Unit 3: Network Layer

IPV4 ADDRESSES:

An IPv4 address is a 32-bit address that uniquely and universally defines the connection of a host or a router to the Internet. IPv4 addresses are unique in the sense that each address defines one, and only one, connection to the Internet.
IPv4 addresses are universal in the sense that the addressing system must be accepted by any host that wants to be connected to the Internet.

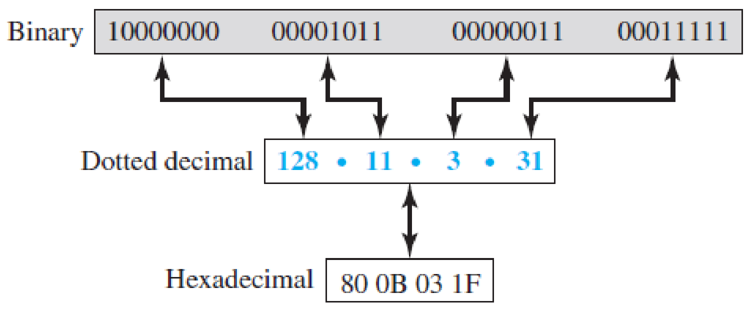
Address Space

An **address space** is the total number of addresses used by the protocol. If a protocol uses bits to define an address, the address space is 2 because each bit can have two different values (0 or 1). IPv4 uses 32-bit addresses, which means that the address space is 232 or 4,294,967,296 (more than four billion). If there were no restrictions, more than 4 billion devices could be connected to the Internet.

Notation

There are three common notations to show an IPv4 address: binary notation (base 2), dotted-decimal notation (base 256), and hexadecimal notation (base 16).

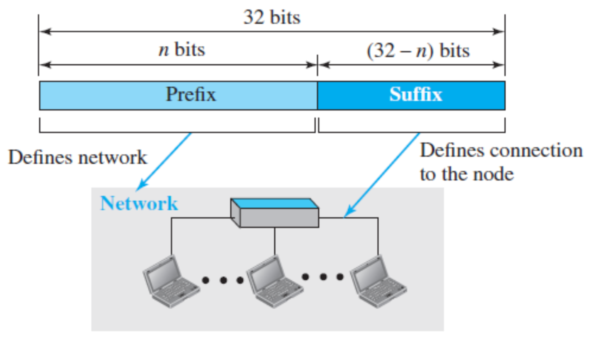
Figure 18.16 *Three different notations in IPv4 addressing*



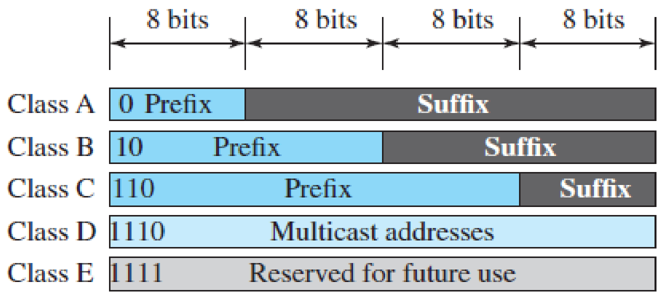
Hierarchy in Addressing

A 32-bit IPv4 address is also hierarchical, but divided only into two parts. The first part of the address, called the , defines the network; the second part of the address, called the , defines the node (connection of a device to the Internet). The prefix length is bits and the suffix length is (32-) bits.

Figure 18.17 *Hierarchy in addressing*



Classful Addressing:



Class	Prefixes	First byte
A	n = 8 bits	0 to 127
B	n = 16 bits	128 to 191
C	n = 24 bits	192 to 223
D	Not applicable	224 to 239
E	Not applicable	240 to 255

Address Depletion:

The reason that classful addressing has become obsolete is address depletion. To understand the problem, let us think about class A. This class can be assigned to only 128 organizations in the world, but each organization needs to have a single network with 16,777,216 nodes. Class B addresses were designed for midsize organizations, but many of the addresses in this class also remained unused. Class C addresses have a completely different flaw in design. The number of addresses that can be used in each network (256) was so small that most companies were not comfortable using a block in this address class.

Subnetting and Supernetting

In subnetting, a class A or class B block is divided into several subnets. Each subnet has a larger prefix length than the original network. For example, if a network in class A is divided into four subnets, each subnet has a prefix of n_{sub} = 10. At the same time, if all of the addresses in a network are not used, subnetting allows the addresses to be divided among several organizations.

While subnetting was devised to divide a large block into smaller ones, supernetting was devised to combine several class C blocks into a larger block to be attractive to organizations that need more than the 256 addresses available in a class C block. This idea did not work either because it makes the routing of packets more difficult.

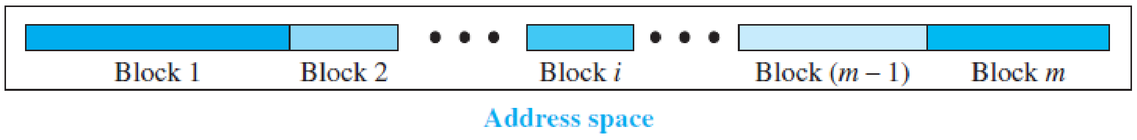
Advantage of Classful Addressing

Given an address, we can easily find the class of the address and, since the prefix length for each class is fixed, we can find the prefix length immediately. In other words, the prefix length in classful addressing is inherent in the address; no extra information is needed to extract the prefix and the suffix.

Classless Addressing:

In 1996, the Internet authorities announced a new architecture called **classless addressing.** In classless addressing, variable-length blocks are used that belong to no classes. We can have a block of 1 address, 2 addresses, 4 addresses, 128 addresses, and so on. In classless addressing, the whole address space is divided into variable length blocks. The prefix in an address defines the block (network); the suffix defines the node (device). Theoretically, we can have a block of 2^0, 2^1, 2^2, . . . , 2^{32} addresses.

Figure 18.19 *Variable-length blocks in classless addressing*

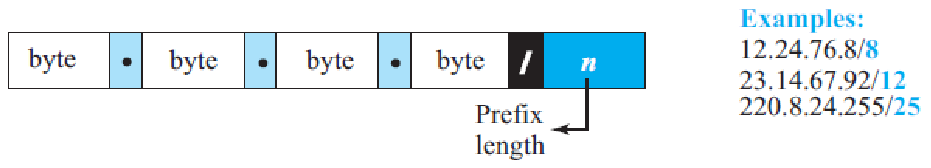


Unlike classful addressing, the prefix length in classless addressing is variable. We can have a prefix length that ranges from 0 to 32. The size of the network is inversely proportional to the length of the prefix. A small prefix means a larger network; a large prefix means a smaller network. the idea of classless addressing can be easily applied to classful addressing. An address in class A can be thought of as a classless address in which the prefix length is 8. An address in class B can be thought of as a classless address in which the prefix is 16, and so on. In other words, classful addressing is a special case of classless addressing.

Prefix Length: Slash Notation:

In this case, the prefix length, , is added to the address, separated by a slash. The notation is informally referred to as and formally as ***classless interdomain routing*** or ***CIDR*** (pronounced cider) strategy.

Figure 18.20 *Slash notation (CIDR)*



Extracting Information from an Address
Three pieces of information about the block to which the address belongs: the number of addresses, the first address in the block, and the last address.

1. The number of addresses in the block is found as = 2^{32-} .
2. To find the first address, we keep the leftmost bits and set the (32 −) rightmost bits all to 0s.
3. To find the last address, we keep the leftmost bits and set the (32 −) rightmost bits all to 1s.

Subnetting

An organization (or an ISP) that is granted a range of addresses may divide the range into several subranges and assign each subrange to a subnetwork (or subnet). Note that nothing stops the organization from creating more levels. A subnetwork can be divided into several sub-subnetworks. A sub-subnetwork can be divided into several sub-sub-subnetworks, and so on.

Designing Subnets
We assume the total number of addresses granted to the organization is N, the prefix length is , the assigned number of addresses to each subnetwork is N_{sub}, and the prefix length for each subnetwork is $_{sub}$. Then the following steps need to be carefully followed to guarantee the proper operation of the subnetworks.

❑ The number of addresses in each subnetwork should be a power of 2.
❑ The prefix length for each subnetwork should be found using the following formula:
first address = (prefix in decimal) × 2^{32-} = (prefix in decimal) × .

$$_{sub} = 32 - \log_2 N_{sub}$$

❑ The starting address in each subnetwork should be divisible by the number of addresses in that subnetwork. This can be achieved if we first assign addresses to larger subnetworks.

Example
An organization is granted a block of addresses with the beginning address 14.24.74.0/24. The organization needs to have 3 subblocks of addresses to use in its three subnets: one subblock of 10 addresses, one subblock of 60 addresses, and one subblock of 120 addresses. Design the subblocks.

Solution

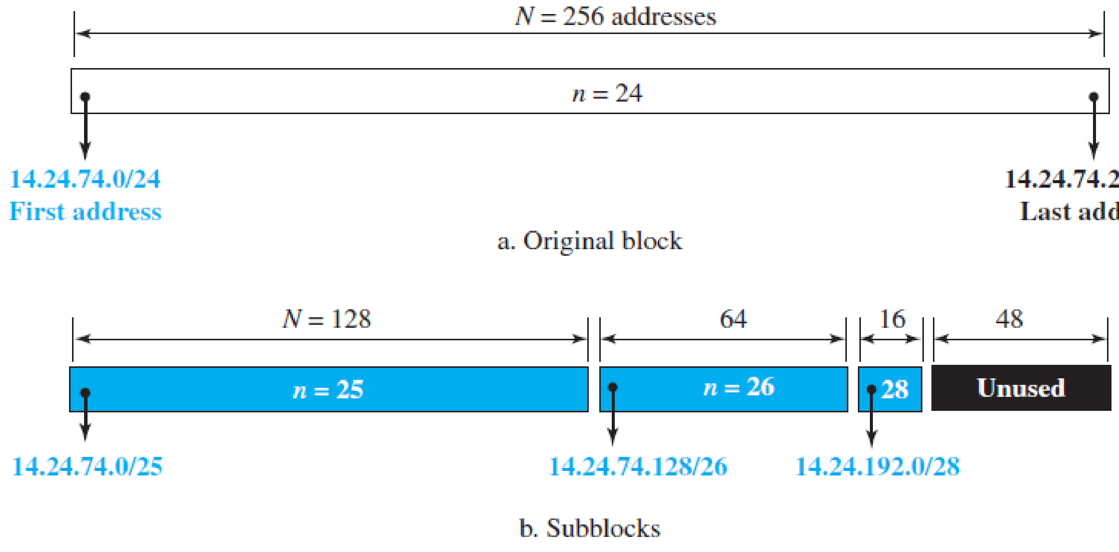
There are $2^{32-24} = 256$ addresses in this block. The first address is 14.24.74.0/24; the last address is 14.24.74.255/24. To satisfy the third requirement, we assign addresses to subblocks, starting with the largest and ending with the smallest one.

a. The number of addresses in the largest subblock, which requires 120 addresses, is not a power of 2. We allocate 128 addresses. The subnet mask for this subnet can be found as $n_1 = 32 - \log_2 128 = 25$. The first address in this block is 14.24.74.0/25; the last address is 14.24.74.127/25.

b. The number of addresses in the second largest subblock, which requires 60 addresses, is not a power of 2 either. We allocate 64 addresses. The subnet mask for this subnet can be found as $n_2 = 32 - \log_2 64 = 26$. The first address in this block is 14.24.74.128/26; the last address is 14.24.74.191/26.

c. The number of addresses in the smallest subblock, which requires 10 addresses, is not a power of 2 either. We allocate 16 addresses. The subnet mask for this subnet can be found as $n_3 = 32 - \log_2 16 = 28$. The first address in this block is 14.24.74.192/28; the last address is 14.24.74.207/28.

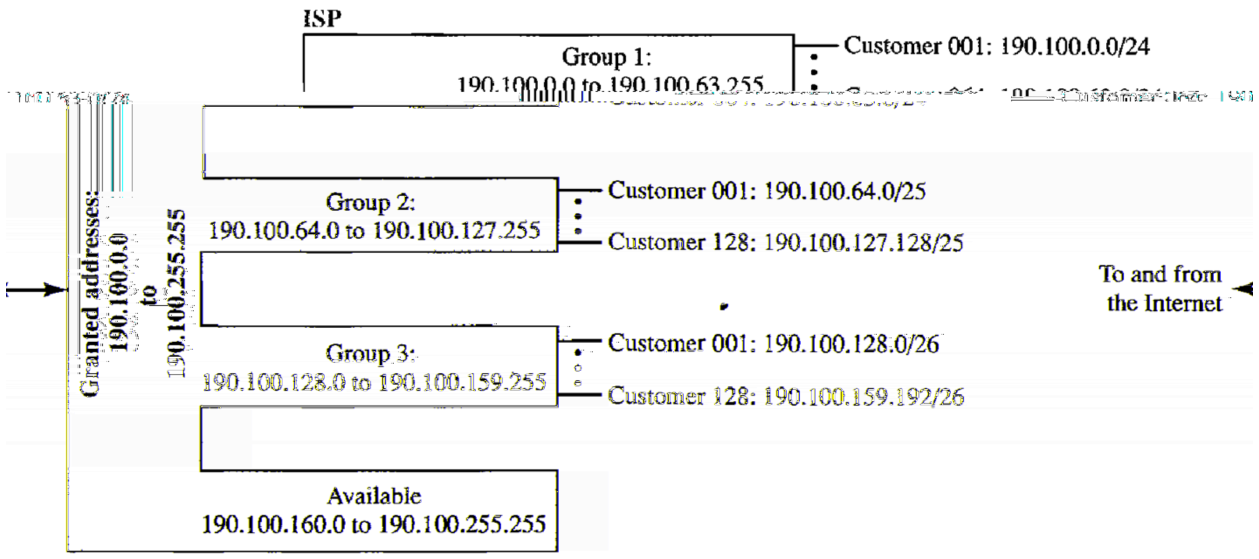
Figure 18.23 *Solution to Example 18.5*



Question: An ISP is granted a block of addresses starting with 190.100.0.0/16 (65,536 addresses). The ISP needs to distribute these addresses to three groups of customers as follows:

a. The first group has 64 customers; each needs 256 addresses.
b. The second group has 128 customers; each needs 128 addresses.
c. The third group has 128 customers; each needs 64 addresses.
Design the subblocks and find out how many addresses are still available after these allocations.

Figure 19.9 *An example of address allocation and distribution by an ISP*


ISP
Group 1:
190.100.0.0 to 190.100.63.255
Customer 001: 190.100.0.0/24
Group 2:
190.100.64.0 to 190.100.127.255
Customer 001: 190.100.64.0/25
Customer 128: 190.100.127.128/25
Group 3:
190.100.128.0 to 190.100.159.255
Customer 001: 190.100.128.0/26
Customer 128: 190.100.159.192/26
Available
190.100.160.0 to 190.100.255.255
Granted addresses:
190.100.0.0
to
190.100.255.255
To and from
the Internet

1. **Group 1**

   For this group, each customer needs 256 addresses. This means that 8 ($\log_2 256$) bits are needed to define each host. The prefix length is then $32 - 8 = 24$. The addresses are

1st Customer:	*190.100.0.0/24*	*190.100.0.255/24*
2nd Customer:	*190.100.1.0/24*	*190.100.1.255/24*
. . .		
64th Customer:	*190.100.63.0/24*	*190.100.63.255/24*

Total $= 64 \times 256 = 16{,}384$

2. **Group 2**

   For this group, each customer needs 128 addresses. This means that 7 ($\log_2 128$) bits are needed to define each host. The prefix length is then $32 - 7 = 25$. The addresses are

1st Customer:	*190.100.64.0/25*	*190.100.64.127/25*
2nd Customer:	*190.100.64.128/25*	*190.100.64.255/25*
. . .		
128th Customer:	*190.100.127.128/25*	*190.100.127.255/25*

Total $= 128 \times 128 = 16{,}384$

3. **Group 3**

   For this group, each customer needs 64 addresses. This means that 6 ($\log_2 64$) bits are needed to each host. The prefix length is then $32 - 6 = 26$. The addresses are

1st Customer:	*190.100.128.0/26*	*190.100.128.63/26*
2nd Customer:	*190.100.128.64/26*	*190.100.128.127/26*
. . .		
128th Customer:	*190.100.159.192/26*	*190.100.159.255/26*

Total $= 128 \times 64 = 8192$

Number of granted addresses to the ISP: 65,536
Number of allocated addresses by the ISP: 40,960
Number of available addresses: 24,576

Network Address Translation (NAT)

A technology that can provide the mapping between the private and universal addresses, and at the same time support virtual private networks, is **Network Address Translation (NAT).** The technology allows a site to use a set of private addresses for internal communication and a set of global Internet addresses (at least one) for communication with the rest of the world.

Figure 18.29 *NAT*

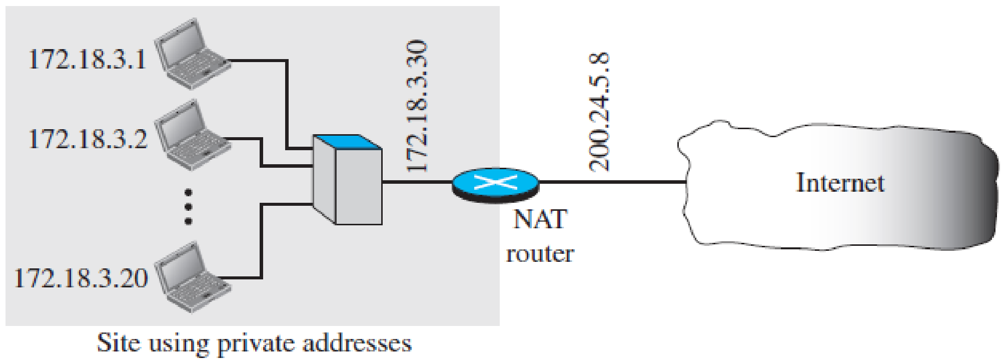
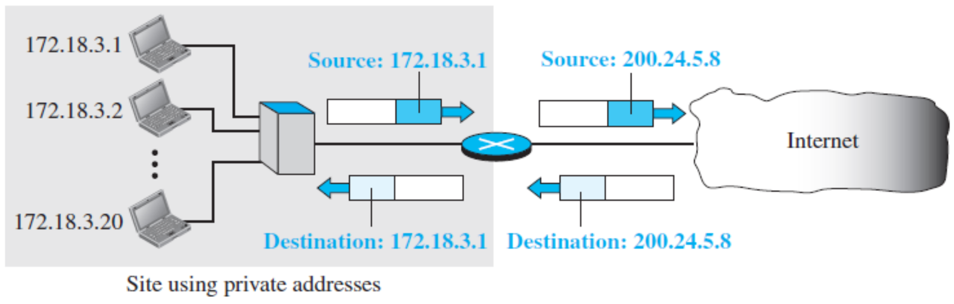


Figure 18.30 *Address translation*



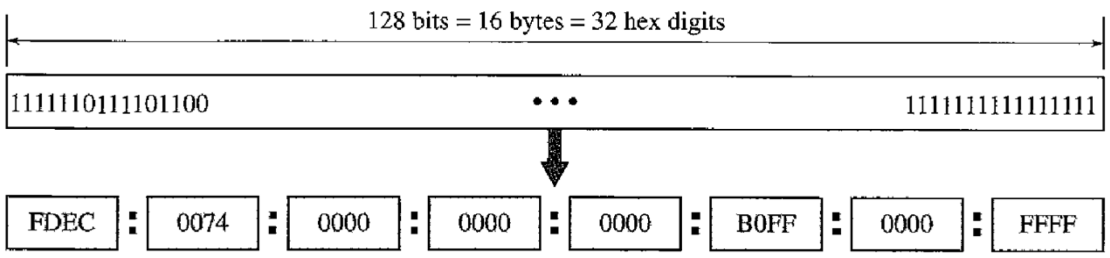
IPv6 ADDRESSING:

The main reason for migration from IPv4 to IPv6 is the small size of the address space in IPv4. An IPv6 address is 128 bits or 16 bytes (octets) long, four times the address length in IPv4.

Representation

The following shows two of these notations: binary and colon hexadecimal.

Figure 19.14 *IPv6 address in binary and hexadecimal colon notation*

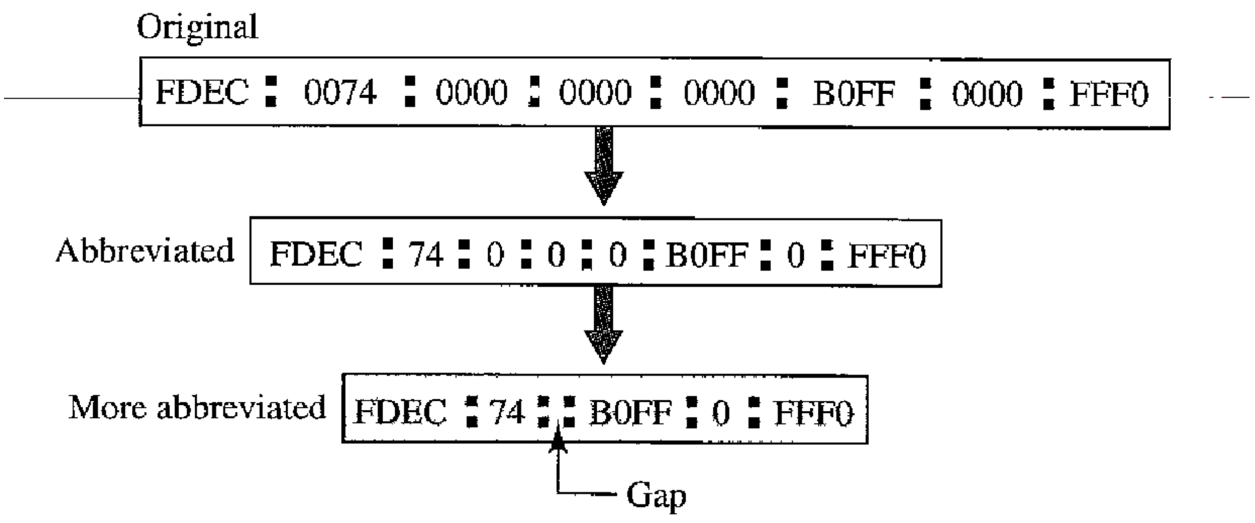


128 bits = 16 bytes = 32 hex digits
1111110111101100
• • •
1111111111111111
FDEC : 0074 : 0000 : 0000 : 0000 : B0FF : 0000 : FFFF

Binary (128 bits)	1111111011110110	...	1111111100000000
Colon Hexadecimal	FEF6:BA98:7654:3210:ADEF:BBFF:2922:FF00		

Abbreviation
Although the IP address, even in hexadecimal format, is very long, many of the digits are zeros. In this case, we can abbreviate the address. The leading zeros of a section (four digits between two colons) can be omitted. Only the leading zeros can be dropped, not the trailing zeros.

Abbreviated IPv6 addresses


Original
FDEC : 0074 : 0000 : 0000 : 0000 : B0FF : 0000 : FFF0
Abbreviated
FDEC : 74 : 0 : 0 : 0 : B0FF : 0 : FFF0
More abbreviated
FDEC : 74 :: B0FF : 0 : FFF0
Gap

FDEC:0:0:0:0:BBFF:0:FFFF $\longrightarrow$ FDEC::BBFF:0:FFFF

Congestion Control

Congestion control refers to techniques and mechanisms that can either prevent congestion before it happens or remove congestion after it has happened. In general, we can divide congestion control mechanisms into two broad categories: **open-loop congestion control** (prevention) and **closed-loop congestion control** (removal).

Open-Loop Congestion Control:
In open-loop congestion control, policies are applied to prevent congestion before it happens. In these mechanisms, congestion control is handled by either the source or the destination.

Retransmission Policy Retransmission is sometimes unavoidable. If the sender feels that a sent packet is lost or corrupted, the packet needs to be retransmitted. Retransmission in general may increase congestion in the network. However, a good retransmission policy can prevent congestion. The retransmission policy and the retransmission timers must be designed to optimize efficiency and at the same time prevent congestion.

Window Policy The type of window at the sender may also affect congestion. The Selective Repeat window is better than the Go-Back- window for congestion control.

Acknowledgment Policy The acknowledgment policy imposed by the receiver may also affect congestion. If the receiver does not acknowledge every packet it receives, it may slow down the sender and help prevent congestion. A receiver may send an acknowledgment only if it has a packet to be sent or a special timer expires. A receiver may decide to acknowledge only packets at a time. the acknowledgments are also part of the load in a network.

Sending fewer acknowledgments means imposing less load on the network.

Discarding Policy A good discarding policy by the routers may prevent congestion and at the same time may not harm the integrity of the transmission. For example, in audio transmission, if the policy is to discard less sensitive packets when congestion is likely to happen, the quality of sound is still preserved and congestion is prevented or alleviated.

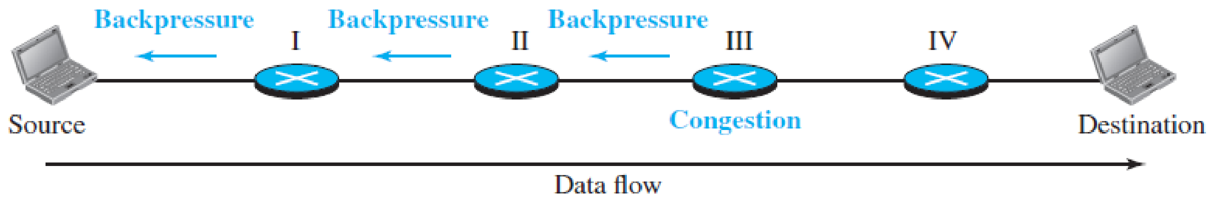
Admission Policy An admission policy, which is a quality-of-service mechanism , can also prevent congestion in virtual-circuit networks. Switches in a flow first check the resource requirement of a flow before admitting it to the network. A router can deny establishing a virtual-circuit connection if there is congestion in the network or if there is a possibility of future congestion.

Closed-Loop Congestion Control

Closed-loop congestion control mechanisms try to alleviate congestion after it happens.

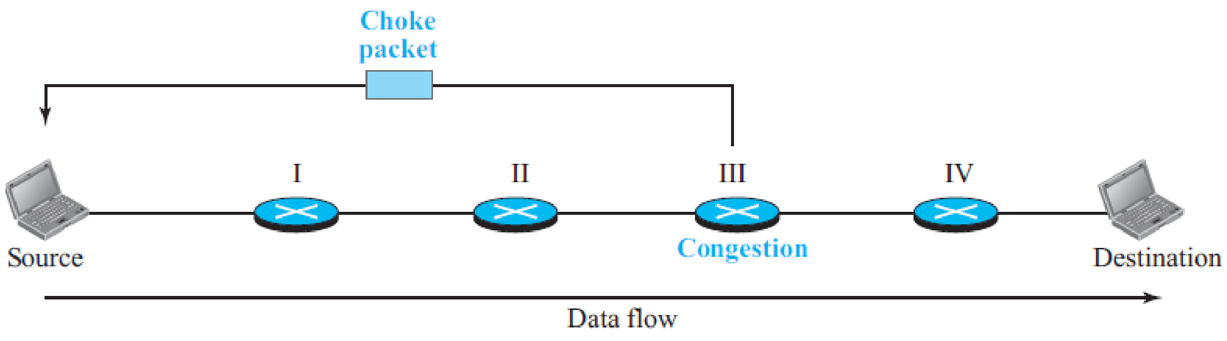
Backpressure The technique of refers to a congestion control mechanism in which a congested node stops receiving data from the immediate upstream node or nodes. This may cause the upstream node or nodes to become congested, and they, in turn, reject data from their upstream node or nodes, and so on. Backpressure is a node-to- node congestion control that starts with a node and propagates, in the opposite direction of data flow, to the source. The backpressure technique can be applied only to virtual circuit networks, in which each node knows the upstream node from which a flow of data is coming.

Figure 18.14 *Backpressure method for alleviating congestion*



Choke Packet A **choke packet** is a packet sent by a node to the source to inform it of congestion. Note the difference between the backpressure and choke-packet methods. In backpressure, the warning is from one node to its upstream node, although the warning may eventually reach the source station. In the choke-packet method, the warning is from the router, which has encountered congestion, directly to the source station. The intermediate nodes through which the packet has traveled are not warned.

Figure 18.15 *Choke packet*



Implicit Signaling In implicit signaling, there is no communication between the congested node or nodes and the source. The source guesses that there is congestion somewhere in the network from other symptoms. For example, when a source sends several packets and there is no acknowledgment for a while, one assumption is that the network is congested. The delay in receiving an acknowledgment is interpreted as congestion in the network; the source should slow down.

Explicit Signaling The node that experiences congestion can explicitly send a signal to the source or destination. The explicit-signaling method, however, is different from the choke-packet method. In the choke-packet method, a separate packet is used for this purpose; in the explicit-signaling method, the signal is included in the packets that carry data. Explicit signaling can occur in either the forward or the backward direction. This type of congestion control can be seen in an ATM network,

ADDRESS MAPPING

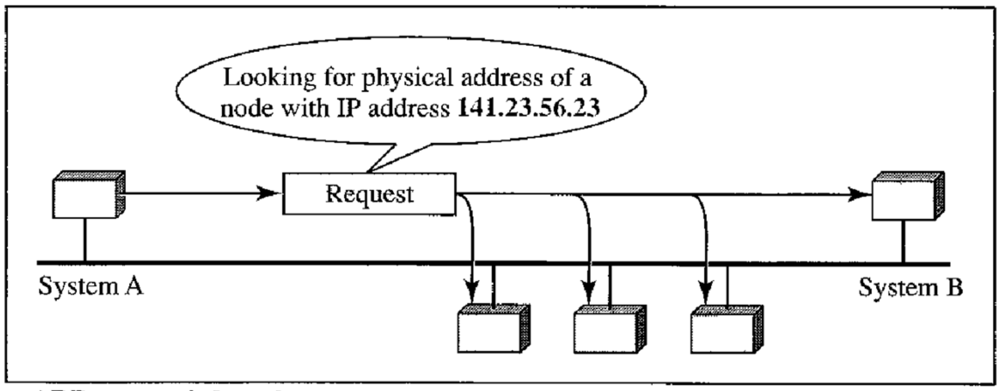
A physical address is a local address. It is called a address because it is usually (but not always) implemented in hardware. An example of a physical address is the 48-bit MAC address in the Ethernet protocol, which is imprinted on the NIC installed in the host or router. The physical address and the logical address are two different identifiers.

Mapping Logical to Physical Address: ARP

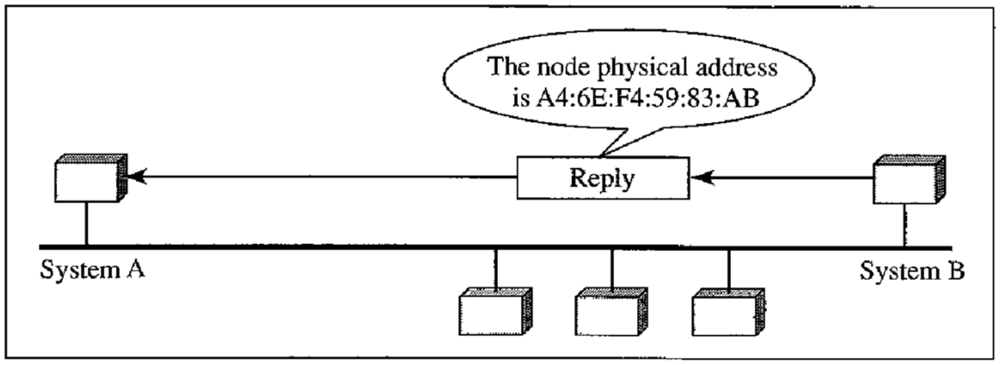
The system on the left (A) has a packet that needs to be delivered to another system (B) with IP address 141.23.56.23. System A needs to pass the packet to its data link layer for the actual delivery, but it does not know the physical address of the recipient. It uses the services of ARP by asking the ARP protocol to send a broadcast ARP request packet to ask for the physical address of a system with an IP address of 141.23.56.23.

This packet is received by every system on the physical network, but only system B will answer it, as shown in Figure 21.1 b. System B sends an ARP reply packet that includes its physical address. Now system A can send all the packets it has for this destination by using the physical address it received.

Figure 21.1 *ARP operation*


Looking for physical address of a node with IP address 141.23.56.23
Request
System A
System B

a. ARP request is broadcast



b. ARP reply is unicast

Four cases using ARP: The following are four different cases in which the services of ARP can be used

1. The sender is a host and wants to send a packet to another host on the same network. In this case, the logical address that must be mapped to a physical address is the destination IP address in the datagram header.

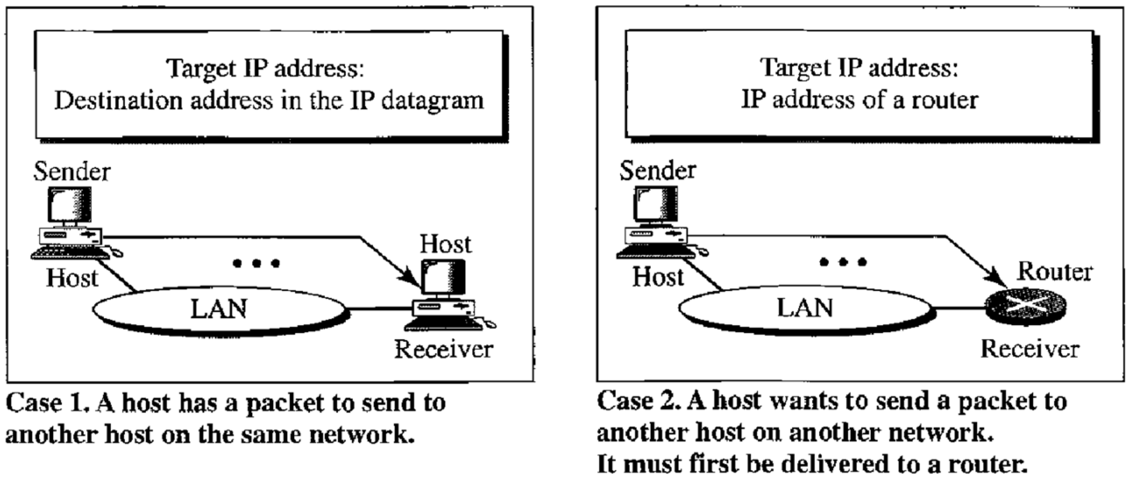
2. The sender is a host and wants to send a packet to another host on another network. In this case, the host looks at its routing table and finds the IP address of the next hop (router) for this destination. If it does not have a routing table, it looks for the IP address of the default router. The IP address of the router becomes the logical address that must be mapped to a physical address.

3. The sender is a router that has received a datagram destined for a host on another network. It checks its routing table and finds the IP address of the next router. The IP address of the next router becomes the logical address that must be mapped to a physical address.

4. The sender is a router that has received a datagram destined for a host on the same network. The destination IP address of the datagram becomes the logical address that must be mapped to a physical address.

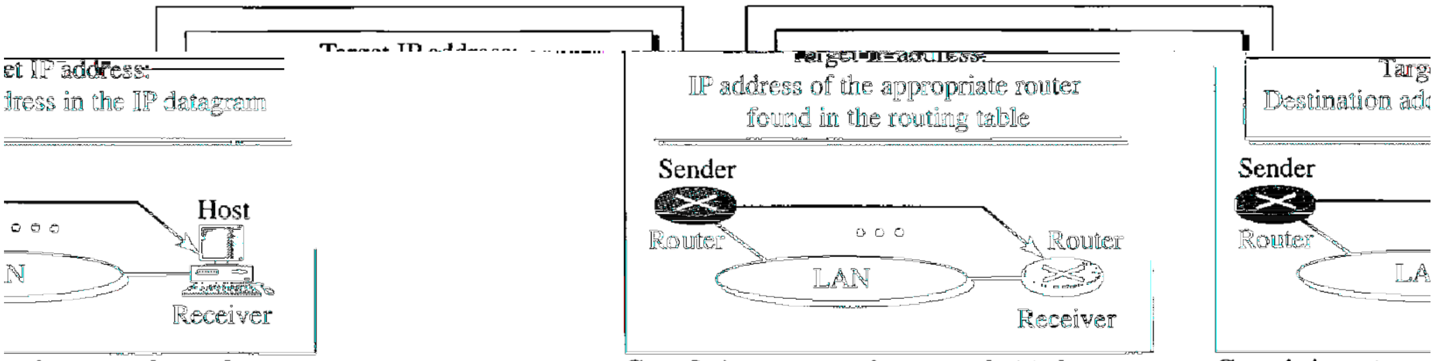
An ARP request is broadcast; an ARP reply is unicast.

Figure 21.4 *Four cases using ARP*



Case 1. A host has a packet to send to another host on the same network.

Case 2. A host wants to send a packet to another host on another network. It must first be delivered to a router.


IP address of the appropriate router
found in the routing table
Sender
Router
LAN
Router
Receiver
Host
Receiver
Sender
Router

ɛceives a packet to be sent me network.

Case 3. A router receives a packet to be sent to a host on another network. It must first be delivered to the appropriate router.

Case 4. A router r to a host on the sa

Mapping Physical to Logical Address: RARP, BOOTP, and DHCP

There are occasions in which a host knows its physical address, but needs to know its logical address. This may happen in two cases:

1. A diskless station is just booted. The station can find its physical address by checking its interface, but it does not know its IP address.
2. An organization does not have enough IP addresses to assign to each station; it needs to assign IP addresses on demand. The station can send its physical address and ask for a short time lease.

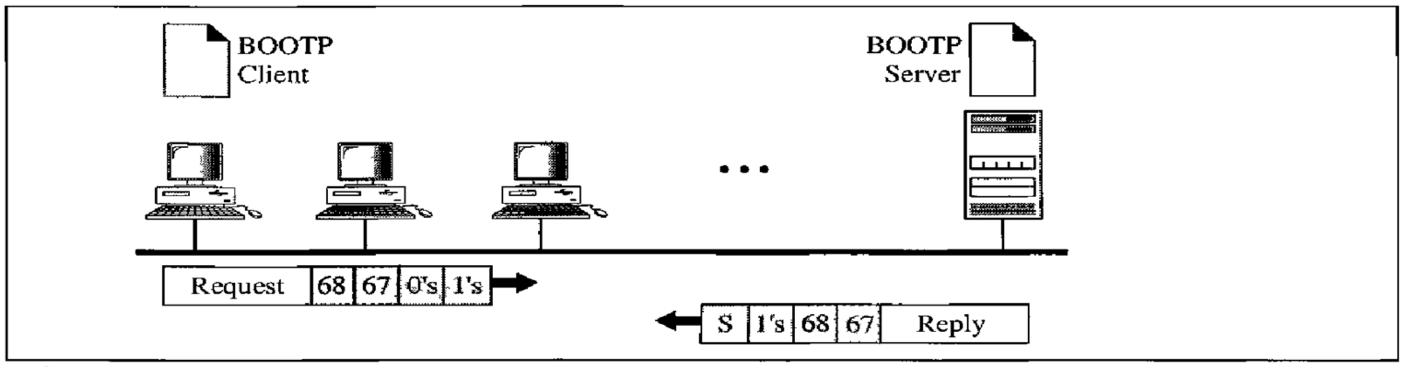
RARP

Reverse Address Resolution Protocol (RARP) finds the logical address for a machine that knows only its physical address. The machine can get its physical address (by reading its NIC, for example), which is unique locally. It can then use the physical address to get the logical address by using the RARP protocol. A RARP request is created and broadcast on the local network. Another machine on the local network that knows all the IP addresses will respond with a RARP reply. The requesting machine must be running a RARP client program; the responding machine must be running a RARP server program.
There is a serious problem with RARP: Broadcasting is done at the data link layer. The physical broadcast address, all 1s in the case of Ethernet, does not pass the boundaries of a network. This means that if an administrator has several networks or several subnets, it needs to assign a RARP server for each network or subnet. This is the reason that RARP is almost obsolete. Two protocols, BOOTP and DHCP, are replacing RARP.

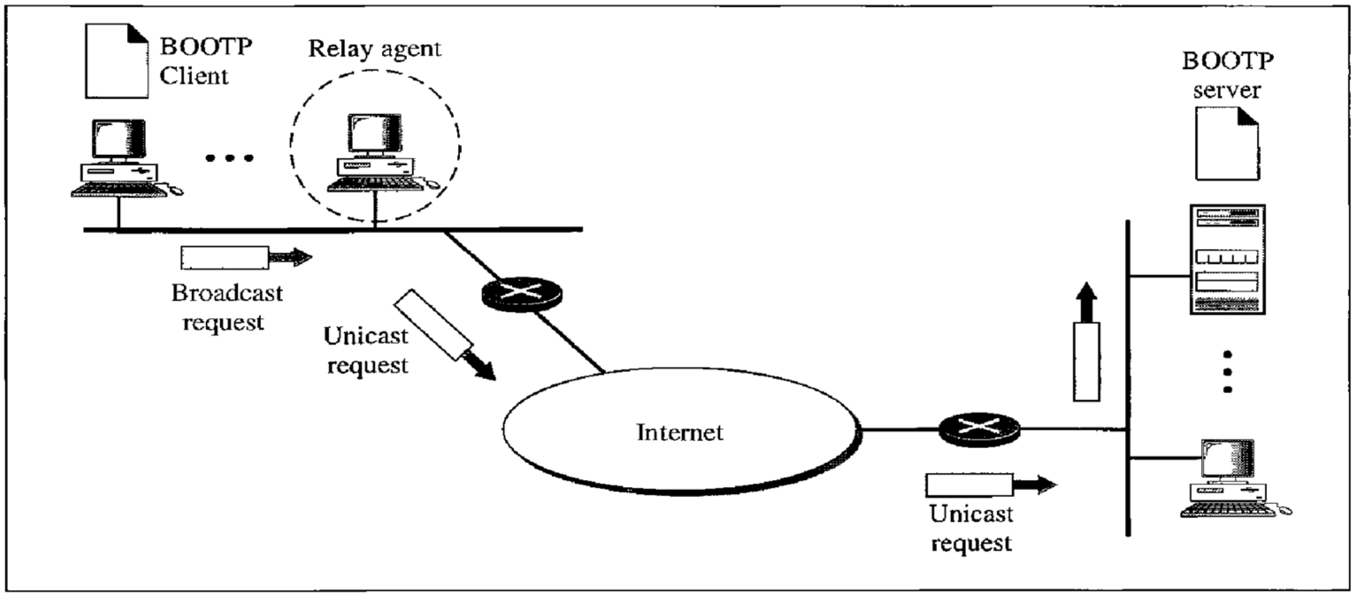
BOOTP

The Bootstrap Protocol (BOOTP) is a client/server protocol designed to provide physical address to logical address mapping. BOOTP is an application layer protocol. The administrator may put the client and the server on the same network or on different networks.

How a client can send an IP datagram when it knows neither its own IP address (the source address) nor the server's IP address (the destination address). The client simply uses all 0s as the source address and all 1s as the destination address.



a. Client and server on the same network



b. Client and server on different networks

One of the advantages of BOOTP over RARP is that the client and server are application-layer processes. As in other application-layer processes, a client can be in one network and the server in another, separated by several other networks. However, there is one problem that must be solved. The BOOTP request is broadcast because the client does not know the IP address of the server. A broadcast IP datagram cannot pass through any router. To solve the problem, there is a need for an intermediary.
One of the hosts (or a router that can be configured to operate at the application layer) can be used as a relay. The host in this case is called a relay agent. The relay agent knows the unicast address of a BOOTP server. When it receives this type of packet, it encapsulates the message in a unicast datagram and sends the request to the BOOTP server.
The Packet carrying a unicast destination address, is routed by any router and reaches the BOOTP server. The BOOTP server knows the message comes from a relay agent because one of the fields in the request message defines the IP address of the relay agent. The relay agent, after receiving the reply, sends it to the BOOTP client.

DHCP

BOOTP is not a dynamic configuration protocol. When a client requests its IP address, the BOOTP server consults a table that matches the physical address of the client with its IP address. This implies that the binding between the physical address and the IP address of the client already exists. The binding is predetermined.

However, what if a host moves from one physical network to another? What if a host wants a temporary IP address? BOOTP cannot handle these situations because the binding between the physical and IP addresses is static and fixed in a table until changed by the administrator. BOOTP is a static configuration protocol.

The Dynamic Host Configuration Protocol (DHCP) has been devised to provide static and dynamic address allocation that can be manual or automatic.

DHCP provides static and dynamic address allocation that can be manual or automatic.

Static Address Allocation In this capacity DHCP acts as BOOTP does. It is backward compatible with BOOTP, which means a host running the BOOTP client can request a static address from a DHCP server. A DHCP server has a database that statically binds physical addresses to IP addresses.

Dynamic Address Allocation DHCP has a second database with a pool of available IP addresses. This second database makes DHCP dynamic. When a DHCP client requests a temporary IP address, the DHCP server goes to the pool of available (unused) IP addresses and assigns an IP address for a negotiable period of time.

Routing:

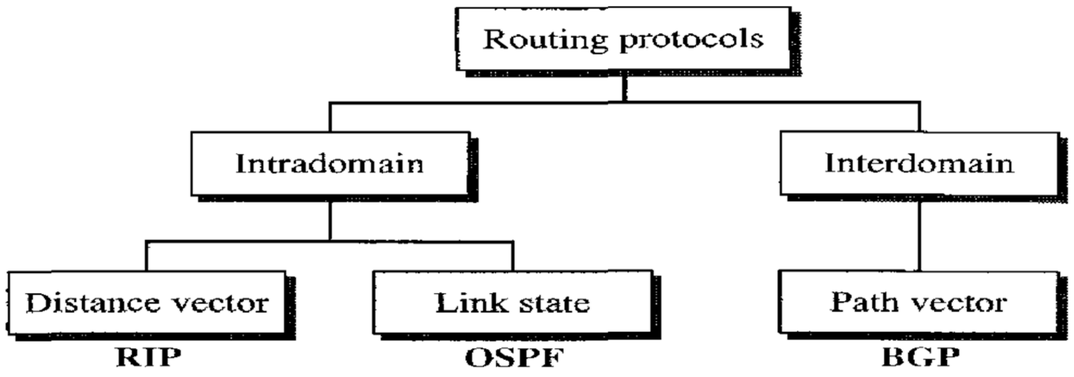
The goal of the network layer is to deliver a datagram from its source to its destination or destinations. If a datagram is destined for only one destination (one-to-one delivery), we have . If the datagram is destined for several destinations (one-to-many delivery), we have .

A routing table can be either static or dynamic. A is one with manual entries.

A on the other hand, is one that is updated automatically when there is a change somewhere in the internet. The tables need to be updated as soon as there is a change in the internet. For instance, they need to be updated when a router is down, and they need to be updated whenever a better route has been found.

Intra and Inter-domain Routing:

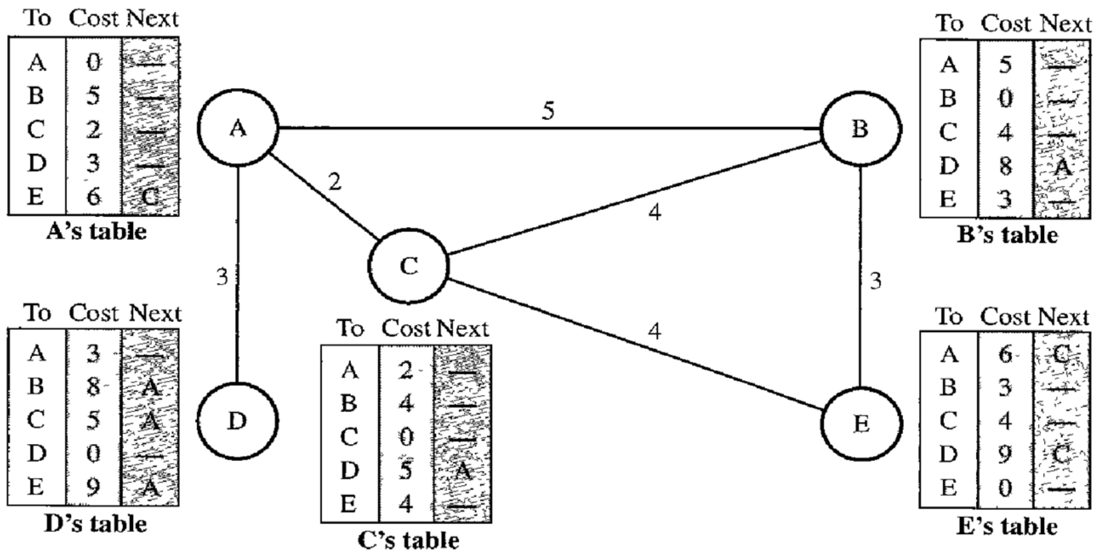
An internet is divided into autonomous systems. An autonomous system (AS) is a group of networks and routers under the authority of a single administration. Routing inside an autonomous system is referred to as intra-domain routing. Routing between autonomous systems is referred to as inter-domain routing. Each autonomous system can choose one or more intra-domain routing protocols to handle routing inside the autonomous system.



Distance Vector Routing:

In distance vector routing, the least-cost route between any two nodes is the route with minimum distance. In this protocol, as the name implies, each node maintains a vector (table) of minimum distances to every node.

Figure 22.14 *Distance vector routing tables*



To	Cost	Next
A	0	—
B	5	—
C	2	—
D	3	—
E	6	C

A's table

To	Cost	Next
A	5	—
B	0	—
C	4	—
D	8	A
E	3	—

B's table

To	Cost	Next
A	3	—
B	8	A
C	5	A
D	0	—
E	9	A

D's table

To	Cost	Next
A	2	—
B	4	—
C	0	—
D	5	A
E	4	—

C's table

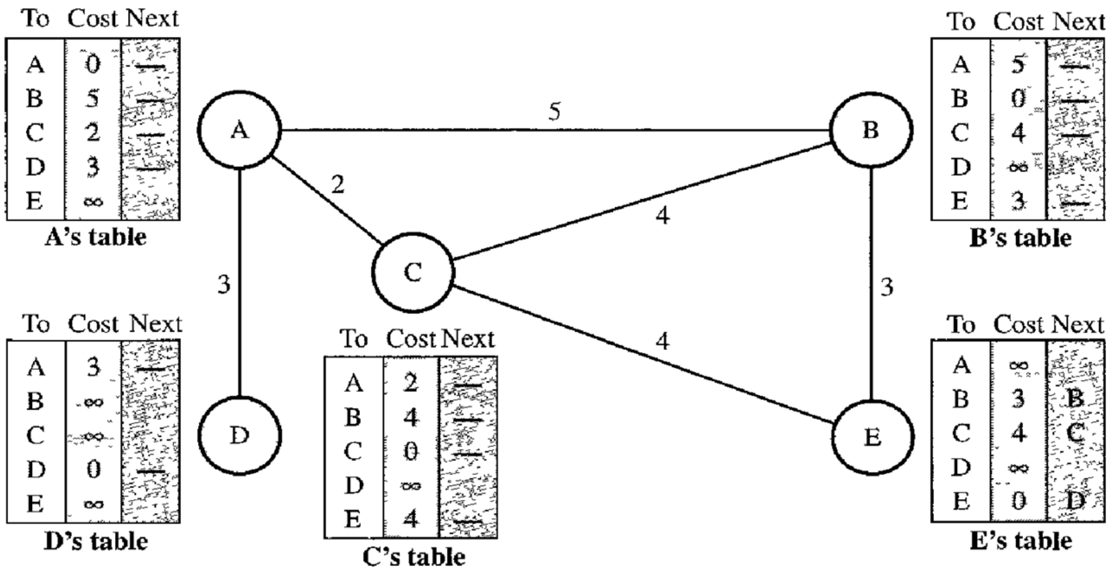
To	Cost	Next
A	6	C
B	3	—
C	4	—
D	9	C
E	0	—

E's table

Initialization

The tables in Figure 22.14 are stable; each node knows how to reach any other node and the cost. At the beginning, however, this is not the case. Each node can know only the distance between itself and its immediate neighbors, those directly connected to it. So for the moment, we assume that each node can send a message to the immediate neighbors and find the distance between itself and these neighbors. The distance for any entry that is not a neighbor is marked as infinite (unreachable).

Figure 22.15 *Initialization of tables in distance vector routing*



Sharing

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

In other words, sharing here means sharing only the first two columns.

In distance vector routing, each node shares its routing table with its immediate neighbors periodically and when there is a change.

Updating

When a node receives a two-column table from a neighbor, it needs to update its routing table.

When to Share

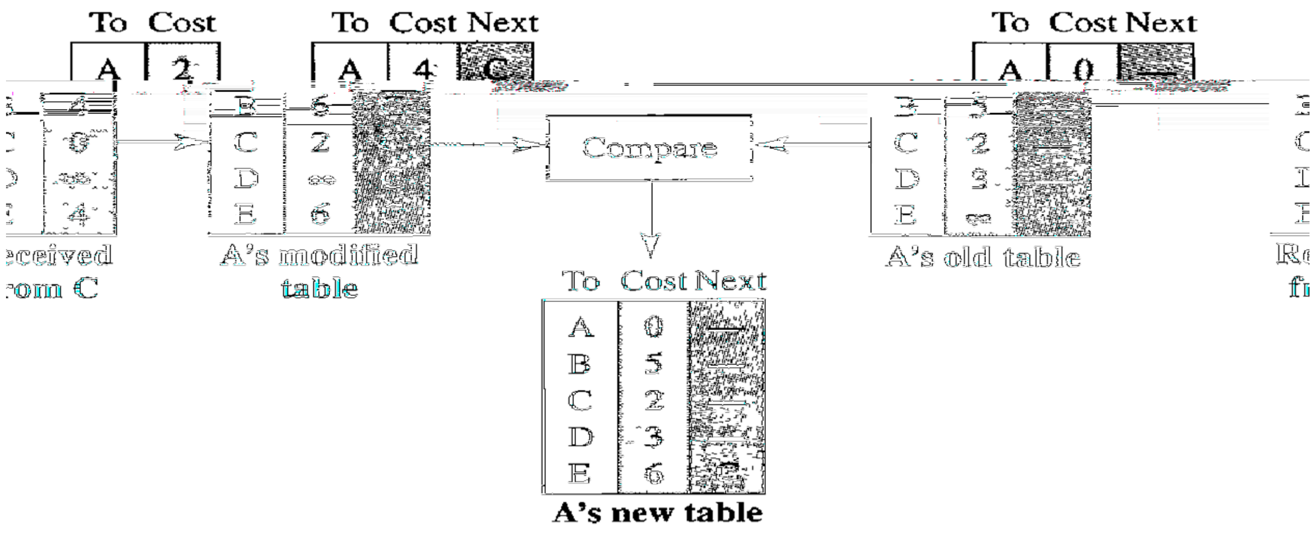
The question now is, When does a node send its partial routing table (only two columns) to all its immediate neighbors? The table is sent both periodically and when there is a change in the table.

Periodic Update A node sends its routing table, normally every 30 s, in a periodic update. The period depends on the protocol that is using distance vector routing.

Triggered Update A node sends its two-column routing table to its neighbors anytime there is a change in its routing table. This is called a triggered update. The change can result from the following.

1. A node receives a table from a neighbor, resulting in changes in its own table after updating.
2. A node detects some failure in the neighboring links which results in a distance change to infinity.

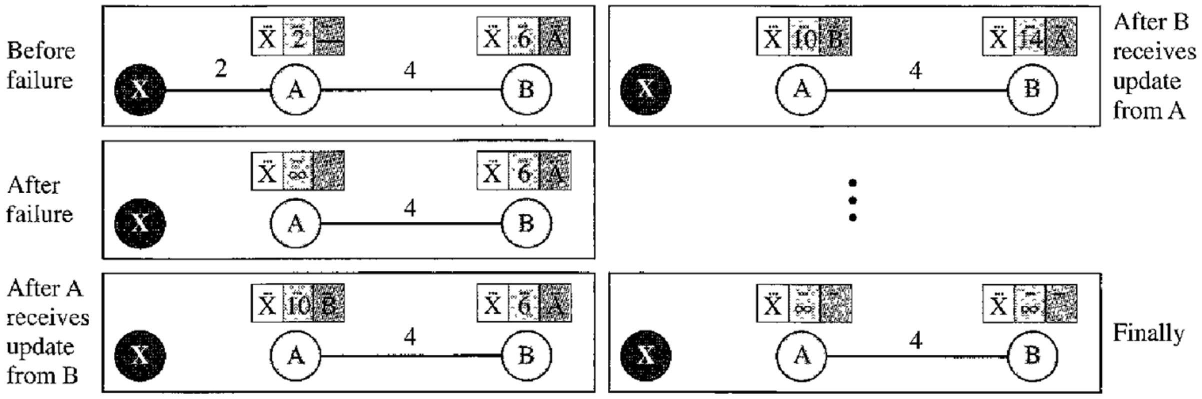
Updating in distance vector routing



Count to Infinity:

A problem with distance-vector routing is that any decrease in cost (good news) propagates quickly, but any increase in cost (bad news) will propagate slowly. For a routing protocol to work properly, if a link is broken (cost becomes infinity), every other router should be aware of it immediately, but in distance-vector routing, this takes some time. The problem is referred to as ***count to infinity***.

Two-Node Loop Instability
A problem with distance vector routing is instability, which means that a network using this protocol can become unstable.



At the beginning, both nodes A and B know how to reach node X. But suddenly, the link between A and X fails. Node A changes its table. If A can send its table to B immediately, everything is fine. However, the system becomes unstable if B sends its forwarding table to A before receiving A's forwarding table. Node A receives the update and, assuming that B has found a way to reach X, immediately updates its forwarding table. Now A sends its new update to B. Now B thinks that something has been changed around A and updates its forwarding table. The cost of reaching X increases gradually until it reaches infinity. At this moment, both A and B know that X cannot be reached. However, during this time the system is not stable. Node A thinks that the route to X is via B; node B thinks that the route to X is via A. If A receives a packet destined for X, the packet goes to B and then comes back to A. Similarly, if B receives a packet destined for X, it goes to A and comes back to B. Packets bounce between A and B, creating a two-node loop problem. A few solutions have been proposed for instability of this kind.

Split Horizon

One solution to instability is called ***split horizon***. In this strategy, instead of flooding the table through each interface, each node sends only part of its table through each interface. If, according to its table, node B thinks that the optimum route to reach X is via A, it does not need to advertise this piece of information to A; the information has come from A (A already knows). Taking information from node A, modifying it, and sending it back to node A is what creates the confusion. In our scenario, node B eliminates the last line of its forwarding table before it sends it to A. In this case, node A keeps the value of infinity as the distance to X. Later, when node A sends its forwarding table to B, node B also corrects its forwarding table. The system becomes stable after the first update: both node A and node B know that X is not reachable.

Poison Reverse

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

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

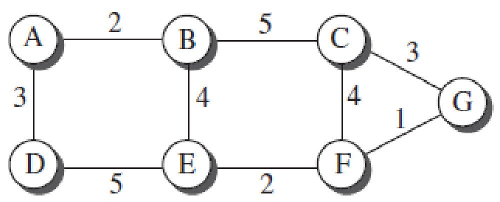
Link-State Routing:

In Link State Routing algorithm the cost associated with an edge defines the state of the link. Links with lower costs are preferred to links with higher costs; if the cost of a link is infinity, it means that the link does not exist or has been broken.

Link-State Database (LSDB)

To create a least-cost tree with this method, each node needs to have a complete of the network, which means it needs to know the state of each link. The collection of states for all links is called the ***link-state database LSDB .*** There is only one LSDB for the whole internet; each node needs to have a duplicate of it to be able to create the least-cost tree. The LSDB can be represented as a two-dimensional array(matrix) in which the value of each cell defines the cost of the corresponding link.

Figure 20.8 *Example of a link-state database*



a. The weighted graph

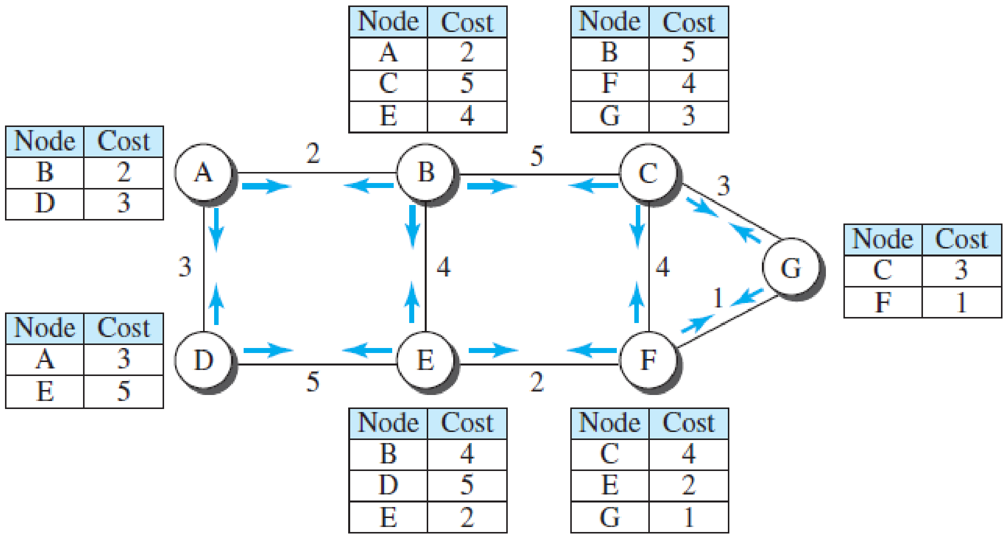
	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

b. Link state database

Now the question is how each node can create this LSDB that contains information about the whole internet. This can be done by a process called **flooding.** Each node can send some greeting messages to all its immediate neighbors (those nodes to which it is connected directly) to collect two pieces of information for each neighboring node: the identity of the node and the cost of the link. The combination of these two pieces of information is called the (LSP); the LSP is sent out of each interface, When a node receives an LSP from one of its interfaces, it compares the LSP with the copy it may already have. If the newly arrived LSP is older than the one it has (found by checking the sequence number), it discards the LSP. If it is newer or the first one received, the node discards the old LSP (if there is one) and keeps the received one. It then sends a copy of it out of each interface except the one from which the packet arrived. This guarantees that flooding stops somewhere in the network (where a node has only one interface).

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

Figure 20.9 *LSPs created and sent out by each node to build LSDB*



Node	Cost
B	2
D	3

Node	Cost
A	2
C	5
E	4

Node	Cost
B	5
F	4
G	3

Node	Cost
C	3
F	1

Node	Cost
A	3
E	5

Node	Cost
B	4
D	5
E	2

Node	Cost
C	4
E	2
G	1

In the distance-vector routing algorithm, each router tells its neighbors what it knows about the whole internet; in the link-state routing algorithm, each router tells the whole internet what it knows about its neighbors.

Formation of Least-Cost Trees

To create a least-cost tree for itself, using the shared LSDB, each node needs to run the famous **Dijkstra Algorithm.** This iterative algorithm uses the following steps:

1. The node chooses itself as the root of the tree, creating a tree with a single node, and sets the total cost of each node based on the information in the LSDB.

2. The node selects one node, among all nodes not in the tree, which is closest to the root, and adds this to the tree. After this node is added to the tree, the cost of all other nodes not in the tree needs to be updated because the paths may have been changed.

3. The node repeats step 2 until all nodes are added to the tree.

TECHNIQUES TO IMPROVE QoS

There are some techniques that can be used to improve the quality of service. four common methods: scheduling, traffic shaping, admission control, and resource reservation.

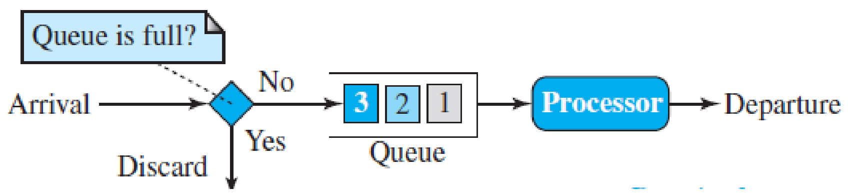
Scheduling

Packets from different flows arrive at a switch or router for processing. A good scheduling technique treats the different flows in a fair and appropriate manner. Several scheduling techniques are designed to improve the quality of service.

FIFO Queuing

In first-in, first-out (FIFO) queuing, packets wait in a buffer (queue) until the node (router or switch) is ready to process them. If the average arrival rate is higher than the average processing rate, the queue will fill up and new packets will be discarded. A FIFO queue is familiar to those who have had to wait for a bus at a bus stop.

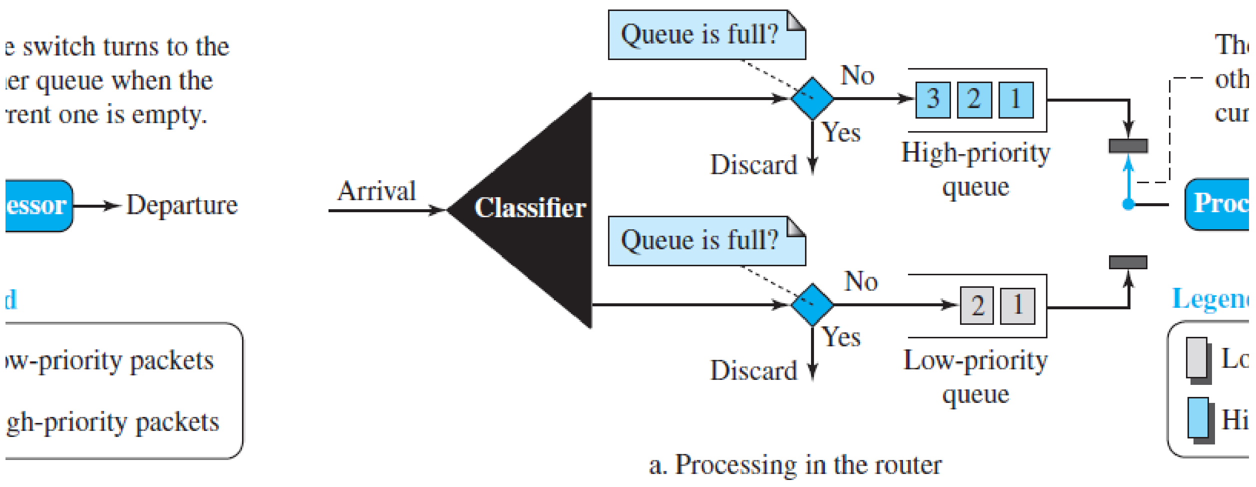
Figure 30.1 *FIFO queue*



Priority Queuing

In priority queuing, packets are first assigned to a priority class. Each priority class has its own queue. The packets in the highest-priority queue are processed first. Packets in the lowest-priority queue are processed last. Note that the system does not stop serving a queue until it is empty.

Figure 30.2 *Priority queuing*

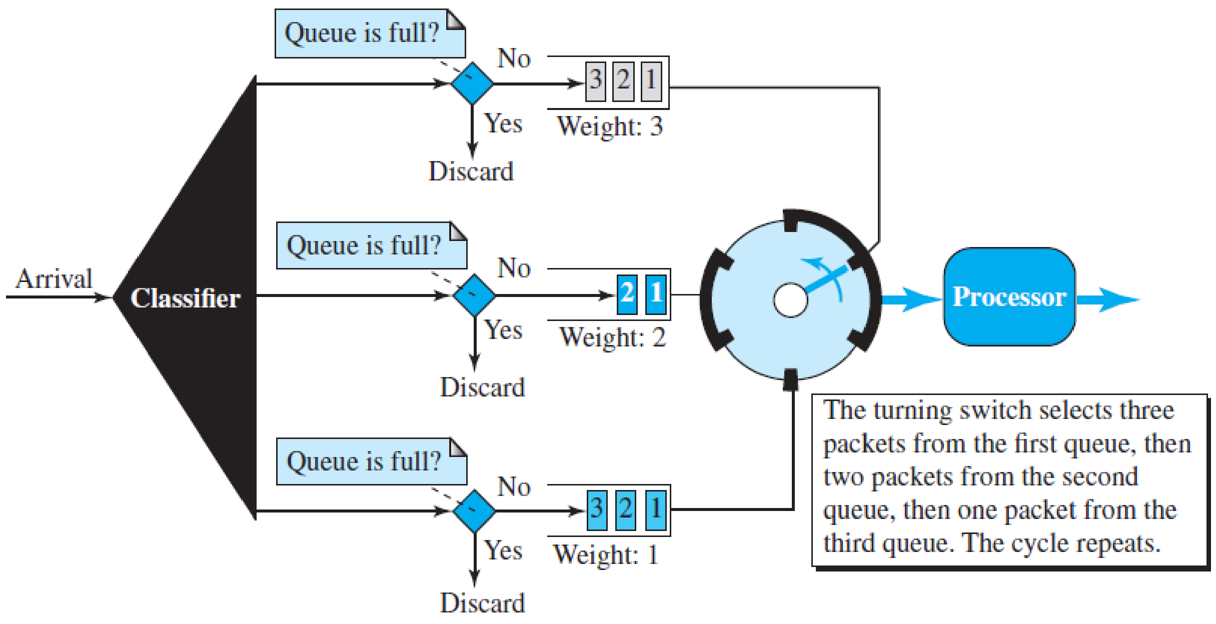


A priority queue can provide better QoS than the FIFO queue because higher priority traffic, such as multimedia, can reach the destination with less delay. However, there is a potential drawback. If there is a continuous flow in a high-priority queue, the packets in the lower-priority queues will never have a chance to be processed. This is a condition called starvation.

Weighted Fair Queuing

A better scheduling method is weighted fair queuing. In this technique, the packets are still assigned to different classes and admitted to different queues. The queues, however, are weighted based on the priority of the queues; higher priority means a higher weight. The system processes packets in each queue in a round-robin fashion with the number of packets selected from each queue based on the corresponding weight. For example, if the weights are 3, 2, and 1, three packets are processed from the first queue, two from the second queue, and one from the third queue. If the system does not impose priority on the classes, all weights can be equal. In this way, we have fair queuing with priority.

Figure 30.3 *Weighted fair queuing*


Queue is full?
No
3 2 1
Yes
Weight: 3
Discard
Queue is full?
No
2 1
Yes
Weight: 2
Discard
Arrival
Classifier
Processor
Queue is full?
No
3 2 1
Yes
Weight: 1
Discard
The turning switch selects three packets from the first queue, then two packets from the second queue, then one packet from the third queue. The cycle repeats.

Traffic Shaping

Traffic shaping is a mechanism to control the amount and the rate of the traffic sent to the network. Two techniques can shape traffic: leaky bucket and token bucket.

Leaky Bucket

If a bucket has a small hole at the bottom, the water leaks from the bucket at a constant rate as long as there is water in the bucket. The rate at which the water leaks does not depend on the rate at which the water is input to the bucket unless the bucket is empty. The input rate can vary, but the output rate remains constant. Similarly, in networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate.

Figure 30.4 *Leaky bucket*

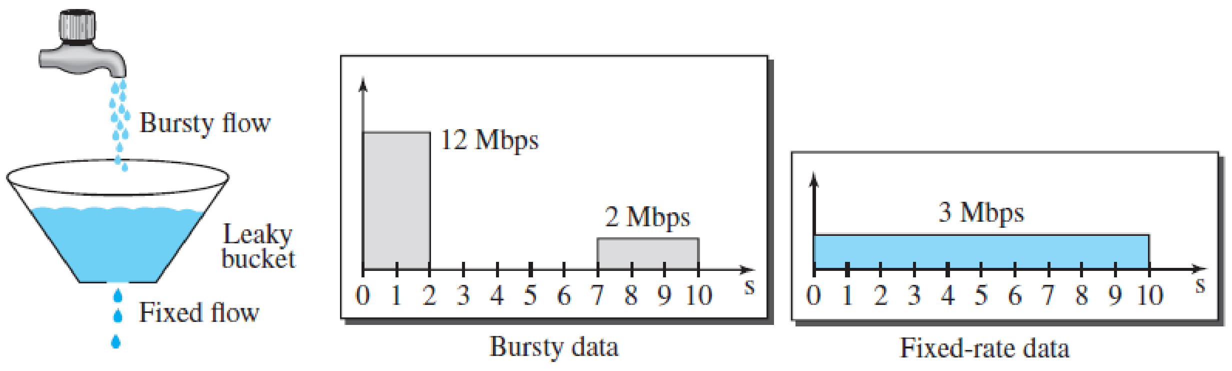
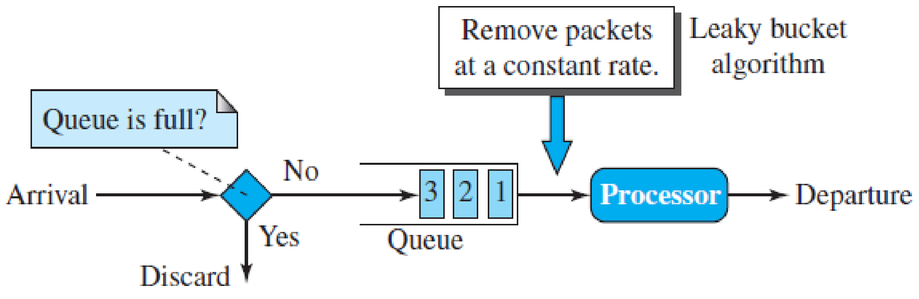


Figure 30.5 *Leaky bucket implementation*



A leaky bucket algorithm shapes bursty traffic into fixed-rate traffic by averaging the data rate. It may drop the packets if the bucket is full.

Token Bucket

The leaky bucket is very restrictive. It does not credit an idle host. For example, if a host is not sending for a while, its bucket becomes empty. Now if the host has bursty data, the leaky bucket allows only an average rate. The time when the host was idle is not taken into account. On the other hand, the **token bucket** algorithm allows idle hosts to accumulate credit for the future in the form of tokens.

Assume the capacity of the bucket is tokens and tokens enter the bucket at the rate of tokens per second. The system removes one token for every cell of data sent. The maximum number of cells that can enter the network during any time interval of length is shown below.

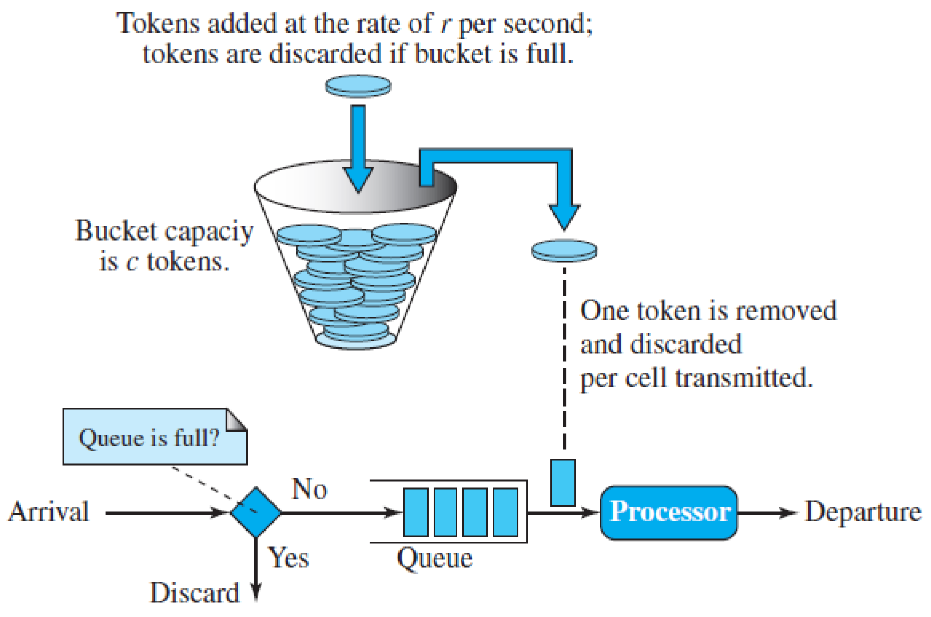
$$\text{Maximum number of packets} = r \times t + c$$

The maximum average rate for the token bucket is shown below.

$$\text{Maximum average rate} = (r \times t + c)/t \text{ packets per second}$$

This means that the token bucket limits the average packet rate to the network.

Figure 30.6 *Token bucket*


Tokens added at the rate of r per second;
tokens are discarded if bucket is full.
Bucket capaciy is c tokens.
One token is removed and discarded per cell transmitted.
Queue is full?
Arrival
No
Yes
Discard
Queue
Processor
Departure

Example 30.2

Let's assume that the bucket capacity is 10,000 tokens and tokens are added at the rate of 1000 tokens per second. If the system is idle for 10 seconds (or more), the bucket collects 10,000 tokens and becomes full. Any additional tokens will be discarded. The maximum average rate is shown below.

$$\textbf{Maximum average rate} = (1000t + 10{,}000)/t$$

The token bucket can easily be implemented with a counter. The counter is initialized to zero. Each time a token is added, the counter is incremented by 1. Each time a unit of data is sent, the counter is decremented by 1. When the counter is zero, the host cannot send data.

The token bucket allows bursty traffic at a regulated maximum rate.

Combining Token Bucket and Leaky Bucket

The two techniques can be combined to credit an idle host and at the same time regulate the traffic. The leaky bucket is applied after the token bucket; the rate of the leaky bucket needs to be higher than the rate of tokens dropped in the bucket.