Module 4
Network Layer
Routing Protocols

BECE401L

Outline

Distance Vector

Link State Routing

Routing Protocols

Routing

Unicast Routing:

- A packet is routed, hop by hop, from its source to its destination by the help of forwarding tables.

Source Host:

- Needs no forwarding table
  - As it delivers its packet to the default router in its local network.

Destination Host:

- Needs no forwarding table either
  - As it receives the packet from its default router in its local network.

Routing a packet from its source to its destination means:

- Routing the packet from a source router (the default router of the source host) to a destination router (the router connected to the destination network).
- There are several routes that a packet can travel from the source to the destination;

What must be determined is which route the packet should take?

Internet As a Graph

To find the best route, An internet can be modeled as a ***graph***.

- A graph in computer science is a set of *nodes* and *edges* (lines) that connect the nodes.

To model an Internet as a Graph:

- Each router as a node
- Each network between a pair of routers as an edge.

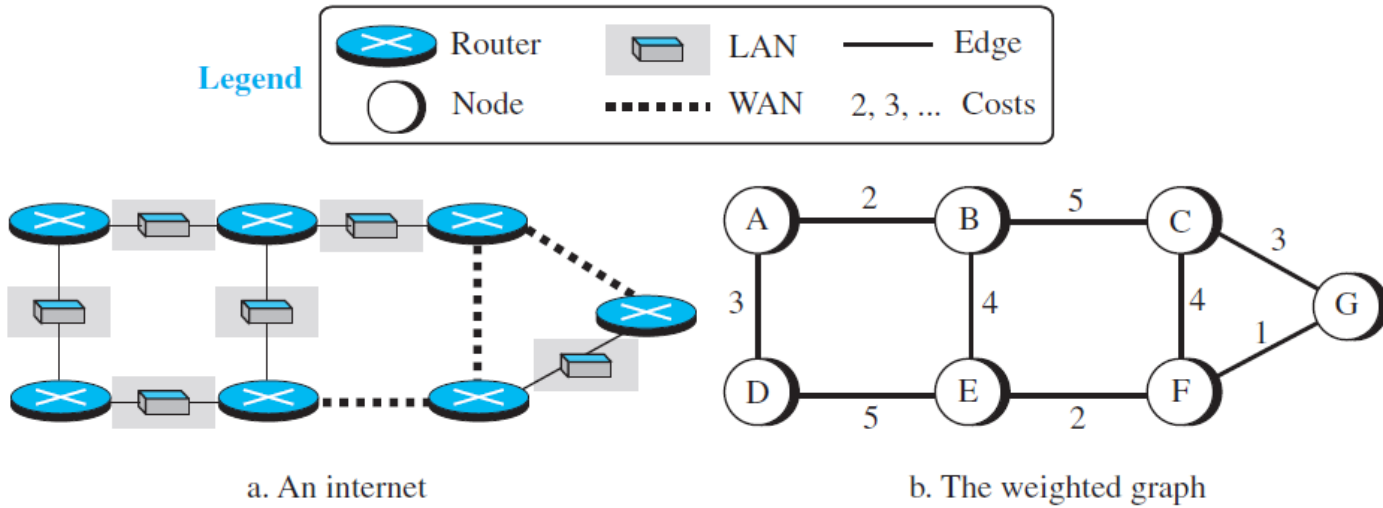


Fig: Internet can be modeled as a graph.

Internet as a *weighted graph*,

- In which each edge is associated with a cost.
  - If a weighted graph is used to represent a geographical area
  - Nodes can be cities and the edges can be roads connecting the cities;
  - Weights, are distances between cities.
- In routing, we assume that there is a **cost associated with each edge**.
- If there is no edge between the nodes, the cost is infinity.

Least Cost Trees

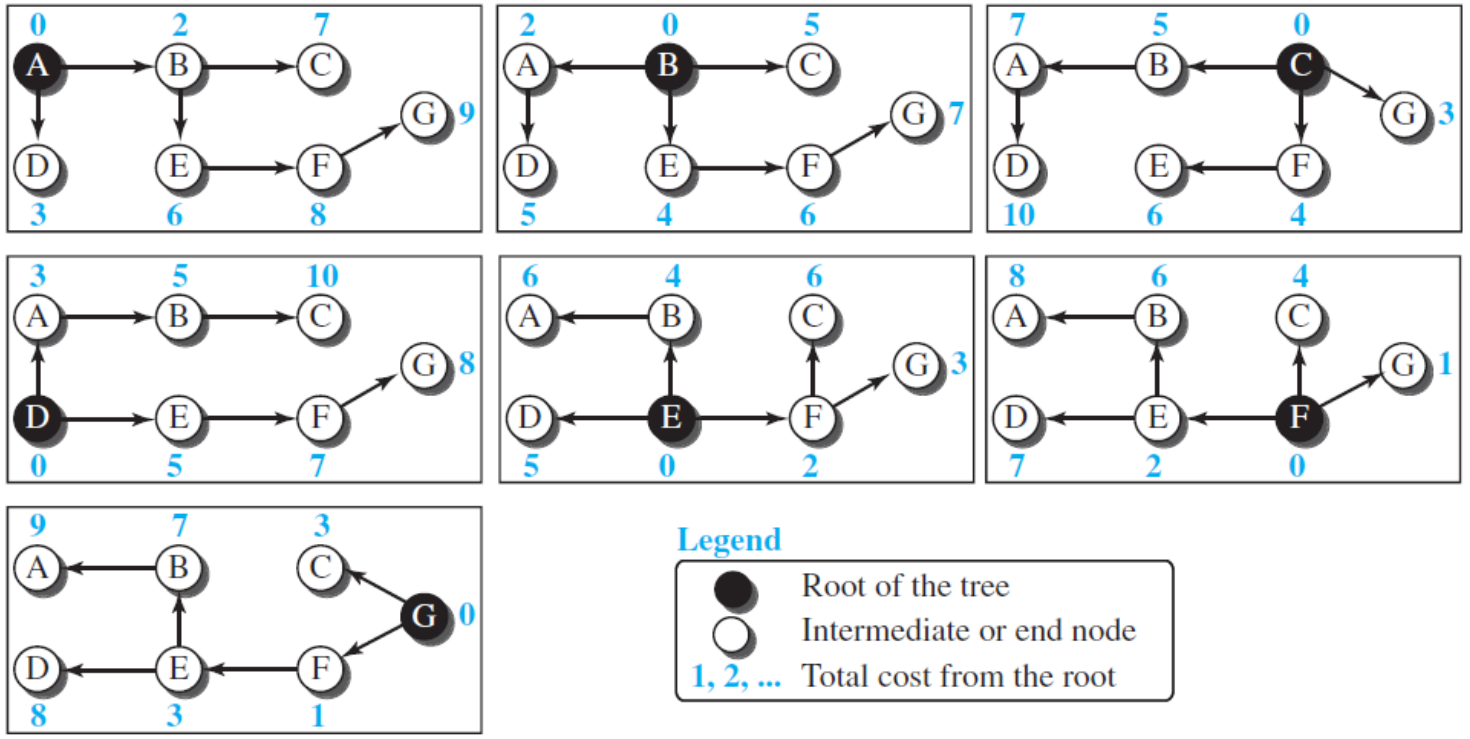
If there are *N* routers in an internet,

- There are (*N* − 1) least-cost paths from each router to any other router.
- **Means we need *N* × (*N* − 1) least-cost paths for the whole internet.**
- **Example:** If we have only 10 routers in an internet, we need 90 least-cost paths.

A better way to see all of these paths is to combine them in a **Least-cost Tree**.

Least-Cost Tree

- A tree with the **source router as the root** that spans the whole graph and in which the path between the root and any other node is the shortest.
- In this way, we can have only one shortest-path tree for each node; we have *N* least-cost trees for the whole internet.



Routing Algorithms

Several routing algorithms have been designed in the past.

Key Differences:

- How these algorithms interpret the least cost.
- The way they create the least-cost tree for each node.

Three major types:

- **Distance Vector Routing**
- **Link State Routing**
- **Path Vector Routing**

Distance Vector Routing

Goal: Try to find the best route.

Firstly, Each node *creates is its own least-cost tree with the primary information it has about its immediate neighbor*.

Then, **Incomplete trees are exchanged between immediate neighbors**

- To make the **trees more and more complete** and to represent the whole internet.

In DV Routing: A router continuously tells all of its neighbors what it knows about the whole internet (even incomplete information).

Now, how incomplete least-cost trees can be combined to make complete ones?

- Bellman-Ford equation
- Concept of distance vectors

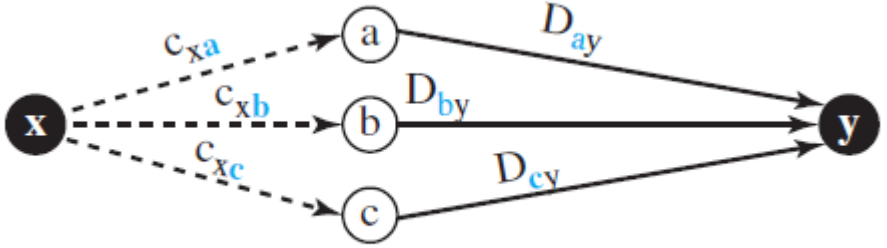
Distance Vector Routing

a. Bellman-Ford equation

Used to **find the least cost (shortest distance)**

- Between a source node, ***x***, and a destination node, ***y***, through some intermediary nodes (**a, b, c, …**).
- When the **costs** between the source and the intermediary nodes and **the least costs** between the intermediary nodes and the destination are given.
- Here,
  - D_{ij} is the shortest distance and C_{ij} is the cost between nodes ***i*** & ***j***.

$$\boldsymbol{D_{xy} = min\{(c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}), \dots\}}$$

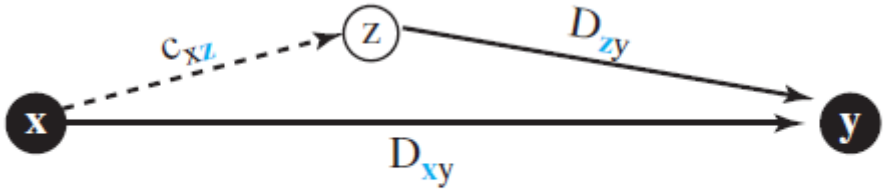


a. General case with three intermediate nodes

In Distance-vector Routing,

- Desired to update an existing least cost with a least cost through an intermediary node, such as z, if the latter is shorter.
- The equation becomes simpler, as:

$$\boldsymbol{D_{xy} = min\{D_{xy}, (c_{xz} + D_{zy})\}}$$



b. Updating a path with a new route

Distance Vector Routing

b. Distance Vectors

Least-cost tree: Combination of *least-cost paths from the root to all destinations*.

- Paths are graphically glued together to form the **tree**.
- Distance-vector routing unglues these paths and creates a ***distance vector***,
  - A one-dimensional array to represent the tree.

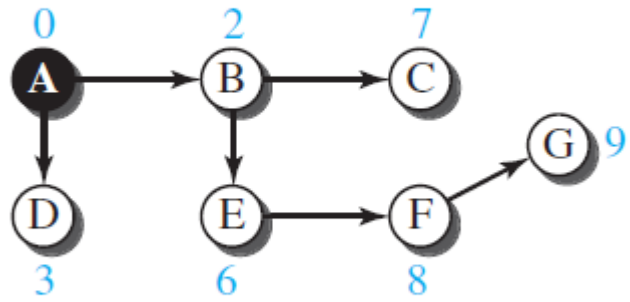
Name of the distance vector defines the root,

Indexes define the destinations,

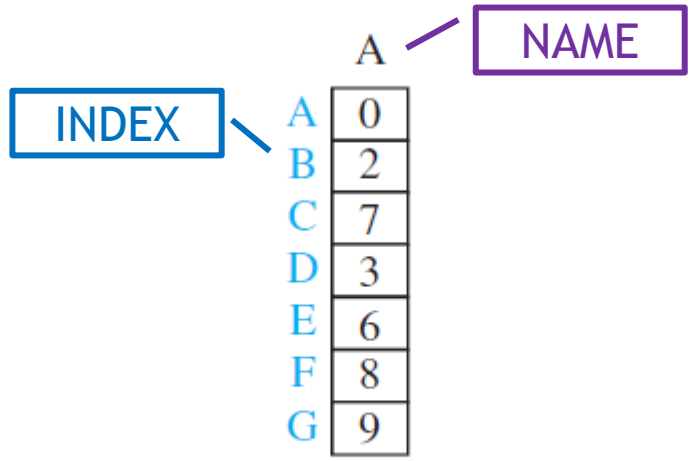
Value of each cell defines the least cost from the root to the destination

To form a tree:

- The node sends some **greeting messages** out of its interfaces
- **Discovers the identity** of the immediate neighbors and the distance **between itself and each neighbor**.
- It then makes a simple distance vector by inserting the discovered distances in the corresponding cells and leaves the value of other cells as infinity.



a. Tree for node A



b. Distance vector for node A

Distance Vector Routing

b. Distance Vectors

Rudimentary vectors cannot help the internet to effectively forward a packet.

Example: Node A thinks that it is not connected to node G; As the corresponding cell shows the least cost of infinity.

To Improve these Vectors:

- The nodes in the internet need to help each other by exchanging information.
- After each node has created its vector, it sends a copy of the vector to all its immediate neighbors.
- After a node receives a distance vector from a neighbor,
  - it updates its distance vector using the Bellman-Ford equation

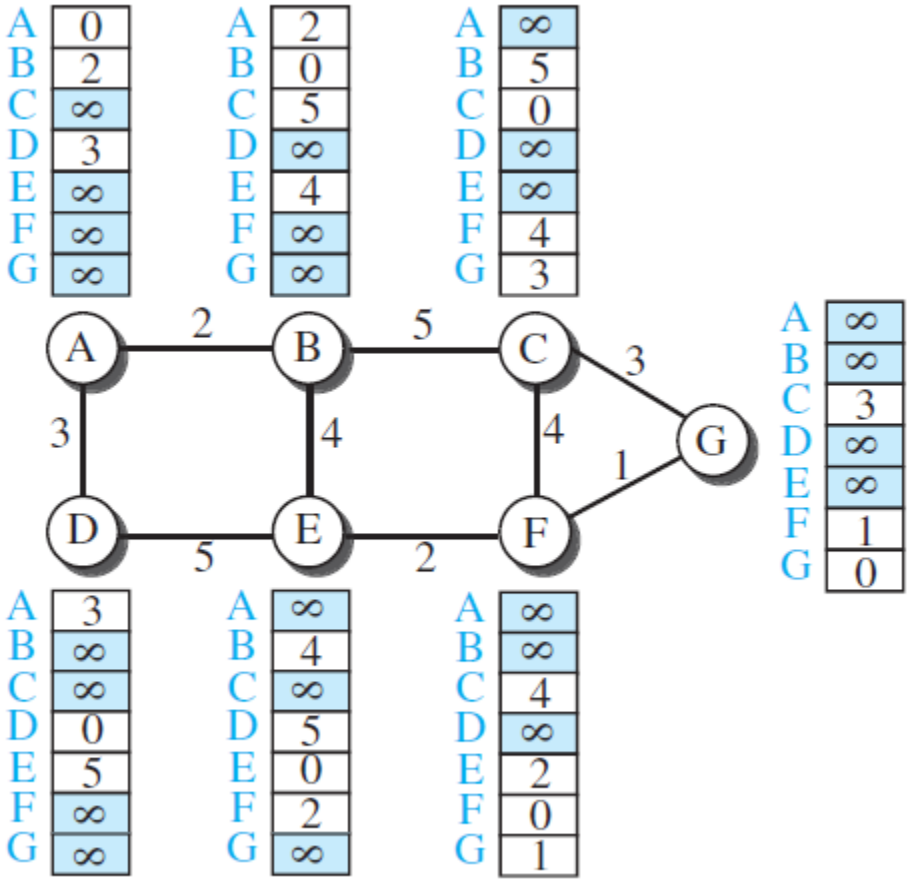
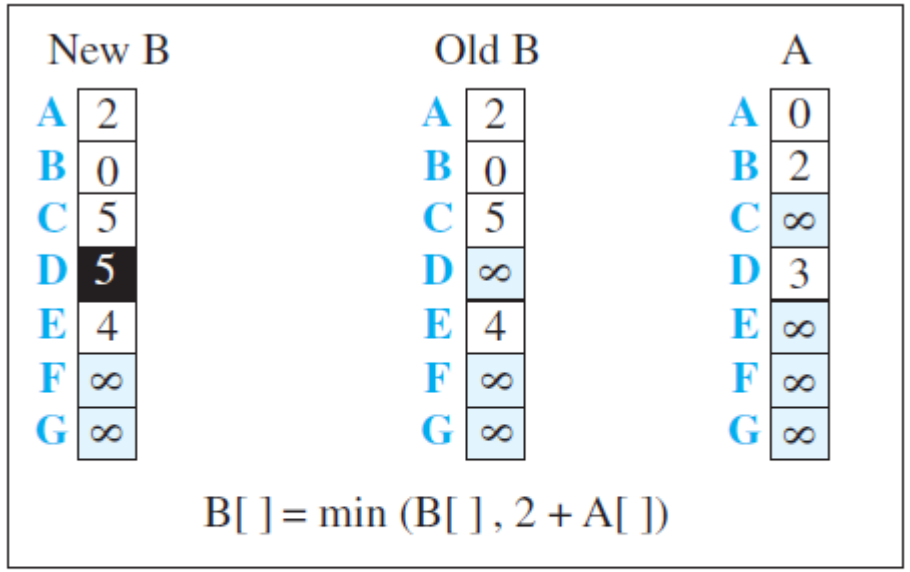


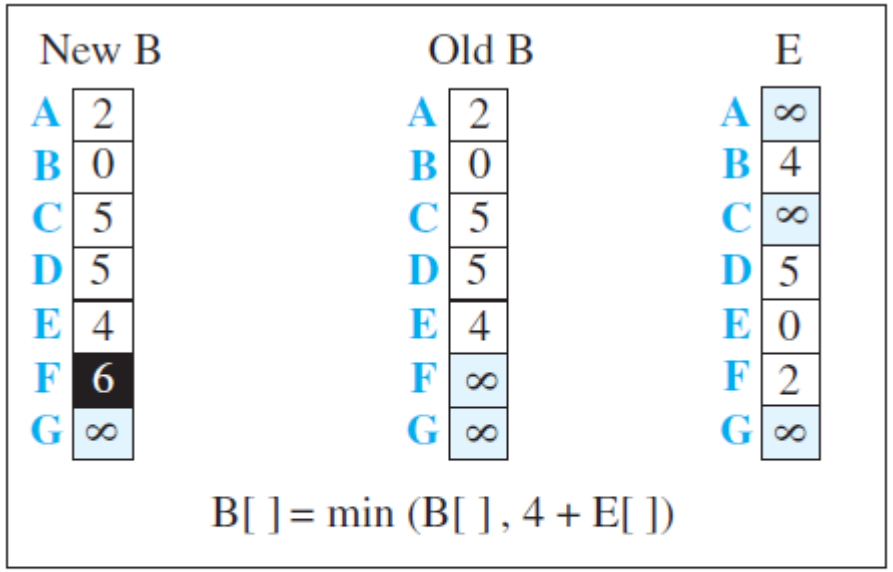
Fig: The first distance vector for an internet

Distance Vector Routing

b. Distance Vectors



a. First event: B receives a copy of A's vector.



b. Second event: B receives a copy of E's vector.

Note:
X[ ]: the whole vector

Fig: *Updating distance vectors*

Link State Routing

- In Link State Routing:
  - **Term *link-state* to define the characteristic of a link that represents a network in the internet.**
  - **In this algorithm, the cost associated with an edge defines the state of the link.**
- **Links with lower costs are preferred**
- **If the cost of a link is infinity:**
  - It means that the link does not exist or has been broken.

Link State Routing

Link-State Database (LSDB)

To create a least-cost tree,

- **Each node needs to have a complete *map* of the network.**
- ***Link-state database (LSDB):*** The collection of states for all links.

There is **only one LSDB for the whole internet**.

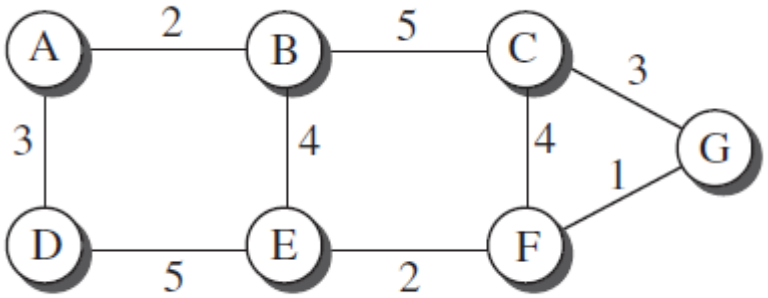
- Each node needs to have a duplicate of it to be able to create the least-cost tree.

LSDB is represented as **a two-dimensional array(matrix)**

- Where value of each cell defines the cost of the corresponding link.

How each node can create this LSDB that contains information about the whole internet?

- This can be done by a process called **flooding**.



a. The weighted graph

	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

b. Link state database

Link State Routing

Link-State Database (LSDB)

Each node can send some greeting messages to all its immediate neighbors to collect two pieces of information for each neighboring node:

- **The identity of the node and**
- **The cost of the link.**

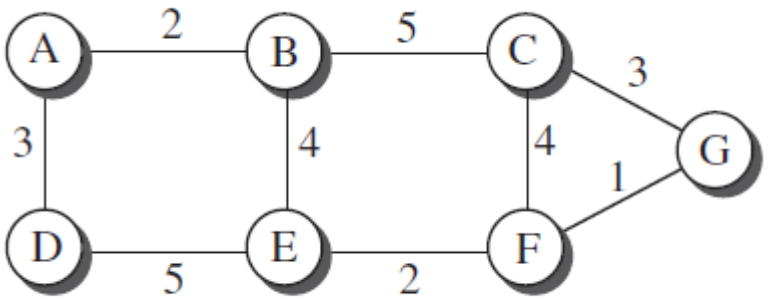
Combination of these two pieces of information is called the *LS packet* (LSP);

- The LSP is sent out of each interface.
- When a node receives an LSP from one of its interfaces:
  - It compares the LSP with the copy it may already have.
- If the newly arrived LSP is older than the one it has it discards the LSP.
- If it is newer or the first one received,
  - The node discards the old LSP (if there is one) and keeps the received one.
  - It then sends a copy of it out of each interface except the one from which the packet arrived.
- This guarantees that flooding stops somewhere in the network.

Each node creates the comprehensive LSDB as shown in Figure.

This LSDB is the same for each node and shows the whole map of the internet.

In other words, a node can make the whole map if it needs to, using this LSDB.



a. The weighted graph

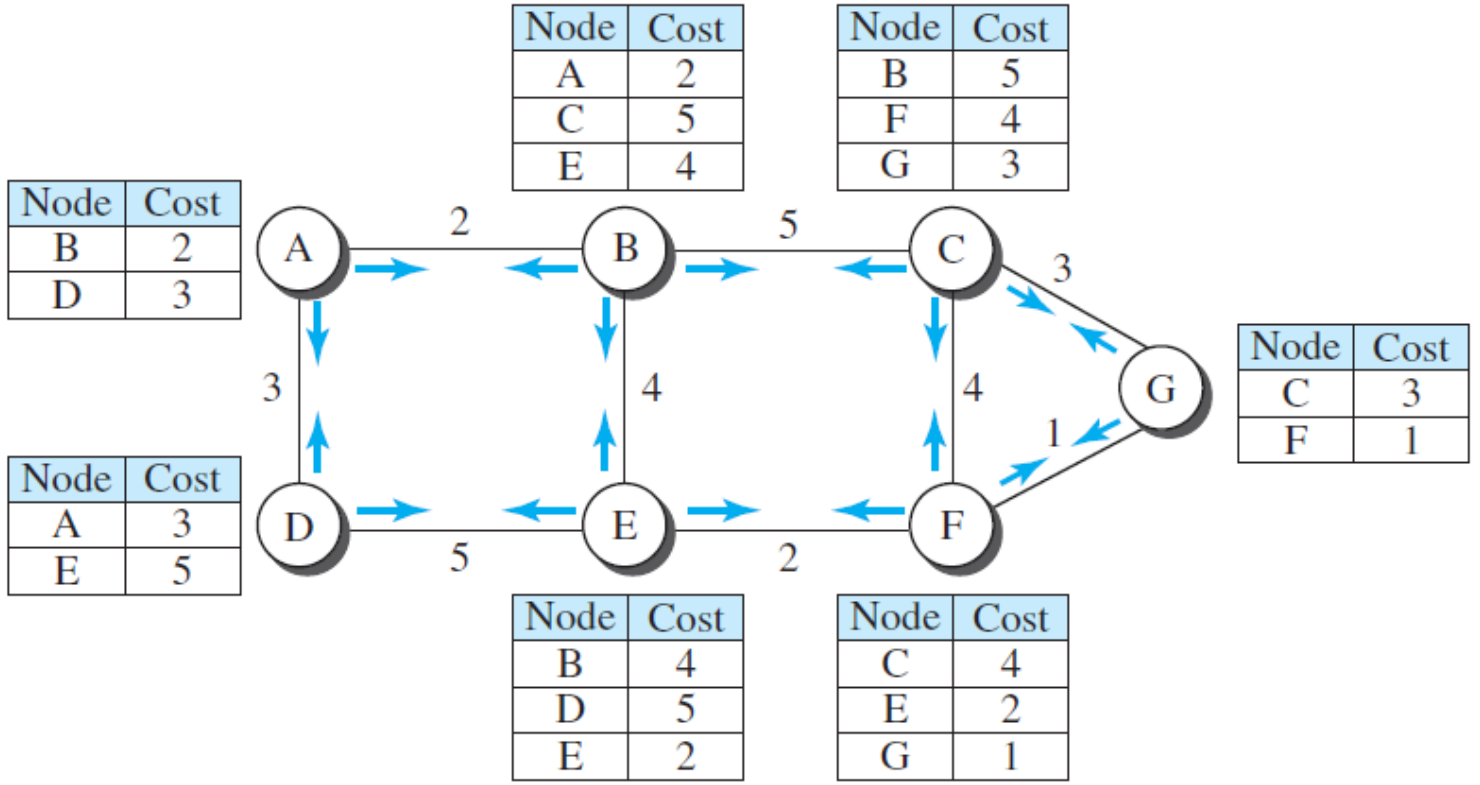
	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

b. Link state database

Link State Routing

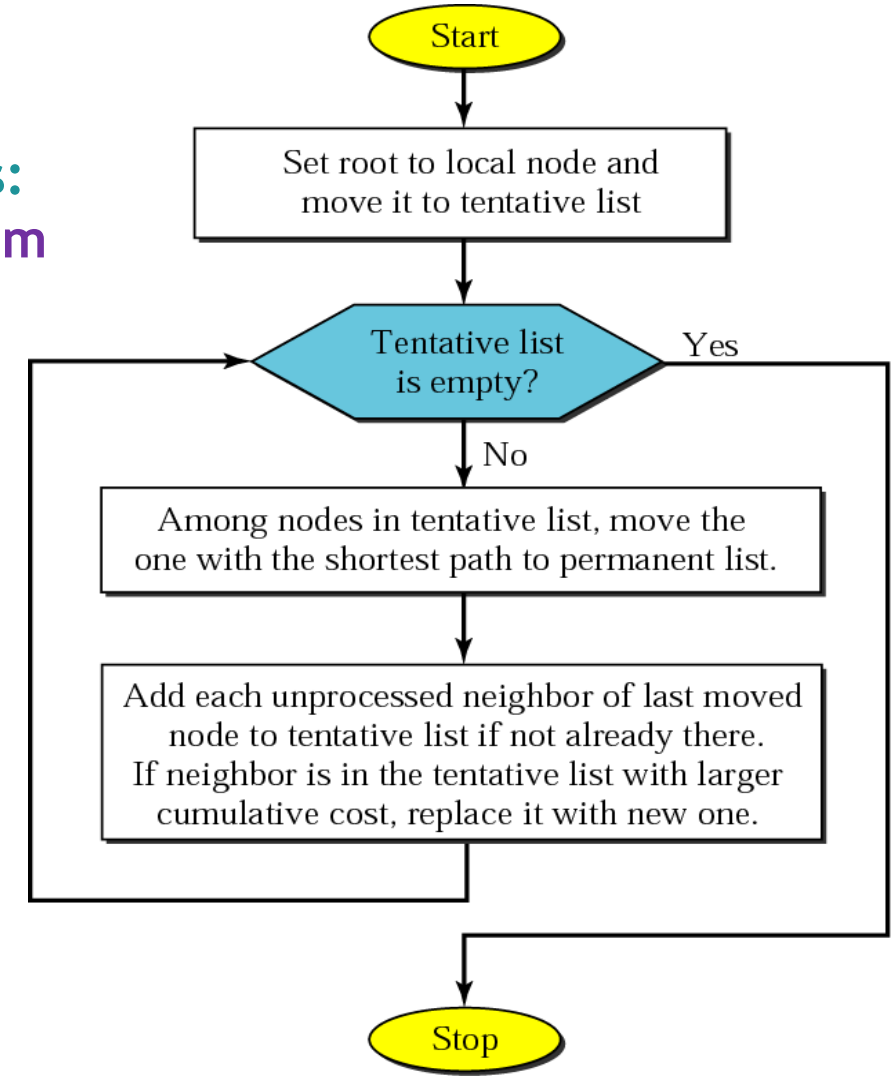
Link-State Database (LSDB)

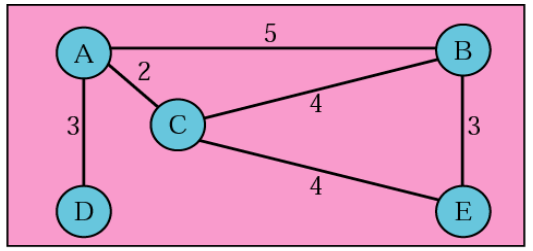
- To create LSP and keep the received one.
  - It then sends a copy of it out of each interface except the one from which the packet arrived.
- **After receiving all new LSPs, each node creates the comprehensive LSDB.**
- This LSDB is the same for each node and shows the whole map of the internet.
- Can compared with the DV routing algo.
  - In the **DV routing algorithm**, each router tells its neighbors what it knows about the whole internet.
  - In the **link-state routing algorithm**, each router tells the whole internet what it knows about its neighbors.



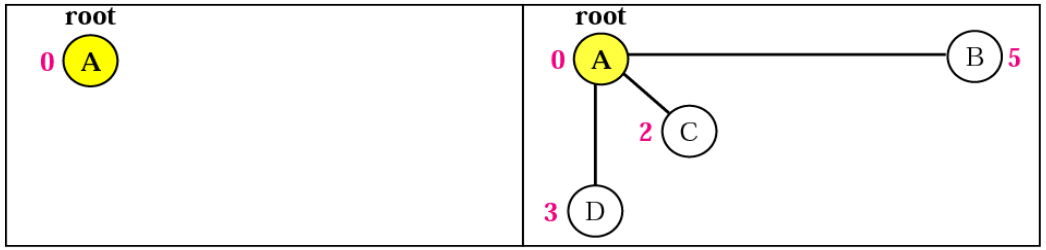
Link State Routing

Formation of Least-Cost Trees: Dijkstra Algorithm



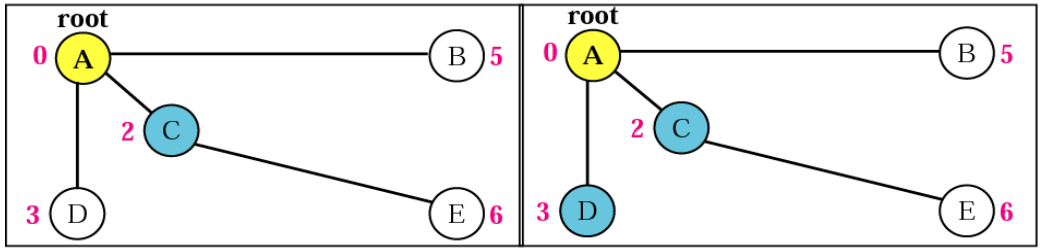


Topology



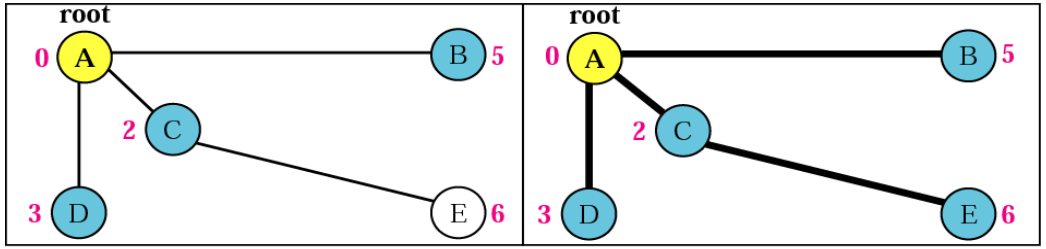
1. Set root to A and move A to tentative list

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



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

4. Move D to permanent list.

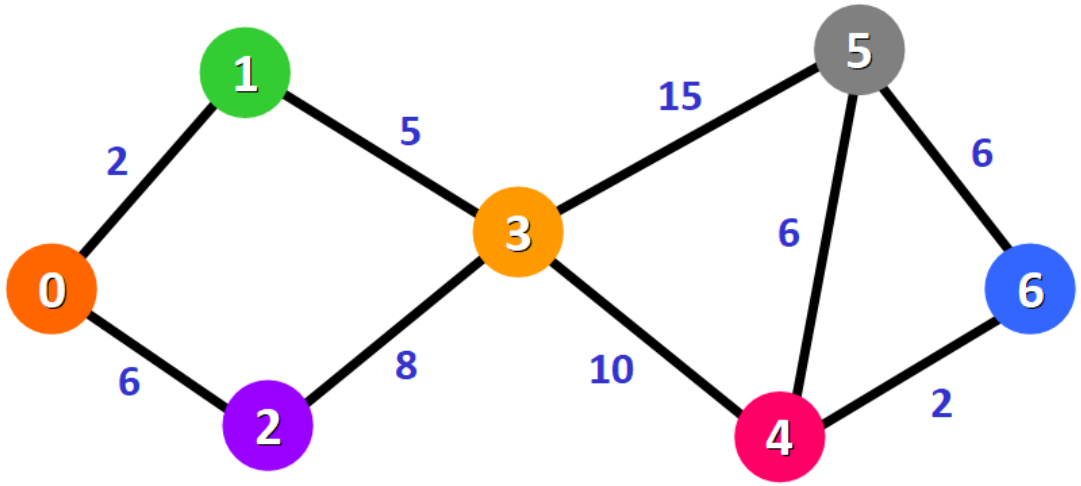


5. Move B to permanent list

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

Problems on Dijkstra Algo

Source/Root Node : 0



The distance from the source node to all other nodes has not been determined yet, so we use the infinity symbol to represent this initially.

Distance:

0: 0
1: ∞
2: ∞
3: ∞
4: ∞
5: ∞
6: ∞

We also have this list to keep track of the nodes that have not been visited yet (nodes that have not been included in the path):

Unvisited Nodes: {0, 1, 2, 3, 4, 5, 6}

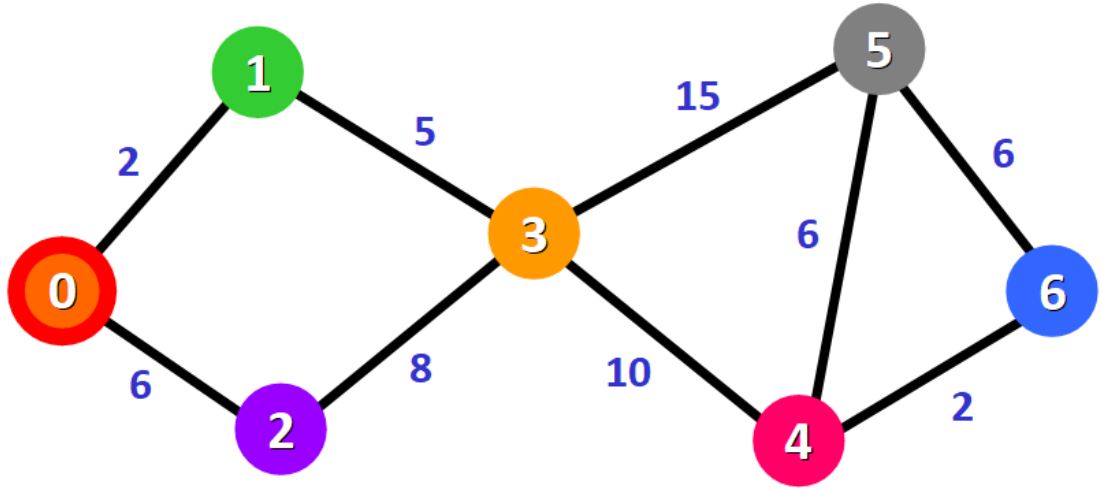
Problems on Dijkstra Algo

Source/Root Node : 0

Since we are **choosing to start at node 0**, we can mark this node as **visited**.

Unvisited Nodes: {~~0~~, 1, 2, 3, 4, 5, 6}

Equivalently, we cross it off from the **list of unvisited nodes** and
Add **a red border to** the corresponding node in diagram:

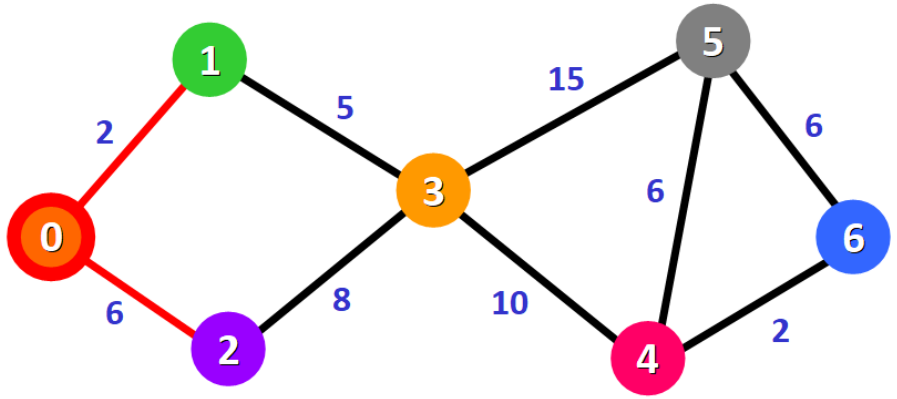


Problems on Dijkstra Algo

Now we need to start checking the **distance from node 0 to its adjacent nodes.**

As you can see, these are nodes 1 and 2 (see the red edges):

Update the distances from node 0 to node 1 and node 2 with the weights of the edges that connect them to node 0 (the source node).
These weights are 2 and 6, respectively:



Distance:

0: 0
1: ~~∞~~ **2**
2: ~~∞~~ **6**
3: ∞
4: ∞
5: ∞
6: ∞

Problems on Dijkstra Algo

Now we need to start checking the **distance from node 0 to its adjacent nodes**.

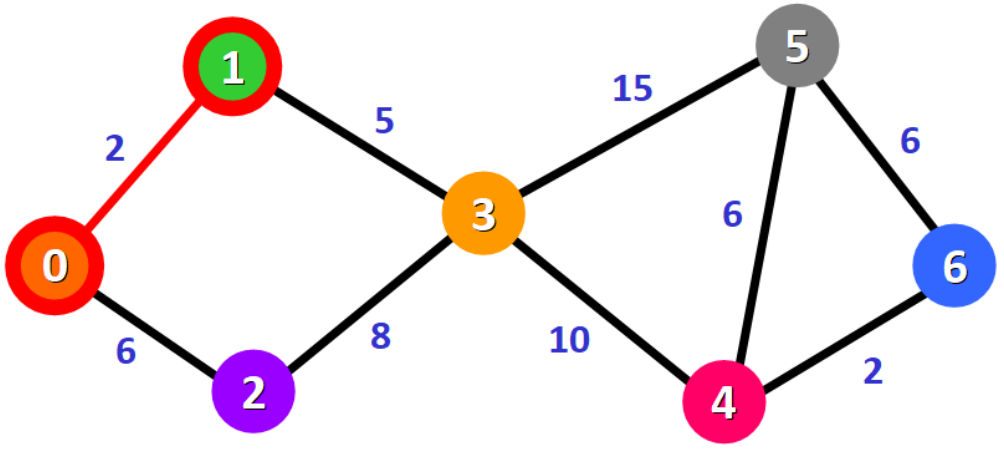
After updating the distances of the adjacent nodes, we need to:

- ***Select the node that is closest to the source node based on the current known distances.***
- ***Mark it as visited.***
- ***Add it to the path.***

If we check the list of distances, ***we can see that node 1 has the shortest distance to the source node (a distance of 2), so we add it to the path***.

In the diagram, we can represent this with a red edge:

We mark it with a red square in the list to represent that it has been "visited" and that we have found the shortest path to this node:



Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ∞
4: ∞
5: ∞
6: ∞

We cross it off from the list of unvisited nodes:

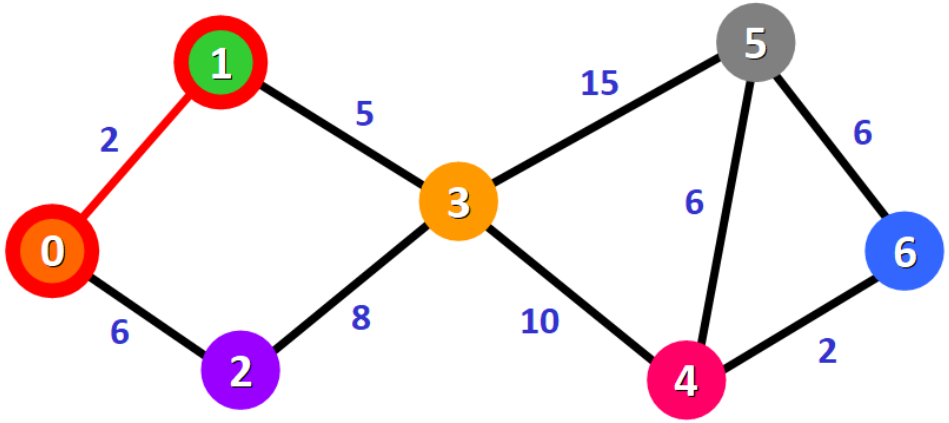
Unvisited Nodes:

{~~0~~, ~~1~~, 2, 3, 4, 5, 6}

Problems on Dijkstra Algo

Node 3 and node 2 are both adjacent to nodes that are already in the path because they are directly connected to node 1 and node 0, respectively.
These are the nodes that we will analyze in the next step.

Since we already have the distance from **the source node to node 2 written down in our list**, we don't need to update the distance this time.
We only need to update the distance from the source node to the new adjacent node (node 3):



Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ∞
5: ∞
6: ∞

This distance is 7.

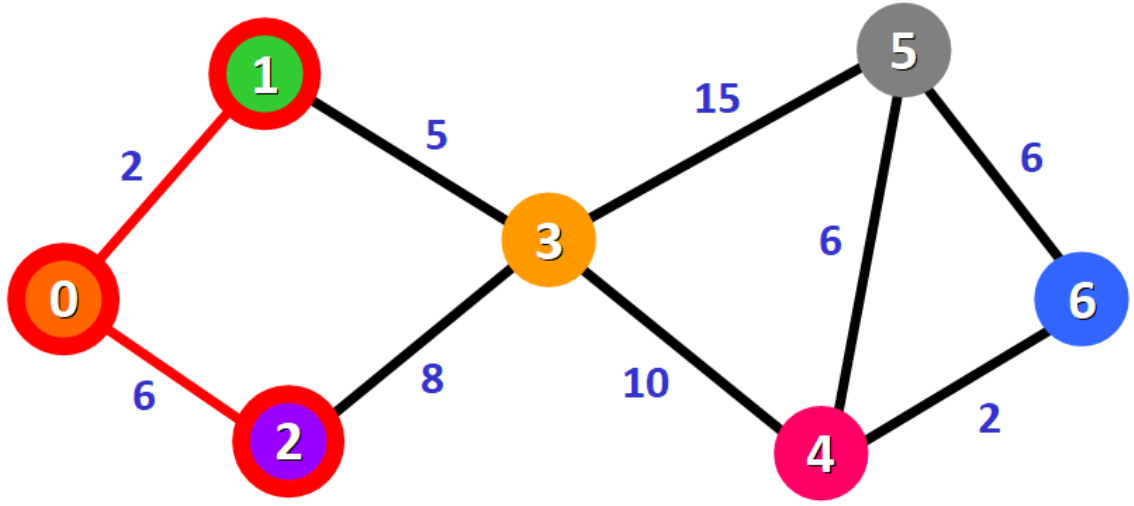
- To find the distance from the source node to another node, we add the weights of all the edges that form the shortest path to reach that node:
- For node 3:
  - The total distance is 7
  - As we add the weights of the edges that form the path 0 -> 1 -> 3
    - (2 for the edge 0 -> 1 and 5 for the edge 1 -> 3).

Problems on Dijkstra Algo

Now that we have the distance to the adjacent nodes, we have to choose which node will be added to the path.
We must select the unvisited node with the shortest (currently known) distance to the source node.
From the list of distances, we can immediately detect that this is node 2 with distance 6:

Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ∞
5: ∞
6: ∞



Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, 3, 4, 5, 6}

Problems on Dijkstra Algo

Now we need to repeat the process to find the **shortest path from the source node to the new adjacent node, which is node 3.**

You can see that we have two possible paths

0 -> 1 -> 3 or

0 -> 2 -> 3.

Let's see how we can decide which one is the shortest path.

Distance:

0: 0

1: ~~∞~~ 2

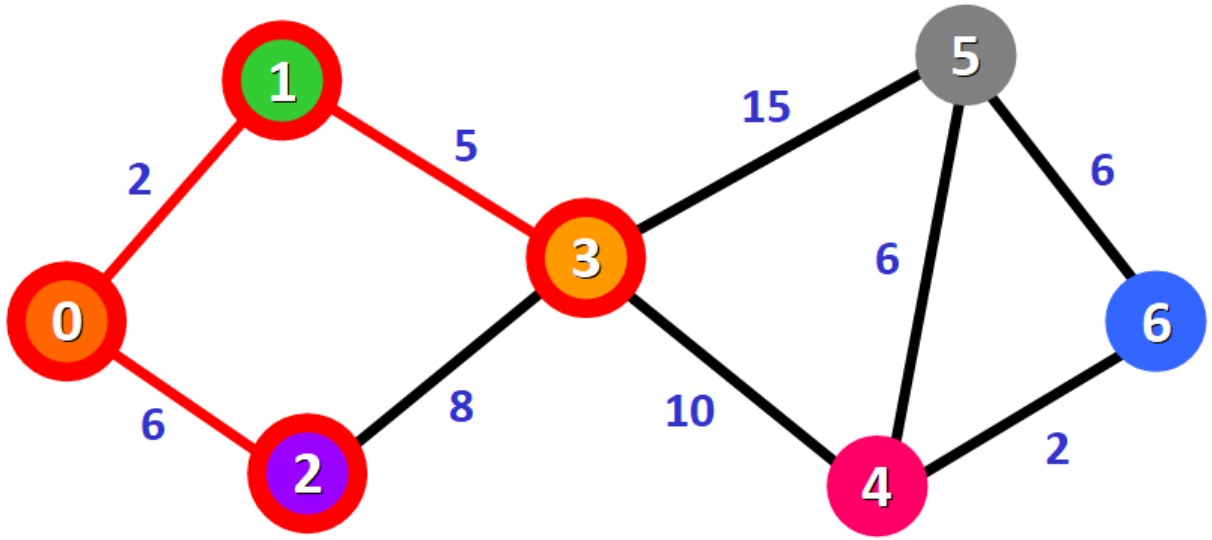
2: ~~∞~~ 6

3: ~~∞~~ 7 from (5 + 2) vs. 14 from (6 + 8)

4: ∞

5: ∞

6: ∞



Problems on Dijkstra Algo

Now we need to repeat the process to find the **shortest path from the source node to the new adjacent node, which is node 3.**

You can see that we have two possible paths

0 -> 1 -> 3 or

0 -> 2 -> 3.

Let's see how we can decide which one is the shortest path.

Distance:

0: 0

1: ~~∞~~ 2

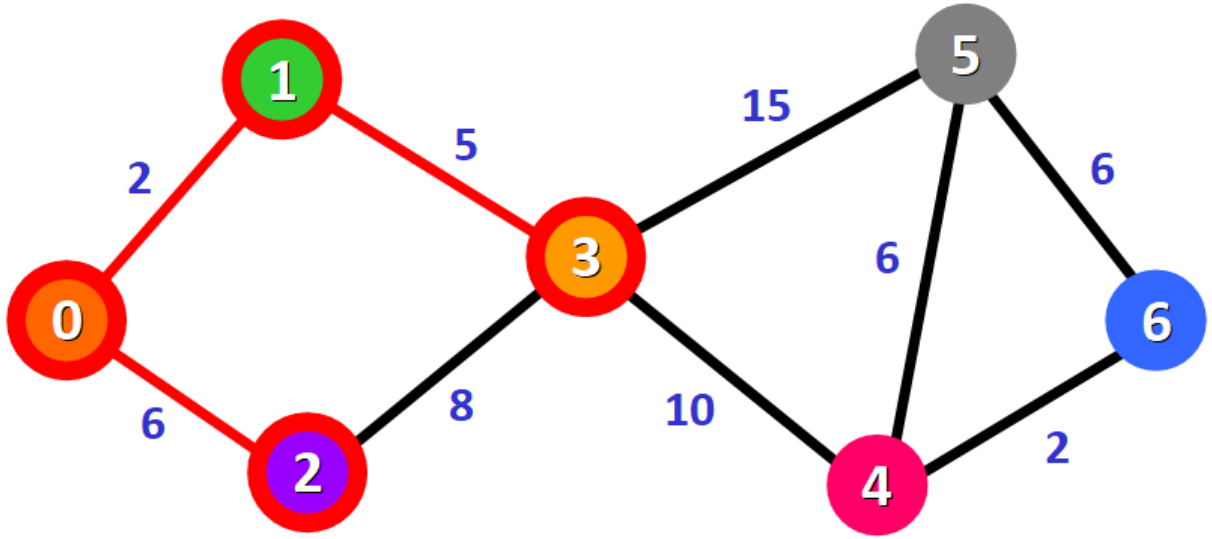
2: ~~∞~~ 6

3: ~~∞~~ 7 from (5 + 2) vs. 14 from (6 + 8)

4: ∞

5: ∞

6: ∞

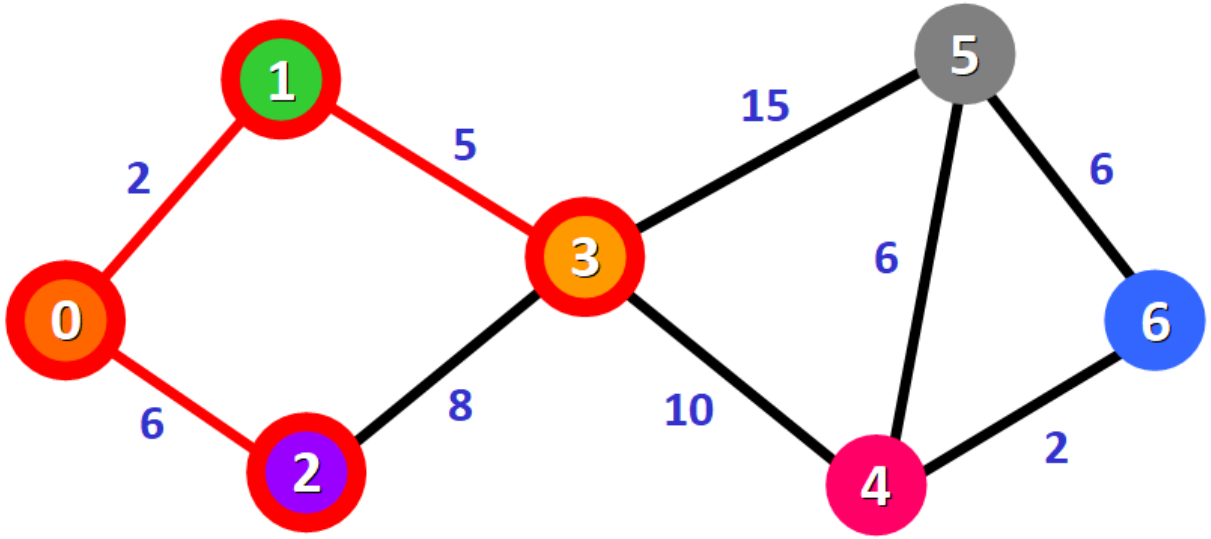


Problems on Dijkstra Algo

We mark this node as visited and cross it off from the list of unvisited nodes:

Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ∞
5: ∞
6: ∞



Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, ~~3~~, 4, 5, 6}

Problems on Dijkstra Algo

We need to check the new adjacent nodes that we have not visited so far.
This time, these nodes are node 4 and node 5 since they are adjacent to node 3.

We update the distances of these nodes to the source node, always trying to find a shorter path, if possible:

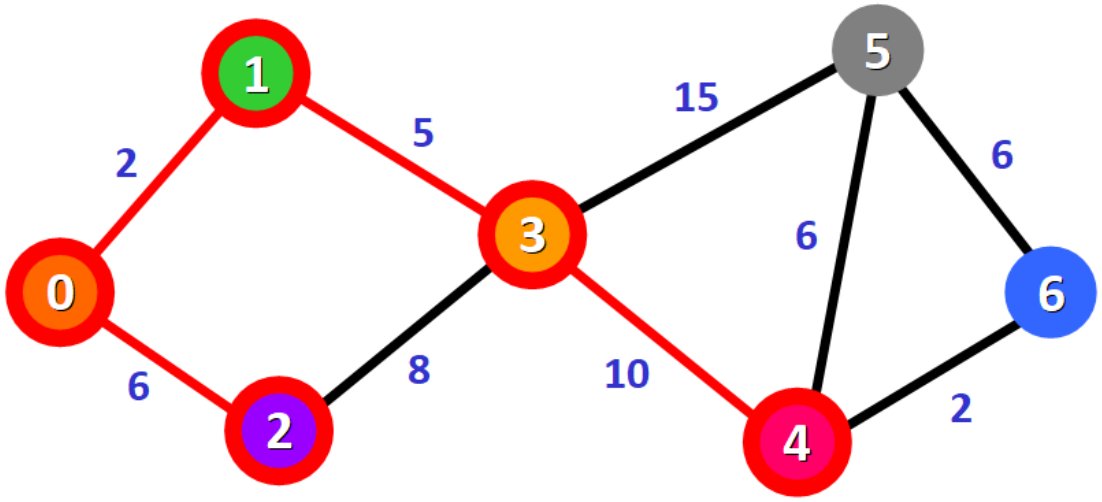
For node 4: the distance is 17 from the path 0 -> 1 -> 3 -> 4.
For node 5: the distance is 22 from the path 0 -> 1 -> 3 -> 5.

Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ~~∞~~ 17 from (2 + 5 + 10)
5: ~~∞~~ 22 from (2 + 5 + 15)
6: ∞

Problems on Dijkstra Algo

We need to choose which unvisited node will be marked as visited now.
In this case, it's node 4 because it has the shortest distance in the list of distances. We add it graphically in the diagram:



We also mark it as "visited" by adding a small red square in the list:

Distance:

0: 0
1: ~~∞~~ 2 ■
2: ~~∞~~ 6 ■
3: ~~∞~~ 7 ■
4: ~~∞~~ 17 ■
5: ~~∞~~ 22
6: ∞

Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, 5, 6}

Problems on Dijkstra Algo

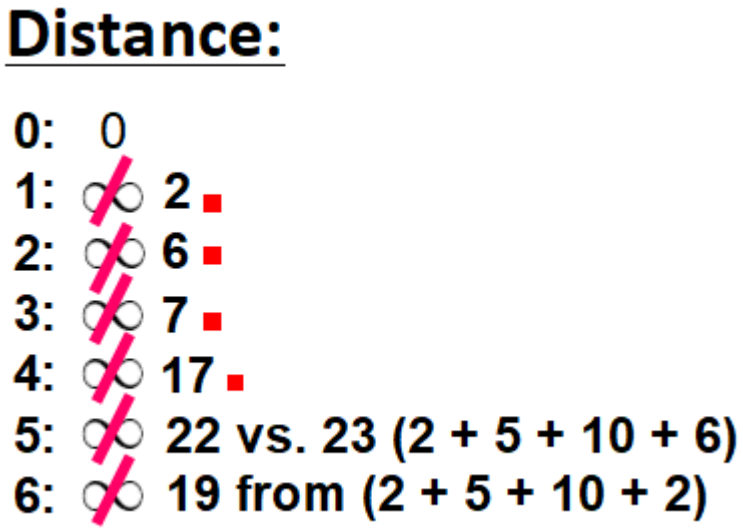
And we repeat the process again. We check the adjacent nodes: node 5 and node 6.
We need to analyze each possible path that we can follow to reach them from nodes that have already been marked as visited and added to the path.

For node 5:

- The first option is to follow the path 0 -> 1 -> 3 -> 5, which has a distance of 22 from the source node (2 + 5 + 15).
- This distance was already recorded in the list of distances in a previous step.
- The second option would be to follow the path 0 -> 1 -> 3 -> 4 -> 5, which has a distance of 23 from the source node (2 + 5 + 10 + 6).
- Clearly, the first path is shorter, so we choose it for node 5.

For node 6:

- The path available is 0 -> 1 -> 3 -> 4 -> 6, which has a distance of 19 from the source node (2 + 5 + 10 + 2).

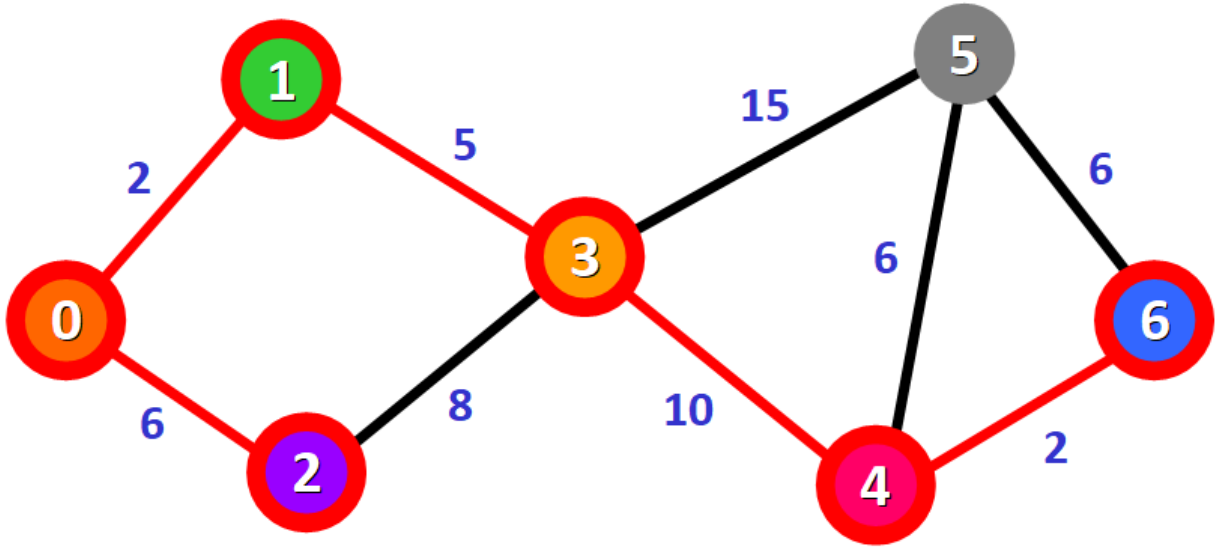


Problems on Dijkstra Algo

We mark the node with the shortest (currently known) distance as visited.
In this case, node 6.

Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ~~∞~~ 17
5: ~~∞~~ 22
6: ~~∞~~ 19



Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, 5, ~~6~~}

Problems on Dijkstra Algo

Only one node has not been visited yet, node 5.
Let's see how we can include it in the path.
There are three different paths that we can take to reach **node 5** from the nodes that have been added to the path:

- **Option 1:** 0 -> 1 -> 3 -> 5 with a distance of 22 (2 + 5 + 15).
- **Option 2:** 0 -> 1 -> 3 -> 4 -> 5 with a distance of 23 (2 + 5 + 10 + 6).
- **Option 3:** 0 -> 1 -> 3 -> 4 -> 6 -> 5 with a distance of 25 (2 + 5 + 10 + 2 + 6).

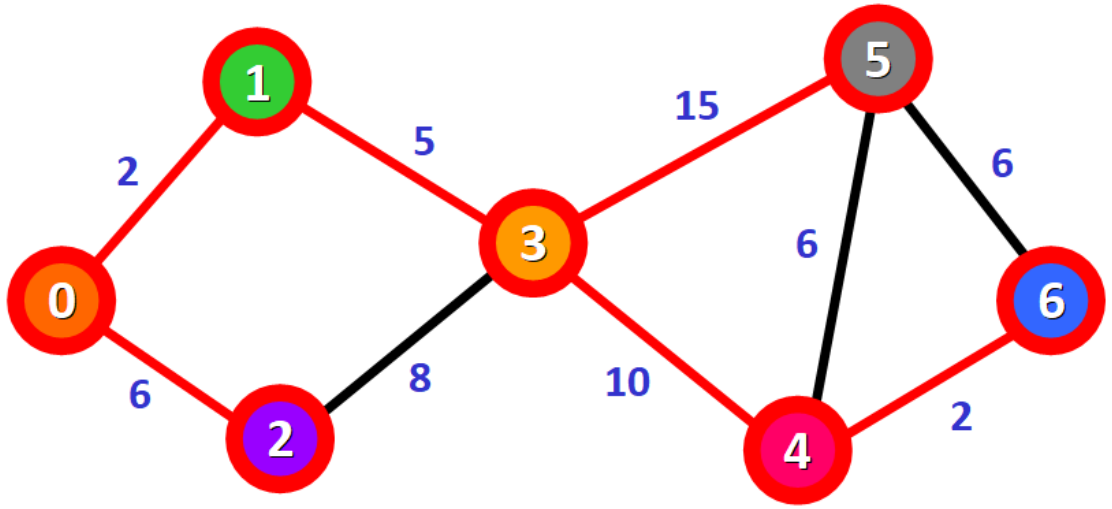
Distance:

0: 0
1: ~~∞~~ **2**
2: ~~∞~~ **6**
3: ~~∞~~ **7**
4: ~~∞~~ **17**
5: ~~∞~~ **22 from (2 + 5 + 15) vs. 23 from (2 + 5 + 10 + 6) vs. 25 from (2 + 5 + 10 + 2 + 6)**
6: ~~∞~~ **19**

We select the shortest path: 0 -> 1 -> 3 -> 5 with a distance of 22.

Problems on Dijkstra Algo

Only one node has not been visited yet, node 5.



Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ~~∞~~ 17
5: ~~∞~~ 22
6: ~~∞~~ 19

Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~}

Internet Structure

Internet has changed from a ***tree-like structure,***

- With a single backbone, to a multi-backbone structure.

First, *backbones* that provide global connectivity.

- Backbones are connected by some *peering points*
- That allow connectivity between backbones.

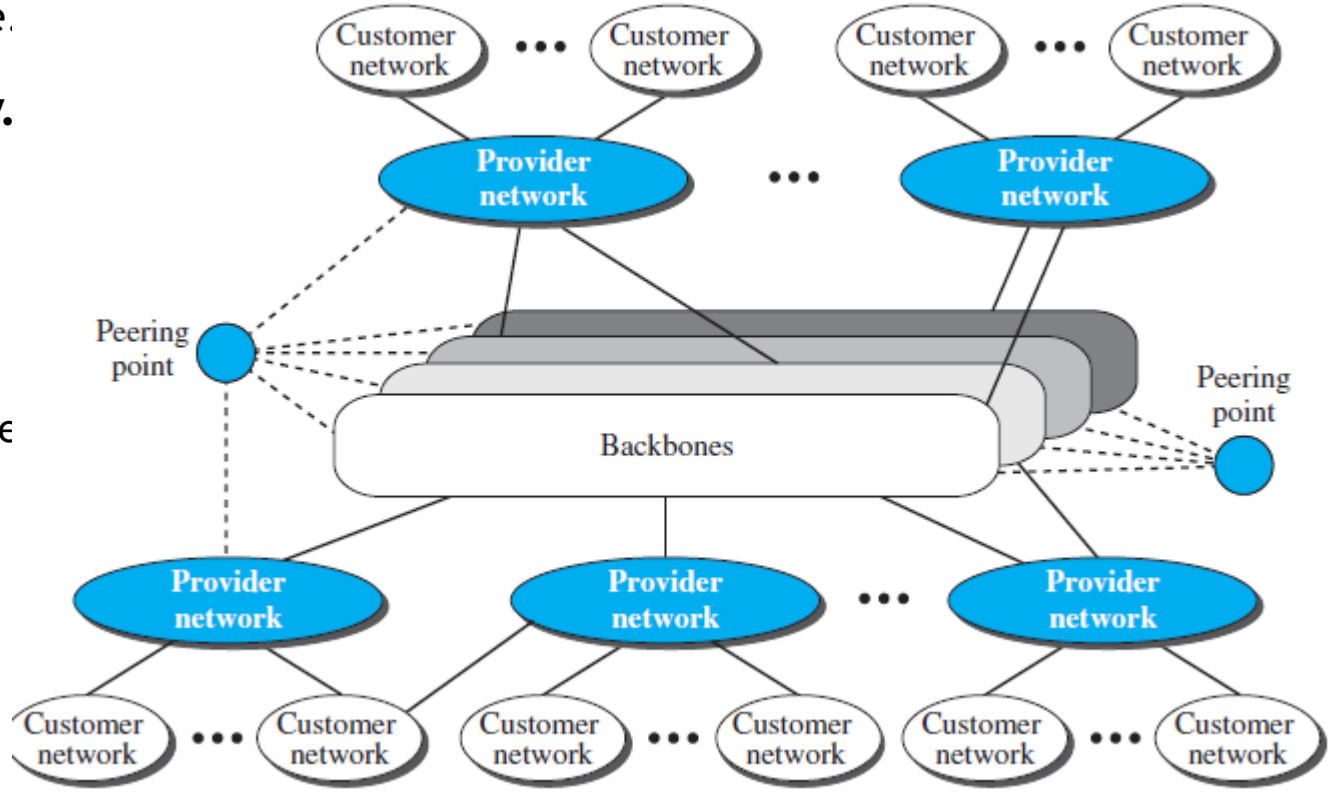
At a lower level, there are some *provider networks*

- That use the backbones for global connectivity but provide services to Internet customers.

Finally, there are some ***customer networks***

- That use the services provided by the provider networks.

Any of these three entities can be called an **Internet Service Provider or ISP**.



Internet Structure

Hierarchical Routing

- Internet cannot be done using a single protocol for two reasons:
- **A Scalability Problem**
  - Means that the size of the **forwarding tables becomes huge**
  - Searching for a destination in a forwarding table becomes time-consuming, and
  - Updating creates a huge amount of traffic.
- **An administrative issue**
  - The administrator needs to have control in its system.

Hierarchical routing means considering each ISP as an **Autonomous System (AS).**

- Each AS can run a routing protocol
  - That meets its needs, but the global Internet runs a global protocol to glue all ASs together.
- The **routing protocol run in each AS** is referred to as ***intra-AS routing protocol***, ***intradomain routing protocol***, or ***interior gateway protocol*** *(IGP)*;
- The **global routing protocol** is referred to as *inter-AS routing protocol, interdomain routing protocol*, or ***exterior gateway protocol (EGP).***
- Presently, the two common intradomain routing protocols are RIP and OSPF.

Internet Structure

Autonomous Systems

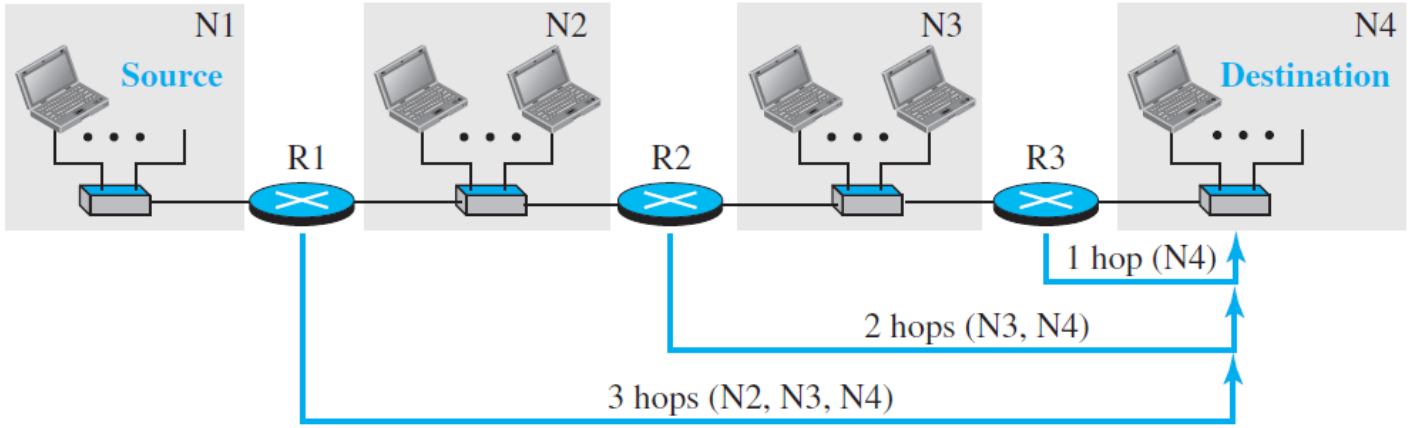
- Each **ISP is an autonomous system** when it comes to managing networks and routers under its control
- Each AS is given **an autonomous number (ASN)** by the ICANN
- **16-bit unsigned integer** that uniquely defines an AS

Categorization:

- According to the way they are connected to other AS.
- ***Stub AS.***
  - A stub AS has **only one connection to another AS.**
  - **Data traffic can be either initiated or terminated in a stub AS; the data cannot pass through it.**
  - A good example of a stub AS is the customer network, which is either the source or the sink of data.
- ***Multihomed AS.***
  - A multihomed AS can have **more than one connection to other ASs**, but it does not allow data traffic to pass through it.

Routing Information Protocol (RIP)

- One of the most widely used intradomain routing protocols
- Based on the distance-vector routing algorithm.
- Started as part of the **Xerox Network System (XNS),** but it was the Berkeley Software Distribution (BSD) version of UNIX.



Hop Count

- **First,** since a router in an **AS needs to know how to forward a packet to different networks (subnets) in an AS**, RIP routers advertise the cost of reaching different networks instead of reaching other nodes in a theoretical graph.
- In other words, the **cost is defined between a router and the network in which the destination host is located.**
- **Second,** to make the implementation of the cost simpler, the cost is defined as the number of hops,
  - Which means the number of networks (subnets) a packet needs to travel through from the source router to the final destination host.
- **In RIP,** the **maximum cost of a path can be 15**, which means 16 is considered as infinity (no connection).

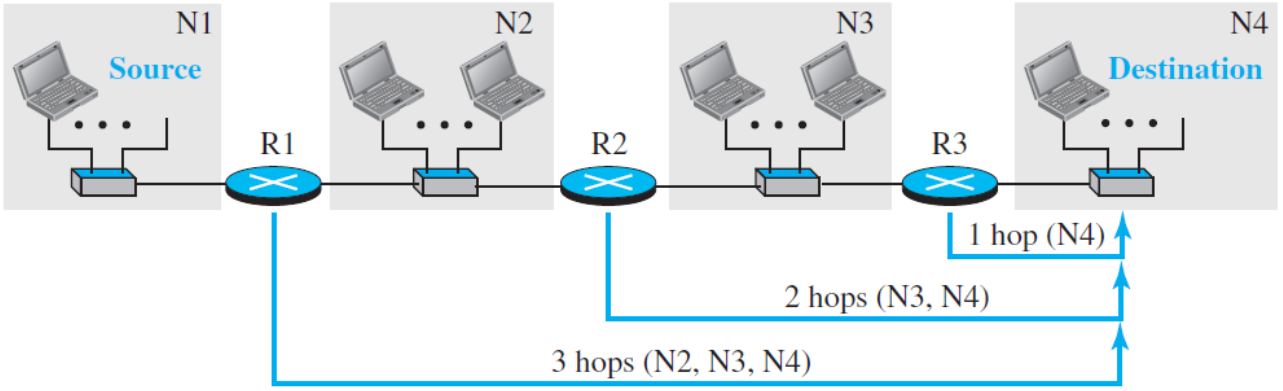
Routing Information Protocol (RIP)

A forwarding table in RIP is a three-column table in which:

- **First column is** the address of the destination network,
- **Second column is** the address of the next router to which the packet should be forwarded, and
- **Third column is** the cost (the number of hops) to reach the destination network.

For example,

- R1 defines that the next router for the path to N4 is R2;
- R2 defines that the next router to N4 is R3;
- R3 defines that there is no next router for this path.
- The tree is then R1 → R2 → R3 → N4.



Forwarding table for R1

Destination network	Next router	Cost in hops
N1	——	1
N2	——	1
N3	R2	2
N4	R2	3

Forwarding table for R2

Destination network	Next router	Cost in hops
N1	R1	2
N2	——	1
N3	——	1
N4	R3	2

Forwarding table for R3

Destination network	Next router	Cost in hops
N1	R2	3
N2	R2	2
N3	——	1
N4	——	1

RIP Timers

RIP uses three timers to support its operation.

The ***periodic timer*** controls the **advertising of regular update messages**.

- Each router has one periodic timer that is randomly set to a number between 25 and 35 seconds.
- The timer counts down; when zero is reached, the update message is sent, and the timer is randomly set once again.

The ***expiration timer*** governs the **validity of a route**.

- When a router receives update information for a route, the **expiration timer is set to 180 seconds** for that particular route.
- Every time a new update for the route is received, the timer is reset.
- If there is a problem on an internet and no update is received within the allotted 180 seconds
  - The route is considered expired and the hop count of the route is set to 16, which means the destination is unreachable.
- Every route has its own expiration timer.

The ***garbage collection timer*** is used to purge a route from the forwarding table.

- When the information about a route becomes invalid
  - The router does not immediately purge that route from its table.
- Instead, it continues to advertise the route with a metric value of 16.
- At the same time, **a garbage collection timer is set to 120 seconds for that route**.
- When the count reaches zero, the route is purged from the table.
- This timer allows neighbors to become aware of the invalidity of a route prior to purging.

RIP Performance

Update Messages

- Have a very simple format and are sent only to neighbors.
- Do not normally create traffic because the routers try to avoid sending them at the same time.

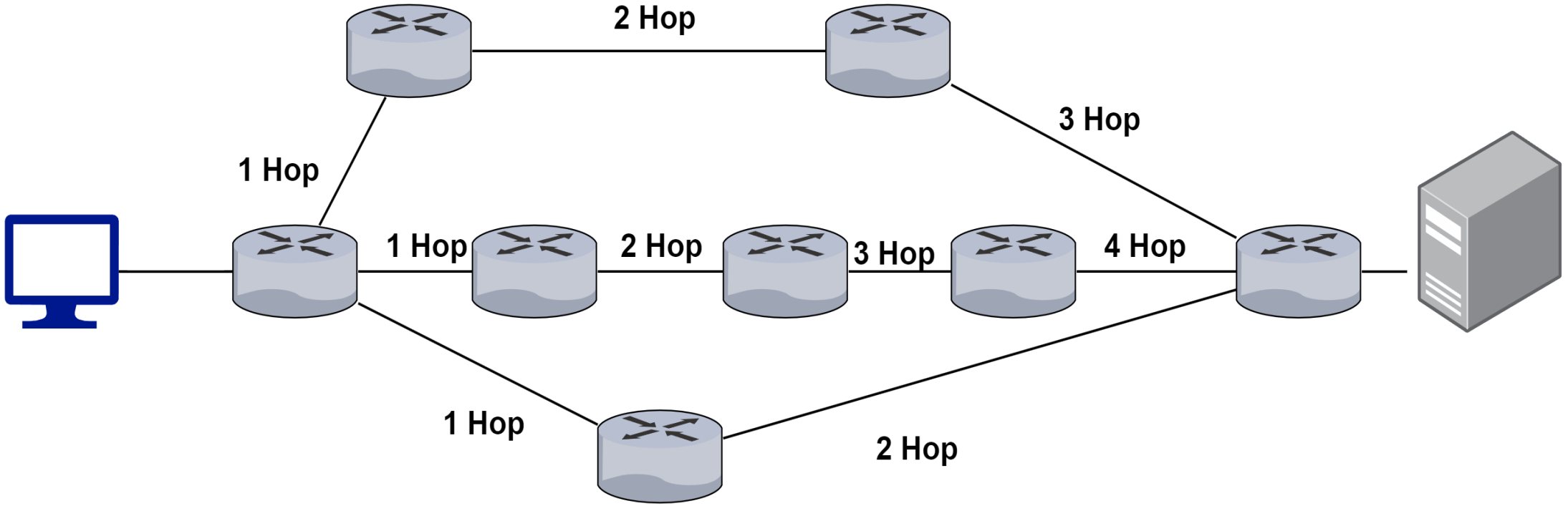
Convergence of Forwarding Tables

- Uses the distance-vector algorithm
  - Which can converge slowly if the domain is large,
  - But, since RIP allows only 15 hops in a domain (16 is considered as infinity), there is normally no problem in convergence.

Robustness

- DV routing is based on the concept that **each router sends what it knows about the whole domain to its neighbors**.
- Means that the calculation of the forwarding table depends on information received from immediate neighbors, which in turn receive their information from their own neighbors.
- If there is a failure or corruption in one router, the problem will be propagated to all routers and the forwarding in each router will be affected.

RIP Example



Reference

Forouzan, A. Behrouz. ***Data Communications & Networking***. 5th Edition. Tata McGraw-Hill Education.

Chapter 20 Unicast Routing

Topic: 20.1 to 20.4