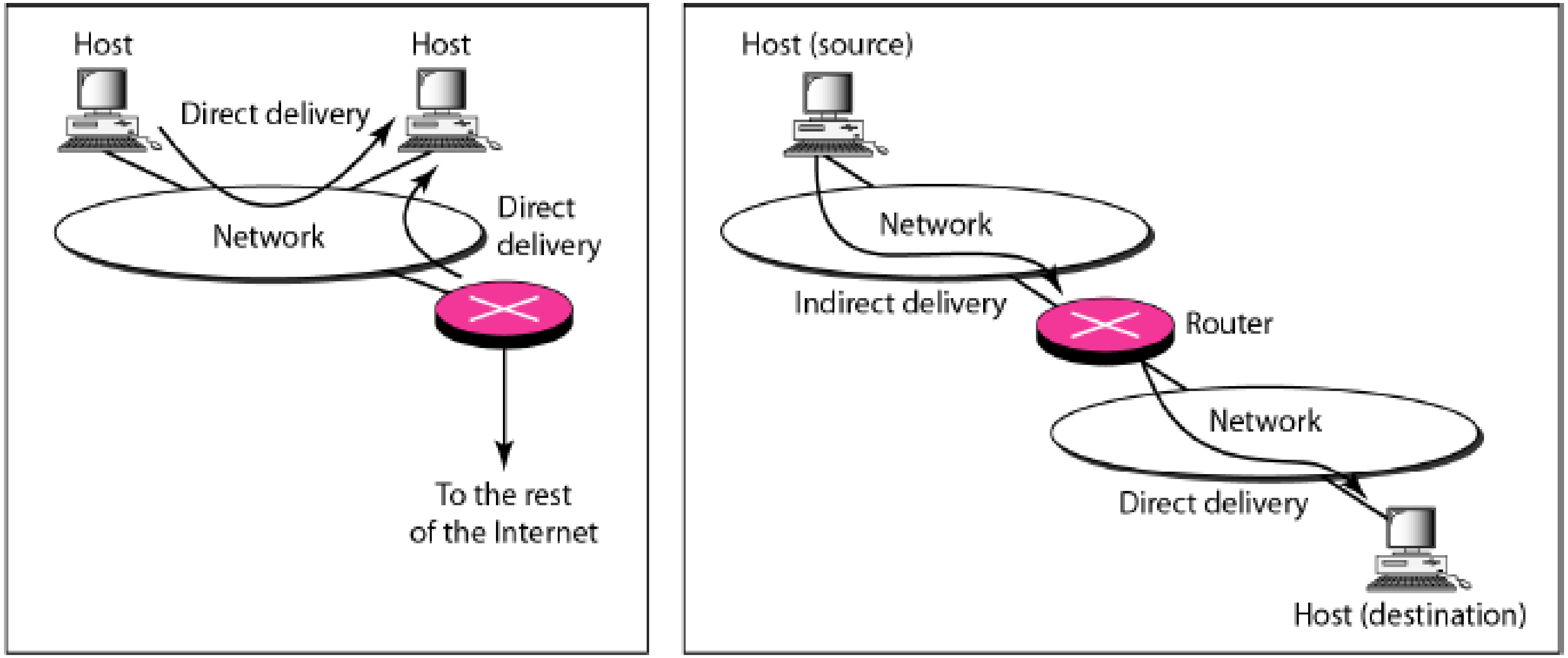
Dr. Vandana Kushwaha

Department of Computer Science

Institute of Science, BHU, Varanasi

DELIVERY

- The **delivery** of a **packet** to its **final destination** is accomplished by using **two different methods** of **delivery**, **direct and indirect**, as shown in Figure below.



a. Direct delivery

b. Indirect and direct delivery

Direct Delivery

- In a **direct delivery**, the **final destination** of the **packet** is a **host** connected to the **same physical network** as the **deliverer.**
- **Direct delivery** occurs :
  - when the **source** and **destination** of the packet are **located** on the **same physical network** or
  - when the **delivery** is **between** the **last router** and the **destination host**.
- The **sender** can easily determine if the **delivery** is **direct**.
- It can extract the **network address** of the **destination** (using the mask) and **compare** this **address** with the **addresses of the networks** to which it is **connected.**
- If a **match** is found, the **delivery** is **direct**.

Indirect Delivery

- If the **destination host** is **not** on the **same network** as the **deliverer,** the **packet** is delivered **indirectly.**
- In an **indirect delivery**, the **packet** goes from **router to router** until it reaches the one connected to the same physical network as its **final destination.**
- A **delivery** always involves **one direct delivery** but **zero or more indirect deliveries**.
- The **last delivery** is always a **direct delivery**.

FORWARDING

- **Forwarding** means to ***place the packet*** *in its* ***route*** *to its* ***destination***.
- **Forwarding** requires a **host** or a **router** to have a **routing table**.
- When a **host** has a **packet** to **send** or when a **router** has **received** a **packet** to be **forwarded**, it **looks** at this **table** to **find** the **route** to the **final destination**.
- However, this **simple solution** is **difficult** today in an **internetwork** such as the **Internet** because the **number of entries** needed in the **routing table** would make **table lookups** **inefficient**.

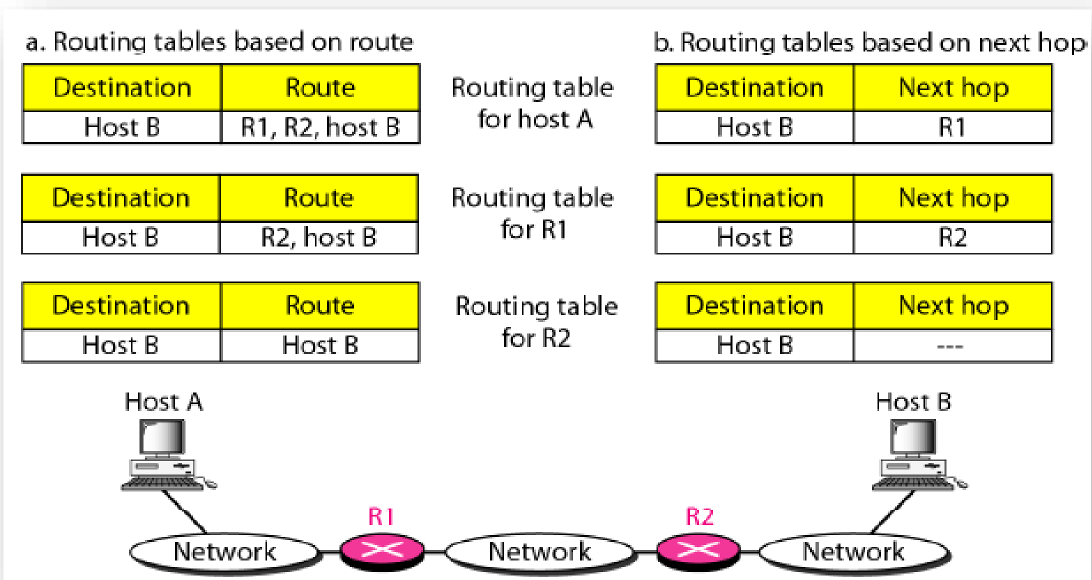
Forwarding Techniques

- **Several techniques** can make the **size** of the **routing table** **manageable** and also **handle** issues such as **security**.
- Following are some **forwarding techniques**:

   1. ***Next-Hop Method versus Route Method***
   2. ***Network-Specific Method versus Host-Specific Method***
   3. ***Default Method***

Next-Hop Method versus Route Method

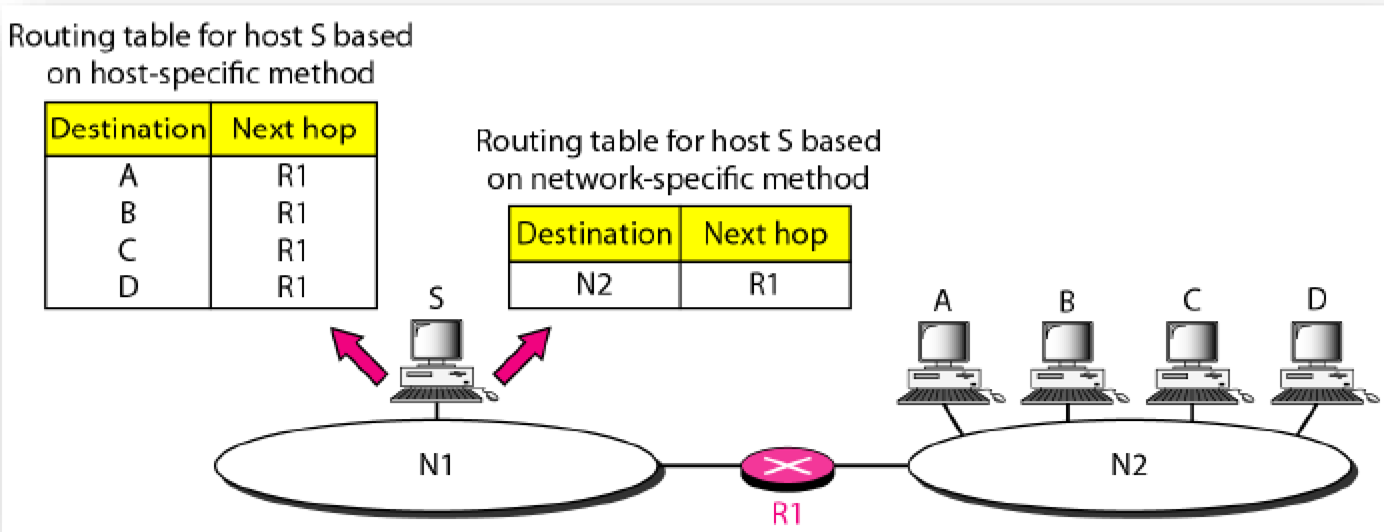
- One **technique** to **reduce** the **contents** of a **routing table** is called the **next-hop method.**
- In this **technique**, the **routing table** holds **only** the **address** of the **next hop** instead of information about the **complete route** (route method).
- The **entries** of a **routing table** must be consistent with one another.



Network-Specific Method versus Host-Specific Method

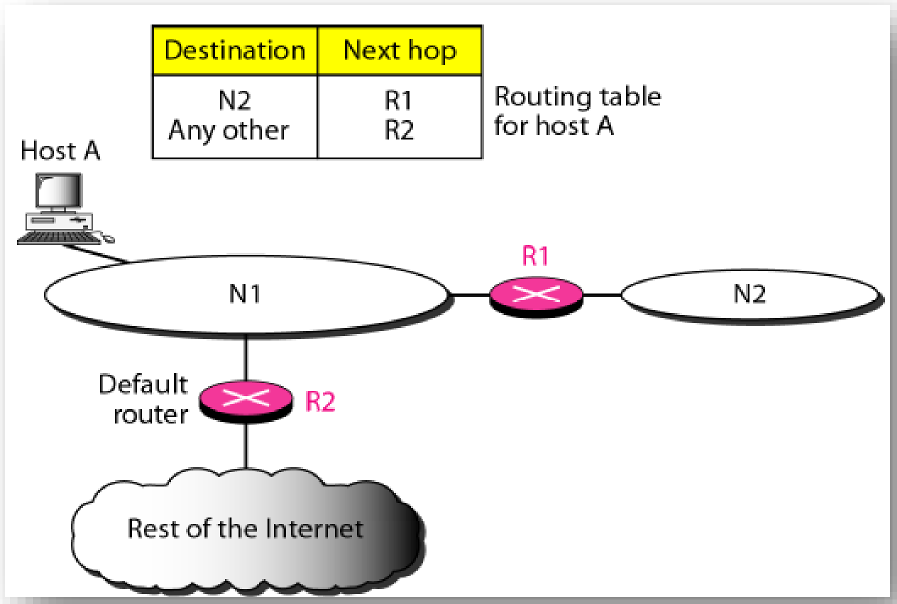
- A **second technique** to **reduce** the **routing table** and **simplify** the **searching process** is called the **network-specific method**.
- In this **method**, instead of having an **entry** for **every destination host** connected to the **same physical network** (host-specific method), we have only **one entry** that defines the **address** of the **destination network** itself.
- In other words, we treat **all hosts** connected to the **same network** as **one single entity.**
- For **example**, if **1000 hosts** are **attached** to the **same network**, only **one entry exists** in the **routing table** instead of **1000.**

Network-Specific Method versus Host-Specific Method



Default Method

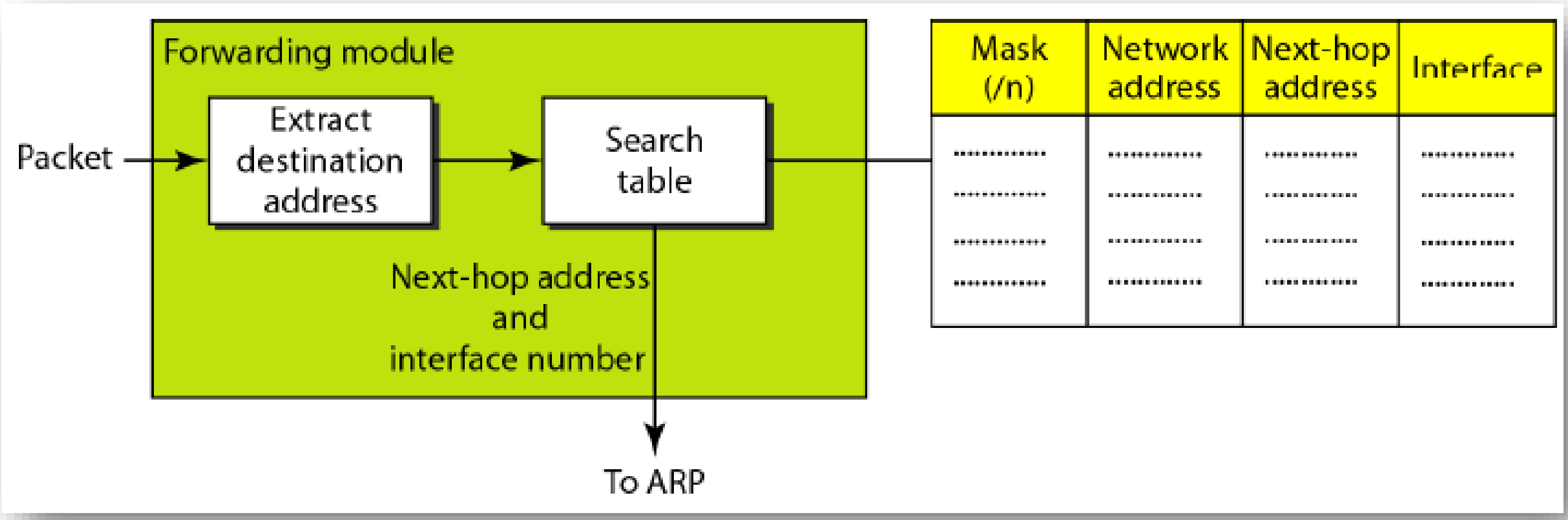
- Another **technique** to **simplify routing** is called the **default method.**
- **Ex. host A** is connected to a **network** with **two routers.**
- **Router R1** routes the packets to **hosts** connected to **network N2.**
- However, for the **rest** of the **Internet**, **router R2** is **used.**
- So instead of listing **all networks** in the **entire Internet**, **host A** can just have **one entry** called the ***default(any other)***.



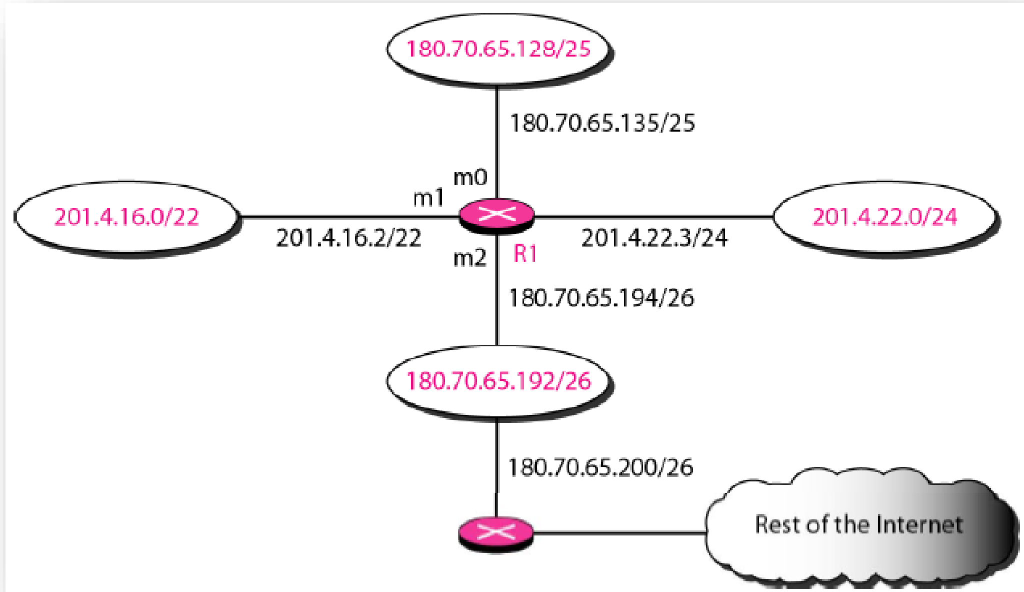
Forwarding Process

- In **classless addressing**, the **routing table** needs to have **one row** of information for each **block** involved.
- The **table** needs to be **searched** based on the **network address** **(first address in the block).**
- Unfortunately, the **destination address** only in the **packet** gives **no clue** **about the network address.**
- To **solve** the **problem**, we need to **include** the **mask** ***(/n)*** in the **table**.
- We need to have an **extra column** that includes the **mask** for the **corresponding block.**
- We need **at least four columns** **in our routing table**; usually there are more.

Forwarding Process



Forwarding Process Example



Mask	*Network Address*	*Next Hop*	*Interface*
/26	180.70.65.192	—	m2
/25	180.70.65.128	—	m0
/24	201.4.22.0	—	m3
/22	201.4.16.0	....	m1
Any	Any	180.70.65.200	m2

Example 1

Show the **forwarding process** if a packet arrives at **R1** in previous Figure with the **destination address** **180.70.65.140**.

Solution

- The **router** performs the **following steps:**

1. The **first mask (/26)** is applied to the **destination address**.

- The result is **180.70.65.128**, which **does not match the corresponding network address**.

2. The **second mask (/25)** is applied to the **destination address**.

- The result is **180.70.65.128**, which **matches** the **corresponding network address**.

- The **next-hop address** (the **destination address** of the packet in this case) and the **interface number m0** are used for **further processing**.

Example 2

Show the **forwarding process** if a **packet** arrives at **R1** with the **destination address** **18.24.32.78.**

Solution

- This time **all masks** are **applied**, one by one, to the **destination address**, but **no matching network address** is **found.**
- When it **reaches** the **end of the table**, the **module** gives the **next-hop address** **180.70.65.200** and interface number **m2.**
- This is probably an **outgoing packet** that needs to be sent, via the **default router**, to someplace else in the **Internet**.

Routing Table

A **host or a router** has a **routing table** which can be either **Static** or **Dynamic**.

Static Routing Table

- A **Static routing table** contains information entered **manually.**
- The **administrator** enters the **route** for each **destination** into the table when a **table** is created.
- It **cannot update automatically** when there is a **change** in the **Internet.**
- The **table** must be **manually altered** by the **administrator**.
- A **static routing table** can be **used** in a **small internet** that **does not change very often**, or in an **experimental internet** for **troubleshooting.**
- It is **poor strategy** to use a **static routing table** in a **big network** such as the **Internet.**

Dynamic Routing Table

- A **Dynamic routing table** is **updated periodically** by using one of the **Dynamic routing protocols** *such as **RIP, OSPF, or BGP***.
- Whenever there is a **change** in the **internet**, such as:
  - *shutdown of a router*
  - *breaking of a link etc.*
- The **Dynamic routing protocols** **update all the tables** in the **routers** (and eventually in the host) **automatically.**
- The **routers** in a big internet such as the **Internet** need to be **updated dynamically** for **efficient delivery** of the **IP packets.**

Format of a Routing Table

- A **routing table** for **classless addressing** has a minimum of **four columns**. However, some of today's routers have even more columns.
- The **number of columns** is **vendor-dependent**, and not all columns can be found in all routers.

Mask	Network address	Next-hop address	Interface	Flags	Reference count	Use
....	....	....	....	....	....	....

- **Mask.** This field defines the mask applied for the entry.
- **Network address.** This field defines the network address to which the packet is finally delivered. In the case of host-specific routing, this field defines the address of the destination host.
- **Next-hop address.** This field defines the address of the **next-hop router** to which the packet is delivered.
- **Interface.** This field shows the **interface id.**

Format of a Routing Table

- **Flags.** This field defines up to **five flags**:
  - **U (up).** The U flag indicates the **router is up and running**. If this flag is not present, it means that the router is down. The packet cannot be forwarded and is discarded.
  - **G (gateway).** The G flag means that the **destination** is **in another network**. The packet is delivered to the **next-hop router** for delivery (**indirect delivery**). When this flag is missing, it means the destination is in this network **(direct delivery**).
  - **H (host-specific).** The H flag indicates that the entry in the network address field is a **host-specific address**. When it is missing, it means that the address is only the **network address** of the destination.
  - **D (added by redirection).** The D flag indicates that routing information for this destination has been **added** to the host routing table by a **redirection message** from **ICMP.**
  - **M (modified by redirection).** The M flag indicates that the routing information for this destination has been **modified by** a redirection message from **ICMP.**

Format of a Routing Table

- **Reference count.**
  - This field gives the **number of users of this route** at the **moment.**
  - For **example**, if five people at the same time are connecting to the same host(destination) from this router, the value of this column is **5.**
- **Use.**
  - This field shows the **number** of **packets transmitted through** this **router** for the **corresponding destination.**

UNICAST ROUTING PROTOCOLS

- A **routing protocol** is a **combination of rules and procedures** that let **routers** in the internet **inform each other** of **changes**.
- It allows **routers** to **share** whatever they know about the **internet** or their **neighborhood.**
- The **sharing of information** allows a **router** at one **geographical location** to know about the **failure** of a **network** at **another location.**
- The **routing protocols** also include **procedures** for **combining information received** from other **routers.**

Optimization

- A **router** receives **a packet** from a **network** and **passes** it to **another network**.
- A **router** is usually **attached** to **several networks.**
- When it **receives** a **packet**, to ***which network should it pass the packet?***
- The **decision is based on optimization**: Which of the available pathways is the **optimum pathway?**
- What is the **definition** of the term ***optimum****?*
- One approach is to **assign a cost(cost metric)** for passing through a network.
- The **metric assigned** depends on the **type of protocol.**

Optimization

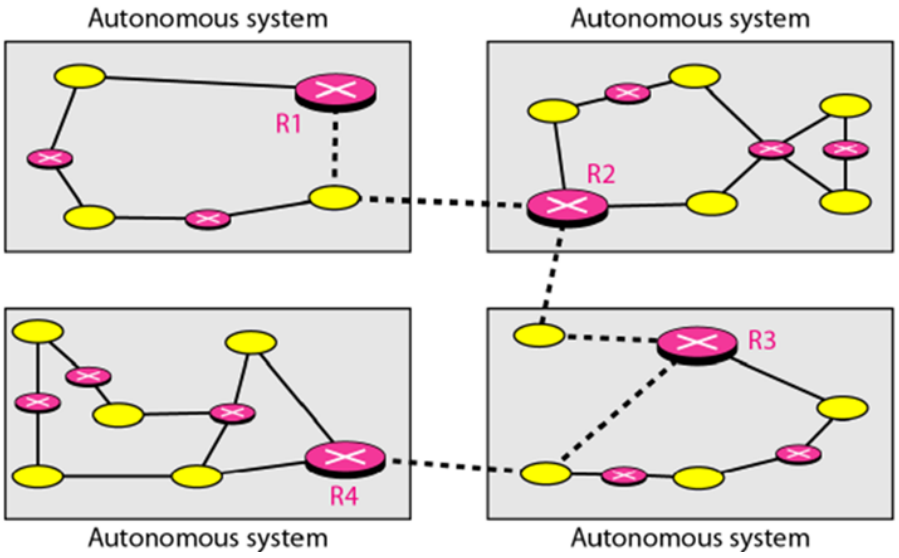
- In **Routing Information Protocol (RIP),** the **cost** of passing through a network is **one hop count**.
- So if a **packet** passes through **10 networks** to reach the **destination**, the **total cost** is **10 hop counts.**
- Other protocols, such as **Open Shortest Path First (OSPF),** allow the **administrator** to **assign a cost** for passing through a **network** based on the **type of service (ToS) required**.
- For **example,**
  - If **maximum throughput** is the **desired type of service**, a **satellite link** has a **lower metric (**more preferred**)** than a **fiber-optic line**.
  - If **minimum delay** is the **desired type of service**, a **fiber-optic line** has a **lower metric (**more preferred**)** than a **satellite link.**

Optimization

- A **satellite link** generally has **high bandwidth** (good throughput) but **high latency** (signal takes ~250 ms one-way due to distance).
- A **fiber-optic link** usually offers both **high bandwidth** and **low latency**.
- When the **ToS = maximum throughput**, the **routing algorithm** (e.g., OSPF, RIP with ToS, or QoS-aware routing) **favors links** that can carry **more data** even if they have **higher delay.**
- If instead **minimum delay** was the desired **ToS**, the **fiber-optic line** would get the **lower metric.**
- **OSPF protocol** allows each router to have **several routing tables** based on the required **type of service.**

Autonomous System

- Today, an **internet** can be so **large** that **one routing protocol cannot handle** the **task** of **updating** the **routing tables** of **all routers.**
- For this reason, an **internet** is **divided** into **autonomous systems.**
- *"An **autonomous system (AS)** is a **group of networks and routers** under the **authority** of a **single administration**".*

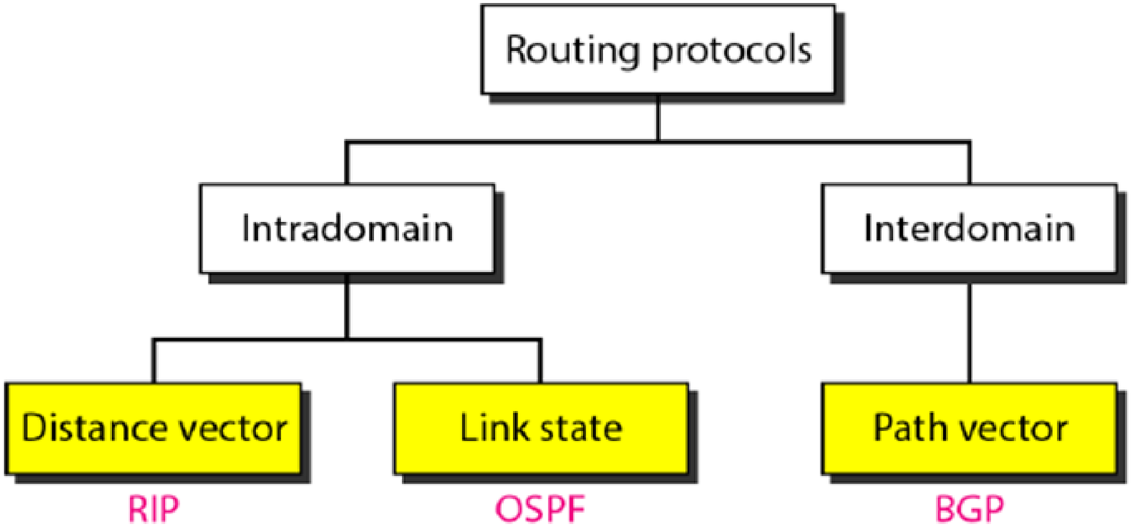


Intra and Inter-domain Routing

- **Routing inside** an **autonomous system** is referred to as **Intradomain routing**.
- Two **intradomain routing algorithms** are **Distance vector routing** and **Link state routing**.
- **Routing between autonomous systems** is referred to as **interdomain routing**.
- One **interdomain routing algorithm** is **Path vector routing**.
- Each **autonomous system** can choose one or more **intradomain routing protocols** to handle **routing inside** the **autonomous system.**
- However, only **one interdomain routing protocol** handles **routing between autonomous systems.**

Popular Routing Protocols

- **Routing Information Protocol (RIP)** is an **implementation** of the **Distance Vector routing algorithm**.
- **Open Shortest Path First (OSPF)** is an **implementation** of the **Link state routing algorithm**.
- **Border Gateway Protocol (BGP)** is an **implementation** of the **Path vector algorithm**

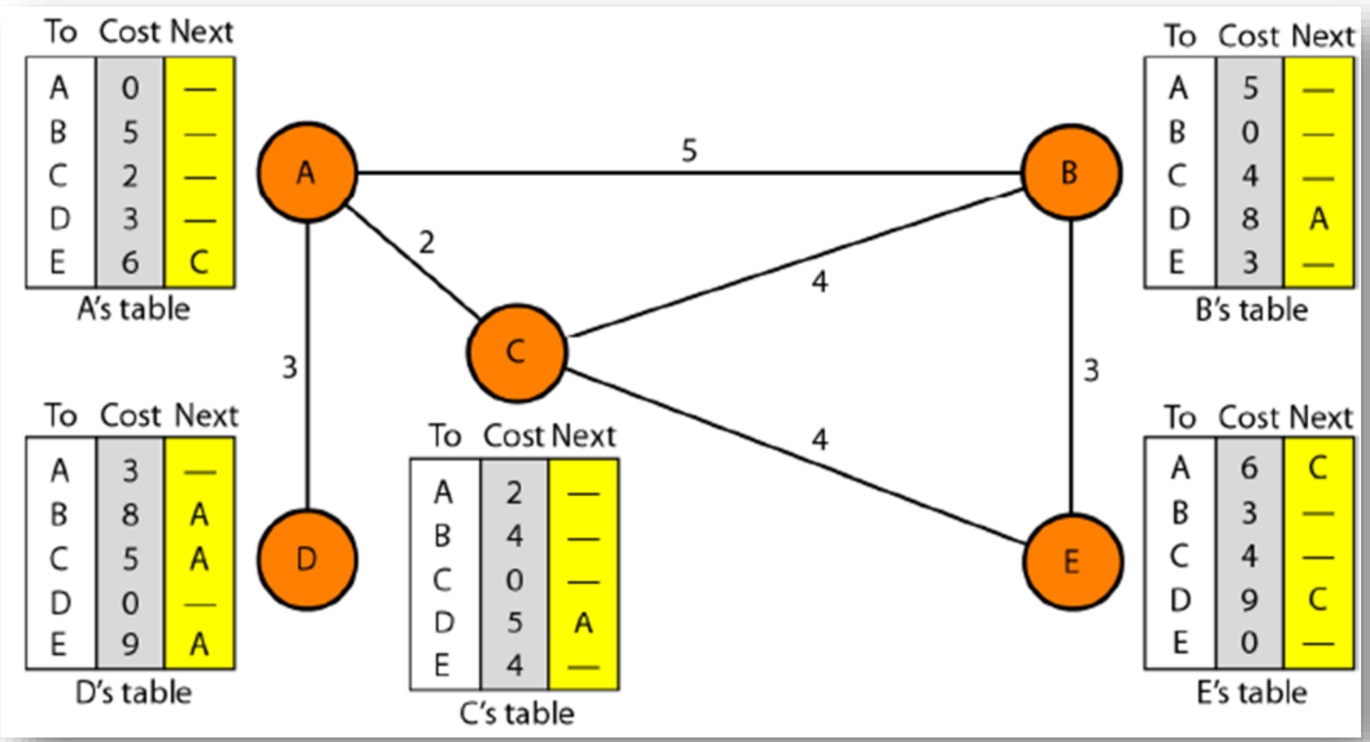


Distance Vector Routing

- In **Distance vector routing**, the **least-cost** **route** between any **two nodes** is the **route** with **minimum distance**.
- In this **protocol**, as the name implies, **each node maintains a vector (table)** of **minimum distances** to every node.
- The ***table*** at **each node** also **guides** the **packets** to the **desired node** by showing the **next hop** in the **route** (next-hop routing).
- We can think of **nodes** as the **cities** in an **area** and the **lines** as the **roads connecting them.**
- A **table** can show a tourist the minimum distance between cities.

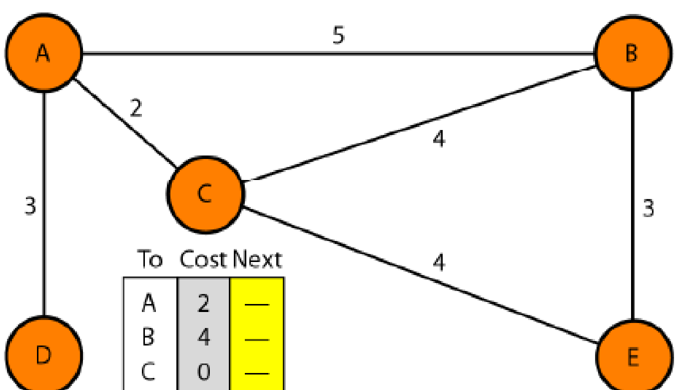
Distance Vector Routing Tables

- The **table** for **node A** shows how we can **reach** any node from this **node.**
- For **example,** our **least cost** to reach **node E** is **6**.
- The **route** passes **through C.**



Step 1: Initialization

- At the **beginning**, **each node** can **know only** the **distance** between **itself** and its **immediate neighbors**, those **directly connected** to it.
- So for the moment, we assume that **each node** can **send** a **message** to the **immediate neighbors** and find the **distance** between itself and these **neighbors.**
- **Figure** below shows the **initial tables** for **each node**. The **distance** for any entry that is **not a neighbor** is marked as **infinite (unreachable).**



A's table

To	Cost	Next
A	0	—
B	5	—
C	2	—
D	3	—
E	∞	

B's table

To	Cost	Next
A	5	—
B	0	—
C	4	—
D	∞	
E	3	—

D's table

To	Cost	Next
A	3	—
B	∞	
C	∞	
D	0	—
E	∞	

C's table

To	Cost	Next
A	2	—
B	4	—
C	0	—
D	∞	
E	4	—

E's table

To	Cost	Next
A	∞	
B	3	B
C	4	C
D	∞	
E	0	D

Step 2: Sharing

- The whole **idea** of **distance vector routing** is the **sharing of information** between **neighbors.**
- Although **node A** does **not know** about **node E**, **node C** does.
- So if **node C** **shares** its **routing table** with **node A**, **node A** can **also know** how to reach **node E.**
- On the other hand, **node C** does **not know** how to reach **node D**, but **node A** does.
- If **node A** **shares** its **routing table** with **node C**, **node C** also **knows** how to reach **node D.**
- In other words, **nodes A** and **C,** as **immediate neighbors**, can **improve** their **routing tables** if they **help** each other.

Step 2: Sharing

- There is only **one problem** while **sharing** the **table.**
- **How much** of the **table** must be **shared** with **each neighbor?**
- The **best solution** for **each node** is to **send** its **entire table** to the **neighbor** and **let** the **neighbor decide** what **part** to **use** and what **part** to **discard.**
- However, the **third column** of a **table (next hop)** is **not useful** for the **neighbor.**
- When the **neighbor** receives a **table,** this **column** needs to be **replaced** with the **sender's id.**
- A **node** therefore **can send only** the **first two columns** of its **table** to any **neighbor.**
- In other words, **sharing** here **means** sharing **only** the **first two columns**.

Step 3: Updating

- When a **node receives** a **two-column table** from a **neighbor,** it needs to **update** its **routing table. Updating** takes **three steps:**

  Step 1.

  - The **receiving node adds** the **cost between itself and the sending node** to **each value** in the **second column** of **received table**.
  - If **node C** claims that its **distance** to a **destination (say B)** is **x meter**, and the **distance between A and** C is **y meter**, then the **distance between A** and **that destination (say B)**, **via C,** is **x + y meter.**

  Step 2.

  - The **receiving node** add the **name** of the **sending node** to **each row** as the **third column**. Now this table is referred as **modified version** of **received table**.

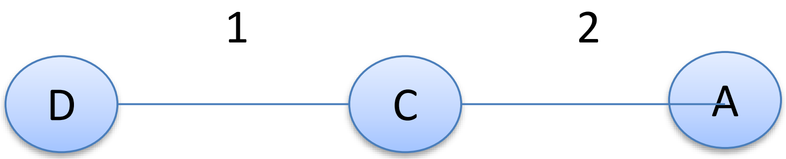
Step 3: Updating

Step 3.

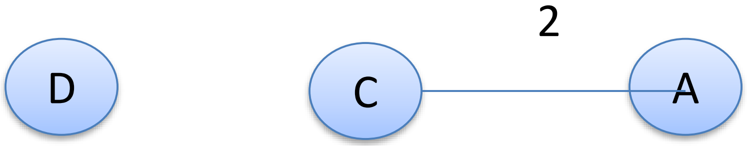
- The **receiving node compares each row of its old table** with the **corresponding row** of the **modified version** of the **received table.**
  - **a.** If the **next-node entry is different** in the above **two tables**, the **receiving node chooses the row with the smaller cost**.
    - If there is a **tie(in cost),** the **old one is kept**.
  - **b.** If the **next-node entry is same**, the **receiving node chooses the new row**.

Updating

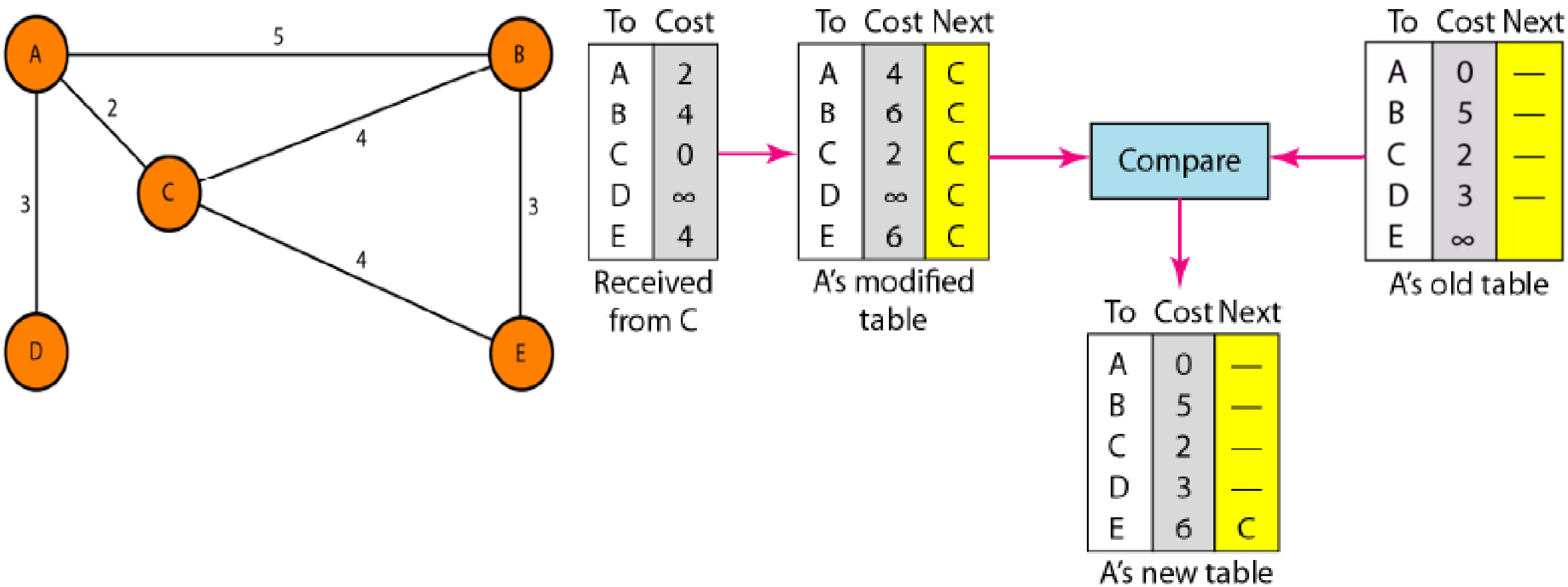
Example: Suppose **node C** has previously **advertised** a **route** to **node D** with **distance 1**.



- Suppose that now there is **no path** between **C** and **D**; **node C** now **advertises** this **route** with a **distance** of **infinity.**
- **Node A** must **not ignore** this **value** even though its **old entry** is **smaller.**
- The **old route** does **not exist** anymore.
- The **new route** has a **distance** of **infinity.**



node A updates its routing table after receiving the partial table from node C.



- **Each node** can **update** its **table** by using the **tables received** from **other nodes.**
- In a short time, if there is no change in the network itself, such as a failure in a link, each node reaches a **stable condition** in which the **contents** of its **table** remains the **same.**

When to Share?

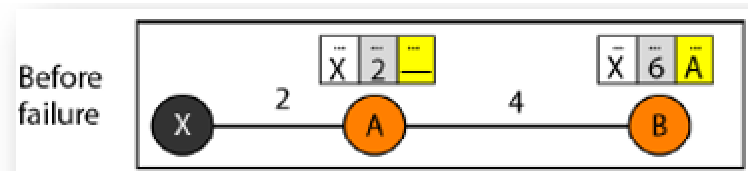
- The **question** now is, **when does a node send its partial routing table** (only two columns) to all its **immediate neighbors**? There are **two types** of **update**:
- **Periodic Update** A node sends its routing table, normally **every 30 sec**, in a periodic update.
- **Triggered Update** A node sends its two-column routing table to its **neighbors anytime** there is a **change** in its **routing table**.
- The **change** can **result** from the **following:**

1. A **node** receives a **table** from a **neighbor,** resulting in **changes** in its **own table** after **updating.**

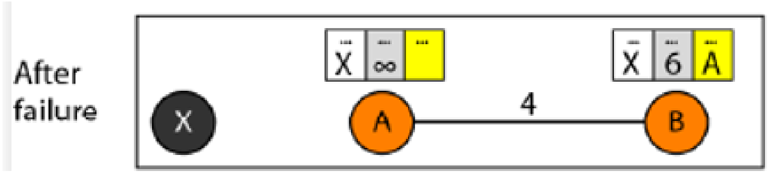
2. A **node** detects some **failure** in the **neighboring links** which results in a **distance** change to **infinity.**

Two-Node Loop Instability

- A **problem** with **distance vector routing** is **instability**, which **means** that a **network** using this **protocol** can become **unstable**.
- To understand the problem, let us look at the **scenario** depicted in **Figure** below.
- At the **beginning**, both **nodes A** and **B** know how to reach **node X.**



- But **suddenly**, the **link** between **A and X fails** and **Node A changes** its **table.**

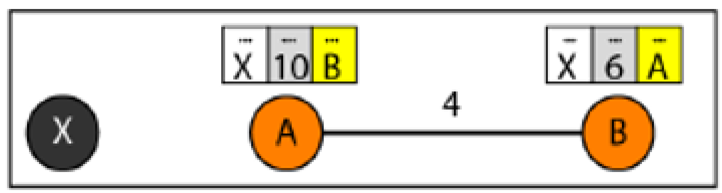


- If **A** can **send** its **table** to **B immediately**, everything is **fine.**

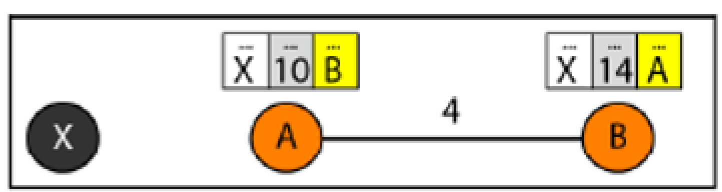
Two-Node Loop Instability

- However, the **system** becomes **unstable** if **B sends** its **routing table** to **A before receiving A's routing table**.
- In such case **Node A receives** the **update** and, **assuming** that **B** has **found a way** to **reach X** and **A immediately updates** its **routing table.**

After A receives update from B



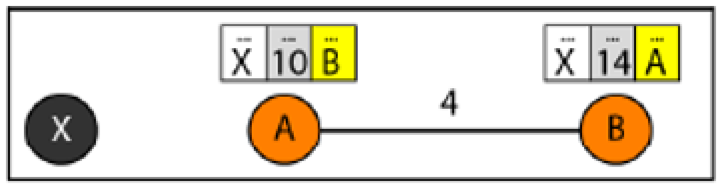
- Based on the <u>**triggered update strategy**</u>, **A sends** its **new update** to **B.**
- Now **B thinks** that something has been **changed** around **A** and **updates** its **routing table.**



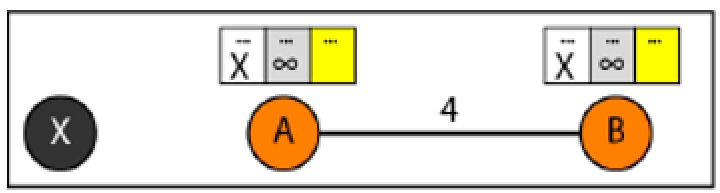
After B receives update from A

Two-Node Loop Instability

- The **cost** of **reaching X increases gradually** until it **reaches infinity.**



After B receives update from A



Finally

- At this **moment**, both **A and B** know that **X cannot** be **reached.**
- However, **during this time** the **system** is **not stable**.
- **Node A thinks** that the **route to X** is **via B**; **node B thinks** that the **route to X** is **via A.**

Two-Node Loop Instability

- If **A receives** a **packet destined** for **X**, it **goes to B** and then **comes back** to **A.**
- Similarly, if **B receives** a **packet destined** for **X**, it **goes to A** and **comes back** to **B.**
- Thus **packets bounce between A and B**, creating a **two-node loop problem.**

Solutions to two node loop instability problem

A few **solutions** have been **proposed** for **instability** of this kind:

1. Defining Infinity

- The **first** obvious **solution** is to **redefine infinity** to a **smaller number**, such as **100.**
- For our previous scenario, the **system** will be **stable** in **less than 20 updates**.
- As a matter of **fact**, most **implementations** of the **distance vector protocol** define the **distance** between **each node** to be **1** and **define 16** as **infinity**.
- However, this **means** that the ***distance vector routing cannot be used in large systems*.**
- The **size of the network**, in each direction, **cannot exceed 15 hops**.

Solutions to two node loop instability problem

2. Split Horizon

- In this strategy, **instead** of **flooding** the **table** through **each interface**, **each node sends only part** of its **table** through each interface.
- If, according to its **table**, ***node B*** *thinks that the* ***optimum route*** *to* ***reach X*** *is* ***via A, it does not need to advertise*** *this* ***piece*** *of* ***information to A;*** *the information has come from A (****A already knows****)*.
- ***Taking information from node A****,* ***modifying it****, and* ***sending it back*** *to* ***node A creates the confusion.***

Solutions to two node loop instability problem

2. Split Horizon

- In our scenario, **node B** **eliminates the last line** of its **routing table** **before** it **sends** it to **A.**
- In this case, **node A** **keeps** the value of **infinity** as the **distance** to **X.**
- Later when **node A** **sends** its **routing table** to **B,** **node B** also **corrects** its **routing table.**
- The **system** **becomes stable** after the **first update**: both **node A and B know** that **X** is **not reachable.**

Solutions to two node loop instability problem

3. Split Horizon and Poison Reverse

- Using the **split horizon** strategy has **one *drawback***.
- Normally, the **distance vector protocol** uses a **timer,** and if **there is no news** about **a route**, the **node deletes** the **route** from its **table.**
- When **node B** in the **previous scenario** **eliminates** the **route to X** from its **advertisement** to **A;**
- ***Node A* *cannot guess*** *that this is* ***due to*** *the* ***split horizon strategy*** *(the source of information was A)* ***or*** *because* ***B has not received*** *any* ***news*** *about* ***X recently.***

Solutions to two node loop instability problem

3. Split Horizon and Poison Reverse

- The **split horizon strategy** can be **combined** with the **poison reverse** **strategy**.
- ***Node B*** *can* ***still advertise*** *the* ***value*** *for* ***X****, but if the* ***source*** *of* ***information*** *is* ***A,*** *it* ***can replace*** *the* ***distance*** *with* ***infinity*** *as a* ***warning: "Do not use this value; what I know about this route comes from you."***

RIP(Routing Information Protocol)

- The **Routing Information Protocol (RIP)** is an **intradomain routing protocol** used **inside an autonomous system.**
- It is a **very simple protocol** based on **distance vector routing**. **RIP** implements **distance vector routing** directly with some considerations:

1. In an **autonomous system**, we are dealing with **routers** and **networks** (links). The **routers** have **routing tables; networks** do not.

2. The **destination** in a **routing table** is a **network,** which means the **first column defines** a **network address.**

3. The **metric** used by **RIP** is **very simple;** the **distance** is defined as the **number of links (networks) to reach the destination**. For this reason, the **metric in RIP** is called a **hop count.**

RIP(Routing Information Protocol)

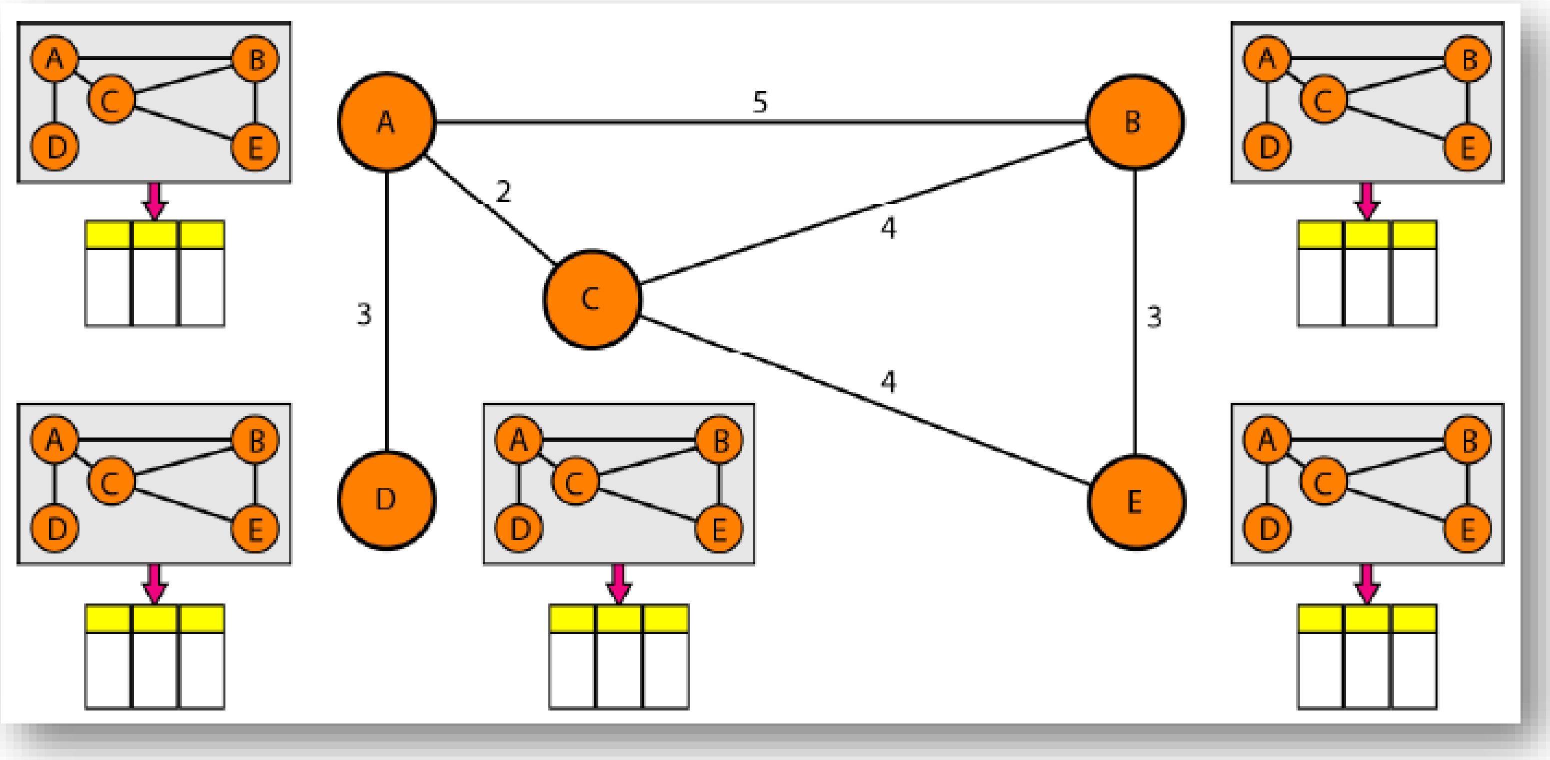
4. Infinity is **defined as 16,** which means that **any route** in an **autonomous system** using **RIP cannot** have **more than 15 hops.**

5. The **next-node column** defines the **address** of the **router** to which the **packet** is to be **sent** to reach its **destination.**

Link State Routing Algorithm

- **Link state routing** has a **different philosophy** from that of **distance vector routing**.
- In **link state routing**, if **each node** in the **domain** has the knowledge of **entire topology** of the **domain :**
  - the **list of nodes and links**,
  - **cost (metric)**, and
  - **condition** of the **links** (up or down)
- The **node** can use ***Dijkstra's algorithm*** to **build** a **routing table**.
- **Figure** on the next slide shows the **concept.**

Link State Routing Algorithm



Link State Routing Algorithm

- The **figure** on **previous slide** shows a **simple domain** with **five nodes.**
- Each **node** uses the **same topology** to create a **routing table**, but the **routing table** for **each node** is **unique** because the **calculations** are based on **different interpretations** of the **topology.**
- This is **analogous** to a **city map.**
- While **each person** may have the **same map**, each needs to take a **different route** to reach her **specific destination.**
- **Link state routing** is based on the **assumption** that, although the global knowledge about the **topology** is not clear, **each node** has **partial knowledge** of its links.
- In other words, the **whole topology** can be **compiled** from the **partial knowledge** of **each node.**

Building Routing Tables

- In **link state routing, four sets of actions** are **required** to **ensure** that **each node** has the **routing table** showing the **least-cost node** to every **other node.**

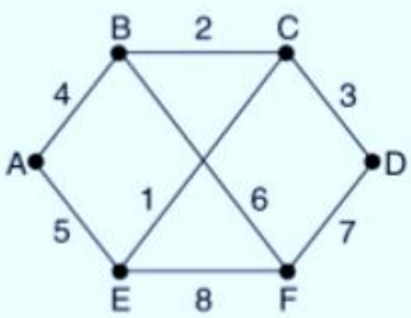
1. **Creation** of the states of the links by each node, called the **link state packet (LSP).**
2. **Dissemination of LSPs** to every other router, called **flooding,** in an efficient and reliable way.
3. **Formation of a shortest path tree** for each node.
4. **Calculation** of a **routing table** based on the **shortest path tree**.

1. Creation of Link State Packet (LSP)

- A **link state packet** can **carry** a **large amount** of **information.**
- For the moment, however, we assume that it carries a minimum **amount of data**:
  - the **node identity,**
  - the **list of links,**
  - a **sequence number**, and
  - **Age**.

- The first two, **node identity** and the **list of links**, are **needed** to make the **topology.**
- The third, **sequence number**, facilitates **flooding** and distinguishes **new LSPs** from **old ones.**
- The fourth, **age**, prevents **old LSPs** from remaining in the domain for **a long time.**

Building Link State Packets



(a)

Link State Packets

A	B	C	D	E	F
Seq.	Seq.	Seq.	Seq.	Seq.	Seq.
Age	Age	Age	Age	Age	Age
B 4	A 4	B 2	C 3	A 5	B 6
E 5	C 2	D 3	F 7	C 1	D 7
	F 6	E 1		F 8	E 8

(b)

(a) A subnet. (b) The link state packets for this subnet.

1. Creation of Link State Packet (LSP)

- **LSPs** are **generated** on ***two occasions***:
- ***When there is a change in the topology of the domain.***
  - **Triggering** of **LSP dissemination** is the main way of quickly informing any node in the domain to update its topology.
- ***On a periodic basis.***
  - The **time period** in this case is **much longer** **(60 min** or **2 hours)** compared to **distance vector routing**.
  - As a matter of fact, there is no actual need for this type of **LSP dissemination**.
  - It is done to ensure that **old information(if any)** is **removed** from the domain.
  - A **longer period** **ensures** that **flooding does not create too much traffic** on the network.

2.Flooding of LSPs

- After a **node** has **prepared** an **LSP**, it must be **disseminated** to **all other nodes**, **not only** to its **neighbors.**
- The process is called **flooding** and based on the following:

1. The **creating node** **sends a copy** of the **LSP** out of **each interface.**

2. A **node** that receives **an LSP compares** it with the **copy** it may **already have**.

- **If** the **newly arrived LSP** is **older** than the **one** it has (found by checking the sequence number), it **discards the LSP**.
- If it is **newer**, the **node** does the following:

  a. It **discards** the **old LSP** and keeps the **new one**.

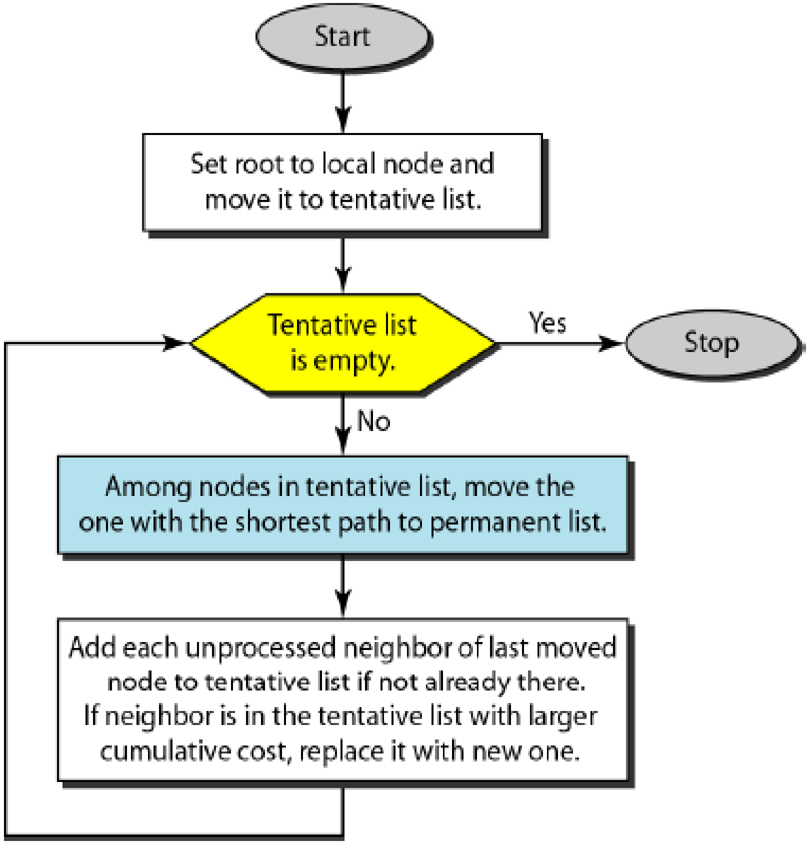
  b. It **sends** a **copy** of it **out** of **each interface except** the **one** from which the **packet arrived.**

3.Formation of Shortest Path Tree using *Dijkstra Algorithm*:

- After **receiving all LSPs**, each **node** will have a **copy of the whole topology**.
- However, the **topology** is **not sufficient** to find the **shortest path** to every other node;
- A **Shortest Path Tree** is **needed.**
- A **tree** is a **acyclic graph** having **nodes** and **links**; **one node** is called the **root.**
- All **other nodes** can be **reached** from the **root** through **only one single route**.
- A **shortest path tree** is a **tree** in which the **path between** the **root** and **every other node** is the **shortest.**
- What we need for **each node** is a **shortest path tree** with that **node** as the **root.**

Dijkstra's Algorithm

- ***Dijkstra Algorithm*** creates a **Shortest Path tree** from a **graph**.
- The **algorithm divides** the **nodes** into **two sets**: **tentative** and **permanent.**
- It **finds** the **neighbors** of a **current node**, **makes them tentative,** examines them, and if they **pass** the **criteria,** makes them **permanent**.

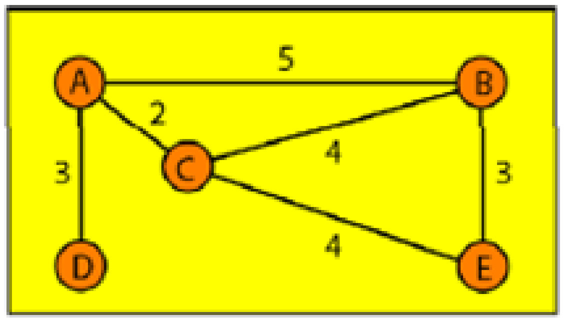


Steps of Applying Dijkstra’s Algorithm

- The following shows the **steps**. At the **end** of each **step,** we show the **permanent (filled circles)** and the **tentative (open circles) nodes** and **lists** with the **cumulative costs.**

  i. ***Permanent list:*** *empty*

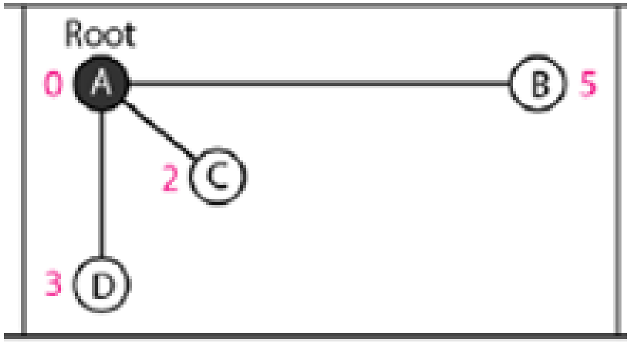
  Tentative list: *A(0)*



1. Set root to A and move A to tentative list.

  ii. ***Permanent list:*** *A(0)*

  Tentative list: *B(5), C(2), D(3)*

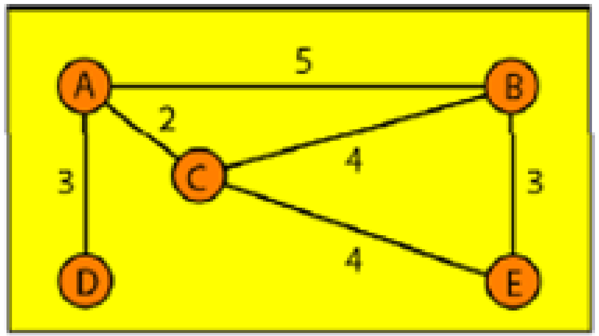


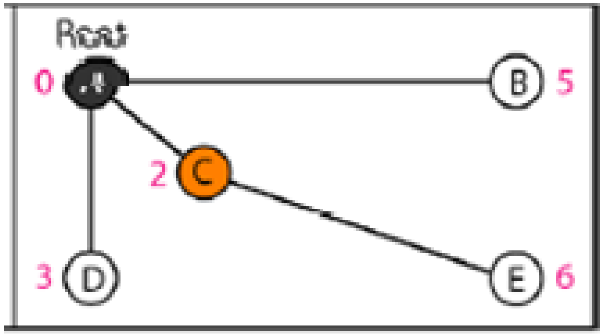
2. Move A to permanent list and add B, C, and D to tentative list.

Steps of Applying Dijkstra’s Algorithm

iii. ***Permanent list:*** *A(0), C(2)*

Tentative list: *B(5), D(3), E(6)*

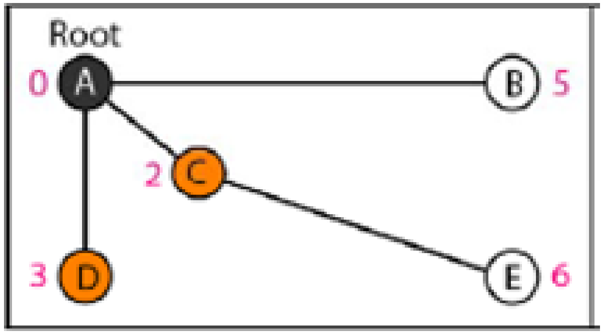




3. Move C to permanent and add E to tentative list.

iv. ***Permanent list:*** *A(0), C(2), D(3)*

Tentative list: *B(5), E(6)*

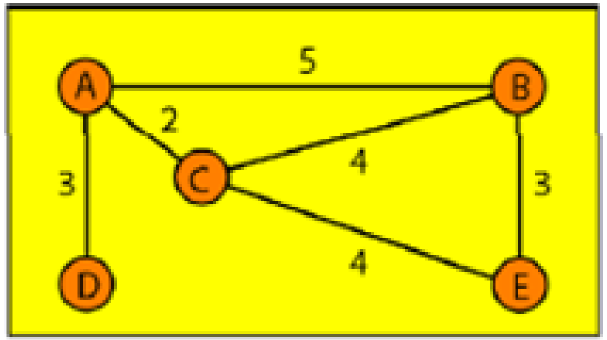


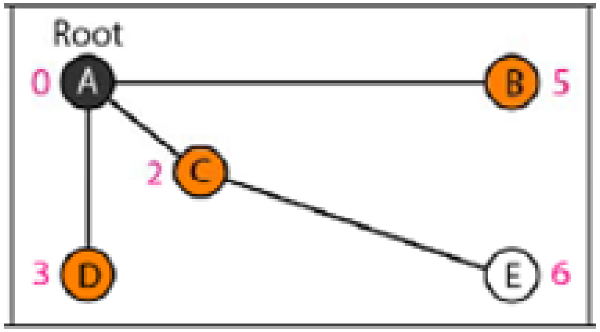
4. Move D to permanent list.

Steps of Applying Dijkstra's Algorithm

v. ***Permanent list:*** *A(0), B(5), C(2), D(3)*

Tentative list: *E(6)*

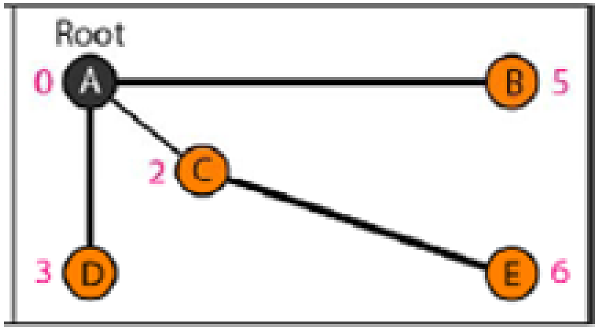




5. Move B to permanent list.

vi. ***Permanent list:*** *A(0), B(5), C(2), D(3), E(6)*

Tentative list: ***empty***



6. Move E to permanent list (tentative list is empty).

4. Calculation of Routing Table from Shortest Path Tree

- Each **node** uses the **shortest path tree algorithm** to construct its **routing table**.
- The **routing table** shows the **cost** of **reaching** each **node** from the **root.**
- **Table** below shows the **routing table** for **node A**.

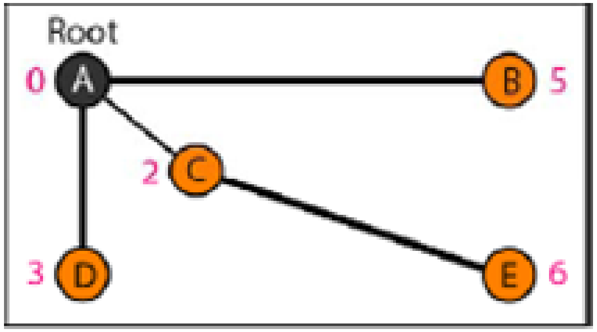


Table. ***Routing table for node A***

Node	*Cost*	*Next Router*
A	0	—
B	5	—
C	2	—
D	3	—
E	6	C

- Both **Distance vector routing** and **Link state routing** end up with the **same routing table** for **node A.**

OSPF(Open Shortest Path First)

- The **Open Shortest Path First** or **OSPF protocol** is an **Intradomain routing protocol** based on **Link state routing algorithm**.
- The **OSPF protocol** allows the **administrator** to assign a **cost**, called the **metric**, to **each route.**
- The **metric** can be based on a **type of service** (minimum delay, maximum throughput, and so on).
- As a matter of fact, a **router** can have **multiple routing tables**, each based on a **different type of service.**

Path Vector Routing

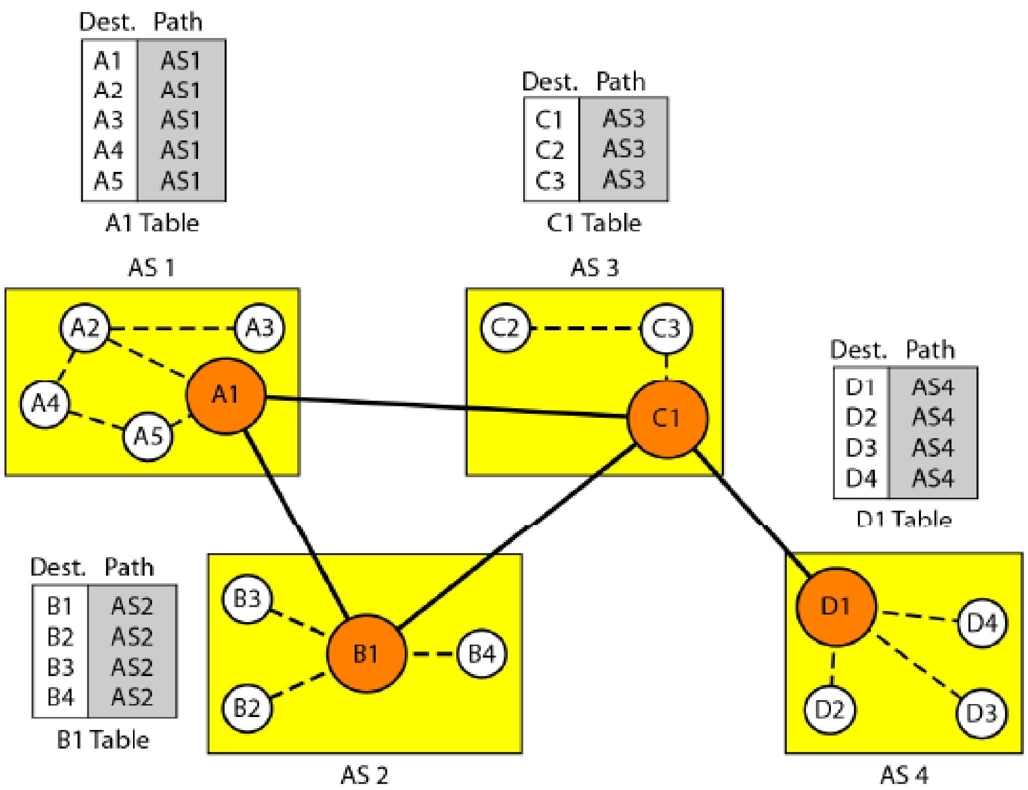
- **Distance vector** and **Link state routing** are both **intradomain routing protocols**.
- They **can be used inside an autonomous system** (AS), but **not between autonomous systems.**
- These **two protocols** are **not suitable for interdomain routing** mostly **because of scalability.**
- **Distance vector routing** is subject to **instability** if there are more than a few **hops** in the **domain** of operation.
- **Link state routing** needs a **huge amount of resources** to **calculate routing tables**.
- It also creates **heavy traffic** because of **flooding**.
- There is a **need** for a **third routing protocol** which we call **Path vector routing**.
- **Path vector routing** proved to be **useful** for **interdomain routing**.

Path Vector Routing

- The **principle** of **path vector routing** is **similar to** that of **distance vector routing**.
- In **path vector routing**, we assume that there is **one node in each autonomous system that acts on behalf of the entire autonomous system**.
- Let us call it the **speaker node**.
- The **speaker node** in an **AS creates** a **routing table** and **advertises** it to **speaker nodes** in the **neighboring ASs.**
- The **idea** is the same as for **distance vector routing** except that **only speaker nodes** in **each AS can communicate with each other**.
- However, what is advertised is **different**.
- A **speaker node advertises** the **path**, **not the metric** of the **nodes**, in its **autonomous system** or other **autonomous systems.**

Path Vector Routing

- At the **beginning**, each **speaker node** can **know** only the **reachability** of **nodes inside** its **autonomous system**. Figure below shows the **initial tables** for each **speaker node** in a **system** made of **four ASs.**



Path Vector Routing

Step 1. Initialization

- Node **A1** is the **speaker node** for **AS1, B1** for **AS2, C1** for **AS3**, and **D1** for **AS4.**
- Node **A1** creates an **initial table** that shows **A1 to A5** are located in **AS1** and can be **reached** through it.
- Node **B1** table shows that **B1 to B4** are located in **AS2** and can be **reached** through **B1**.
- And so on.

Path Vector Routing

Step 2. Sharing

- Just as in **distance vector routing**, in **path vector routing**, a **speaker** in an **autonomous system** **shares** its **table** with **immediate neighbors**.
- **Node A1** shares its table with **nodes B1** and **C1.**
- **Node C1** shares its table with **nodes D1, B1**, and **A1.**
- **Node B1** shares its table with **C1** and **A1.**
- **Node D1** shares its table with **C1.**

Path Vector Routing

Step 3. Updating

- When a **speaker node receives a two-column table from a neighbor**, it **updates its own table** by **adding** the nodes that are not in its **routing table** and **adding** its **own autonomous system** and the **autonomous system** that **sent** the **table.**
- After a while **each speaker** has a **table** and knows how to reach **each node** in other **ASs.** (Figure on next slide shows the tables for each **speaker node** after the **system** is **stabilized**.)
- According to the figure, if **router A1** receives a **packet** for **nodes A3**, it knows that the **path** is in **AS1** (the packet is at home); but if it **receives** a **packet** for **D1,** it knows that the **packet** should go from **AS1, to AS2**, and then **to AS4.**
- The **routing table** shows the **path completely**. On the other hand, if node **D1** in **AS4** receives a packet for node **A2**, it knows it should go through **AS4, AS3**, and **AS1.**

Stabilized tables for three Autonomous Systems

Dest.	Path
A1	AS1
…	
A5	AS1
B1	AS1-AS2
…	…
B4	AS1-AS2
C1	AS1-AS3
…	…
C3	AS1-AS3
D1	AS1-AS2-AS4
…	…
D4	AS1-AS2-AS4

A1 Table

Dest.	Path
A1	AS2-AS1
…	
A5	AS2-AS1
B1	AS2
…	…
B4	AS2
C1	AS2-AS3
…	…
C3	AS2-AS3
D1	AS2-AS3-AS4
…	…
D4	AS2-AS3-AS4

B1 Table

Dest.	Path
A1	AS3-AS1
…	
A5	AS3-AS1
B1	AS3-AS2
…	…
B4	AS3-AS2
C1	AS3
…	…
C3	AS3
D1	AS3-AS4
…	…
D4	AS3-AS4

C1 Table

Dest.	Path
A1	AS4-AS3-AS1
…	
A5	AS4-AS3-AS1
B1	AS4-AS3-AS2
…	…
B4	AS4-AS3-AS2
C1	AS4-AS3
…	…
C3	AS4-AS3
D1	AS4
…	…
D4	AS4

D1 Table