Module – 3

The Network layer: What's Inside a Router?: Input Processing, Switching, Output Processing, Where Does Queuing Occur? Routing control plane, IPv6,A Brief foray into IP Security, Routing Algorithms: The Link-State (LS) Routing Algorithm, The Distance-Vector (DV) Routing Algorithm, Hierarchical Routing, Routing in the Internet, Intra-AS Routing in the Internet: RIP, Intra-AS Routing in the Internet: OSPF, Inter/AS Routing: BGP, Broadcast and Multicast Routing: Broadcast Routing Algorithms and Multicast.

3.1 What's inside a router ?

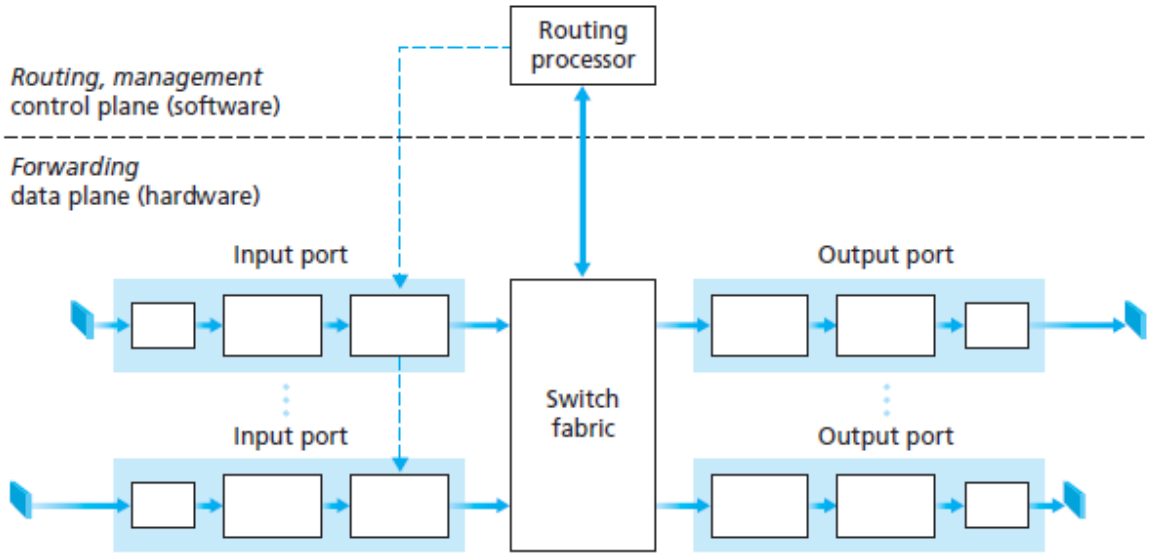


Figure 3.1 Router architecture

A high-level view of a generic router architecture is shown in Figure above. Four router components can be identified:

• ***Input ports.*** An input port performs several key functions. It performs the physical layer function of terminating an incoming physical link at a router; this is shown in the leftmost box of the input port and the rightmost box of the output port in Figure above.

An input port also performs link-layer functions needed to interoperate with the link layer at the other side of the incoming link; this is represented by the middle boxes in the input and output ports.

The lookup function is also performed at the input port; this will occur in the rightmost box of the input port. Here the forwarding table is consulted to determine the router output port to which an arriving packet will be forwarded via the switching fabric.

• *Switching fabric.* The switching fabric connects the router's input ports to its output ports. This switching fabric is completely contained within the router

• *Output ports.* An output port stores packets received from the switching fabric and transmits these packets on the outgoing link by performing the necessary link-layer and physical-layer functions.

• *Routing processor.* The routing processor executes the routing protocols, maintains routing tables and attached link state information and computes the forwarding table for the router.

It also performs the network management functions we distinguished between a router's forwarding and routing functions. A router's input ports, output ports and switching fabric together implement the forwarding function, as shown in Figure.

These forwarding functions are collectively referred to as the **router forwarding plane**.

3.1.1 Input Processing :

The lookup performed in the input port is central to the router's operation—it is here that the router uses the forwarding table to look up the output port to which an arriving packet will be forwarded via the switching fabric.

The forwarding table is computed and updated by the routing processor, with a shadow copy typically stored at each input port.

The forwarding table is copied from the routing processor to the line cards over a separate bus (e.g., a PCI bus) indicated by the dashed line from the routing processor to the input line cards in Figure below.

With a shadow copy, forwarding decisions can be made locally, at each input port, without invoking the centralized routing processor on a per-packet basis and thus avoiding a centralized processing bottleneck.

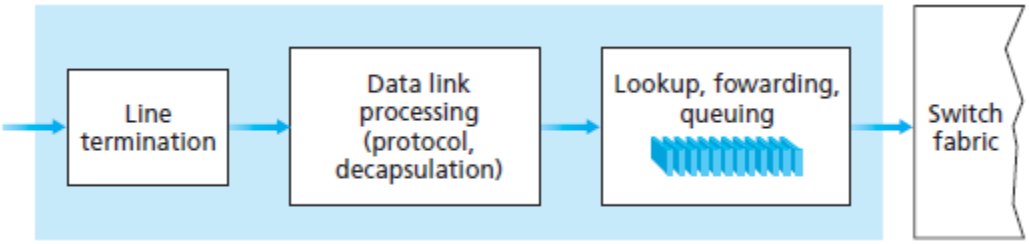


Figure 3.2: Input port processing

Once a packet's output port has been determined via the lookup, the packet can be sent into the switching fabric.

In some designs, a packet may be temporarily blocked from entering the switching fabric if packets from other input ports are currently using the fabric.

A blocked packet will be queued at the input port and then scheduled to cross the fabric at a later point in time.

Important action in input port processing are :

(1) physical- and link-layer processing must occur ;

(2) the packet's version number, checksum and time-to-live field

(3) counters used for network management must be updated.

3.1.2 Switching

The switching fabric is at the very heart of a router, as it is through fabric that the packets are actually switched (that is, forwarded) from an input port to an output port.

Switching can be accomplished in following ways, as shown in Figure below:

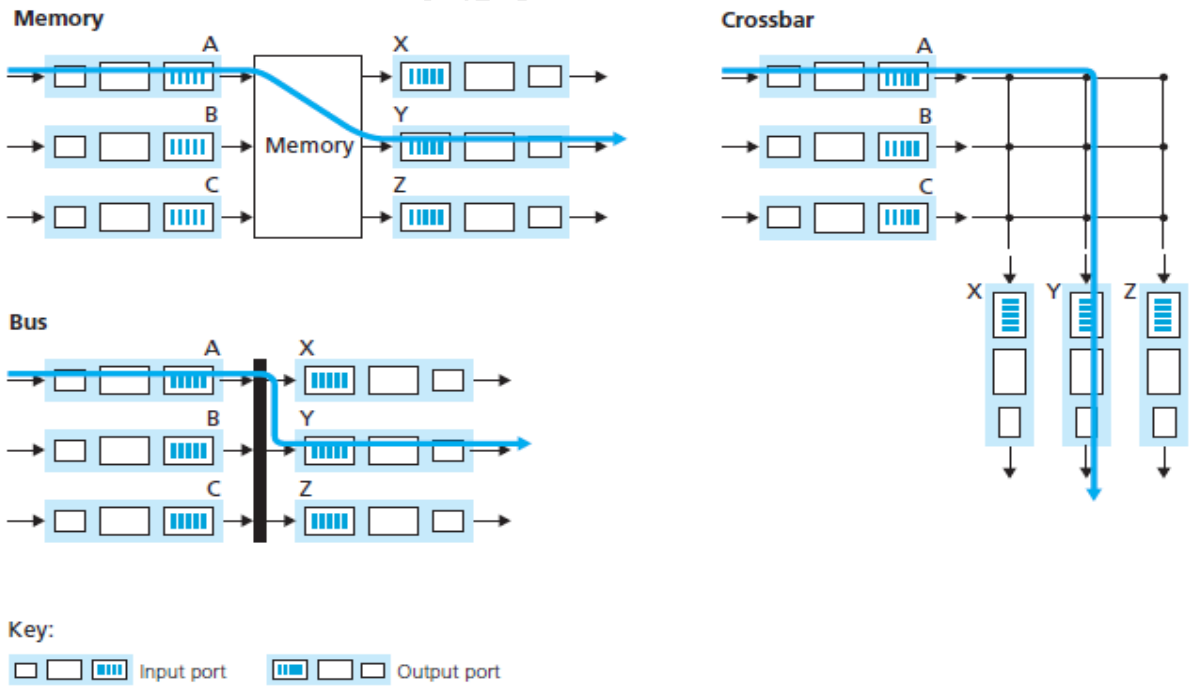


Figure 3.3: Three switching techniques

- ***Switching via memory***. Switching between input and output ports is done under direct control of the CPU (routing processor). An input port with an arriving packet first signals the routing processor via an interrupt.

  The packet is then copied from the input port into processor memory. The routing processor then extracts the destination address from the header, looking up the appropriate output port in the forwarding table and copies the packet to the output port's buffers.

  If the memory bandwidth is such that B packets per second can be written into, or read from, memory, then the overall forwarding throughput (the total rate at which packets are transferred from input ports to output ports) must be less than $B/2$.

- ***Switching via a bus.*** In this approach, an input port transfers a packet directly to the output port over a shared bus, without intervention by the routing processor.

  This is done by having the input port pre-pend a switch-internal label (header) to the packet indicating the local output port to which this packet is being transferred and transmitting the packet onto the bus.

  The packet is received by all output ports, but only the port that matches the label will keep the packet. The label is then removed at the output port, as this label is only used within the switch to cross the bus.

  If multiple packets arrive to the router at the same time, each at a different input port, all but one must wait since only one packet can cross the bus at a time.

  Since every packet must cross the single bus, the switching speed of the router is limited to the bus speed;

- ***Switching via an interconnection network***. One way to overcome the bandwidth limitation of a single, shared bus is to use a more sophisticated interconnection network, such as those that have been used in the past to interconnect processors in a multiprocessor computer architecture.

  A crossbar switch is an interconnection network consisting of $2N$ buses that connect N input ports to N output ports, as shown in Figure above.

  Each vertical bus intersects each horizontal bus at a cross point, which can be opened or closed at any time by the switch fabric controller.

When a packet arrives from port A and needs to be forwarded to port Y, the switch controller closes the crosspoint at the intersection of busses A and Y, and port A then sends the packet onto its bus, which is picked up (only) by bus Y.

A packet from port B can be forwarded to port X at the same time, since the A-to-Y and B-to-X packets use different input and output busses.

Thus, crossbar networks are capable of forwarding multiple packets in parallel.

However, if two packets from two different input ports are destined to the same output port, then one will have to wait at the input, since only one packet can be sent over any given bus at a time.

3.1.3 Output Processing

Output port processing, shown in Figure below, takes packets that have been stored in the output port's memory and transmits them over the output link.

This includes selecting and de-queueing packets for transmission, and performing the needed link layer and physical-layer transmission functions.

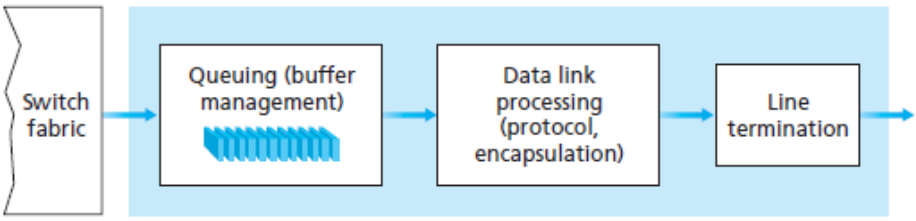


Figure 3.4: Output port processing

3.1.4 Where Does Queueing Occur?

The location and extent of queueing (either at the input port queues or the output port queues) will depend on the traffic load, the relative speed of the switching fabric, and the line speed.

As these queues grow large, the router's memory can eventually be exhausted and **packet loss** will occur when no memory is available to store arriving packets.

Suppose that the input and output line speeds (transmission rates) all have an identical transmission rate of R_{line} packets per second, and that there are N input ports and N output ports.

Assume that all packets have the same fixed length, and the packets arrive to input ports in a synchronous manner.

That is, the time to send a packet on any link is equal to the time to receive a packet on any link, and during such an interval of time, either zero or one packet can arrive on an input link.

Define the switching fabric transfer rate R_{switch} as the rate at which packets can be moved from input port to output port. If R_{switch} is N times faster than R_{line}, then only negligible queuing will occur at the input ports.

This is because even in the worst case, where all N input lines are receiving packets, and all packets are to be forwarded to the same output port, each batch of N packets (one packet per input port) can be cleared through the switch fabric before the next batch arrives.

Suppose R_{switch} is N times faster than R_{line}. Packets arriving at each of the N input ports are destined to the same output port.

The time it takes to send a single packet onto the outgoing link, N new packets will arrive at this output port. Since the output port can transmit only a single packet in a unit of time (the packet transmission time), the N arriving packets will have to queue (wait) for transmission over the outgoing link. Then N more packets can possibly arrive in the time it takes to transmit just one of the N packets that had just previously been queued.

Eventually, the number of queued packets can grow large enough to exhaust available memory at the output port, in which case packets are dropped.

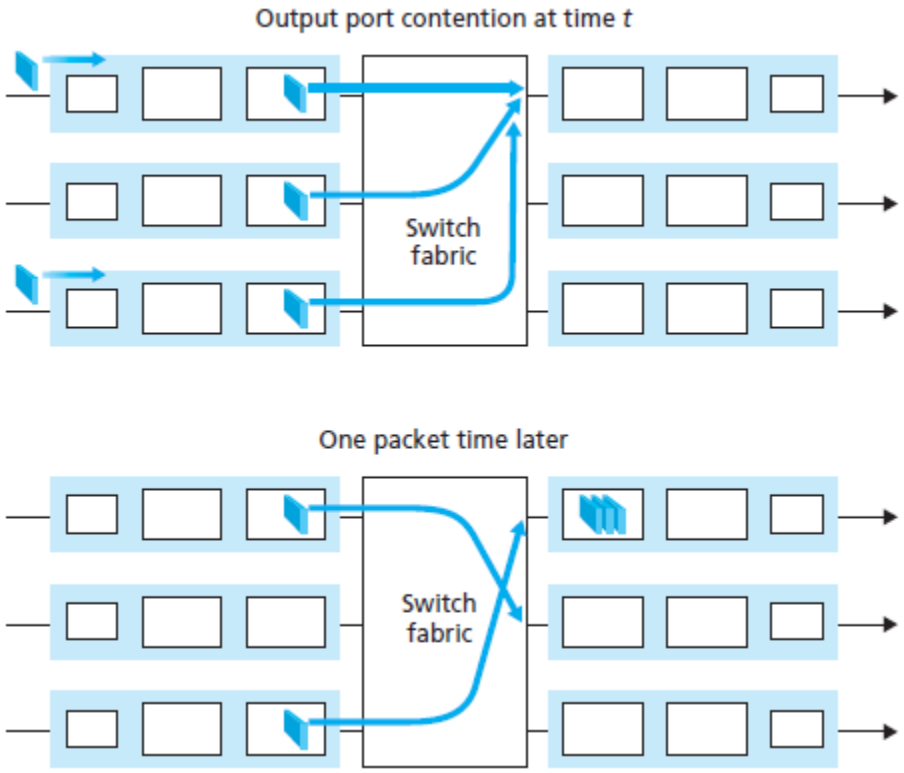


Figure 3.5: Output port queuing

Output port queuing is illustrated in Figure 3.5. At time *t*, a packet has arrived at each of the incoming input ports, each destined for the uppermost outgoing port.

Assuming identical line speeds and a switch operating at three times the line speed, one time unit later (that is, in the time needed to receive or send a packet), all three original packets have been transferred to the outgoing port and are queued awaiting transmission.

In the next time unit, one of these three packets will have been transmitted over the outgoing link. In example, two *new* packets have arrived at the incoming side of the switch; one of these packets is destined for this uppermost output port.

Given that router buffers are needed to absorb the fluctuations in traffic load. Buffer sizing was that the amount of buffering *(B)* should be equal to an average round-trip time (*RTT*, say 250 msec) times the link capacity *(C)*.

Thus, a 10 Gbps link with an RTT of 250 msec would need an amount of buffering equal to $B = RTT \cdot C$ = 2.5 Gbits of buffers.

When there are a large number of TCP flows (*N*) passing through a link, the amount of buffering needed is $B = RTT \times C/\sqrt{N}$

With a large number of flows typically passing through large backbone router links, the value of *N* can be large, with the decrease in needed buffer size becoming quite significant.

A consequence of output port queuing is that a **packet scheduler** at the output port must choose one packet among those queued for transmission. This selection might be done on first-come-first-served (FCFS) scheduling, or a more sophisticated scheduling discipline such as weighted fair queuing (WFQ), which shares the outgoing link fairly among the different end-to-end connections that have packets queued for transmission.

Similarly, if there is not enough memory to buffer an incoming packet, a decision must be made to either drop the arriving packet (a policy known as **drop-tail**) or remove one or more already-queued packets to make room for the newly arrived packet.

One of the widely implemented Active Queue Management AQM algorithms is the **Random Early Detection** (**RED**) algorithm.

Under RED, a weighted average is maintained for the length of the output queue. If the average queue length is less than a minimum threshold, *minth*, when a packet arrives, the packet is admitted to the queue.

Conversely, if the queue is full or the average queue length is greater than a maximum threshold, *maxth*, when a packet arrives, the packet is marked or dropped.

Finally, if the packet arrives to find an average queue length in the interval [*minth*, *maxth*], the packet is marked or dropped with a probability that is typically some function of the average queue length, *minth*, and *maxth*.

If the switch fabric is not fast enough (relative to the input line speeds) to transfer *all* arriving packets through the fabric without delay, then packet queuing can also occur at the input ports, as packets must join input port queues to wait their turn to be transferred through the switching fabric to the output port.

Consider a crossbar switching fabric and suppose that

(1) all link speeds are identical

(2) that one packet can be transferred from any one input port to a given output port in the same amount of time it takes for a packet to be received on an input link.

(3) packets are moved from a given input queue to their desired output queue in an FCFS manner.

Multiple packets can be transferred in parallel, as long as their output ports are different. However, if two packets at the front of two input queues are destined for the same output queue, then one of the packets will be blocked and must wait at the input queue—the switching fabric can transfer only one packet to a given output port at a time.

Figure below shows an example in which two packets (darkly shaded) at the front of their input queues are destined for the same upper-right output port.

Suppose that the switch fabric chooses to transfer the packet from the front of the upper-left queue. In this case, the darkly shaded packet in the lower-left queue must wait.

But not only must this darkly shaded packet wait, so too must the lightly shaded packet that is queued behind that packet in the lower-left queue, even though there is *no* contention for the middle-right output port (the destination for the lightly shaded packet). This phenomenon is known as **head-of-the-line** (**HOL**) **blocking** in an input-queued switch—a queued packet in an input queue must wait for transfer through the fabric (even though its output port is free) because it is blocked by another packet at the head of the line.

Due to HOL blocking, the input queue will grow to unbounded length under certain assumptions as soon as the packet arrival rate on the input links reaches only 58 percent of their capacity.

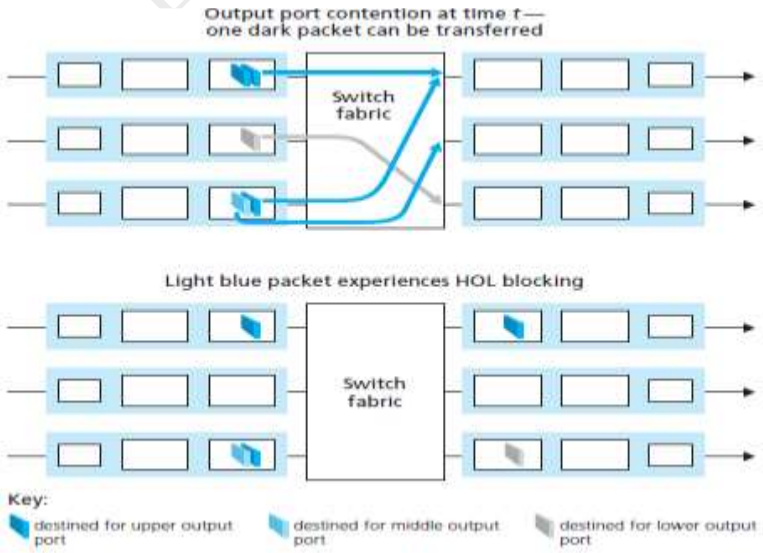


Figure 3.6: HOL blocking at an input queued switch

3.1.5 The Routing Control Plane

Router control plane architectures in which part of the control plane is implemented in the routers (e.g., local measurement/reporting of link state, forwarding table installation and maintenance) along with the data plane and part of the control plane can be implemented externally to the router (e.g., in a centralized server, which could perform route calculation).

A well-defined API dictates how these two parts interact and communicate with each other. By separating the software control plane from the hardware data plane (with a minimal router-resident control plane) can simplify routing by replacing distributed routing calculation with centralized routing calculation, and enable network innovation by allowing different customized control planes to operate over fast hardware data planes.

3.2 IPv6

In the early 1990s, the Internet Engineering Task Force began an effort to develop a successor to the IPv4 protocol. A prime motivation for this effort was the realization that the 32-bit IP address space was beginning to be used up, with new subnets and IP nodes being attached to the Internet (and being allocated unique IP addresses) at a breathtaking rate. To respond to this need for a large IP address space, a new IP protocol, IPv6, was developed.

IPv6 Datagram Format

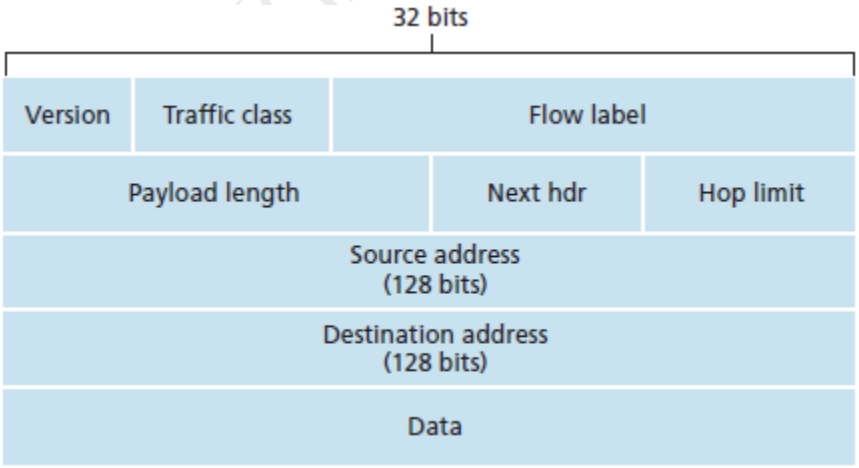


Figure 3.7: IPv6 datagram format

The format of the IPv6 datagram is shown in Figure above.

The most important changes introduced in IPv6 datagram format are:

• ***Expanded addressing capabilities***. IPv6 increases the size of the IP address from 32 to 128 bits.

IPv6 has introduced a new type of address, called an anycast address, which allows a datagram to be delivered to any one of a group of hosts.

• ***A streamlined 40-byte header***. The 40-byte fixed-length header allows for faster processing of the IP datagram.

• ***Flow labeling and priority.***

IPV6 allows "labelling of packets belonging to particular flows for which the sender requests special handling, such as a non default quality of service or real-time service."

For example, audio and video transmission might likely be treated as a flow. Traditional applications, such as file transfer and e-mail, might not be treated as flows.

The IPv6 header also has an 8-bit traffic class field. This field, like the TOS field in IPv4, can be used to give priority to certain datagrams within a flow, or it can be used to give priority to datagrams from certain applications over datagrams from other applications.

The following fields are defined in IPv6:

• ***Version***. This 4-bit field identifies the IP version number. IPv6 carries a value of 6 in this field.

•***Traffic class***. This 8-bit field is similar to the TOS field we saw in IPv4.

• ***Flow label.*** *T*his 20-bit field is used to identify a flow of datagrams.

• ***Payload length***. This 16-bit value is treated as an unsigned integer giving the number of bytes in the IPv6 datagram following the fixed-length, 40-byte datagram header.

• ***Next header***. This field identifies the protocol to which the contents (data field) of this datagram will be delivered. The field uses the same values as the protocol field in the IPv4 header.

• ***Hop limit.*** The contents of this field are decremented by one by each router that forwards the datagram. If the hop limit count reaches zero, the datagram is discarded.

• ***Source and destination addresses***. The various formats of the IPv6 128-bit address

• ***Data***. This is the payload portion of the IPv6 datagram. When the datagram reaches its destination, the payload will be removed from the IP datagram and passed on to the protocol specified in the next header field.

Following are the differences between IPV4 & IPV6 :

• ***Fragmentation/Reassembly***. IPv6 does not allow for fragmentation and reassembly at intermediate routers; these operations can be performed only by the source and destination. If an IPv6 datagram received by a router is too large to be forwarded over the outgoing link, the router simply drops the datagram and sends a "Packet Too Big" ICMP error message (see below) back to the sender.

The sender can then resend the data, using a smaller IP datagram size. Fragmentation and reassembly is a time-consuming operation; removing this functionality from the routers and placing it squarely in the end systems considerably speeds up IP forwarding within the network.

• ***Header checksum***. *T*he transport-layer (for example, TCP and UDP) and link-layer (for example, Ethernet) protocols in the Internet layers perform checksumming, the designers of IP probably felt that this functionality was sufficiently redundant in the network layer that it could be removed.

Since the IPv4 header contains a TTL field (similar to the hop limit field in IPv6), the IPv4 header checksum needed to be recomputed at every router. As with fragmentation and reassembly, was a costly operation in IPv4.

• ***Options***. An options field is no longer a part of the standard IP header. Options field is one of the possible next headers pointed to from within the IPv6 header. That is, just as TCP or UDP protocol headers can be the next header within an IP packet. The removal of the options field results in a fixed-length, 40-byte IP header.

3.2.1 Transitioning from IPV4 to IPV6 :

IPv6-capable nodes has a dual-stack approach, where IPv6 nodes have a complete IPv4 implementation. Such a node, referred to as an IPv6/IPv4 node , has the ability to send and receive both IPv4 and IPv6 datagrams.

When interoperating with an IPv4 node, an IPv6/IPv4 node can use IPv4 datagrams; when interoperating with an IPv6 node, it can speak IPv6.

IPv6/IPv4 nodes must have both IPv6 and IPv4 addresses. They will be able to determine whether the node is IPv6-capable or IPv4-only.

In the dual-stack approach, if either the sender or the receiver is only IPv4- capable, an IPv4 datagram must be used.

It is possible that two IPv6-capable nodes can end up, sending IPv4 datagrams to each other. This is illustrated in Figure below.

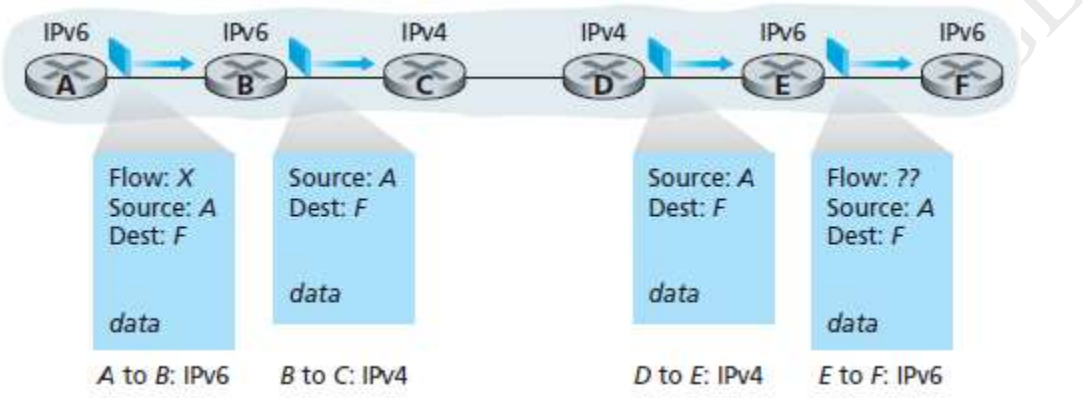


Figure 3.8: A dual-stack approach

Suppose Node A is IPv6-capable and wants to send an IP datagram to Node F, which is also IPv6-capable. Nodes A and B can exchange an IPv6 datagram.

Node B must create an IPv4 datagram to send to C. Certainly, the data field of the IPv6 datagram can be copied into the data field of the IPv4 datagram and appropriate address mapping can be done.

In performing the conversion from IPv6 to IPv4, there will be IPv6-specific fields in the IPv6 datagram that have no counterpart in IPv4.

The information in these fields will be lost. Thus, even though E and F can exchange IPv6 datagrams, the arriving IPv4 datagrams at E from D do not contain all of the fields that were in the original IPv6 datagram sent from A.

An alternative to the dual-stack approach, is known as **tunneling**.

Tunneling can solve the problem noted above, allowing, for example, E to receive the IPv6 datagram originated by A. The basic idea behind tunneling is the following.

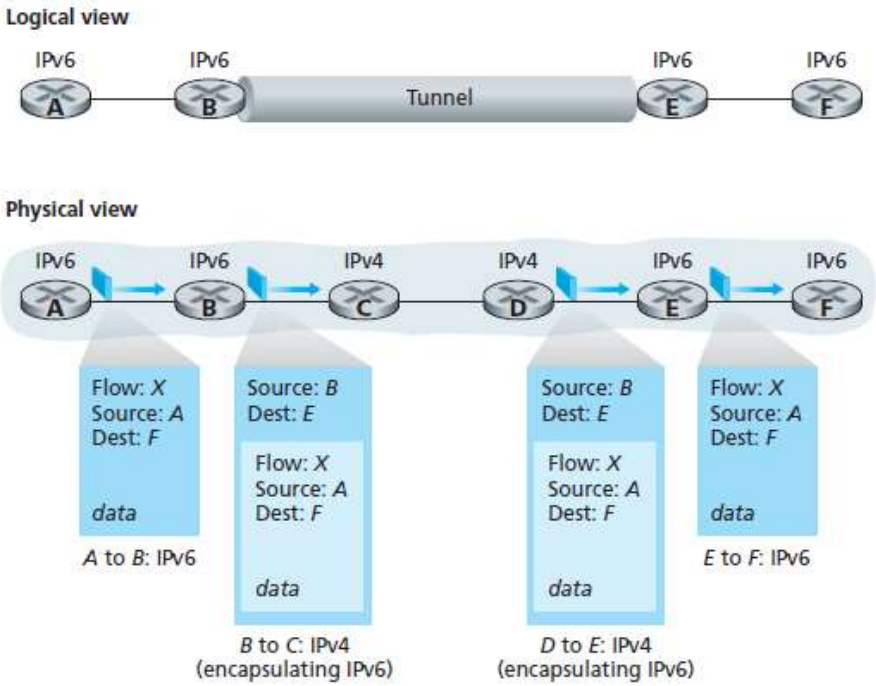


Figure 3.9: Tunneling

Suppose two IPv6 nodes (for example, B and E in Figure above) want to interoperate using IPv6 datagrams but are connected to each other by intervening IPv4 routers.

The intervening set of IPv4 routers between two IPv6 routers is referred as a **tunnel**, as illustrated in Figure above.

With tunneling, the IPv6 node on the sending side of the tunnel (for example, B) takes the *entire* IPv6 datagram and puts it in the data (payload) field of an IPv4 datagram.

This IPv4 datagram is then addressed to the IPv6 node on the receiving side of the tunnel (for example, E) and sent to the first node in the tunnel (for example, C).

The intervening IPv4 routers in the tunnel route this IPv4 datagram among themselves, unaware that the IPv4 datagram itself contains a complete IPv6 datagram.

The IPv6 node on the receiving side of the tunnel eventually receives the IPv4 datagram, determines that the IPv4 datagram contains an IPv6 datagram, extracts the IPv6 datagram, and then routes the IPv6 datagram.

3.2.2 IP Security

IPsec, is a secure network-layer protocols .

IPsec has been designed to be backward compatible with IPv4 and IPv6. If two hosts want to securely communicate, IPsec needs to be available only in those two hosts.

On the sending side, the transport layer passes a segment to IPsec. IPsec then encrypts the segment, appends additional security fields to the segment, and encapsulates the resulting payload in an ordinary IP datagram.

The sending host then sends the datagram into the Internet, which transports it to the destination host. There, IPsec decrypts the segment and passes the unencrypted segment to the transport layer.

The services provided by an IPsec session include:

• *Cryptographic agreement*. Mechanisms that allow the two communicating hosts to agree on cryptographic algorithms and keys.

• *Encryption of IP datagram payloads*. When the sending host receives a segment from the transport layer, IPsec encrypts the payload. The payload can only be decrypted by IPsec in the receiving host.

• *Data integrity*. IPsec allows the receiving host to verify that the datagram's header fields and encrypted payload were not modified while the datagram was in route from source to destination.

• *Origin authentication*. When a host receives an IPsec datagram from a trusted source, the host is assured that the source IP address in the datagram is the actual source of the datagram.

3.3 Routing Algorithms

We now turn our attention to the other major topic of this chapter, namely, the network layer's critical routing function. Whether the network layer provides a datagram service (in which case different packets between a given source-destination pair may take different routes) or a VC service (in which case all packets between a given source and destination will take the same path), the network layer must nonetheless determine the path that packets take from senders to receivers.

A host is attached directly to one router, the **default router** for the host (also called the **first-hop router** for the host).

Whenever a host sends a packet, the packet is transferred to its default router. We refer to the default router of the source host as the **source router** and the default router of the destination host as the **destination router**.

The purpose of a routing algorithm is: given a set of routers, with links connecting the routers, a routing algorithm finds a "good" path i.e least cost path from source router to destination router.

A **graph** $G = (N,E)$ is a set N of nodes and a collection E of edges, where each edge is a pair of nodes from N.

In the context of network-layer routing, the nodes in the graph represent routers—the points at which packet-forwarding decisions are made—and the edges connecting these nodes represent the physical links between these routers.

An edge's cost reflects the physical length of the corresponding link .

Consider figure below , For any edge (x,y) in E, we denote $c(x,y)$ as the cost of the edge between nodes x and y. If the pair (x,y) does not belong to E, we set $c(x,y) = \infty$.

Only undirected graphs (i.e., graphs whose edges do not have a direction) are considered, so that edge (x,y) is the same as edge (y,x) and that $c(x,y) = c(y,x)$. Also, a node y is said to be a **neighbor** of node x if (x,y) belongs to E.

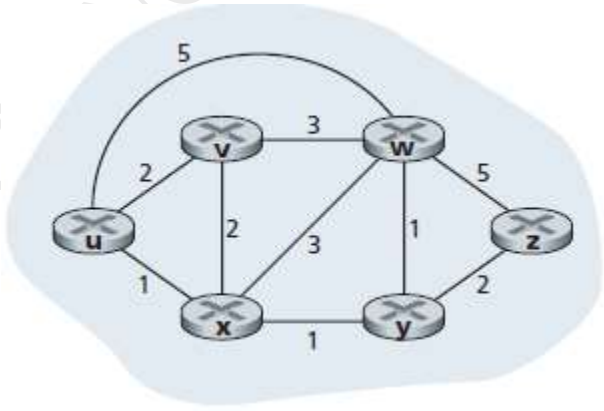


Figure 3.10: Abstract graph model of a computer network

A path in a graph $G = (N,E)$ is a sequence of nodes $(x1, x2,..., xp)$ such that each of the pairs $(x1,x2)$, $(x2,x3),...,(xp\text{-}1,xp)$ are edges in E.

The cost of a path $(x1,x2,..., xp)$ is the sum of all the edge costs along the path, that is, $c(x1,x2) + c(x2,x3) + ... + c(xp\text{-}1,xp)$.

Given any two nodes *x* and *y*, there are typically many paths between the two nodes, with each path having a cost. One or more of these paths is a **least-cost path**. For example, the least-cost path between source node *u* and destination node *w* is (*u*, *x*, *y*, *w*) with a path cost of 3.

Classification of routing algorithm:

Routing algorithms can be classified according to whether they are global or decentralized.

- A **global routing algorithm** computes the least-cost path between a source and destination using complete, global knowledge about the network. The algorithm takes the connectivity between all nodes and all link costs as inputs.

  Algorithms with global state information are referred to as **link-state** (**LS**) **algorithms**, since the algorithm must be aware of the cost of each link in the network.

- In a **decentralized routing algorithm**, the calculation of the least-cost path is carriedout in an iterative, distributed manner. No node has complete information about the costs of all network links.

  Each node begins with only the knowledge of the costs of its own directly attached links. Through an iterative process of calculation and exchange of information with its neighboring nodes, a node gradually calculates the least-cost path to a destination or set of destinations.

  The decentralized routing algorithm is called a **distance-vector (DV) algorithm**, because each node maintains a vector of estimates of the costs (distances) to all other nodes in the network.

Routing algorithms is classified according to whether they are static or dynamic.

- In **static routing algorithms**, routes change very slowly over time, often as a result of human intervention.
- **Dynamic routing algorithms** change the routing paths as the network traffic loads or topology change.

Routing algorithms is classified according to whether they are load sensitive or load-insensitive.

- In a **load-sensitive algorithm**, link costs vary dynamically to reflect the current level of congestion in the underlying link. If a high cost isassociated with a link that is currently congested, a routing algorithm will tend to choose routes around such a congested link.

- Internet routing algorithms (such as RIP, OSPF, and BGP) are **load-insensitive**, as a link's cost does not explicitly reflect its current level of congestion.

3.3.1 The Link-State (LS) Routing Algorithm

Link costs are available as input to the LS algorithm. This is accomplished by having each node broadcast link-state packets to *all* other nodes in the network, with each link-state packet containing the identities and costs of its attached links.

The link-state routing algorithm is known as ***Dijkstra's algorithm.***

Dijkstra's algorithm is iterative and has the property that after the *k*th iteration of the algorithm, the least-cost paths are known to *k* destination nodes, and among the least-cost paths to all destination nodes, these *k* paths will have the *k* smallest costs.

The following notations are defined:

• $D(v)$: cost of the least-cost path from the source node to destination v as of this iteration of the algorithm.

• $p(v)$: previous node (neighbor of v) along the current least-cost path from the source to v.

• N' : subset of nodes; v is in N' if the least-cost path from the source to v is definitively known.

The global routing algorithm consists of an initialization step followed by a loop. The number of times the loop is executed is equal to the number of nodes in the network.

Upon termination, the algorithm will have calculated the shortest paths from the source node u to every other node in the network.

Link-State (LS) Algorithm for Source Node *u*

```
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) = ∞
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
```

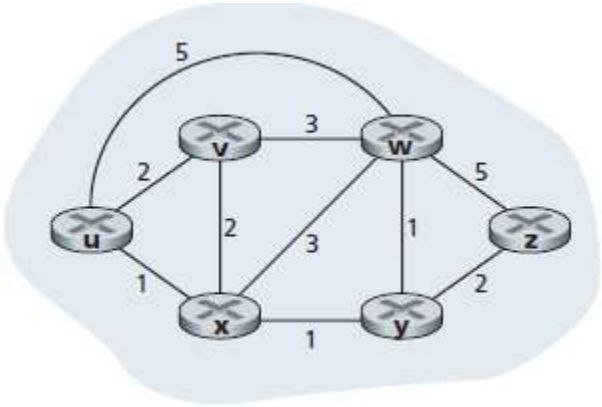


Figure 3.11: Network graph

Consider the network in graph above and compute the least-cost paths from *u* to all possible destinations.

A tabular summary of the algorithm's computation is shown in Table below, where each line in the table gives the values of the algorithm's variables at the end of the iteration.

Consider the following few first steps:

• In the initialization step, the currently known least-cost paths from *u* to its directly attached neighbors, *v*, *x*, and *w*, are initialized to 2, 1, and 5, respectively.

The cost to *w* is set to 5 since this is the cost of the direct link from *u* to *w*. The costs to *y* and *z* are set to infinity because they are not directly connected to *u*.

• In the first iteration, consider the nodes that are not yet added to the set *N*_ and find that node with the least cost as of the end of the previous iteration.

That node is *x*, with a cost of 1, and thus *x* is added to the set *N*_.

Line 12 of the LS algorithm is then performed to update *D*(*v*) for all nodes *v*, yielding the results shown in the second line (Step 1) in Table. The cost of the path to *v* is unchanged.

The cost of the path to *w* (which was 5 at the end of the initialization) through node *x* is found to have a cost of 4. Hence this lower-cost path is selected and *w*'s predecessor along the shortest path from *u* is set to *x*. Similarly, the cost to *y* (through *x*) is computed to be 2, and the table is updated accordingly.

• In the second iteration, nodes *v* and *y* are found to have the least-cost paths (2), and we break the tie arbitrarily and add *y* to the set *N*_ so that *N*_ now contains *u*, *x*, and *y*. The cost to the remaining nodes not yet in *N*_, that is, nodes *v*, *w*, and *z*, are updated via line 12 of the LS algorithm, yielding the results shown in the third row in the Table 4.3.

• And so on. . . .

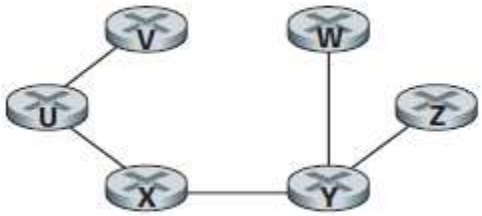
When the LS algorithm terminates, we have, for each node, its predecessor along the least-cost path from the source node.

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					

Figure 3.12: Running the link-state algorithm on the network in figure 3.11

The forwarding table in a node, say node *u*, can then be constructed from this information by storing, for each destination, the next-hop node on the least-cost path from *u* to the destination.

Figure 3.13 below shows the resulting least-cost paths and forwarding table in *u* for the network shown in above figure .



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

Figure 3.13: Least cost path and forwarding table for nodule u

Computational Complexity : In the first iteration, we need to search through all *n* nodes to determine the node, *w*, not in *N*_ that has the minimum cost.

In the second iteration, we need to check *n* – 1 nodes to determine the minimum cost;

In the third iteration *n* – 2 nodes, and so on.

Overall, the total number of nodes searched through over all the iterations is $n(n + 1)/2$, and thus the preceding implementation of the LS algorithm has worst-case complexity of order *n* squared: $O(n2)$.

3.3.2 The Distance-Vector (DV) Routing Algorithm

Distance vector (DV) algorithm is iterative, asynchronous, and distributed.

Each node receives some information from one or more of its ***directly attached*** neighbours, performs a calculation, and then distributes the results of its calculation back to its neighbours.

It is *iterative i.e* process continues until no more information is exchanged between neighbours.

The algorithm is *asynchronous* i.e it does not require all of the nodes to operate in lockstep with each other.

Let $d_x(y)$ be the cost of the least-cost path from node *x* to node *y*. Then the least costs are related by the Bellman-Ford equation, namely,

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

where the ***minv*** in the equation is taken over all of *x*'s neighbours.

After traveling from *x* to *v*, if we then take the least-cost path from *v* to *y*, the path cost will be $c(x,v) + d_v(y)$.

Since we must begin by travelling to some neighbour *v*, the least cost from *x* to *y* is the minimum of $c(x,v) + d_v(y)$ taken over all neighbours *v*.

Evaluate for source node *u* and destination node *z* in Figure. The source node *u has* three neighbours*: **nodes v, x, and w.***

dv(z) = 5, dx(z) = 3, and dw(z) = 3.

Substitute these values into Equation above, along with the costs c(u,v) = 2, c(u,x) = 1, and c(u,w) = 5, gives *:*

du(z) = min{2 + 5, 5 + 3, 1 + 3} = 4 ;

The basic idea is as follows:

Each node x begins with Dx(y), an estimate of cost of the least-cost path from itself to node y, for all nodes in N.

Let **Dx = [Dx(y): y in N]** be node x's distance vector, which is the vector of cost estimates from x to all other nodes, y, in N.

With the DV algorithm, each node x maintains the following routing information:

• For each neighbour v, the cost c(x,v) from x to directly attached neighbour, v

• Node x's distance vector, that is, Dx = [Dx(y): y in N], containing x's estimate of its cost to all destinations, y, in N

• The distance vectors of each of its neighbours, that is, Dv = [Dv(y): y in N] for each neighbour v of x.

Each node sends a copy of its distance vector to each of its neighbours. When a node *x* receives a new distance vector from any of its neighbours *v*, it saves *v*'s distance vector, and then uses the Bellman-Ford equation to update its own distance vector as follows:

$D_x(y) = \min_v\{c(x,v) + D_v(y)\}$ for each node *y* in *N*

If node *x*'s distance vector has changed as a result of this update step, node *x* will then send its updated distance vector to each of its neighbours, which can update their own distance vectors.

Distant Vector Algorithm:

At each node, x :

```
1    Initialization:
2        for all destinations y in N:
3            Dx(y) = c(x,y) /* if y is not a neighbour then c(x,y) = ∞ */
4        for each neighbour w
5            Dw(y) = ? for all destinations y in N
6        for each neighbour w
7            send distance vector Dx = [Dx(y): y in N] to w
8
```

```
9     loop
10        wait (until I see a link cost change to some neighbour w or
11              until I receive a distance vector from some neighbour w)
12
13        for each y in N:
14              Dx(y) = minv{c(x,v) + Dv(y)}
15
16        if    Dx(y) changed for any destination y
17              send distance vector Dx = [Dx(y): y in N] to all neighbours
18
19        forever
```

In the DV algorithm, a node *x* updates its distance-vector estimate when it either sees a cost change in one of its directly attached links or receives a distance vector update from some neighbour.

But to update its own forwarding table for a given destination *y*, what node *x* needs to know is not the shortest-path distance to *y* but instead the neighbouring node $v^*(y)$ that is the next-hop router along the shortest path to *y*.

The next-hop router $v^*(y)$ is the neighbour *v* that achieves the minimum in Line 14 of the DV algorithm.

Thus, in Lines 13–14, for each destination *y*, node *x* also determines $v^*(y)$ and updates its forwarding table for destination *y*.

The LS algorithm is a global algorithm in the sense that it requires each node to first obtain a complete map of the network before running the Dijkstra algorithm.

The DV algorithm is *decentralized* and does not use such global information.

Only information a node will have is the costs of the links to its directly attached neighbours and information it receives from these neighbours. Each node waits for an update from any neighbour (Lines 10–11), calculates its new distance vector when receiving an update (Line 14), and distributes its new distance vector to its neighbours (Lines 16–17).

Figure 3.14 below illustrates the operation of the DV algorithm for the simple three node network shown at the top of the figure. The operation of the algorithm is illustrated in a synchronous manner, where all nodes simultaneously receive distance vectors from their neighbours, compute their new distance vectors, and inform their neighbours if their distance vectors have changed.

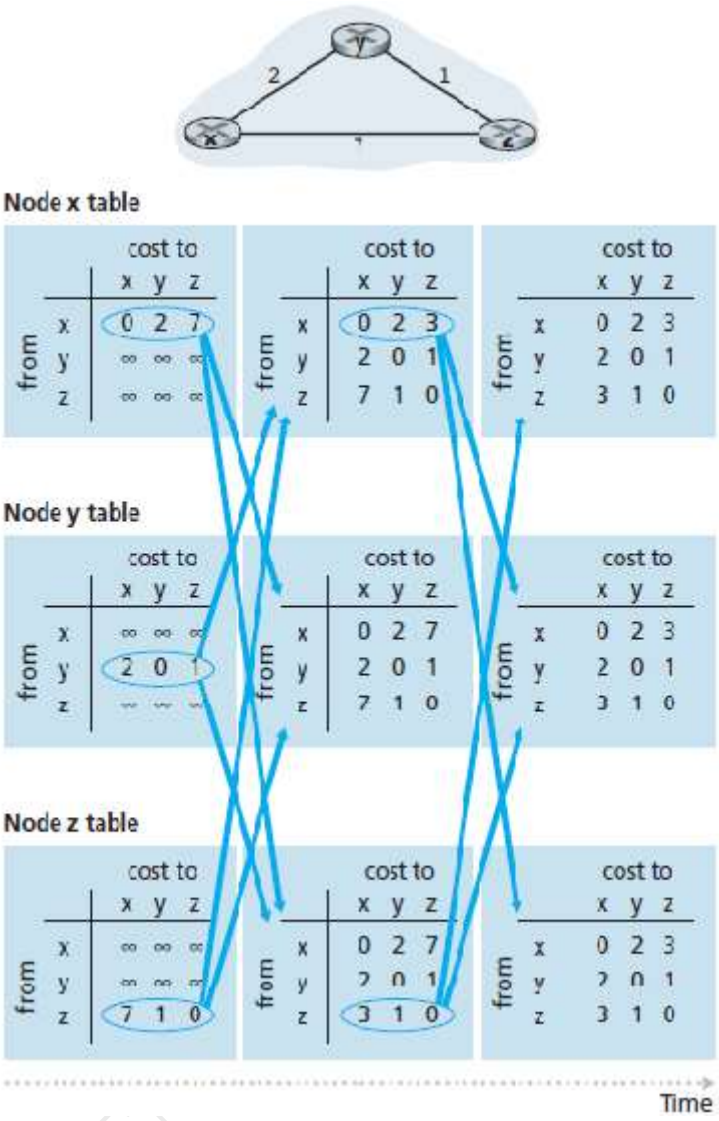


Figure 3.14 Operation of DV algorithm

The leftmost column of the figure displays three initial routing tables for each of the three nodes. For example, the table in the upper-left corner is node *x*'s initial routing table. Within a specific routing table, each row is a distance vector—specifically, each node's routing table includes its own distance vector and that of each of its neighbours. Thus, the first row in node *x*'s initial routing table is $\boldsymbol{Dx = [Dx(x), Dx(y), Dx(z)] = [0, 2, 7].}$

The second and third rows in this table are the most recently received distance vectors from nodes *y* and *z*, respectively. Because at initialization node *x* has not received anything from node *y* or *z*, the entries in the second and third rows are initialized to infinity.

After initialization, each node sends its distance vector to each of its two neighbours.

This is illustrated in Figure above by the arrows from the first column of tables to the second column of tables. For example, node *x* sends its distance vector $Dx = [0, 2, 7]$ to both nodes *y* and *z*. After receiving the updates, each node recomputes its own distance vector.

For example, node *x* computes

$Dx(x) = 0$

$Dx(y) = \min\{c(x,y) + Dy(y), c(x,z) + Dz(y)\} = \min\{2 + 0, 7 + 1\} = 2$

$Dx(z) = \min\{c(x,y) + Dy(z), c(x,z) + Dz(z)\} = \min\{2 + 1, 7 + 0\} = 3$

The second column therefore displays, for each node, the node's new distance vector along with distance vectors just received from its neighbours.

For example, that node *x*'s estimate for the least cost to node *z*, $Dx(z)$, has changed from 7 to 3.

For node *x*, neighbouring node *y* achieves the minimum in line 14 of the DV algorithm; thus at this stage of the algorithm, we have at node *x* that $v^*(y) = y$ and $v^*(z) = y$.

After the nodes recompute their distance vectors, they again send their updated distance vectors to their neighbours (if there has been a change).

This is illustrated in Figure above by the arrows from the second column of tables to the third column of tables.

Only nodes *x* and *z* send updates: node *y*'s distance vector didn't change so node *y* doesn't send an update. After receiving the updates, the nodes then recompute their distance vectors and update their routing tables, which are shown in the third column.

The process of receiving updated distance vectors from neighbours, recomputing routing table entries, and informing neighbours of changed costs of the least-cost path to a destination continues until no update messages are sent. At this point, since no update messages are sent, no further routing table calculations will occur and the algorithm will enter a quiescent state; that is, all nodes will be performing the wait in Lines 10–11 of the DV algorithm.

Distance-Vector Algorithm: Link-Cost Changes and Link Failure

When a node running the DV algorithm detects a change in the link cost from itself to a neighbour (Lines 10–11), it updates its distance vector (Lines 13–14) and, if there's a change in the cost of the least-cost path, informs its neighbours (Lines 16–17) of its new distance vector.

Figure (a) below illustrates a scenario where the link cost from *y* to *x* changes from 4 to 1.

Focus is only on *y*' and *z*'s distance table entries to destination *x*. The DV algorithm causes the following sequence of events to occur:

• At time *t*0, *y* detects the link-cost change (the cost has changed from 4 to 1), updates its distance vector, and informs its neighbours of this change since its distance vector has changed.

• At time *t*1, *z* receives the update from *y* and updates its table. It computes a new least cost to *x* (it has decreased from a cost of 5 to a cost of 2) and sends its new distance vector to its neighbours.

• At time *t*2, *y* receives *z*'s update and updates its distance table. *y*'s least costs do not change and hence *y* does not send any message to *z*. The algorithm comes to a quiescent state.

Thus, only two iterations are required for the DV algorithm to reach a quiescent state.

Consider *increase in* link cost. Suppose that the link cost between *x* and *y* increases from 4 to 60, as shown in Figure (b) below.

1. Before the link cost changes, $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$, and $D_z(x) = 50$. At time *t*0, *y* detects the link-cost change (the cost has changed from 4 to 60) y computes its new minimum-cost path to *x* to have a cost of

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

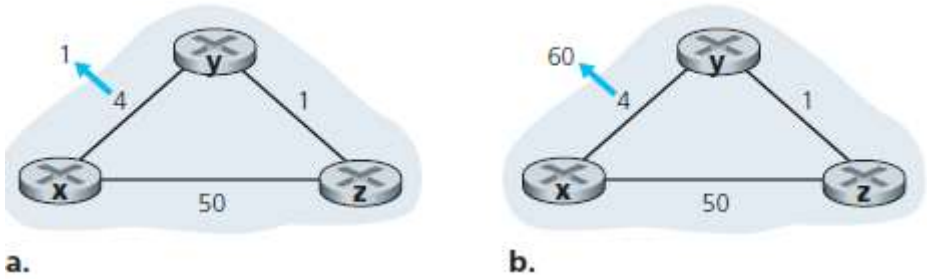


Fig 3.15 Changes in link cost

New cost via *z* is *wrong.* But the only information node *y* has is that its direct cost to *x* is 60 and that *z* has last told *y* that *z* could get to *x* with a cost of 5. So in order to get to *x, y* would now route through *z,* fully expecting that *z* will be able to get to *x* with a cost of 5. As of *t*1, a routing loop is -- in order to get to *x, y* routes through *z,* and *z* routes through *y*.

A routing loop is like a black hole—a packet destined for *x* arriving at *y* or *z* as of *t*1 will bounce back and forth between these two nodes forever

2. Since node *y* has computed a new minimum cost to *x,* it informs *z* of its new distance vector at time *t*1.

3. After *t*1, *z* receives *y*'s new distance vector, which indicates that *y*'s minimum cost to *x* is 6. *z* knows it can get to *y* with a cost of 1 and hence computes a new least cost to *x* of $D_z(x) = \min\{50 + 0, 1 + 6\} = 7$. Since *z*'s least cost to *x* has increased, it then informs *y* of its new distance vector at *t*2.

4. In a similar manner, after receiving *z*'s new distance vector, *y* determines $D_y(x) = 8$ and sends *z* its distance vector. *z* then determines $D_z(x) = 9$ and sends *y* its distance vector, and so on.

Distance-Vector Algorithm: Adding Poisoned Reverse

If *z* routes through *y* to get to destination *x*, then *z* will advertise to *y* that its distance to *x* is infinity, that is, *z* will advertise to *y* that $D_z(x) = \infty$ (even though *z* knows $D_z(x) = 5$ in truth). *z* will continue telling this to *y* as long as it routes to *x* via *y*. Since *y* believes that *z* has no path to *x*, *y* will never attempt to route to *x* via *z*, as long as *z* continues to route to *x* via *y*.

Poisoned reverse solves the particular looping problem encountered before in Figure (b) above. As a result of the poisoned reverse, *y*'s distance table indicates $D_z(x) = \infty$.

When the cost of the (*x*, *y*) link changes from 4 to 60 at time *t*0, *y* updates its table and continues to route directly to *x*, albeit at a higher cost of 60, and informs *z* of its new cost to *x*, that is, $D_y(x) = 60$.

After receiving the update at *t*1, *z* immediately shifts its route to *x* to be via the direct (*z*, *x*) link at a cost of 50. Since this is a new least-cost path to *x*, and since the path no longer passes through *y*, *z* now informs *y* that $D_z(x) = 50$ at *t*2.

After receiving the update from *z*, *y* updates its distance table with $D_y(x) = 51$. Also, since *z* is now on *y*'s least-cost path to *x*, *y* poisons the reverse path from *z* to *x* by informing *z* at time *t*3 that $D_y(x) = \infty$ (even though *y* knows that $D_y(x) = 51$ in truth).

A Comparison of LS and DV Routing Algorithms

In the DV algorithm, each node talks to *only* its directly connected neighbours, but it provides its neighbours with least-cost estimates from itself to *all* the nodes (that it knows about) in the network.

In the LS algorithm, each node talks with *all* other nodes (via broadcast), but it tells them *only* the costs of its directly connected links.

Differences between LS and DV algorithm :

N is the set of nodes (routers) and *E* is the set of edges (links).

• ***Message complexity.*** LS requires each node to know the cost of each link in the network. This requires O(|N| |E|) messages to be sent.

If a link cost changes, the new link cost must be sent to all nodes. The DV algorithm requires message exchanges between directly connected neighbours at each iteration.

The time needed for the algorithm to converge can depend on many factors. When link costs change, the DV algorithm will propagate the results of the changed link cost only if the new link cost results in a changed least-cost path for one of the nodes attached to that link.

• ***Speed of convergence***. *Complexity* of LS is O(|N|2). The DV algorithm can converge slowly and can have routing loops while the algorithm is converging. DV also suffers from the count-to-infinity problem.

• ***Robustness***. Under LS, a router could broadcast an incorrect cost for one of its attached links (but no others). A node could also corrupt or drop any packets it received as part of an LS broadcast. But an LS node is computing only its own forwarding tables; other nodes are performing similar calculations for themselves. This means route calculations are separated under LS, providing a degree of robustness. Under DV, a node can advertise incorrect least-cost paths to any or all destinations.

3.3.3 Hierarchical Routing

One router is indistinguishable from another i.e all routers executes the same routing algorithm to compute routing paths through the entire network.

In practice, this model and its view of a homogenous set of routers all executing the same routing algorithm is simple for two important reasons:

• ***Scale***. As the number of routers becomes large, the overhead involved in computing, storing and communicating routing information is prohibitive.

Internet consists of hundreds of millions of hosts. Storing routing information at each of these hosts would clearly require enormous amounts of memory. The overhead required to broadcast LS updates among all of the routers in the public Internet would leave no bandwidth left for sending data packets!

• ***Administrative autonomy***. An organization should be able to run and administer its network as it wishes, while still being able to connect its network to other outside networks.

Both the above problems can be solved by organizing routers into autonomous systems (ASs), with each AS consisting of a group of routers that are typically under the same administrative control.

Routers within the same AS all run the same routing algorithm (for example, an LS or DV algorithm) and have information about each other.

The routing algorithm running within an autonomous system is called an intra autonomous system routing protocol. It is necessary, to connect ASs to each other, and thus one or more of the routers in an AS will have the added task of being responsible for forwarding packets to destinations outside the AS; these routers are called gateway routers.

Figure below provides a simple example with three ASs: AS1, AS2, and AS3.

In this figure, the heavy lines represent direct link connections between pairs of routers. The thinner lines hanging from the routers represent subnets that are directly connected to the routers. AS1 has four routers—1a, 1b, 1c, and 1d— which run the intra-AS routing protocol used within AS1.

Thus, each of these four routers knows how to forward packets along the optimal path to any destination within AS1. Similarly, autonomous systems AS2 and AS3 each have three routers.

Intra-AS routing protocols running in AS1, AS2, and AS3 need not be the same. The routers 1b, 1c, 2a, and 3a are all gateway routers.

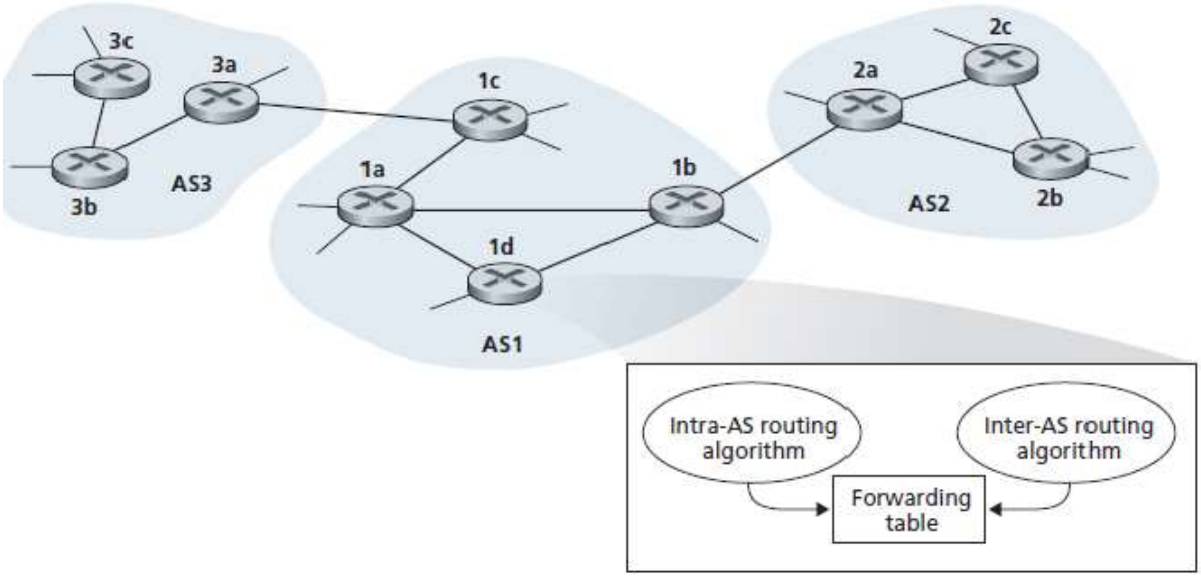


Fig 3.16: Example of interconnected autonomous system

The gateway router, upon receiving the packet, forwards the packet on the one link that leads outside the AS. The AS on the other side of the link then takes over the responsibility of routing the packet to its ultimate destination.

As an example, suppose router 2b in Figure above receives a packet whose destination is outside of AS2. Router 2b will then forward the packet to either router 2a or 2c, as specified by router 2b's forwarding table, which was configured by AS2's intra-AS routing protocol.

The packet will eventually arrive to the gateway router 2a, which will forward the packet to 1b. Once the packet has left 2a, AS2's job is done with this one packet.

AS1 needs :

(1) to learn which destinations are reachable via AS2 and which destinations are reachable via AS3.

(2) to propagate this reachability information to all the routers within AS1, so that each router can configure its forwarding table to handle external-AS destinations.

These two tasks—obtaining reachability information from neighbouring ASs and propagating the reachability information to all routers internal to the AS—are handled by the inter-AS routing protocol. Since the inter-AS routing protocol involves communication between two ASs, the two communicating ASs must run the same inter-AS routing protocol.

Consider a subnet *x* and suppose that AS1 learns from the inter-AS routing protocol that subnet *x* is reachable from AS3 but is *not* reachable from AS2.

AS1 then propagates this information to all of its routers. When router 1d learns that subnet *x* is reachable from AS3, and hence from gateway 1c, it then determines, from the information provided by the intra-AS routing protocol, the router interface that is on the least-cost path from router 1d to gateway router 1c. Say this is interface *I*. The router 1d can then put the entry (*x*, *I*) into its forwarding table.

Hot Potato Routing :

In hot-potato routing, the AS gets rid of the packet (the hot potato) as quickly as possible. This is done by having a router send the packet to the gateway router that has the smallest router-to-gateway cost among all gateways with a path to the destination.

Eg : Hot-potato routing, running in 1d, would use information from the intra-AS routing protocol to determine the path costs to 1b and 1c, and then choose the path with the least cost. Once this path is chosen, router 1d adds an entry for subnet *x* in its forwarding table.

Figure below summarizes the steps for adding the new entry for subnet *x* to the forwarding table of router 1d(refer prev fig) .

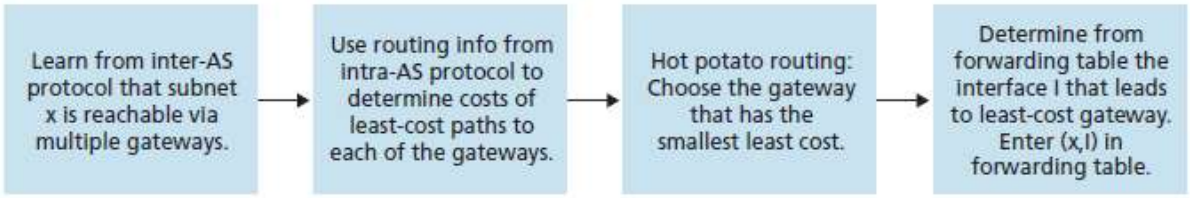


Fig 3.17: Steps in adding an outside AS destination in a routers forwarding table

When an AS learns about a destination from a neighbouring AS, the AS can advertise this routing information to some of its other neighbouring ASs.

For example, suppose AS1 learns from AS2 that subnet *x* is reachable via AS2. AS1 could then tell AS3 that *x* is reachable via AS1. In this manner, if AS3 needs to route a packet destined to *x*, AS3 would forward the packet to AS1, which would in turn forward the packet to AS2.

The problems of scale and administrative authority are solved by defining autonomous systems. Within an AS, all routers run the same intra-AS routing protocol. The ASs run the same inter-AS routing protocol.

The problem of scale is solved because an intra-AS router need only know about routers within its AS.

The problem of administrative authority is solved since an organization can run intra-AS routing protocol it chooses; Each pair of connected ASs needs to run the same inter-AS routing protocol to exchange reachability information.

3.4 Routing in the Internet

3.4.1 Intra-AS Routing in the Internet: RIP

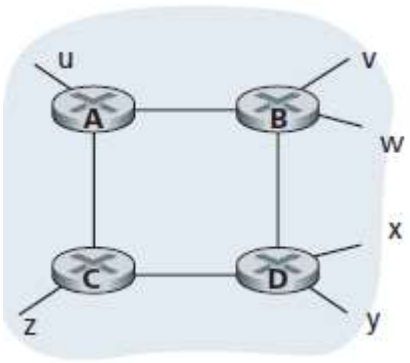
An intra-AS routing protocol is used to determine how routing is performed within an autonomous system (AS). Intra-AS routing protocols are also known as interior gateway protocols.

Two routing protocols have been used extensively for routing within an autonomous system in the Internet: the Routing Information Protocol (RIP) and Open Shortest Path First (OSPF).

RIP is a distance-vector protocol that operates in a manner very close to the idealized DV protocol. In RIP (and also in OSPF), costs are from source router to a destination subnet.

RIP uses the term *hop*, which is the number of subnets traversed along the shortest path from source router to destination subnet, including the destination subnet. Figure below illustrates

an AS with six leaf subnets. The table in the figure indicates the number of hops from the source A to each of the leaf subnets.



Destination	Hops
u	1
v	2
w	2
x	3
y	3
z	2

Fig 3.18: Number of hops from source router A to various subnets

The maximum cost of a path is limited to 15, thus limiting the use of RIP to autonomous systems that are fewer than 15 hops in diameter.

In DV protocols, neighbouring routers exchange distance vectors with each other. The distance vector for any one router is the current estimate of the shortest path distances from that router to the subnets in the AS.

In RIP, routing updates are exchanged between neighbours approximately every 30 seconds using a RIP response message.

The response message sent by a router or host contains a list of up to 25 destination subnets within the AS, as well as the sender's distance to each of those subnets. Response messages are also known as RIP advertisements.

Consider the portion of an AS shown in Figure 3.19. In this figure, lines connecting the routers denote subnets. Only selected routers (*A, B, C,* and *D*) and subnets (*w, x, y, and z) are labeled.* Dotted lines indicate that the AS continues on;

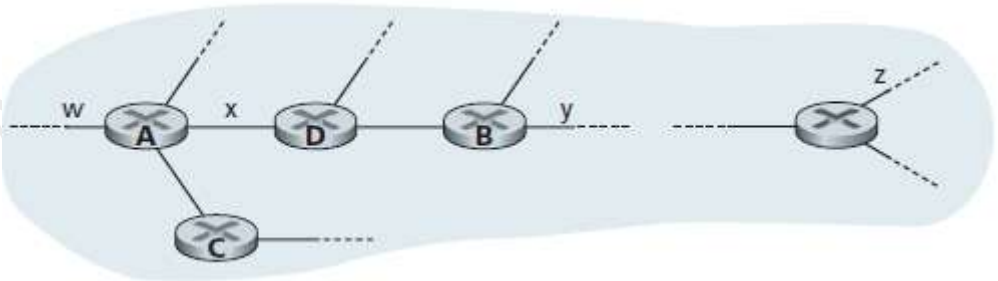


Fig 3.19: Portion of an autonomous systrem

Each router maintains a RIP table known as a routing table. A router's routing table includes both the router's distance vector and the router's forwarding table.

Figure below shows the routing table for router *D*.

The routing table has three columns :

- The first column is for the destination subnet.
- The second column indicates the identity of the next router along the shortest path to the destination subnet,
- The third column indicates the number of hops to get to the destination subnet along the shortest path.

For this example, the table indicates that to send a datagram from router *D* to destination subnet *w*, the datagram should first be forwarded to neighbouring router *A;* the table also indicates that destination subnet *w* is two hops away along the shortest path.

Similarly, the table indicates that subnet *z* is seven hops away via router *B*. A routing table will have one row for each subnet in the AS.

Advertisement from D :

Destination Subnet	Next Router	Number of Hops to Destination
w	A	2
y	B	2
z	B	7
x	—	1
....	....	....

Fig 3.20 Routing table in router D before receiving advertisement from router A

Suppose that 30 seconds later, router *D* receives from router *A* the advertisement shown in Figure below. This advertisement is the routing table information from router *A*!

This information indicates, in particular, that subnet *z* is only four hops away from router *A*. Router *D,* upon receiving this advertisement, merges the advertisement (Figure below) with the old routing table (Figure above).

In particular, router *D* learns that there is now a path through router *A* to subnet *z* that is shorter than the path through router *B*. Thus, router *D* updates its routing table to account for the shorter shortest path, as shown in Figure below.

Advertisement from A :

Destination Subnet	Next Router	Number of Hops to Destination
z	C	4
w	—	1
x	—	1
....	....	....

Fig 3.21: Advertisement from router A

RIP routers exchange advertisements approximately every 30 seconds. If a router does not hear from its neighbour at least once every 180 seconds, that neighbour is considered to be no longer reachable;

Destination Subnet	Next Router	Number of Hops to Destination
w	A	2
y	B	2
z	A	5
....	....	....

Fig 3.22: Routing table in router D after receiving advertisement from router A

RIP modifies the local routing table and then propagates this information by sending advertisements to its neighbouring routers (the ones that are still reachable). A router can also request information about its neighbour's cost to a given destination using RIP's request message. Routers send RIP request and response messages to each other over UDP using port number 520.

RIP uses a transport-layer protocol (UDP) on top of a network layer protocol (IP) to implement network-layer functionality (a routing algorithm).

Figure below shows RIP implementation in a UNIX system, for example, a UNIX workstation serving as a router. A process called *routed* executes RIP, that is, maintains routing information and exchanges messages with *routed* processes running in neighbouring routers.

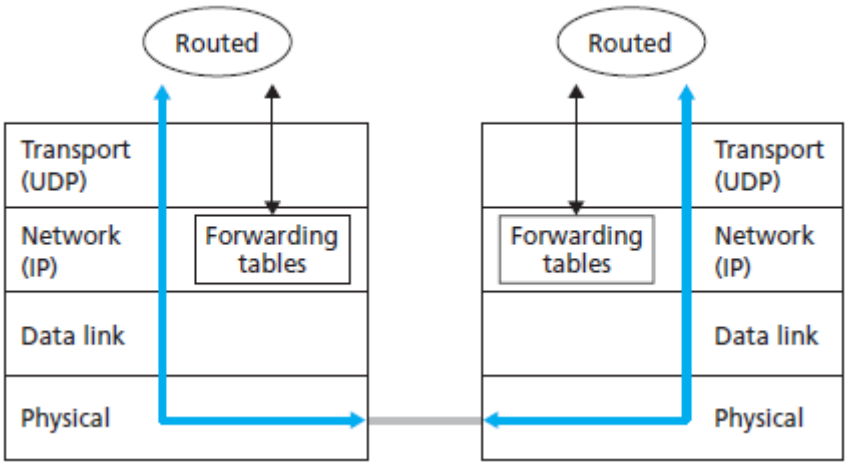


Fig 3.23 Implementation of RIP as routed daemon

3.4.2 Intra-AS Routing in the Internet: OSPF

OSPF routing is widely used for intra-AS routing in the Internet.

The Open in OSPF indicates that the routing protocol specification is publicly available.

The most recent version of OSPF,version 2.

OSPF is a link-state protocol that uses flooding of link-state information and a Dijkstra least-cost path algorithm.

With OSPF, a router constructs a complete topological map (that is, a graph) of the entire autonomous system. The router then locally runs Dijkstra's shortest-path algorithm to determine a shortest-path tree to all *subnets*, with itself as the root node.

Individual link costs are configured by the network administrator. The administrator might choose to set all link costs to 1, thus achieving minimum-hop routing, or might choose to set the link weights to be inversely proportional to link capacity.

With OSPF, a router broadcasts routing information to *all* other routers in the autonomous system, not just to its neighbouring routers.

A router broadcasts link state information whenever there is a change in a link's state. It also broadcasts a link's state periodically (at least once every 30 minutes), even if the link's state has not changed.

OSPF advertisements are contained in OSPF messages that are carried directly by IP

The OSPF protocol also checks that links are operational (via a HELLO message that is sent to an attached neighbour) and allows an OSPF router to obtain a neighbouring router's database of network-wide link state.

Some of the advances embodied in OSPF include the following:

• ***Security***. Exchanges between OSPF routers (for example, link-state updates) can be authenticated. With authentication, only trusted routers can participate in the OSPF protocol within an AS, thus preventing malicious intruders from injecting incorrect information into router tables.

By default, OSPF packets between routers are not authenticated and could be forged. Two types of authentication can be configured—simple and MD5.

Simple authentication: The same password is configured on each router. When a router sends an OSPF packet, it includes the password in plaintext.

MD5 authentication is based on shared secret keys that are configured in all the routers. For each OSPF packet that it sends, the router computes the MD5 hash of the content of the OSPF packet appended with the secret key.

Then the router includes the resulting hash value in the OSPF packet. The receiving router, using the preconfigured secret key, will compute an MD5 hash of the packet and compare it with the hash value that the packet carries, thus verifying the packet's authenticity. Sequence numbers are also used with MD5 authentication to protect against replay attacks.

Multiple same-cost paths : When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used.

• *Integrated support for unicast and multicast routing.* Multicast OSPF (MOSPF) provides extensions to OSPF to provide for multicast routing

MOSPF uses the existing OSPF link database and adds a new type of link-state advertisement to the existing OSPF link-state broadcast mechanism.

• ***Support for hierarchy within a single routing domain***. *T*he most significant advance in OSPF is the ability to structure an autonomous system hierarchically.

An OSPF autonomous system can be configured hierarchically into areas.

Each area runs its own OSPF link-state routing algorithm, with each router in an area broadcasting its link state to all other routers in that area.

Within each area, one or more area border routers are responsible for routing packets outside the area.

Lastly, exactly one OSPF area in the AS is configured to be the backbone area.

The primary role of the backbone area is to route traffic between the other areas in the AS. The backbone always contains all area border routers in the AS and may contain non border routers as well.

Inter-area routing within the AS requires that the packet be first routed to an area border router (intra-area routing), then routed through the backbone to the area border router that is in the destination area, and then routed to the final destination.

3.4.5 Inter-AS Routing: BGP

The Border Gateway Protocol version 4, is the standard inter-AS routing protocol. It is referred to as BGP4 or simply as BGP. As an inter-AS routing protocol BGP provides each AS a means to :

1. Obtain subnet reachability information from neighbouring ASs.
2. Propagate the reachability information to all routers internal to the AS.
3. Determine "good" routes to subnets based on the reachability information and on AS Policy.

BGP allows each subnet to advertise its existence to the rest of the Internet.

BGP Basics

In BGP, pairs of routers exchange routing information over semipermanent TCP connections using port 179. The semi-permanent TCP connections for the network in graph(refer fig 1 in hierarchical routing) are shown in Figure below.

There is one such BGP TCP connection for each link that directly connects two routers in two different ASs;

Thus, in Figure below, there is a TCP connection between gateway routers 3a and 1c and another TCP connection between gateway routers 1b and 2a. There are also semi permanent BGP TCP connections between routers within an AS.

Figure below displays a common configuration of one TCP connection for each pair of routers internal to an AS, creating a mesh of TCP connections within each AS.

For each TCP connection, the two routers at the end of the connection are called BGP peers, and the TCP connection along with all the BGP messages sent over the connection is called a BGP session.

Furthermore, a BGP session that spans two Ass is called an external BGP (eBGP) session, and a BGP session between routers in the same AS is called an internal BGP (iBGP) session. In Figure below, the eBGP sessions are shown with the long dashes; the iBGP sessions are shown with the short dashes.

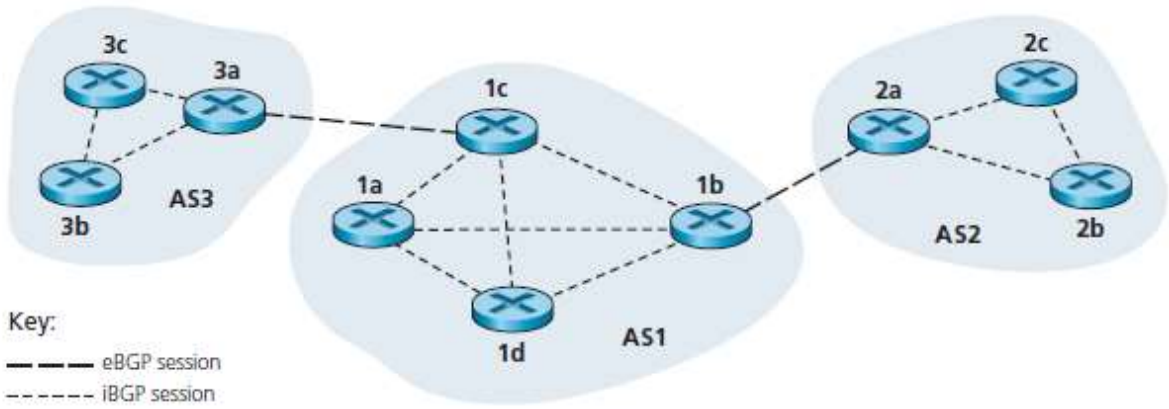


Fig 3.24 eBGP and iBGP sessions

BGP allows each AS to learn which destinations are reachable via its neighbouring ASs. In BGP, destinations are not hosts but instead are CIDRized prefixes, with each prefix representing a subnet or a collection of subnets.

Thus, for example, suppose there are four subnets attached to AS2: 138.16.64/24, 138.16.65/24, 138.16.66/24, and 138.16.67/24. Then AS2 could aggregate the prefixes for these four subnets and use BGP to advertise the single prefix to 138.16.64/22 to AS1.

Suppose that only the first three of those four subnets are in AS2 and the fourth subnet, 138.16.67/24, is in AS3.

Using the eBGP session between the gateway routers 3a and 1c, AS3 sends AS1 the list of prefixes that are reachable from AS3; and AS1 sends AS3 the list of prefixes that are reachable from AS1.

Similarly, AS1 and AS2 exchange prefix reachability information through their gateway routers 1b and 2a. When a gateway router (in any AS) receives eBGP-learned prefixes, the gateway router uses its iBGP sessions to distribute the prefixes to the other routers in the AS.

Thus, all the routers in AS1 learn about AS3 prefixes, including the gateway router 1b. The gateway router 1b (in AS1) can therefore re-advertise AS3's prefixes to AS2. When a router (gateway or not) learns about a new prefix, it creates an entry for the prefix in its forwarding table.

Path Attributes and BGP Routes

In BGP, an autonomous system is identified by its globally unique autonomous system number (ASN) .

When a router advertises a prefix across a BGP session, it includes with the prefix a number of BGP attributes.

Thus, BGP peers advertise routes to each other.

Two of the more important attributes are AS-PATH and NEXT-HOP:

• ***AS-PATH***. This attribute contains the ASs through which the advertisement for the prefix has passed. When a prefix is passed into an AS, the AS adds its ASN to the ASPATH attribute.

For example, consider Figure above and suppose that prefix 138.16.64/24 is first advertised from AS2 to AS1;

if AS1 then advertises the prefix to AS3, AS-PATH would be AS2 AS1. Routers use the AS-PATH attribute to detect and prevent looping advertisements;

Specifically, if a router sees that its AS is contained in the path list, it will reject the advertisement.

• **NEXT- HOP** : Providing the critical link between the inter-AS and intra-AS routing protocols, the NEXT-HOP attribute is of important use. *The NEXT-HOP is the router interface that begins the AS-PATH*.

Refer above Figure. Consider the gateway router 3a in AS3 when advertises a route to gateway router 1c in AS1 using eBGP. The route includes the advertised prefix, say *x*, and an AS-PATH to the prefix.

This advertisement also includes the NEXT-HOP, which is the IP address of the router 3a interface that leads to 1c.

Consider when router 1d learns about this route from iBGP.

After learning about this route to *x*, router 1d may want to forward packets to *x* along the route.

Router 1d may want to include the entry (*x*, *l*) in its forwarding table, where *l* is its interface that begins the least-cost path from 1d towards the gateway router 1c.

To determine *l*, 1d provides the IP address in the NEXT-HOP attribute to its intra-AS routing module.

Intra-AS routing algorithm has determined the least-cost path to all subnets attached to the routers in AS1, including to the subnet for the link between 1c and 3a.

From this least-cost path from 1d to the 1c-3a subnet, 1d determines its router interface *l* that begins this path and then adds the entry (*x*, *l*) to its forwarding table.

Thus, NEXT-HOP attribute is used by routers to configure their forwarding tables.

• Figure below illustrates another situation where the NEXT-HOP is needed. In this figure, AS1 and AS2 are connected by two peering links.

A router in AS1 could learn about two different routes to the same prefix *x*. These two routes could have the same AS-PATH to *x*, but could have different NEXT-HOP values corresponding to the different peering links.

Using the NEXT-HOP values and the intra-AS routing algorithm, the router can determine the cost of the path to each peering link, and then apply hot-potato routing to determine the appropriate interface.

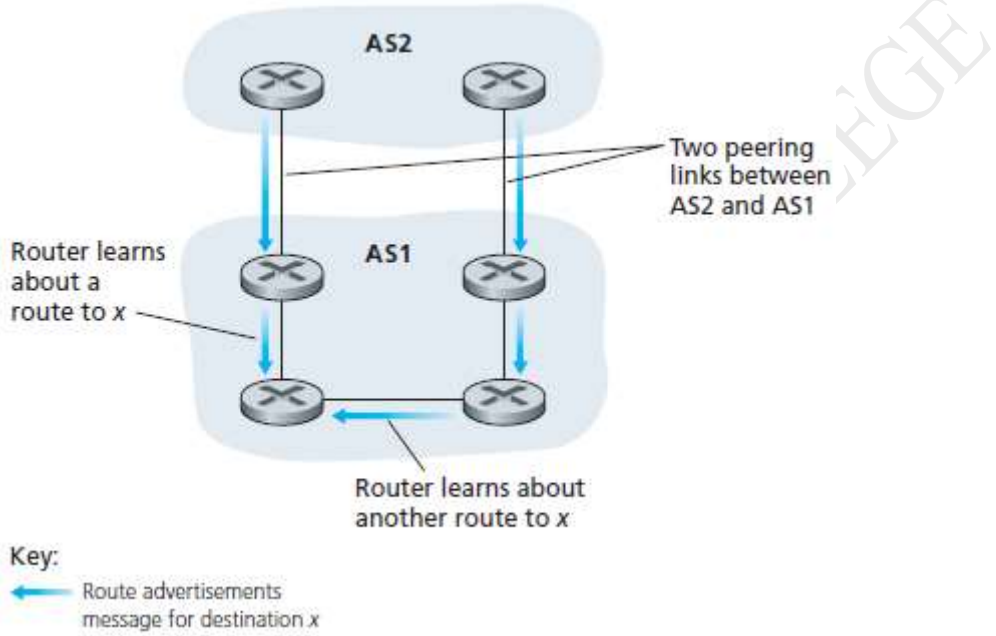


Figure 3.25: NEXT-HOP Attributes in advertisements are used to determine which peering link to use

BGP Route Selection

BGP uses eBGP and iBGP to distribute routes to all the routers within ASs.

From this distribution, a router may learn about more than one route to any one prefix, in which case the router must select one of the possible routes.

The input into this route selection process is the set of all routes that have been learned and accepted by the router.

If there are two or more routes to the same prefix, then BGP sequentially invokes the following elimination rules until one route remains:

• Routes are assigned a local preference value as one of their attributes. The local preference of a route could have been set by the router or could have been learned by another router in the same AS. The routes with the highest local preference values are selected.

• From the remaining routes (all with the same local preference value), the route with the shortest AS-PATH is selected. If this rule were the only rule for route selection, then BGP would be using a DV algorithm for path determination, where the distance metric uses the number of AS hops rather than the number of router hops.

• From the remaining routes (all with the same local preference value and the same AS-PATH length), the route with the closest NEXT-HOP router is selected. Here, closest means the router for which the cost of the least-cost path, determined by the intra-AS algorithm, is the smallest. This process is called hot-potato routing.

• If more than one route still remains, the router uses BGP identifiers to select the route;

Routing Policy

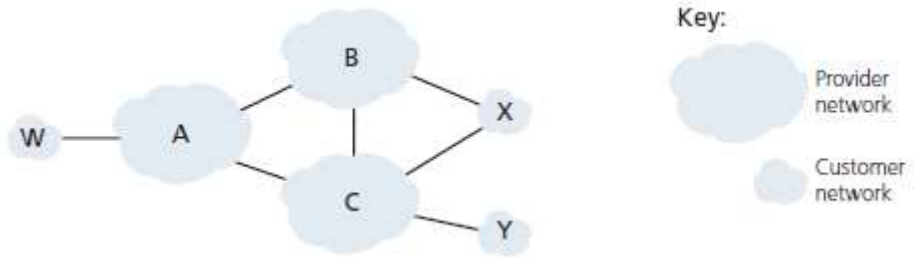


Fig 3.26 Simple BGP Scenario

Figure above shows six interconnected autonomous systems: A, B, C, W, X, and Y. It is important to note that A, B, C, W, X, and Y are ASs, not routers.

Assume that autonomous systems W, X, and Yare stub networks and that A, B, and C are backbone provider networks. Also assume that A, B, and C, all peer with each other, and provide full BGP information to their customer networks.

All traffic entering a stub network must be destined for that network, and all traffic leaving a stub network must have originated in that network. W and Y are clearly stub networks.

X is a multihomed stub network, since it is connected to the rest of the network via two different providers .

However, like Wand Y, X itself must be the source/destination of all traffic leaving/entering X.

In particular, X will function as a stub network if it advertises (to its neighbours B and C) that it has no paths to any other destinations except itself.

Even though X may know of a path, say XCY, that reaches network Y, it will *not* advertise this path to B.

Since B is unaware that X has a path to Y, B would never forward traffic destined to Y (or C) via X.

This simple example illustrates how a selective route advertisement policy can be used to implement customer/provider routing relationships.

Consider a provider network, say AS B. Suppose that B has learned (from A) that A has a path AW to W.

B can thus install the route BAW into its routing information base.

Clearly, B also wants to advertise the path BAW to its customer, X, so that X knows that it can route

65 to W via B.

But if B advertise the path BAW to C then C could route traffic to W via CBAW. If A, B, and C are all backbone providers, then B might rightly feel that it should not have to shoulder the burden (and cost!) of carrying transit traffic between A and C.

B might rightly feel that it is A's and C's job (and cost!) to make sure that C can route to/from A's customers via a direct connection between A and C.

3.5 Broadcast and Multicast Routing

In broadcast routing, the network layer provides a service of delivering a packet sent from a source node to all other nodes in the network; multicast routing enables a single source node to send a copy of a packet to a subset of the other network nodes.

3.5.1 Broadcast Routing Algorithms

Perhaps the most straightforward way to accomplish broadcast communication is for the sending node to send a separate copy of the packet to each destination, as shown in Figure 3.27 (a).

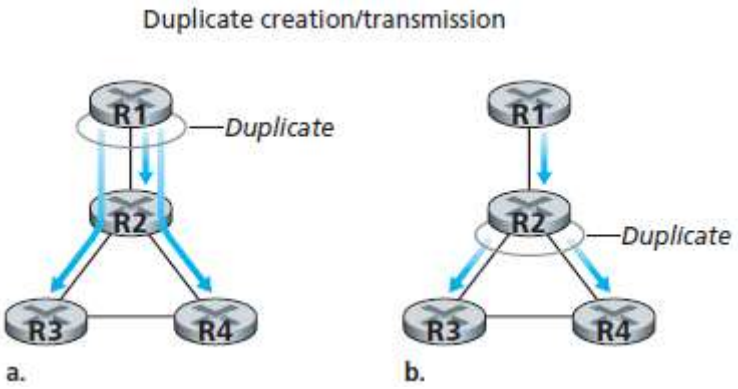


Fig 3.27: Source duplication v/s in network duplication

Given N destination nodes, the source node simply makes N copies of the packet, addresses each copy to a different destination, and then transmits the N copies to the N destinations using unicast routing.

- This N-way unicast approach to broadcasting is simple—no new network-layer routing protocol, packet-duplication, or forwarding functionality is needed.
- There are, however, several drawbacks to this approach. The first drawback is its inefficiency. If the source node is connected to the rest of the net work via a single link, then N separate copies of the (same) packet will traverse this single link.
- It would clearly be more efficient to send only a single copy of a packet over this first hop and then have the node at the other end of the first hop make and forward any additional needed copies. That is, it would be more efficient for the network nodes themselves (rather than just the source node) to create duplicate copies of a packet.
- For example, in Figure 3.27 (b), only a single copy of a packet traverses the R1-R2 link. That packet is then duplicated at R2, with a single copy being sent over links R2-R3 and R2-R4.
- An implicit assumption of N-way-unicast is that broadcast recipients, and their addresses, are known to the sender. But how is this information obtained? Most likely, additional protocol mechanisms (such as a broadcast membership or destination-registration protocol) would be required. This would add more overhead and, importantly, additional complexity to a protocol that had initially seemed quite simple.
- A final drawback of N-way-unicast relates to the purposes for which broadcast is to be used. Link-state routing protocols use broadcast to disseminate the link-state information that is used to compute unicast routes. Clearly, in situations where broadcast is used to create and update unicast routes, it would be unwise to rely on the unicast routing infrastructure to achieve broadcast.

Uncontrolled Flooding

The most noticeable technique for achieving broadcast is a flooding approach in which the source node sends a copy of the packet to all of its neighbours.

- When a node receives a broadcast packet, it duplicates the packet and forwards it to all of its neighbours (except the neighbour from which it received the packet).
- Clearly, if the graph is connected, this will eventually deliver a copy of the broadcast packet to all nodes in the graph.
- Although this scheme is simple and elegant, it has a fatal flaw.
    - If the graph has cycles, then one or more copies of each broadcast packet will cycle indefinitely. For example, in Figure 4.43, R2 will flood to R3, R3 will flood to R4, R4 will flood to R2, and R2 will flood (again!) to R3, and so on. This simple scenario

results in the endless cycling of two broadcast packets, one clockwise, and one counter clockwise.

- When a node is connected to more than two other nodes, it will create and forward multiple copies of the broadcast packet, each of which will create multiple copies of itself (at other nodes with more than two neighbours), and so on. This broadcast storm, resulting from the endless multiplication of broadcast packets, would eventually result in so many broadcast packets being created that the network would be rendered useless.

Controlled Flooding

The key to avoiding a broadcast storm is for a node to judiciously choose when to flood a packet and (e.g., if it has already received and flooded an earlier copy of a packet) when not to flood a packet.

This can be done in one of several ways.

- In sequence-number-controlled flooding, a source node puts its address (or other unique identifier) as well as a broadcast sequence number into a broadcast packet, then sends the packet to all of its neighbours.
  - Each node maintains a list of the source address and sequence number of each broadcast packet it has already received, duplicated, and forwarded.
  - When a node receives a broadcast packet, it first checks whether the packet is in this list.
  - If so, the packet is dropped; if not, the packet is duplicated and forwarded to all the node's neighbours
  - The Gnutella protocol, uses sequence-number-controlled flooding to broadcast queries in its overlay network.
- A second approach to controlled flooding is known as reverse path forwarding (RPF) [Dalal 1978], also sometimes referred to as reverse path broadcast (RPB).
  - When a router receives a broadcast packet with a given source address, it transmits the packet on all of its outgoing links (except the one on which it was received) only if the packet arrived on the link that is on its own shortest unicast path back to the source.
  - Otherwise, the router simply discards the incoming packet without forwarding it on any of its outgoing links.

- Such a packet can be dropped because the router knows it either will receive or has already received a copy of this packet on the link that is on its own shortest path back to the sender.
- RPF need only know the next neighbour on its unicast shortest path to the sender; it uses this neighbour's identity only to determine whether or not to flood a received broadcast packet.
    - Figure below illustrates RPF.

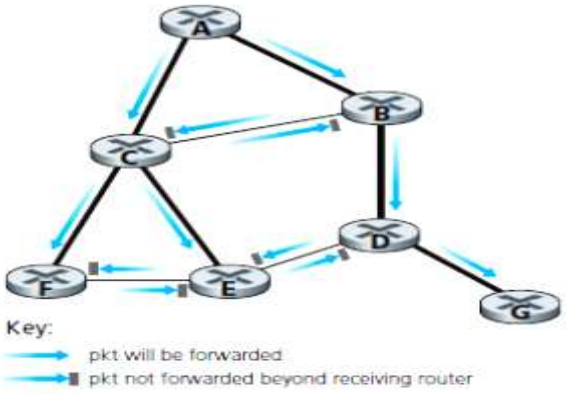


Fig 3.28: Reverse path forwarding

- Suppose that the links drawn with thick lines represent the least-cost paths from the receivers to the source (A).
- Node A initially broadcasts a source-A packet to nodes C and B.
- Node B will forward the source-A packet it has received from A (since A is on its least-cost path to A) to both C and D.
- B will ignore (drop, without forwarding) any source-A packets it receives from any other nodes (for example, from routers C or D).
- Consider node C, which will receive a source-A packet directly from A as well as from B.
- Since B is not on C's own shortest path back to A, C will ignore any source-A packets it receives from B.
- On the other hand, when C receives a source-A packet directly from A, it will forward the packet to nodes B, E, and F.

Spanning-Tree Broadcast

While sequence-number-controlled flooding and RPF avoid broadcast storms, they do not completely avoid the transmission of redundant broadcast packets. For example, in Figure, nodes B, C, D, E, and F receive either one or two redundant packets.

Ideally, every node should receive only one copy of the broadcast packet. Examining the tree consisting of the nodes connected by thick lines in Figure 3.29, if broadcast packets were forwarded only along links within this tree, each and every network node would receive exactly one copy of the broadcast packet. This tree is an example of a spanning tree—a tree that contains each and every node in a graph.

More formally, a spanning tree of a graph G = (N,E) is a graph G` = (N,E`) such that E` is a subset of E, G` is connected, G` contains no cycles, and G` contains all the original nodes in G.

- If each link has an associated cost and the cost of a tree is the sum of the link costs, then a spanning tree whose cost is the minimum of all of the graph's spanning trees is called a minimum spanning tree.

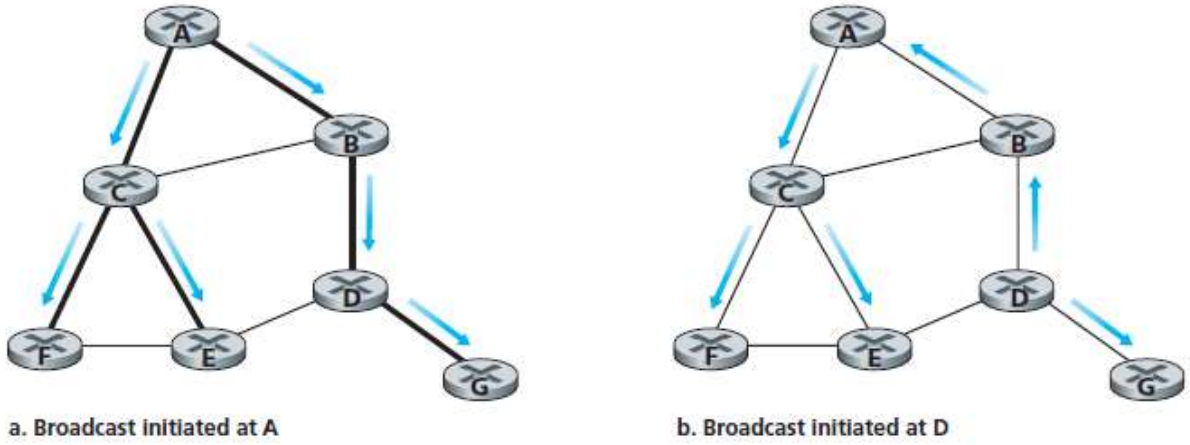


Fig 3.29: Broadcast along a spanning tree

- Thus, another approach to providing broadcast is for the network nodes to first construct a spanning tree. When a source node wants to send a broadcast packet, it sends the packet out on all of the incident links that belong to the spanning tree.
- A node receiving a broadcast packet then forwards the packet to all its neighbours in the spanning tree (except the neighbour from which it received the packet). Not only does spanning tree eliminate redundant broadcast packets, but once in place, the spanning tree can be used by any node to begin a broadcast, as shown in Figures 3.30 (a) and 3.30 (b).
- Note that a node need not be aware of the entire tree; it simply needs to know which of its neighbours in G are spanning-tree neighbours.

- The main complexity associated with the spanning-tree approach is the creation and maintenance of the spanning tree.
- Numerous distributed spanning-tree algorithms have been developed [Gallager 1983, Gartner 2003].
    - In the center-based approach to building a spanning tree, a center node (also known as a rendezvous point or a core) is defined.
    - Nodes then unicast tree-join messages addressed to the center node.
    - A tree-join message is forwarded using unicast routing toward the center until it either arrives at a node that already belongs to the spanning tree or arrives at the center.
    - In either case, the path that the tree-join message has followed defines the branch of the spanning tree between the edge node that initiated the tree-join message and the center.

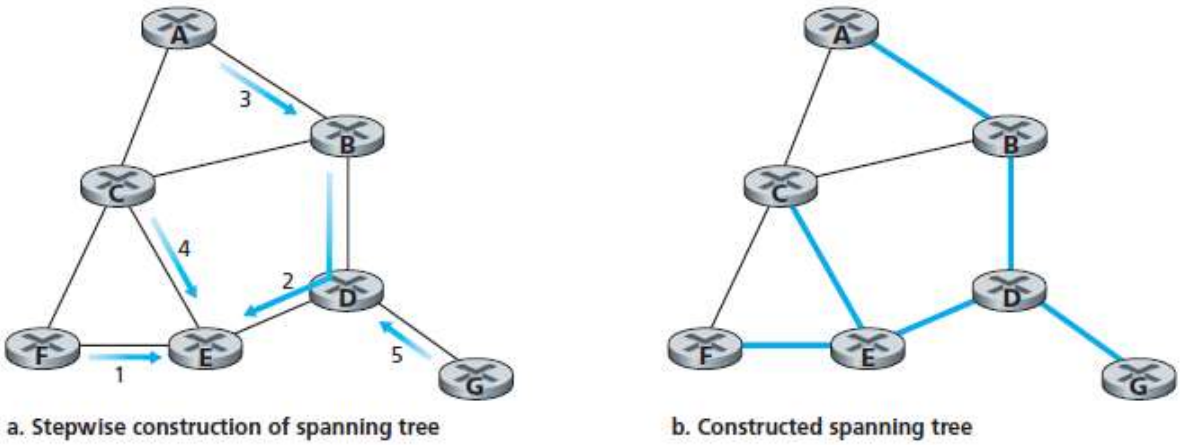


Figure 3.30: illustrates the construction of a center-based spanning tree.

- Suppose that node E is selected as the center of the tree. Suppose that node F first joins the tree and forwards a tree-join message to E.
- The single link EF becomes the initial spanning tree. Node B then joins the spanning tree by sending its tree-join message to E.
- Suppose that the unicast path route to E from B is via D. In this case, the tree-join message results in the path BDE being grafted onto the spanning tree.
- Node A next joins the spanning group by forwarding its tree-join message towards E. If A's unicast path to E is through B, then since B has already joined the spanning tree, the arrival of A's tree-join message at B will result in the AB link being immediately grafted onto the spanning tree.

- Node C joins the spanning tree next by forwarding its tree-join message directly to E. Finally, because the unicast routing from G to E must be via node D, when G sends its tree-join message to E, the GD link is grafted onto the spanning tree at node D.

3.5.2 Multicast

Multicast service is where a multicast packet is delivered to only a subset of network nodes.

- A number of emerging network applications require the delivery of packets from one or more senders to a group of receivers.
- These applications include
  - bulk data transfer (for example, the transfer of a software upgrade from the software developer to users needing the upgrade),
  - streaming continuous media (for example, the transfer of the audio, video, and text of a live lecture to a set of distributed lecture participants),
  - shared data applications (for example, a whiteboard or teleconferencing application that is shared among many distributed participants),
  - data feeds (for example, stock quotes),
  - Web cache updating, and interactive gaming (for example, distributed interactive virtual environments or multiplayer games).
- In multicast communication, we are immediately faced with two problems— how to identify the receivers of a multicast packet and how to address a packet sent to these receivers.

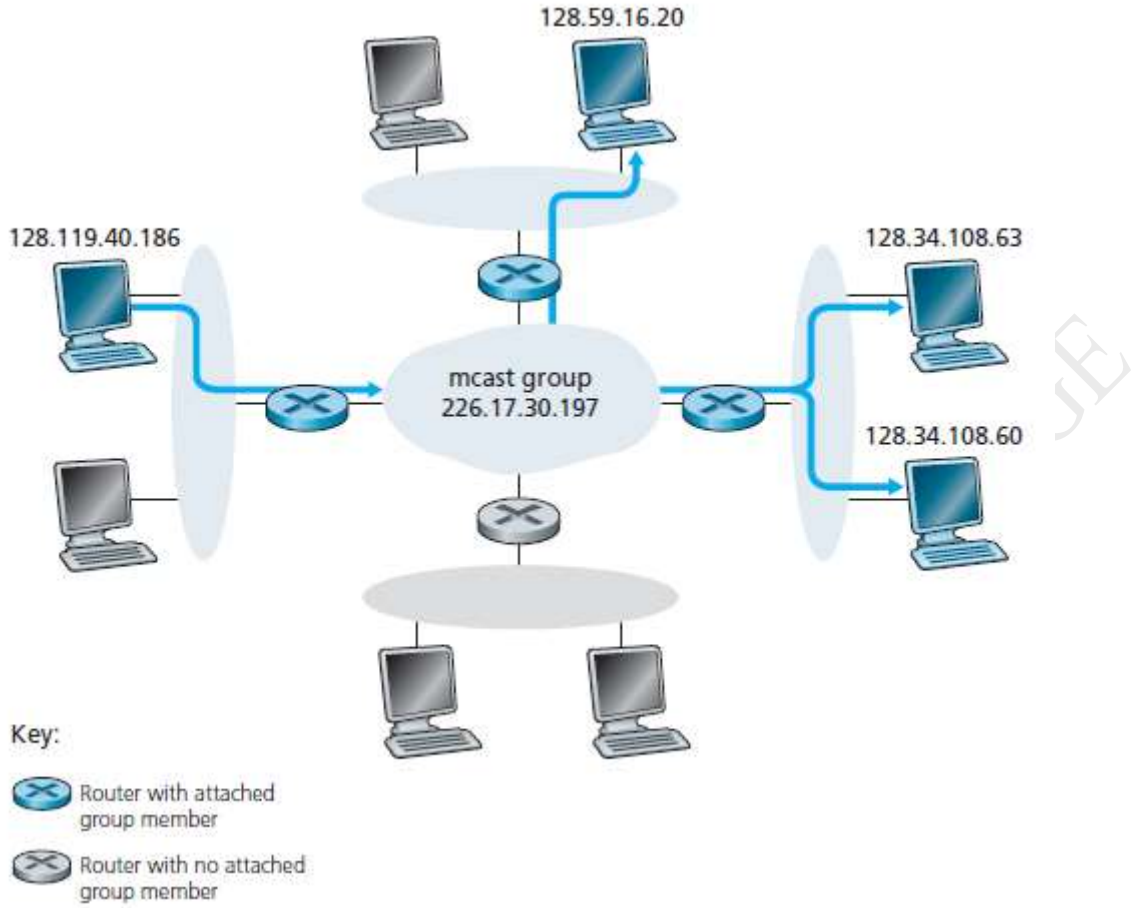


Fig 3.31 The multicast group: A datagram addressed to the group is delivered to all members of the multicast group

- In the case of unicast communication, the IP address of the receiver (destination) is carried in each IP unicast datagram and identifies the single recipient; in the case of broadcast, all nodes need to receive the broadcast packet, so no destination addresses are needed. But in the case of multicast, we now have multiple receivers.
- Explicit identification of the receivers by the sender also requires that the sender know the identities and addresses of all of the receivers
- A multicast packet is addressed using address indirection. That is, a single identifier is used for the group of receivers, and a copy of the packet that is addressed to the group using this single identifier is delivered to all of the multicast receivers associated with that group.
- In the Internet, the single identifier that represents a group of receivers is a class D multicast IP address. The group of receivers associated with a class D address is referred to as a multicast group.
- The multicast group abstraction is illustrated in Figure 3.31. Here, four hosts (shown in shaded color) are associated with the multicast group address of 226.17.30.197 and will

receive all datagrams addressed to that multicast address. The difficulty that we must still address is the fact that each host has a unique IP unicast address that is completely independent of the address of the multicast group in which it is participating.

Internet Group Management Protocol

The IGMP protocol version 3 [RFC 3376] operates between a host and its directly attached router as shown in Figure below.

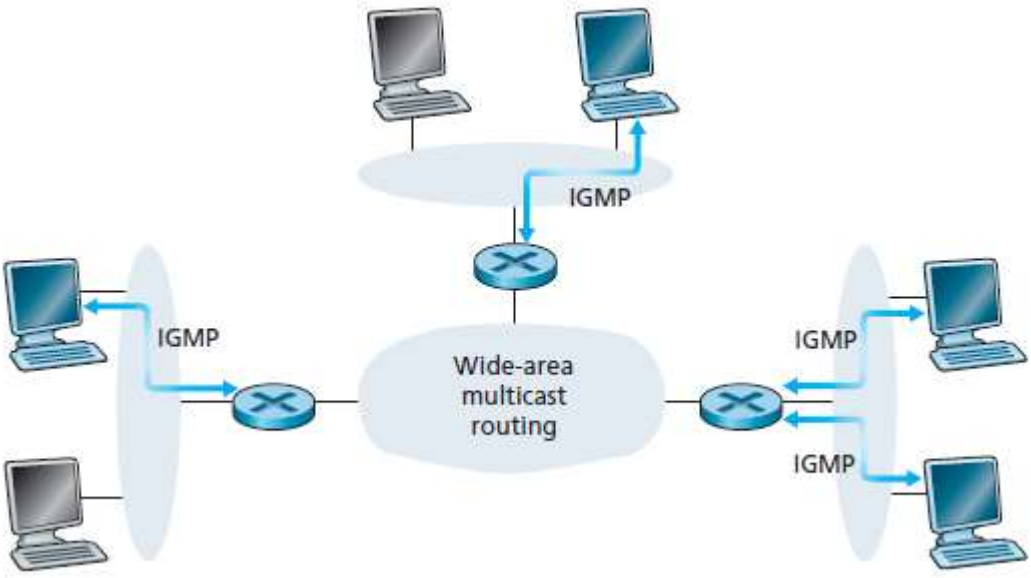


Figure 3.32: The two components of network-layer multicast in the Internet: IGMP and multicast routing protocols

Figure shows three first-hop multicast routers, each connected to its attached hosts via one outgoing local interface. This local interface is attached to a LAN in this example, and while each LAN has multiple attached hosts, at most a few of these hosts will typically belong to a given multicast group at any given time.

- IGMP provides the means for a host to inform its attached router that an application running on the host wants to join a specific multicast group.
- Given that the scope of IGMP interaction is limited to a host and its attached router, another protocol is clearly required to coordinate the multicast routers (including the attached routers) throughout the Internet, so that multicast datagrams are routed to their final destinations.
- IGMP has only three message types. Like ICMP, IGMP messages are carried (encapsulated) within an IP datagram, with an IP protocol number of 2.

- The membership query message is sent by a router to all hosts on an attached interface (for example, to all hosts on a local area network) to determine the set of all multicast groups that have been joined by the hosts on that interface.
- Hosts respond to a membership_query message with an IGMP membership_report message. membership_report messages can also be generated by a host when an application first joins a multicast group without waiting for a membership_query message from the router.
- The final type of IGMP message is the leave_group message. This message is optional.

Multicast Routing Algorithms

The multicast routing problem is illustrated in Figure below.

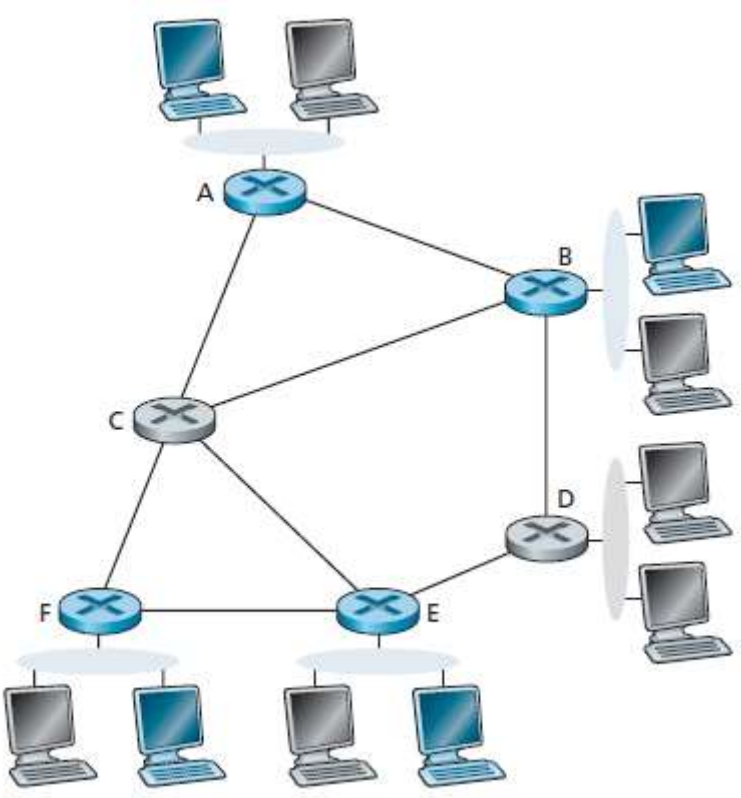


Fig 3.33: Multicast hosts , their attached routers and other routers

Hosts joined to the multicast group are shaded in color; their immediately attached router is also shaded in color.

- As shown in Figure above, only a subset of routers (those with attached hosts that are joined to the multicast group) actually needs to receive the multicast traffic.
- In Figure 3.33, only routers A, B, E, and F need to receive the multicast traffic.
- Since none of the hosts attached to router D are joined to the multicast group and since router C has no attached hosts, neither C nor D needs to receive the multicast group traffic.
- The goal of multicast routing, then, is to find a tree of links that connects all of the routers that have attached hosts belonging to the multicast group. Multicast packets will then be routed along this tree from the sender to all of the hosts belonging to the multicast tree.
- The tree may contain routers that do not have attached hosts belonging to the multicast group.

Two approaches have been adopted for determining the multicast routing tree.

- **Multicast routing using a group-shared tree.**
  - As in the case of spanning-tree broadcast, multicast routing over a group-shared tree is based on building a tree that includes all edge routers with attached hosts belonging to the multicast group.

- In practice, a center-based approach is used to construct the multicast routing tree, with edge routers with attached hosts belonging to the multicast group sending (via unicast) join messages addressed to the center node.
- As in the broadcast case, a join message is forwarded using unicast routing toward the center until it either arrives at a router that already belongs to the multicast tree or arrives at the center.
- All routers along the path that the join message follows will then forward received multicast packets to the edge router that initiated the multicast join.
- Acritical question for center-based tree multicast routing is the process used to select the center.

- **Multicast routing using a source-based tree.**
  - While group-shared tree multicast routing constructs a single, shared routing tree to route packets from all senders, the second approach constructs a multicast routing tree for each source in the multicast group.
  - In practice, an RPF algorithm (with source node x) is used to construct a multicast forwarding tree for multicast datagrams originating at source x.
  - The RPF broadcast algorithm we studied earlier requires a bit of tweaking for use in multicast.
  - Consider router D in Figure 4.50. Under broadcast RPF, it would forward packets to router G, even though router G has no attached hosts that are joined to the multicast group. While this is not so bad for this case where D has only a single downstream router, G, imagine what would happen if there were thousands of routers downstream from D! Each of these thousands of routers would receive unwanted multicast packets.
  - The solution to the problem of receiving unwanted multicast packets under RPF is known as pruning. A multicast router that receives multicast packets and has no attached hosts joined to that group will send a prune message to its upstream router. If a router receives prune messages from each of its downstream routers, then it can forward a prune message upstream.

Multicast Routing in the Internet

The first multicast routing protocol used in the Internet was the Distance-Vector Multicast Routing Protocol (DVMRP).

- DVMRP implements source-based trees with reverse path forwarding and pruning.
- DVMRP uses an RPF algorithm with pruning, as discussed above.
- Most widely used Internet multicast routing protocol is the Protocol-Independent Multicast (PIM) routing protocol, which explicitly recognizes two multicast distribution scenarios.
- In dense mode [RFC 3973], multicast group members are densely located; that is, many or most of the routers in the area need to be involved in routing multicast datagrams. PIM dense mode is a flood-and-prune reverse path forwarding technique similar in spirit to DVMRP.
- In sparse mode [RFC 4601], the number of routers with attached group members is small with respect to the total number of routers; group members are widely dispersed. PIM sparse mode uses rendezvous points to set up the multicast distribution tree.
- In source-specific multicast (SSM) [RFC 3569, RFC 4607], only a single sender is allowed to send traffic into the multicast tree, considerably simplifying tree construction and maintenance.