NEAR EAST UNIVERSITY

Faculty of Engineering

Depertment of Computer Engineering

INTERNET: Architecture, Addressing, Protocols, Routing

Graduation Project
COM-400

Student: Aslı Topcu

Supervisor: Prof. Dr. Fakhraddin Mamedov

Lefkoşa-2000

CONTENTS

AKNOWLEDGEMENTS

I wish to thank Prof. Dr. Fakhraddin Mamedov, for intellectual support, encouragement, and enthusiasm which made this thesis possible, and for his patience in correcting both my stylistic and scientific errors.

I thank all my teachers for them help and them patience in correcting my thesis.

I thank my family for their continious encouragement.

I thank all my friends espicially Damla Özdemir, Hatice Köseliören, Naim Aladağ, Mehmet Karakan...., etc. Who helped me alot.

To all of them, all my love and respectations.

ABSTRACT

INTERNET

The Internet is growing rapidly in the number of computers and networks it connects, in the volume of traffic it carries, and in the range of applications it enables. The main protocols of the Internet are the Internet protocol (IP) and the transmission control protocol (TCP). Many companies build isolated networks also using TCP/IP These networks are called intranets. Such networks may be attached to the Internet, usually through a "firewall" that protects the security of the intranet. We call a network that uses the TCP/IP protocols an internet.

We take for granted that computers attached to a common link can exchange packets. Computers attached by a LAN other than an Ethernet or by a point-to-point link or another packet delivery service such as an ATM virtual circuit can also exchange packets. Usually, such packet transfers are not reliable, but have a small packet error rate.

We review the architecture of TCP/IP networks. We then explain the addressing and routing. We discuss the transport layer. After this basic discussion of the major Internet protocols, we present a few complements on some aspects of these protocols. All these complements can be skipped without harm to the understanding the rest of the book.

INTRODUCTION

A few of the major milestones in the history of the Internet are the following.

· 1962: Paul Baran, of the Rand Corporation, proposes packet switching as a robust networking mechanism.

· 1969: The Department of Defense Advanced Research Projects Agency funds a project on packet switched networks. The first four nodes of ARPANET are connected.

· 1974: Vint Cerf and Bob Kahn publish the basic mechanisms of TCP

· 1982: The protocol suite TCP/IP is defined for ARPANET.

· 1984: Domain name system is introduced.

· 1986: NSFNET, the backbone (at 56 kbps) of Internet is created by the National Science Foundation (NSF).

· 1992: The World Wide Web (WWW), designed by Tim Berners-Lee, is released b CERN, the European Organization for Nuclear Research.

The telephone networks were designed in a centralized way by major corporations. In contrast, the evolution of the Internet has been decentralized, almost chaotic. No one can estimate reliably the number of hosts on the networks or the number of users. Such a distributed growth is possible because of the specific protocols that control the operations of Internet, as we explain in this chapter.

Figure 1.1 illustrates the Internet in the United States around 1990. The highest part of the figure shows NSFNET, the backbone of the network. Regional networks are attached to the nodes of the backbone. This arrangement is sketched in Fiqure 1.2.

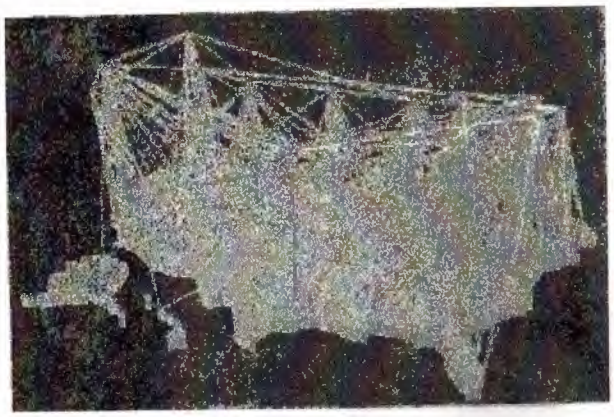


figure 1.1.

In 1997, the Internet structure was as shown in Figure 1.2. As the figure shows, Internet has three backbones that are maintained by network service providers. Users are connected to the Internet through a large number of Internet service providers.

The Internet is a collection of interconnected subnetworks. The terminology is that the Internet consists of *autonomous systems* that are connected together by routers called *border gateways.* Roughly, an autonomous system is a network that is managed by an independent authority. Thus, the Berkeley campus network is an autonomous system. To go from one host to another, a packet may have to go across a number of autonomous systems and there may be a few possible alternative paths.

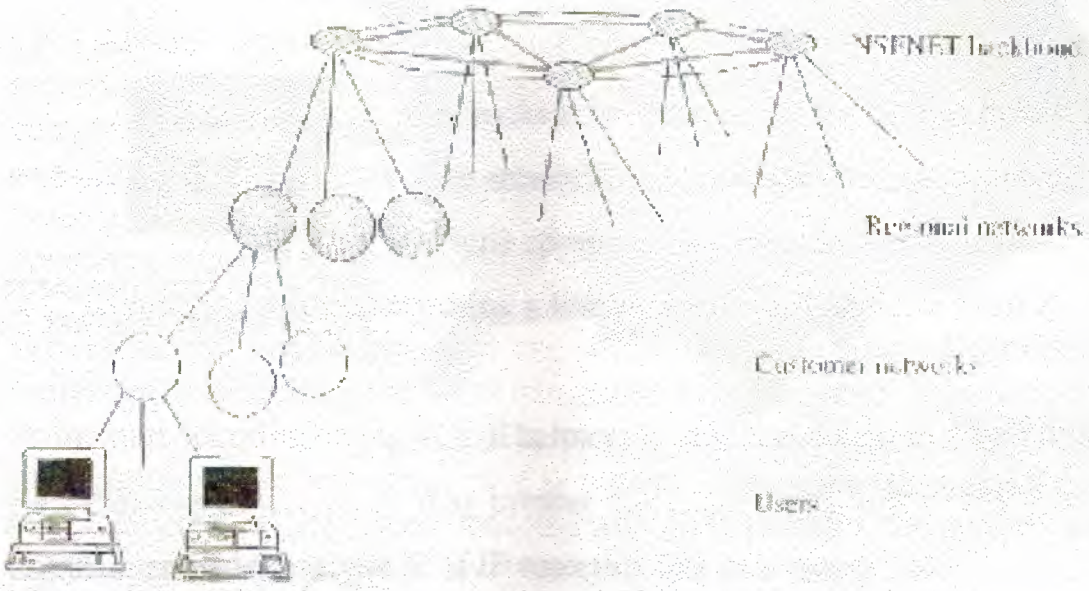


Figure 1.2.

1.ARCHITECTURE

The architecture of TCP/IP networks is shown in Figure 1.3. We comment briefly on each of the layers.

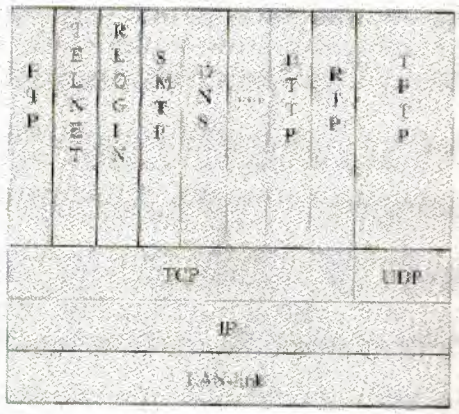
1.1 LAN Link Layer

The LAN-link layer transmits packets between a specific pair of nodes. In most cases, these two nodes are attached to a common physical channel. For instance, if you use your home computer to communicate with your office çomputer, you may be using a link protocol such as SLIP or PPP

To understand some networks, it helps to extend the notion of a link to a packet delivery system that may involve more than one physical link. For instance, consider a network of IP routers interconnected by an ATM (permanent) virtual circuit. In looking at this network as an internet, focusing on the TCP/IP protocols, it is convenient to view the ATM network as implementing links between pairs of IP routers. Thus, although each ATM link may consist of several physical links interconnected with ATM switches that perform some routing and flow control functions, the IP layer sees this link as *a simple* link.

The LAN-link layer delivers packets, possibly with occasional transmission errors. The main characteristics of a link are its maximum transfer unit (MTU), its packet error rate (PER), and its transmission rate. The MTU is the maximum size of packets that the link can transmit. For instance, the MTU of Ethernet is 1500 bytes. That of SLIP is 512 bytes. Typical transmission rates range from 9600 bps for a slow modem to 622

Mbps for a fast ATM LAN. This rate is 1 Gbps for the gigabit Ethernet. Some specialized links and



TCP/IP architecture.

figure 1.3

high-speed LANs have larger transmission rates. The PER is very small for most wired links (a small fraction of 1 percent). For wireless links, this PER can be larger and losses are often bursty. That is, errors. tend to occur in groups.

1.1.1 Network Layer

The network layer of an internet, the *Internet protocol* (IP), implements the end-to-end delivery of packets of up to 64 kbytes. IP supervises the addressing of nodes and the routing of packets. The nodes of an internet use ICMP (Internet control message protocol) to supervise the delivery of packets.

The routers route the packets one at a time. Each packet carries its full source and destination addresses. The routers maintain routing tables. The router checks its routing table to find the next link for the packet based on its destination address. This form of routing is called *datagram* routing. When an IP packet is larger than the MTU of a link, IP fragments the

packet and transmits the fragments one at a time. The destination host reassembles the packet from the fragments.

One major design objective of ARPANET, which evolved into the Internet, was that the network should be robust. Thus, router failures should have minimal consequences. Accordingly, the routers should remember as little information as possible, for that infor- mation is lost in case of failure. Since a packet carries its destination address, a router can determine where it should send that packet next by consulting a map of the network, without requiring a memory of previous packets.

In a virtual circuit network, after a connection is set up, the packets carry only a virtual circuit number. The router must remember how it must handle packets with a given virtual circuit number. We say that an ATM switch has a state ând that an IP router is *stateless.* Note that the terminology is somewhat oversimplified, since the IP routers remember a routing table. The point of this distinction is that the IP router does not keep track of individual connections as an ATM switch does.

Thus, one design rule of TCP/IP networks is to minimize the information that the network elements maintain to make the network robust.

As an analogy, if a mailman discovers a misplaced bag of envelopes, he can resume the delivery of the envelopes without having to check any records. All the information he needs is on the envelopes. Similarly, if an envelope is mistakenly sent to Paris, France, instead of Paris, Texas, the error can be corrected without having to trace a paper trail.

1.2 Transport

The transport layer supervises the end-to-end delivery of packets. The two end systems (the two ends of a transmission) implement the transport layer. The IP routers are unaware of this layer.

The Internet protocol suite implements two transport layer protocols: the transmission control protocol (TCP) and the user datagram protocol (UDP). UDP delivers packets from source to destination. The only error control that UDP provides is that it checks whether the packet arrived correctly and discards it if it did not.

TCP is a more sophisticated transport protocol than UDP TCP performs two main tasks: controlling errors and controlling the flow of packets. TCP controls errors by arranging for erroneous packets to be retransmitted. It controls the flow by slowing down the transmissions when it detects congestion. Thus, the end systems perform the important tasks of error control and congestion/flow control, not the IP routers. This choice reflects a design principle: "Do not ask the network to do what you can do yourself."

1 This principle is consistent with what we learned in the network layer: keep the network as simple as possible and let the end systems perform most of the tasks. This approach makes the network as scalable as possible. Since the internal network elements are unaware of connections, the number of connections can grow without bounds: As more end systems are attached, they can manage the additional connections.

This "keep it simple" principle is prevalent in the design of internets. One interesting question in network engineering is whether this principle might not ultimately limit the capabilities of **TCP/IP** networks in an unacceptable way.

1.3 Applications

The applications layer implements information delivery services and accessory services that user applications need. We briefly comment on a few representative applications.

FTP

Three basic operations are şupported by file transfer protocol (FTP): A user on a machine can send a file to another computer, get a file from that computer, and transfer files between two remote machines. FTP is usually used interactively. It provides a large number of options for creating, changing, or consulting a remote directory, deleting and retrieving a remote file, choosing the transfer mode (stream, block, or compressed), and sending files.

The stream mode is used by default. The file is sent without modification. The block mode partitions the file to be transferred into blocks. This mode is used to simplify data recovery in case of an error. Finally, the compressed mode is used to avoid sending long strings of repeated characters (e.g., spaces). This mode uses the Lempel-Ziv algorithm.
FTP uses two TCP connections: one for the commands/responses and the other for the data transfers/acknowledgments. A host has an FTP process constantly running and ready to process commands. These commands reach the machine on a TCP connection, using a special port number (21). An FTP request from another machine may require the user to be authenticated by a password.

SMTP

The simple mail-transfer protocol (SMTP) is used to transfer electronic mail messages between hosts. A mail server process is always running, ready to handle messages.

SMTP accepts a message from the user along with a list of destinations. A copy is then sent to each destination, except when different users are on the same host. In that case, the message is sent only once, together with the list of destinations on the corresponding host. When the delivery of a message is not successful, SMTP will attempt delivery a number of times on successive days before giving up and indicating failure to deliver it to the user.

TELNET

TELNET is the virtual terminal protocol of Internet. It enables a user to simulate a direct connection from a terminal to a remote host.

This function is implemented by defining a standard character code for the network. In TELNET, this is ASCII (the usual symbols plus a set of control codes). This standard code corresponds to *a virtual ter ninal* that all the hosts are able, it is assumed, to interact with. This interaction takes place by converting the characters sent by the actual terminal into the network standard. Similarly, the virlual terminal input is translated into the input expected by the host.

The transmission is done by means of TCP. Special commands can be sent as expedited data so as to bypass queued data.

The TELNET virlual terminal is a basic scroll-mode terminal. That is, when the end of a line is reached, a new line starts and the others move up, as in most video displays. Many screen commands such as "home" and "clear screen" are not supported. The selection of these basic terminal

actions guarantees that most existing terminals can support the features of the virtual termitial. The disadvantage is that it limits the capabilities of the terminal. In order to make the connection of more sophisticated terminals possible, TELNET has an *option negotiation* phase that allows users to agree on a set of options to be supported by the virtual terminal. For instance, the *echo* (i.e., displaying the sender's commands on the sender's screen) can be handled either locally or by the remote machine. In the latter case, each character is sent in a separate TCP packet to the remote machine and is then sent back for echoing. In the case of local echoing, the characters can be sent together in one packet. The *sizes* of the output (line length, page size) can also be negotiated.

rcp

The *remote file copy* (rcp) command is used to copy files between machines. It can also be used to copy files from one remote machine to another remote machine. An option allows the user to copy all the files of a subdirectory.

rsh

The *remote shell* (rsh) command is used to execute a command on a remote machine and to see the results on the local output. Thus, *rsh* first connects to the remote machine, then sends the command to the machine, returns the result of the command, and finally terminates the connection when the command is executed.

rlogin

rlogin is the remote log in command. It is used to connect the local terminal to a remote host. The remote host will verify that the user is authorized to log in and will then execute the log in without asking for a password.

The echoing is done by the remote machine. Special commands to control the flow of data (the "stop" S and the "resume" Q commands) are sent as expeditsd data.

TFTP

The *trivial file transfer protocol* enables the transfer of files between two processes over UDP A file to be transferred is decomposed into blocks of up to 512 bytes. Each block is sent as a UDP packet together with a block number to enable the receiver to reassemble the file. The blocks are acknowledged by the receiver. The sender retransmits blocks that are not acknowledged before a timeout.

HTML

HTML is *a hypertext marking language.* Figure 1.4 shows the a simple HTML source code of a page and, to its right, a screen capture of that page. The source code is almost self- explanatory. Note that the link "http://www.eecs.berkeley.edu/~wlr" specifies the protocol (http), the server (www.eecs.berkeley.edu), and a document (here indicated by the link wlr).

HTTP

The *hypertext transfer protocol* HTTP uses TCP to get documents from a server. You use HTTP when you click on a hypertext link on a web page using a web browser.

The sequence of operations is as we explained in our discussion of TCP: Your web browser is the client and the remote machine indicated by the link is the server. When you click on the link, HTTP opens a connection with the server (with a three-way handshake), the server checks for authorizations and asks for your password if necessary, then the server sends you the requested document and closes its connection to the client, and then the client closes the connection to the server.

Future versions of HTTP will enable a user to start a few connections at a time and will also enable the same elient/server to use the same TCP connection for multiple documents. Such modifications will improve the utilization of the network links.

RTP

RTP, *a real-time transfer protocol,* is designed to transmit audio and video over the Internet with little latency. The source encodes the video or audio signal and compresses it. The source then places that signal into packets whose header specifies the compression algorithm

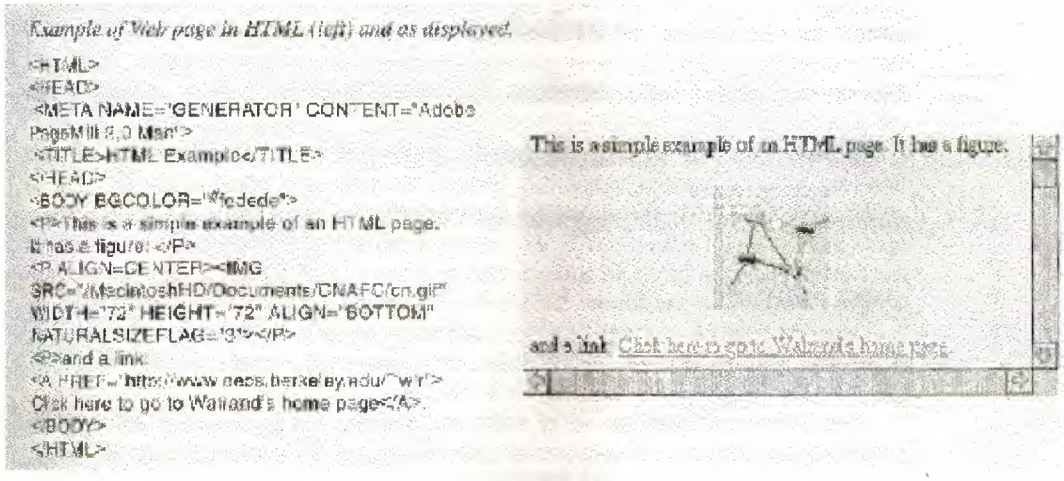


figure 1.4

and the time when the packet was produced. The receiver buffers the packets to absorb the delay jitter in the network, decompresses them, and plays them back at the appropriate time.

Note on Java

Java is a programming language that was designed at Sun Microsystems. The initial motiva- tion was that Java should be a lightweight programming language that can run on small con- trollers embedded in consumer appliances as well as in powerful desktop computers. To be universal, Java

runs on a virtual machine that is simulated by the actual computer. In this way, the same code can be executed on all computers, independently of their operating system.

The importance of Java in network applications is that it could potentially resolve the compatibility problem that new applications face. A document that requires a specific viewer can be sent with its viewer written in Java so that all receivers can view it.

2.DOMAINS AND ADDRESSES

To ensure that packets arnve at the correct destination, the network must know the location of a destination before selecting a path leading to it. Eeach node has a unique network address and a unique name. In the Internet, the name has a hierarchical structure based on that of the name-granting authorities; the network addresses have a hierarchical structure that is geographical.

2.1

Names

The network-more precisely, a server in the network-must translate the destination name into a network location. To find the network address of a destination, the sending computer consults a directory, as you do to find a telephone number. The organization of the auto- matic directory service for the Internet is similar to that of telephone directory assistance. Telephone operators are responsible for the information in one area. To learn a telephone number anywhere in the United States, you need only know the area code, say XYZ, of that number and call the directory assistance of that area (1-XYZ-555-1212). Thus, the directory database is partitioned by area codes. This partition localizes the assistance calls and simplifies the updating of the database.

Similarly, the host names in Internet are grouped into *domains.* Note that the division into domains is not geographical. Instead, the division is based on the hierarchical organization structure of the authorities that supervise the naming of nodes. When a host needs the address of some other host, it places a request to a local *name server* process that runs on a computer whose address must be known. This process checks whether the node name belongs to the domain for which it is responsible. If so, the server replies with the requested network address. Otherwise, the name server determines which other name server is responsible for the name's domain, and it forwards the request to that other server. When it gets the address, the host caches it for future use.

This decomposition of the set of nodes into domains can be extended further by de- composing the domains into subdomains and subdomains into subsubdomains, and so on.

Such a decomposition is called *hierarchical naming.* The structure of the Internet names is shown in the left part of Figure 1.5. The name space is divided into domains and each domain into subdomains. The domains are .com (private companies), .edu (educational in- stitutions), .gov (governmental agencies), .int (international organizations), .mil (military), .net (network service providers), .org (nonprofit organizations), and countries such as .be (Belgium), .ca (Canada), .ch (Switzerland), .fr (France), .in (India), .jp (Japan), **aild** .uk (United Kingdom). For instance. berkeley.edu is the subdomain "Berkeley campus" of the domain "educational institutions:' The subdomains can be further divided. For instance, eecs.berkeley.edu is the collection of names in the Electrical Engineering and Computer Sciences Department of berkeley.edu. The names are managed by the Internet Assigned Number Authority (IANA), which distributes the allocation authority to three organizations called

Internet registries (IRs): RIPE for Europe (http://www.ripe.net), APNIC for Asia and the Pacific (http://www.apnic.net), and InterNIC for the United States and the rest of the world (http://www.internic.net). To connect to the Internet, a company asks its service provider for a collection of names. The provider in turn gets the names from its IR. Each domain is equipped with *a domain name server* (DNS) which maintains the list of Inter- net addresses of its names. In late 1996, the Internet Engineering Technical Forum was considering adding new domain names.

As an illustration, let us examine how an application such as email or WWW run- ning in a computer A in France finds the Internet âddress of computer B with name diva.eecs.berkeley.edu. (See the right part of Figure 1.5.) Computer A first searches a cache for diva.eecs.berkeley.edu. If that cache contains the IP address of that computer, the procedure is terminated. Otherwise, A asks its local name server, say C, for the address. Computer C asks the root name server (duplicated in France) for the address of the name server, say *D,* of berkeley.edu. Computer C then asks *D* for the address of the name server, say E of eecs.berkeley.edu. Finally, C asks E for the address of B and it gives that address to A.

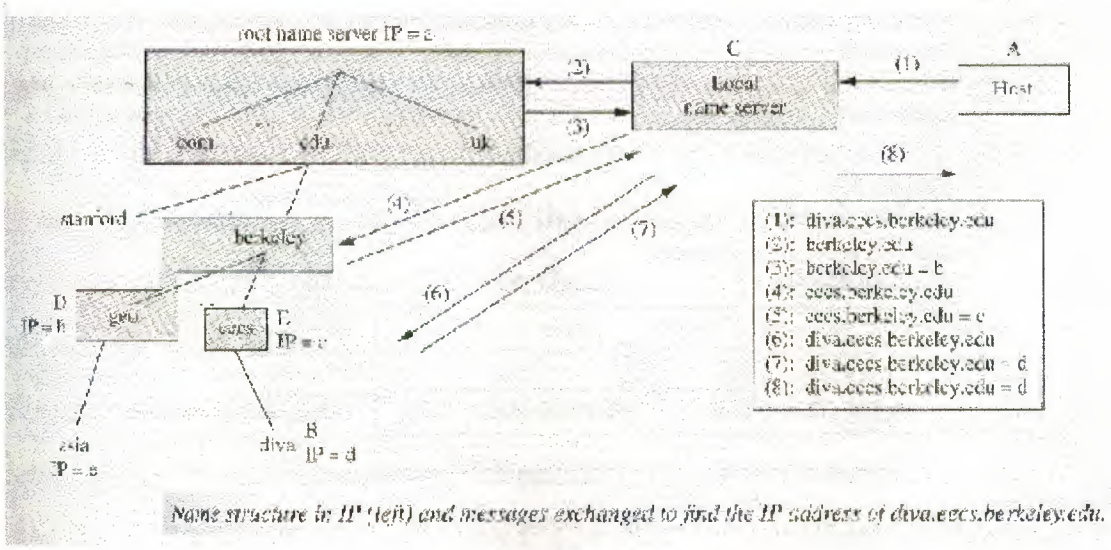


figure 1.5

network A resource (a file to be accessed with a specific protocol) is identified by a resource identifier. The most common resource identifier is the URL (uniform resource locator). The URL specifies the protocol (e.g., FTP or HTTP) and the location of the resource. The location is the network address the computer and the path name of the file in the directory of the computer. For instance, http://www.eecs.berkeley.edu/"wlr is the URL of my home page. In this URL, http specifies that the protocol to be used is the hypertext transfer protocol, www.eecs.berkeley.edu is the name of the WWW server of the EECS Department, and "wlr is a link to my homepage.

2.2 Addresses

Addressing in Internet has gone through three steps: class-based addresses, subnetting, and classless addressing. We explain and justify these three steps.

17,18,19,22.3 Class-Based Addressing

The class-based Internet addresses have the form "network.host:' For

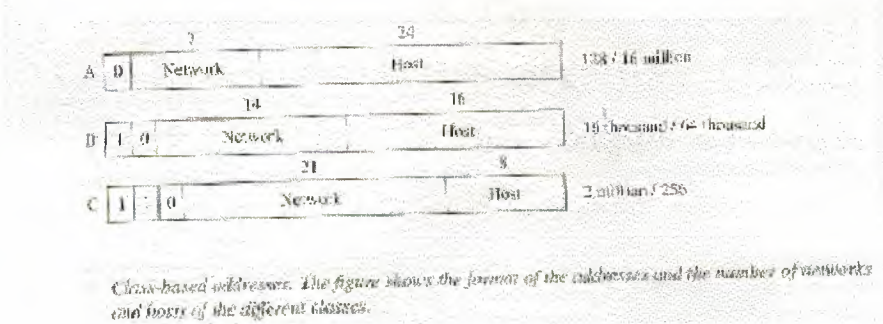
corresponds to two successive 8-bit words with decimal values 128 and 32 (thus, 1000'0000'0010'0000). Similarly, the address of U.C. Riverside is 192.31.146. The rule is that small networks have large addresses, since fewer bits are needed to discriminate the few hosts on such networks, while large networks have small addresses; the balance of the address bits are used for the host address.

There are three classes of networks shown in Figure 1.6: large, or class A (network address = 8 bits, host address on network = 24 bits); medium, or class B (16, 16); and small, or class C (24; 8). A fourth class of addresses, D, is reserved for multicast groups.

One problem with these address classes is that they correspond to network sizes that are not well matched to what users need: class A networks are huge and class C networks are very small; there are almost no more free class B addresses. One partial solution to this problem is subnetting, which we explain next.

2.4 Subnetting

Subnetting enables a large network, say class B, to be split into subnetworks. With subnet- ting, five 2000-node subnetworks can share to the same class B address instead of using five different class B addresses. For instance, the U.C. Berkeley network partitions its hosts into subnetworks; the subnetwork is indicated by the third byte and the host on that network is



Class-based addresses. The figure shows the format of the addresses and the number of networks and hosts of the different classes.

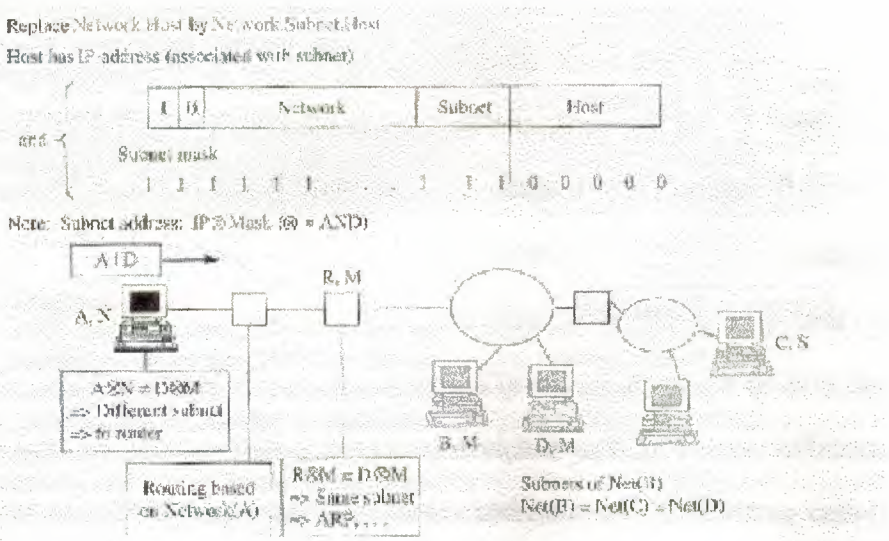
figure 1.6

identified by the last byte. Thus, 128.32:134 is a particular subnetwork on the U.C. Berkeley network and 128.32.134.56 is a computer on that subnetwork.

When subnetting is used, a host is given an IP address and a subnet mask, as shown in Figure 1.7. For instance, the computer 128.32.134.56 also has the mask 255.255.255.0, that is, 24 ones followed by 8 zeros. By looking at the first bits of the address (100..), one finds that this address is of class B and, consequently, that the network part of the address is 128.32. In addition, by using the mask, one finds that the subnetwork of that computer is 128.32.134.

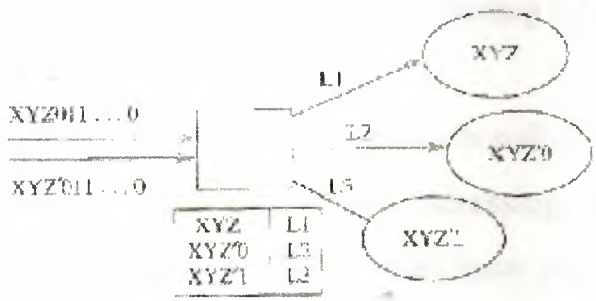
Figure 1.7 shows a packet that goes from a host with IP address A and mask N to a host with IP address D and mask M. To send that packet, host A compares A ® N and D ® M. If these values are different, then A is not on the same subnetwork as D. Accordingly, A must send the packet first to the router of the subnetwork A ® N. The router finds the route to reach D by checking the entry of the routing table that corresponds to the network part of the address D. Eventually, the packet reaches the router R. Since R ® M = D ® M, the router knows that D is on the same subnet. Router R then consults its address resolution protocol (ARP) table, or uses ARP, to find the media access control (MAC) address of D and sends the packet to D over the LAN. For instance, assume that computer A with address 128.32.152.26 and mask 255.255. 255.0 needs to send a packet to computer B with address 128.32.134.56. Computer A finds out that 128.32.152.26 ® 255.255.255.0 = 128.32.152:0 is not the same as 128.32.134.56 ® 255.255.255.0 = 128.32.134. Thus, computer A knows that it is not on the same subnetwork as computer B. Computer A then sends the packet to the router of its subnetwork.

This subnetting procedure uses addresses more efficiently. However, an even better scheme is explained next.



Subnetting. Each host has an IP address and a subnet mask. The subnetwork of the host determined by a bit-by-bit "and"-of the IP address and of the mask.

figure 1.7



Supernetting in CIDR. In this addressing scheme, the router checks the longest prefix matching entry in its routing table.

figure 1.8

2.5 CIDR

To make the addressing more flexible, the Internet designers defined more general prefix- based addresses called CIDR (for *classLess interdomain routing).* The general idea of CIDR is that the addresses are organized as a prefix-free code. That is, the initial segment of an address can define a domain if shortening it does not define another domain. Thus, the initial prefix, say the first 3 bits, could specify the continent, the next 7 bits the country. If the country is France, then the next 7 bits could be the department. Depending on the department, a number of bits then define the town, and so on. If the country is Belgium, then 4 bits suffice to define the province, then a few for the town, and so on. The routing can then be based on the structure of the address.

Figure 1.8 illustrates CIDR. The figure shows a supernet which consists of all the hosts with addresses that share a prefix XYZ. Another supernet corresponds to the prefix XYZ'0, and a third supernet corresponds to the prefix XYZ' 1. The entries of the routing table correspond to the longest matching prefixes.

The 32-bit addresses are in principle sufficient to accommodate about 4 billion hosts. With class-based addressing, the utilization of the addresses was very inefficient and we would have run out of addresses by this time. Internet experts predict that with an efficient use of CIDR (which requires renumbering of hosts), the 32-bit addresses may be sufficient for the foreseeable future.

The next version of IP uses 128-bit addresses. Its main motivation is to overcome the limitations of the 32-bit addresses. It is possible that CIDR will make the 128-bit addresses unnecessary, thus eliminating the principal justification for IPv6.

3. INTERNET PROTOCOLS

The Internet protocol is the main network layer protocol of an internet. This protocol supervises the routing of packets to their destination. The network nodes exchange control packets using ICMP (Internet control message protocol) to implement IP We explain the services that IP provides in Section 3.4.1. in patticular we discuss the format of IP packets
The routing in Internet is hierarchical and divides nodes into subnetworks called *au tonomous systems* (ASs) that are interconnected by *border gateways.* An AS is a subnetwork controlled by a single organization, such as a university campus or a company network. Within the AS, the routing algorithm is OSPF (open shortest path first). A number of routers still use RIP (routing information proto- col), a routing protocol based on the Bellman-Ford algorithm, which we explain between ASs, the routers use a routing algorithm called the border gateway protocol **(BGP).** This algorithm, is implemented by a router in each AS. contain additional material on routing algorithms.

3.1 IP Datagrams and ICMP

IP is a datagram protocol. It delivers datagrams of up to 64 kbytes. That is, the layer above the network layer-the transport layer-gives IP a datagram of up to 64 kbytes together with a network destination IP address. If all goes well, IP in the destination node eventually gives the datagram to the transport layer. A few things may go wrong. For instance, the destination may be unreachable or a packet may loop in the network. In such a case, the Internet control message protocol (ICMP) informs the source host. ICMP specifies the format of control messages and when routers should send them. ICMP messages are delivered by IP.
Transmission errors may also corrupt the packet or a packet may arrive at a

router that is full and get dropped. In these situations, IP cannot inform the source of the packets it lost and it is up to the transport layer to take corrective actions.

Next, we explain the format of the header of an IP packet.

IP Header The packets contain an *IP header.* The basic header, without options, is illustrated in Figure 3.12, where each tick indicates a bit position. The *version* field allows new versions of the IP protocol to be installed while the network is operational. The *Internet header length* (IHL) indicates how long the header is. The Service type field specifies the quality of service desired (e.g., low delay, high throughput, high reliability); few routers offer this choice. Identification, Flag, and Fragment offset allow reassembly of fragmented datagrams (see below). The Time to live field indicates how long the packet can still remain in the network. Each router decrements the value in that field by one when it gets the packet, and discards the packet if the value reaches zero (ICMP then informs the source). The Protocol field indicates what higher level protocol is contained in the data portion of the IP packet (e.g., TCP, UDP, or ICMP). The Header checksum is a checksum calculated on the bytes in the IP header. Higher-level protocols must control the errors in their data. The Source and Destination network addresses indicate the sending host and the intended recipient host for this datagram.

3.2 Fragmentation

Links have a maximum size of packets they can transmit. This maximum size is the link's *maximum transfer unit* (MTU).

If a packet that arrives at a router is larger than the MTU of the outgoing link, the router fragments the packet. The destination host reassembles the packet. Thus, a packet may be fragmented a few times between the source

and destination. The specific numbering scheme of the fragments makes the reassembly procedure straightforward and independent of the number of fragmentations. Looking at the IP header in Figure 1.9 we see that the

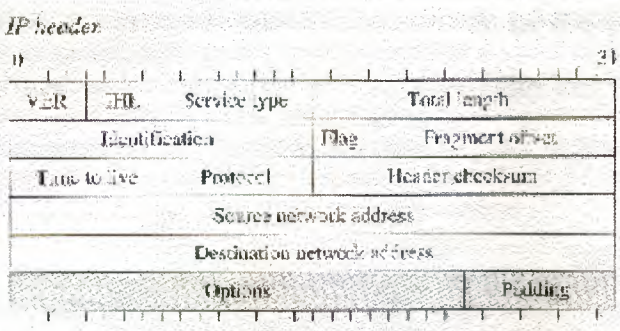


figure 1.9

fragments are marked with the identification number of the original datagram and with the offset of the fragment relative to the first byte of the datagram.

We illustrate the procedure in an example. Consider a datagram with identificatior number X and length L. Assume that a router first fragments this datagram into four pieces with lengths *K, K, K, K'*, respectively. Here, *K* is the MTU of the outgoing link of the router that fragments the packet and $K' = L - 3K <\_ K$. The first fragment is marked with *(X,* 0), the second with *(X, K),* the third with *(X, 2K),* and the fourth with (X, 3K) to indicate the identity X of the original datagram and the offsets of the fragments. Now assume that these fragments reach a router with an outgoing link that has an MTU equal to *M* where $M < K$. Let us assume that $2M < K = 2M + M' <$ 3M. The third fragment-that marked with *(X, 2K)* after the first fragmentation-get decomposed into three fragments. These fragments have lengths M, *M, M'* and are marked with *(X, 2K), (X, 2K -I- M), (X, 2K + 2M),* respectively. Two points should be noted (1) the router can figure out the offsets of the new fragments from the original offset anc the new MTU

and (2) the destination has no problem reassembling these fragments.

OSPF

The open shortest path first algorithm is based on the shortest path algorithm of Dijkstra This algorithm constructs the shortest path from one node of a graph to all the other node of the graph. The length of a path is defined as the sum of the lengths of the links along the path.

3.3 A Physical Model

There is an easy way to visualize the operations of the algorithm. Imagine a collection of N balls that are attached to each other by strings. The strings have different lengths. We select one ball and we call it "ball 1:' We want to find the shortest path from ball 1 to each of the other balls. To find these paths we put down all the balls on the floor and we start lifting ball 1 slowly. We call the next ball to rise from the floor "ball 2:' When ball 2 rises from the floor, we have found the shortest path from ball 1 to ball 2: it is the string that attaches balls 1 and 2 together. As we continue to lift ball 1, another ball, ball 3, rises from the floor. The shortest path from ball 1 to ball 3 is either a link from 1 to 3 or is made up of a link from 1 to 2 and another from 2 to 3. Continuing in this way, eventually we have lifted n balls from the floor. When next ball, ball $n + 1$, rises, we have found the shortest path from 1 to n + 1. Eventually, we find the shortest paths from 1 to all the other balls.

3.4 Dijkstra's Shortest Path

The algorithmic translation of the physical process that we just described is called *Dijkstra's shortest path algorithm.* The algorithm is based on the following observation: The distance from ball 1 to ball n cannot decrease after ball n rises from the floor. Consequently, if we keep track of an estimated distance from ball 1 to ball n, we know that this estimated

distance eventually settles to the shortest distance, and that this happens when the ball rises from the floor.

How do we know which ball rises next? Say that ball k rises and let d be the distance from ball 1 to ball k. We update the estimates of the distances from ball 1 to the balls attached to ball *k* as follows. If some ball j is not up yet and is attached to ball *k* with a string of length s, we replace the current estimated distance x from 1 to j by the minimum of *x* and d + s. When we have updated all the neighbors of ball *k,* we must find the ball that will rise next. That ball is the ball on the floor, say ball p, with the current smallest estimated distance away from ball 1. We can then continue the process with ball p.

We first explain the algorithm on the simple network shown in the left part of Figure 1.10. The objective is to find the shortest paths from A to all the other nodes. The lengths of the individual links are marked next to them. On such a small network, a simple inspection shows that the shortest path from A to the bottom node has length 5 and goes through the right middle node: Dijkstra's algorithm is a systematic procedure for discovering such a shortest path even in a large network.

Next to each node, we mark the current estimate of the length of the shortest path from A to that node. The symbol of infinity means that no path to that node has been found yet. The algorithm starts by considering the node with the smallest label, in this case A. The algorithm explores the links going out of that node. The left link leads to the left middle node which can then be reached from A with a path of length 3. Accordingly, we reduce the label of that node from infinity to 3. Since the label is reduced by using that left link out of A, we mark the link as being the current candidate for the shortest path to the left middle node.

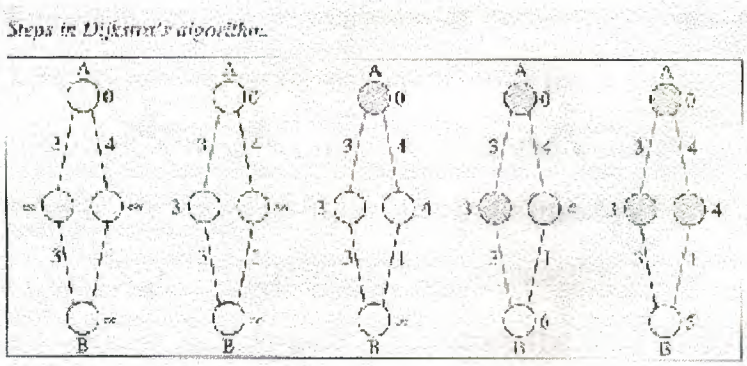


figure 1.10

A similar step is performed for the right link out of A. We then shade node A to indicate that we have examined all its outgoing links. At the next step, we examine the unshaded node with the current smallest label. That node is the left middle node. We examine the outgoing link of that node and find a candidate shortest path to the bottom node with length 3 + 3 = 6. We shade that node and then examine the unshaded node with the smallest label: the right middle node. With the link out of that node, we find a shorter path to the bottom node. Accordingly, we "unmark" the link from the left middle node to the bottom node because we know that the link is not on the shortest path to the bottom node. In addition, we mark the link from the right middle node to the bottom node. Since the bottom node has no outgoing link, the algorithm terminates. The marked links define the shortest paths from A to the other nodes and the node labels are the lengths of the shortest paths from A to the nodes.

Next we illustrate Dijkstra's shortest path algorithm with the example shown in Figure 1.11. We start with node A as the source and we mark each node with an estimated distance from A to the node. The initial

estimates are infinite, except that of node A which is 0. The first node to "rise from the floor" is node A. We shade that node to remember that it is off the floor and we update the estimate of all the neighbors B, C, D of A. For instance, we replace the current infinite estimate of B by the sum of the estimate of node A (equal to 0) plus the length 4 of the link from A to B. The new estimates (4, 3, 2) are underlined in the second part of the figure. At that point, we determine the unshaded node with the smallest estimate. That node is node D and is therefore the next node to rise from the floor; we shade that node D. Note that all the links from node A to its neighbors are colored blue

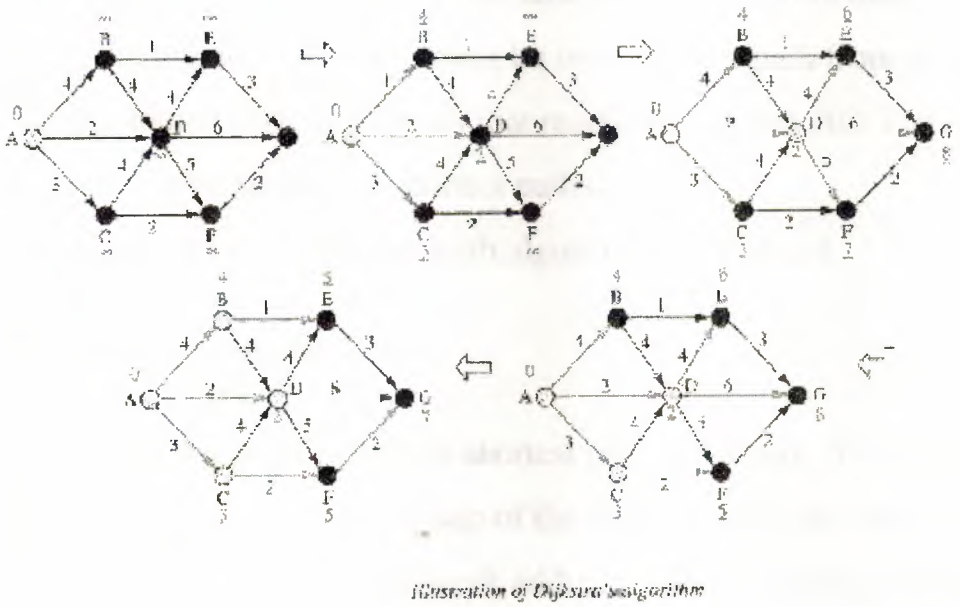


figure 1.11

to remember that these are candidates for shortest paths from A to other nodes since these links have provided us with the smallest estimates of the distances to the nodes B, C, D so far. In the third part of the figure, we

explore the neighbors of node D and we update their estimates and color the links from D to these neighbors blue. In the fourth part of the figure, we have located node C as the unshaded node with the smallest estimate; we shade C and we explore its neighbors and update their estimates. Note that the link CF produces a smaller estimate (6) for node R Accordingly, we change the link DF from blue to black and we color the link CF as blue. In the last part of the figure, we locate the unshaded node B with the smallest estimate; we shade node B and update the estimates of its neighbors. Note that the link from B to D does not reduce the estimate of node D because 4 -I- 4 > 2. However, the link from **B** to E reduces the estimate of E because 4 + 1 < 6. Consequently, to reflect this reduced estimate, we color the link from **B** to E blue and we change the link from **D** to E from blue to black. Indeed, the link from **D** to E cannot be on a shortest path from A to E since it yields a longer path than a path that reaches E via link BE. The left-most part of the figure shows the shortest paths.

We analyze Dijkstra's shortest path algorithm in Section 4.

3.5 Implementation of OSPF

Each router executes Dijkstra's shortest path algorithm. To execute the algorithm each router needs a map of the network with the link lengths. Each router knows its own network address, which is configured when the router is set up. Each router talks to its neighbors to find their network addresses. As the network is in operation, the router calculates a metric for each output link. In the simplest case, the metric is 1 when the link is operating and is infinite when it is not. A more accurate metric takes into account the link transmission rate and the average delay to go through its buffer. The metric is selected by the network administrator and all the routers (in a given AS) use the same metric.

The routers of an AS can also execute Dijkstra's algorithm for a few different metrics. For instance, the metrics can be measures of reliability, transmission rate, and delay. After running Dijkstra's algorithm for each of these three metrics, each router knows the most reliable path, the path with the largest throughput, and the path with the smallest delay to each of the other routers in the AS. A packet can then specify which criterion the routers should use to select its path. We continue our discussion assuming a single metric, but it can be adapted easily to the case of multiple metrics. Router i prepares a message that lists the metrics of its outgoing links. The router appends a sequence number s to the message *M.* Thus, *M* might look like this (we do not explain the actual message formats in bits and bytes):

M = (ilslk1, dilkz, dzl ... *Ikm,* ***dm)·***

In this notation, k1 identifies a neighbor of router i and d1 is the estimated metric of the link from *i* to k1, the meaning of *k2, d2, . . . , km, dm* is similar. Router *i* sends message *M* on each of its outgoing links.

When a router j gets such a message, it checks the original source i of that message. If i = j, then router j discards the packet. Otherwise, it checks the largest sequence number of routing messages from i that it has received to date. If that sequence number is smaller than s, indicating that *M* is a new message from *i,* router j updates that largest sequence number, stores the message for its own use, and sends a copy along each link other than the incoming link of the message. Eventually, the message is sent once to all the reachable nodes. This distribution method to all the other routers is called *flooding.*

As the router gets messages, it can construct a map of the network and execute Dijkstra's algorithm. A router knows when it gets an updated message by checking the sequence number and it then knows that it should

run the algorithm again. A router can send a new message when a link metric has changed significantly enough.

3.6 BORDER GATEWAY PROTOCOL

The routing between ASs is based on the *border gateway protocol* **(BGP).** In contrast with OSPF, BGP is a distributed protocol. That is, the routers that execute the protocol have different information about the network. Another key difference with OSPF is that BGP is *a preferred path* algorithm instead of being *a metric* algorithm, as we explain below.

A First Look

Let us start with a rough sketch of how BGP works: We then refine that sketch. Say that the routers want to construct a path to some destination host **D. At** some step of the algorithm, some of the autonomous systems have identified preferred paths to **D.**

Consider an AS X that is attached to the AS Y and the AS U. (See Figure3.15.) The AS Y advertises to X the path that Y prefers to reach D by sending the message, [Y, X, U; 17]. In this message, (Y, X, U) is the path that Y prefers to reach D and 17 is the estimated metric of that path, say the expected delay (in milliseconds). Similarly, the AS U advertises its preferred path to D by sending the message [U; 18].

When it gets these messages, X must select the path is prefers to reach D. Assume that X chooses to send a packet destined to D to Y because the advertised metric of that path is smaller than that advertised by U. When Y gets the packet; it sends it along its preferred path. Accordingly, Y sends the packet first to X. Consequently, the packet will loop between X and Y However, if X examines the path advertised by Y, it notices that this path goes throngh X, so that X should not choose that path. Thus X will select

the path going through U as its preferred path and it will advertise it as [X, U; 23], where the metric

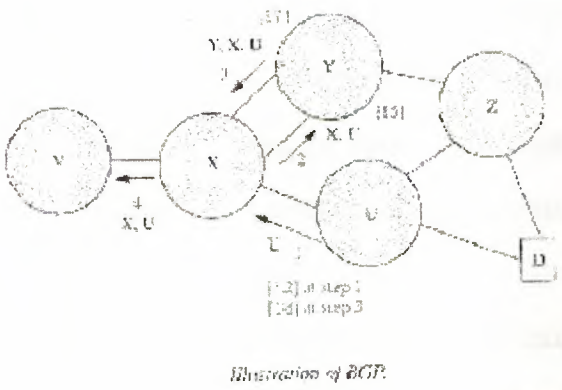


figure 1.12

23 is obtained by adding to 18 the metric of the connection from the entry of V into X to the exit of X toward U (which we assume to be 5 here). This metric (5) is internal to the AS X and is derived by the routers of X running OSPR The metrics that the different ASs use do not need to be identical. However, the metrics through an AS used to update the BGP path metrics should be consistent: a longer path inside X should have a larger metric. Note that the metrics that Y and U advertise to X are inconsistent with the topology of the paths. Indeed, since the path from Y to D goes first through X then through U, its metric should be larger than the metric of the path from U to D. Such inconsistencies may result from dated estimates. The figure illustrates a sequence of updates that leads to inconsistent estimates. At step 1, U advertises a metric equal to 12 to reach D. At step 2, X propagates that estimate to Y At step 3, Y uses that estimate to calculate the metric 17 that it advertises to X. At the same time, step 3, U updates its estimate of the metric to D and advertises the new estimate (18) to X. The increase in the metric is due to a buildup of congestion inside U that Y is not aware of yet at step 3.

3.7 Implementation

An AS may be connected to other AS s with more than one border router. One router inside the AS is selected by the network manager to execute BGP. That router is called the BGP speaker. As in OSPF, BGP can use a number of different metrics. In addition, each AS can maintain lists of ASs that it does not want to send its packets through. For instance, some ASs may belong to a commercial competitor and security or reliability might be doubted.

Note the economy of information exchange in BGP Instead of flooding the network, each AS sends only short messages to its neighbors. Distributed algorithms run the risk of creating loops. We saw in the previous section that if X bases its path selection only on the advertised metrics, then the packets it sends enter a loop. Such a loop is created when the nodes have inconsistent estimates of metrics. BGP prevents loops by advertising the paths in addition to their metrics.

3.7.1 Plug and Play: DHCP

A number of study rooms and lecture rooms on our campus are equipped with Ethernet outlets. A student can plug a laptop computer into the Ethernet outlet and start using electronic mail, Web browsers, and file transfer applications. The protocol that makes such connections possible is called the dynamic host configuration protocol (DHCP) and is known under the descriptive name of plug and play.

Remember that to send a packet on the Internet, a computer must first have a network address. This address is associated with the location of the computer, specifically with the network to which the computer is attached. In our example, the laptop computer must find the Ethernet router and ask for a network address. The laptop computer sends a special broadcast

(DHCP) message asking "Router, give me a network address:' The router maintains a pool of free network addresses and gives one to the laptop with a specified time to live (say 20 minutes). The laptop can then use the standard Internet applications. As the time to live gets close to zero, the laptop asks the router for a new 20 minutes, which is normally granted. If the student unplugs the laptop, the network address returns to the pool of free addresses when its time to live expires because the laptop does not ask for an extension. The same mechanism can be used with a wireless Ethernet.

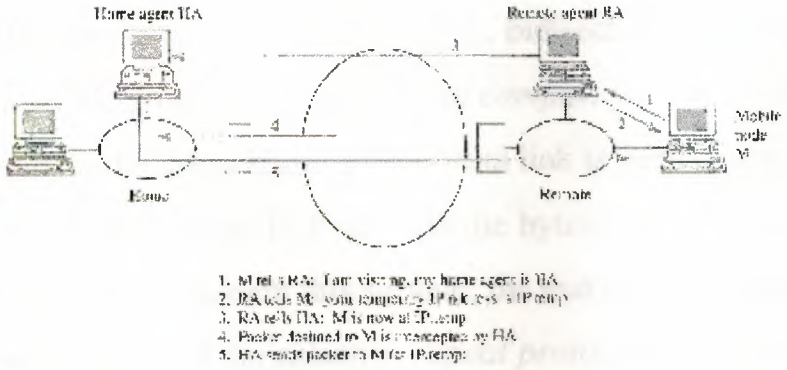


figure 1.13
illustration of mobile IP.

3.7.2 *Mobile IP*

The DHCP that wejust explained can be combined with another protocol so that the student's email is automatically forwarded to the laptop. This protocol, mobile IP, works as follows The student is assumed to have a home router where the computer is normally attached When the student moves to another location, the computer automatically registers with ac agent on the network the student is visiting. That agent gives the student the temporar5 network address that we saw in DHCR.

4. End to end transmission

We learned that the network layer implements the end-to-end delivery of packets. The transport layer supervises two aspects of this delivery: error and congestion/flow control.

4.1 Errors

Errors occur in the delivery of packets for two reasons. First, transmission errors corrupt packets. Second, a router discards a packet that arrives when the buffer is full.

Two versions of the transport layer are implemented in the Internet: TCP the *trans- mission control protocol,* and UDP, the *user datagram protocol.* TCP establishes a full-duplex (i.e., bidirectional) virtual byte link between two com- puters that we call *end computers* to recall that they are at both ends of the connection. That virtual link is perfect, except for the delays of the bytes. That is, TCP delivers the bytes without errors and in the correct order. To implement this service, the end computers use a retransmission protocol called the *selective repeat protocol* (SRP) that we explain below. UDP delivers datagrams and makes no attempt to retransmit erroneous packets.

4.2 Congestion and Flow Control

Flow control is the procedure a source uses to adjust its transmission rate so as not to overwhelm the receiver. In TCP, the receiver tells the source the number of packets it is willing to receive.

Congestion control is the mechanism that sources use to limit the congestion in nodes of the network. In TCP, the sources use the delays of acknowledgments as indicators of congestion and adjust the window size of the retransmission protocol accordingly, as we explain below.

4.2.1 Retransmission Protocol

The selective repeat protocol attempts to retransmit only the packets that do not get correctly to the destination, either because of transmission errors or because they are dropped by a full router.

The source and destination computers implement SRP. When the source sends a packet, it starts a timer. The destination sends back an acknowledgment for every correct packet that it receives. The source assumes that a packet fails to reach the destination correctly when the acknowledgment does not come back within some time called the *timeout.* The source then retransmits a copy of that packet.

More precisely, the source uses a window size W for SRP The source sends the packets 1, 2, . . . , W and waits for the acknowledgment of packet 1 before it sends packet W + 1. If everything goes well, the source gets back the acknowledgments of packets before a timeout and it can keep on transmitting. If the acknowledgment of some packet i fails to arnve within a timeout, the source retransmits that packet.

When it gets a correct packet, the destination sends an acknowledgment with a sequence number equal to K + 1 if the destination has received the packets 1, 2, . . . , K but has not received K+1. Thus, if the destination receives the packets 1, 2, . . . , *K, K+2, K+3, K+4,* then it sends the acknowledgments with sequence numbers 2, 3, . . . , K + 1, K + 1, K + 1, K + 1. In such a situation we say that the last three acknowledgments are *duplicate acknowledgments.* The destination delivers the packets 1, 2, . . . , K that it got in order and stores the packets *K + 2, K + 3,* K + 4 to be able to deliver them after packet *K* + 1 that it is waiting for. When the sender sees three duplicate acknowledgments, it assumes that packet K + 1 has been lost and it retransmits that packet, without waiting for a timeout of the acknowledgment K+1. This mechanism is called fast retransmit. Section

6.11 explains the protocols in more detail.

4.3 TCP

TCP implements a full-duplex byte link between two end computers. The virtual link transports streams of bytes. The application that uses TCP injects a stream of bytes and listen to the stream coming toward it. The stream is error-free, at least as long as the connection stays up. A computer may have a number of connections open at the same time. These TCP connections are differentiated by a port number. Some applications are always listening to the same port number. For instance, the electronic mail process listens to port number 25. Other applications use a temporary port number that is allocated by a "port server" when the application is invoked. Figure 1.14 shows how an application sees TCP.

We view TCP as setting up two connections: one from the client to the server and one from the server to the client. Here are typical steps in a TCP connection.

I . The client starts up a duplex connection with the server and requests a document.

2. The server sends the document to the client.

3. The server closes the connection to the client. 4. The client closes the connection to the server.

This sequence of events is shown in Figure 3.18. The sequence in the figure shows the acknowledgments that are needed because the network is not fully reliable. When the server sends data to the client, the server sends a stream of bytes to TCP These bytes enter a buffer and TCP puts a group of these bytes into a segment that it gives to IP when the buffet is full. (See Figure 3.17.) The size of the buffer is MSS, for maximum segment size.

MSS can be specified as an option at the start of a connection. The default value of MSS is 536 bytes for most links; it is 1460 bytes for an

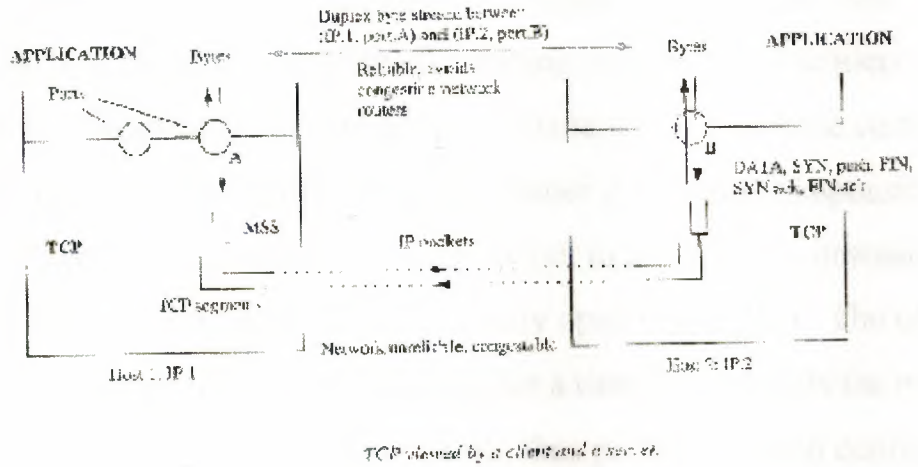


figure 1.14

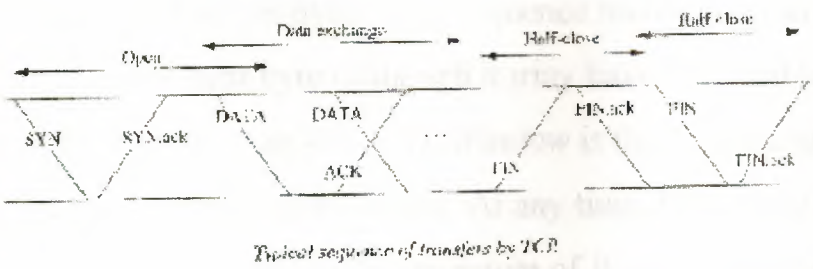


figure 1.15

Ethernet and 256 bytes for SLIP The source can issue *a PUSH* command to TCP to avoid having to wait until the buffer is full. When it gets such a command, TCP puts whatever bytes it has in its buffer into a segment it gives to IP.

Note the control messages (SYN, SYN.ack, FIN, FIN.ack) that the two end computers exchange: These control messages are contained in the TCP header of the packets. This header has the following components:

[S.port ~ D.port ~ Seq ~ Ack ~ FLAG ~ Window ~ CKS ~ URG]

In this header, *S. port* and *D. port* are the source and destination port numbers. *Seq* and Ack are the packet and acknowledgment sequence numbers. Specifically, the source computer numbers the bytes it sends (in the byte stream) consecutively, starting with an initial sequence number that is determined by a clock-driven counter in the source computer. The objective of this initial sequence number is to make it impossible for a delayed packet in transit in the network to arrive at the destination and be mistaken for a packet of the currently open connection. The clock-driven sequence number wraps around after a time that exceeds the maximum lifetime of a packet in the network, thus preventing such confusions. *Seq* is the sequence number of the first byte in the packet and *Ack* is the sequence number of the next byte that the destination expects. That is, the destination has received all the bytes with sequence numbers up to *Ack* - 1 but has not received the next byte (although it iriay have received bytes with sequence numbers larger than *Ack + 1). Window* is the window size that the destination is willing to accept. At any time, the source must use SRP with a window size equal to the minimum of *Window* and the window size *W* calculated by the slow start and congestion avoidance algorithm. *CKS* is a checksum that the source computes on the full TCP packet, including the data. *URG* is a pointer to the last urgent byte in the stream. The *FLAG* indicates whether the packet is a start of connection *(FLAG* = SYN), the acknowledgment of a start of connection *(FLAG* = SYN.ack), an urgent packet *(FLAG = Urgent),* a push command *(FLAG = PUSH),* an end of connection *(FLAG* = FIN), the acknowledgment of an end of connection *(FLAG* = FIN.ack), or *a reset* to foree the termination of a connection *(FLAG = reset).*

The protocol speci6es what happens when an acknowledgment fails to

arrive. Without going into all the details, here are the main points:

· The start of the connection is based on *a three-way handshake* and proceeds as follows. The client asks for a connection to the server by specifying the client's IP address and port number, the server's IP address and port number, and the initial sequence number of the connection based on a clock register in the client. The server sends back an acknowledgment with the same addresses and port numbers and sequence number. The client then sends the first data packet with the initial sequence number. This three-way handshake prevents the client and server from being confused by unrelated delayed packets that might reach them. Such delayed packets would not have the same sequence numbers.

· If the start is not acknowledged in time, then the client tries again after **d** seconds. If this fails, the client tries three more times, every 24 seconds, and then gives up.

· The data transfer is supervised by SRP

· Each half of the connection is closed by a two-way handshake: "I am closing my connection to you" followed by an acknowledgment.

· After the server sends the acknowledgment of the close of connection from the

çlient to the server, the server stays alive for some time to make sure the client gets the acknowledgment and does not send again its close message, expecting another acknowledgment.

If you want to write client/server network applications, you need to learn how to use TCP and IP Specifically, you need to know how to start a connection, how to exchange data, and how to close a connection. You can find the implementation details in many books (e.g., Stevens, *UNIX Network Programming,* Prentice-Hall, 1990, and Stevens, *TCPlIP*

Illustrated, Addison-Wesley, 1994).

4.4 UDP

The *user datagram protocol* is a datagram communication service built on top of IP It adds multiplexing and error detection to the IP capabilities. As TCP, UDP uses 16-bit port numbers to distinguish source and destination processes.

The UDP header contains 16 error-detection bits which are set to zero when they are not used. The UDP header also specifies the IP source and destination addresses that UDP obtains from IP (which selects the path and, in particular, the network interfaces which determine the source and destination IP addresses). In contrast to TCP, UDP does not use acknowledgments; it does not retransmit erroneous packets or control the flow.

4.5 Complement 1: Link Protocols

In this section we explain two protocols that are commonly used to connect a computer to the Internet with a point-to-point link. Typically, the link is a telephone line and the transmission takes place with modems at each end of the line.

4.5.1 SLIP

SLIP, the *serial line IP,* is a protocol to transport IP packets between computers attached to a common point-to-point link. Many people use SLIP to access the Internet from their home computer either via an Internet service provider or via an office computer.

SLIP starts and ends every IP packet with a special ASCII character *OxCO* called END. If a byte of the IP packet is the END character, SLIP replaces

it by the two characters ESC END where ESC = *Oxdb*. If a byte of the IP packet is the SLIP character ESC, SLIP replaces it with the two characters *Oxdb, Oxdd.*

When the receiver gets the bytes, it detects the start of the IP packet when it sees the first END. When it sees ESC, the receiver knows that this byte should be skipped and it looks at the next byte; if that next byte is END, then the receiver knows that this END is a byte inside the IP packet and does not confuse it with the end of packet; if the next byte is *Oxdd,* then the receiver knows that a byte *Oxdb* was inside the IP packet.

For the two computers to be able to talk to each other using SLIP, they must know each other's IP address. Because the SLIP frames have no field for the frame type, the link can only be used for SLIP Moreover, SLIP frames have no error detection field. Errors must be controlled by higher layers or by the line framing that modern modems introduce.

Most SLIP implementations support a variation called *compressed SLIP* (CSLIP). CSLIP recognizes that many packets on the link belong to the same TCP connection and avoids replicating the headers. To avoid repeating headers, CSLIP maintains the state of up to 16 TCP connections and transmits only the changes in the fields of the headers that change in subsequent packets of a given TCP connection.

4.5.2 PPP

PPP, the *point-to-point protocol,* is becoming increasingly popular and is replacing SLIP in many point-to-point links.

PPP, like SLIP allows IP packets to be transported over a point-to-point link. The link can be either an asynchronous link-a link that transmits one ASCII character at a time-or a synchronous link that transmits large

packets. PPP enables the two devices to negotiate options such as deciding which protocol is used on the link, agreeing on the compression of headers as in CSLIP, and agreeing to skip addresses and control fields of IP packets. In addition, PPP packets contain an error detection field (a 2-byte CRC).

4.6 Analysis of Dijkstra's Shortest Path Algorithm

In this section we give a formal definition of Dijkstra's shortest path algorithm and we analyze its properties. Specifically, we prove that it produces the shortest paths and we analyze its numerical complexity.

We are given a set **S** of *N* nodes and a set of link lengths between these nodes: *{L (i , j), i, j E* **S}.** By convention, *L(i, j)* = infinity if there is no link from *i* to j. We want to find the shortest path from some node a E **S** to each of the other nodes.

We initialize the algorithm by letting n = 1 and defining

bl = a, dl(a) = 0, U(1) = *{a},* F(1) = S-U(1), and *dı(i)* = infinity, *Pı(i) = ld* for *i* E F(1) We then determine

N(n) = {İ E F(n)I L(bn, .i) < infinity}

and

dn+ı(ı) = min{dn(i), *dn(bn) -I- L(bn, i)}* for *i E N(n).***t** (3.2) Moreover,

*Pn+ı(i) = { **Pn(t)*** if *dn < dn(bn) + L(bn, i),*(3.3)

{bn} if dn > *dn(bn) -1- L(bn, i).*

We set i

n=n+1(3.4) and

bn+ı = arg min**{dn+ı(i),** i E F(n)}(3.5)

and we define

U(n+1)=U(n)U{bn+ı}andF(n-I-1)=S-U(n+1).(3.6) Finally, we set n=n+1(3.7) ı and we repeat steps (3.1) to (3.7) until n = *N + 1.*

When the algorithm has executed, *dN (i)* is the length of the shortest path from node *a* to *i,* for *i ~ a.* Also, *(PN (i), i)* is the link that reaches *i* along the shortest path from *a* to *i* .

We now describe the connection between this formal definition and the physical description of Section 3.4.2. The set U(n) is the set of balls that are "up" at step n whereas **F(n)** is the set of balls that are still on the floor. The ball bn rises from the floor at step n, and N(n) is the set of its neighbors that are still on the floor. In step (3.2) we update the estimates of the nodes in N(n) and, if the estimate of node i is reduced, we note in step (3.2) that *(bn, i)* is on the shortest path from *a* to *i.* Indeed, the ball bn that just lifted from the floor yields a shorter path to i than the previously known ones. In step (3.5) we find bn+ı, the next ball to lift up.

Next we prove that the algorithm produces the shortest paths.

4.6 Shortest Paths

We prove that the algorithm produces the shortest paths by induction on n. That is, we show that at each n the algorithm has found the shortest paths to the nodes in U(n) and that these paths have the lengths dn(i) for i E U(n). This statement is certainly true at time n = 1 since the shortest path from node a to *a* has length 0. Assume that the statement is true at time *n.* We claim that it must then be true at time n + 1. To prove this claim, we argue by contradiction. That is, we assume that there is a path from *a* to *f : = bn+ı* that has length strictly less than *dn+ı (f).* We show that this cannot be by considering the last node in U(n) along that path. Call this node j and let *m* < *n* be such that bm = *j.* Let also *k* be the next node along that path to bn+ı.

We have

dm (.1) -I- L(j, k) = dn (j) -I- L (j, k) = dn+ı (k) ? dn+ı (f).

Indeed, if this inequality did not hold, then *k* would have been selected before *f* by steps (3.2) and (3.6). This contradicts the existence of a shorter path to f and concludes the proof.

4.7 Routing Algorithms

The Bellman-Ford algorithm is a distributed algorithm that routers can use to discover the shortest paths to a destination. This algorithm was implemented until 1983 in the Internet under the name of the routing informa6on protocol (RIP). It was then replaced by OSPF because of a few undesirable properties of RIP that we discuss after having described the algorithm and analyzed its main properties.

Description

The problem formulation is identical to that of Dijkstra's shortest path algorithm. One is given a set S of *N* nodes and a set of link lengths between these nodes: *{L(i, j), i,* ***j E S}.*** By convention, *L (i , j)* = infinity if there is no link from *i* to j . The nodes, which represent routers, want to discover the shortest paths to a destination D E **S.**

The Bellman-Ford algorithm is distributed. That is, the routers have different and incomplete information about the graph. At each step of the algorithm, a node computes an estimate of its shortest distance to the destination and advertises that estimate to its neighbors.

To simplify the description of the algorithm, we assume that all the steps are synchronized. We also assume that the links are duplex. That is, we assume that if *L(i, j) <infinity,* then *L(j, i)* < infinity for any *i* and j. Thus, if there is a link from *i* to j, then there is a link from j to i. The algorithm

performs correctly without the synchronization and the duplex link assumptions, as you can verify.

At step 0, node *i* has the estimate xo *(i)* = infinity for all *i* E **S** with *i is not equal to* ***D*** and that *xo(D) = 0.* The nodes advertise these estimates to their neighbors.

At step *n* _> 1, node j updates its estimate of its distance to *D* to some value *xn (j)* and advertises that value to its neighbors. At step *n* + 1, node *k* examines the messages that it just received from its neighbors at the previous step n. For instance, j might be a neighbor of k-that is, *L(k, j)* < infnity and j informs *k* at step *n* that the shortest estimated distance from j to D is equal to *xn(j).* Node *k* gets similar messages from all its neighbors. Node *k* then knows that if it chooses to send a message to *D* by first sending it to its neighbor j, then the distance that the message will travel to reach *D* is estimated to be *L(k, j)+-xn (j).* Indeed, this estimate is equal to the distance *L (k, j)* from *k* to j plus the estimated shortest distance *xn (j)* from j to D as advertised by j at the previous step. Since node *k* can choose the neighbor to which it sends the message destined to D, distance to D as xn+1(k) = ~*{L(i, j) +xn(j), j* E S}, k E **S,** k is not equal to *D.*

This set of *N* - 1 equations, computed in parallel by *N* - 1 nodes, produces the next estimates {xn+1 (i), i E **S,** i ¢ *D}.* These equations are the Bellman-Ford algorithm.

An Example

Figure 3.19 illustrates the steps of the Bellman-Ford algorithm. The network on the left shows the link lengths and the current estimated lengths of the shortest paths from all the nodes to the bottom node.

The objective is to discover the shortest path from all the nodes to the bottom node. The algorithm proceeds from the destination (the bottom

node) to the other nodes. In the first step, shown in the second drawing from the left in Figure 1.16, the bottom node informs its neighbors that its distance to the destination is 0. When they get that information, the middle nodes can update their estimates of their shortest distance to the destination. These estimates correspond to paths that go through the links that we mark thick. At the next step (the right-most drawing), the middle nodes inform their upstream neighbor (the top node) of their estimated distance to the destination. The top node then updates its estimate and the shortest path uses the right outgoing link from the top node. We mark that link thick. The algorithm stops at that step and the marked links are the shortest paths to the destination from all the nodes, with their lengths indicated by the node labels.

Distributed versus Centralized

The implementation of the Bellman-Ford algorithm that we described is *distributed.* This means that the different nodes have incomplete information about the state of the network. It is possible to implement the Bellman-Ford algorithm centrally. To do this, all the nodes broadcast, by flooding, the lengths of their outgoing links to all the other nodes. All the nodes can then execute the Bellman-Ford algorithm on the basis of the full information.

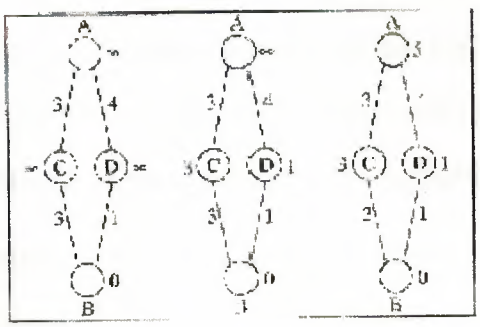


Illustration of the Bellman-Ford algorithm.

figure 1.16

Analysis

We show that this algorithm converges in finite time to the correct estimates. That is, there some $m <$ infinity such that for all $n >= m$, *xn(i)* is the shortest distance from *i* to *D* for all f E **S.**

Consider some arbitrary *i is not equal toD* and assume that there is a path *u* from *i* to *D* with $n >= 1$ links. A moment of reflection shows that at step *n* of the algorithm, node i calculates an estimate that cannot be larger than the length of the path pi. Indeed, at step l, Ç e last node before *D* along *n* gets the message from *D* and calculates an estimate equal to the length of the last link of pi and sends that estimate to the previous node of n. **At** step 2, that previous node calculates an estimate that cannot be larger than the sum of the lengths of the last two links of pi, and so on. Thus, at step *n,* the estimate of node *i* is the minimum of the lengths of paths of up to *n* links from *i* to *D.* Since the shortest path from *i* to *D* is loop-free, the estimate of *i* must reach the shortest distance from *i* to *D* after *m (i)* steps, where *m (i)* denotes the maximum length of a loop-free path from *i* to *D.* Consequently, the algorithm converges in at most *m* steps, where $m = \max\{m(i), i \text{ E } S\}$.

Adaptation

One can show that this algorithm converges for arbitrary nonnegative inital estimates *xo(i), i* E **S** with xo(D) = 0. To see this, consider the algorithm started with the ini- tial estimates *xo(i)* ~ 0 for all *i* E **S** and denote by *{y"(i), i* E **S}** the estimates at time *n* that result from these initial values. You can verify by induction over n that

$yn(i) < yn+ı(i), i \text{ E } \mathbf{S}, n > 0$ (3.9)

and

$V(1)=min\{L(i,J)+V(j),i$ **ES).** (3.11)

Since $V(D) = 0$, these equalities imply that $V(i)$ is the shortest distance from i to D for all i E **S,** so that $V(i) = L(i)$.

Finally, consider arbitrary but nonnegative estimates $xo(i)$ with xo(D) = 0 and denote by $zn(i)$ the estimate of node i at time n that results from these initial estimates. You can verify by induction on n that

$yn(i) < z''(i) < xn(i)$, i E **S,** $n > 0$. (3.12)

Since the estimates x" (i) and $yn(i)$ converge to the same value $L(i)$, it follows that the estimates zn (i) also converge to $L(i)$.

The practical importance of this remark is as follows. Assume that the nodes execute the algorithm and that their estimates have reached some values. Imagine that the situation in the network changes and that this change results in different link lengths. The nodes keep on executing the algorithm and we know from the result of this remark that the estimates will converge to the new shortest distances.

Slow Convergence

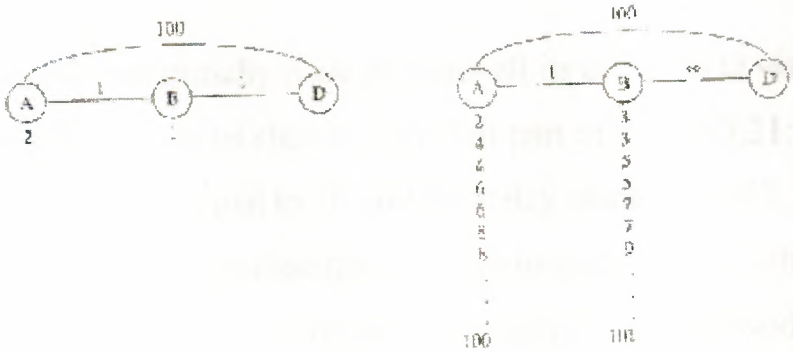
Although the Bellman-Ford algorithm converges to the correct values of the shortest distances to the destination, the convergence may be very slow, specially in the case of a link failure. Figure 3.20 illustrates this slow convergence. In the example shown in the figure, the destination is D and the nodes maintain estimates of the shortest distance to D by running : the Bellman-Ford algorithm.

The left part of the figure shows the estimates after the algorithm has conveiged when , L(A, B) = 1 = L(B, D) and L(A, D) = 100. In the right part of the figure, it is assumed , that the link (A < **D)** fails and has now an

infinite length. The numbers below the nodes A ' and **B** show the estimates that these nodes compute at the successive steps of the algorithm. As you see, it takes 98 steps for the algorithm to converge to the correct values.

Oscillations

Another undesirable property of the Bellman-Ford algorithm is that it may result in oscillations. The example in Figure 1.18 shows what we mean by oscillations and explains the phenomenon. The figure shows a node A attached to two routers Rl and RZ connected to some destination D with two links. Node A has a large amount of traffic to send to D, say B bps. We assume that the delay on link (R1, D) and on link (R2, D) is equal to 1 when almost



Slow convergence of Bellman-Ford algorithm.

figure 1.17

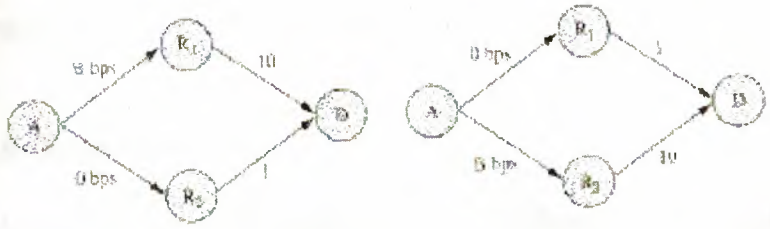


figure 1.18

no traffic is sent along that link. Thus, 1 is the propagation time of a signal along the link. We also assume that the delay is equal to 10 when the link carries **B** bps. For simplicity, we neglect the delays on the links (A, Rl) and (A, R2).

Assume that initially node A sends all its traffic to D via node Rı . The situation is then as shown in the left part of Figure 3.21: The delay along link (Rl, D) is equal to 10 and the delay along link (R2, D) is equal to 1. These routers advertise these delays to node A which then finds out that the preferred path to D starts with RZ and not with Rl. Node A then sends all its traffic to RZ and the new situation is that shown in the right part of the figure. At that point, Rı and RZ advertise the delays 1 and 10, respectively. This leads node A to send all its traffic to node Rl. As a result of this algorithm, the situation alternates between the left and right parts of the figure.

We say that the routing algorithm oscillates between two states. Note that the two states are not desirable. A better routing strategy would consist in node A sending half of its traffic to Rl and the other half to R2. This desirable routing can be approached if the Bellman-Ford algorithm is

modified so that node A can shift only a small fraction of its traffic in each iteration of the algorithm. However, this modification makes the algorithm even slower to reach to link failures.

4.8 Spanning Tree

A number of applications require finding a spanning tree in a network. For instance, the transparent routing in bridged LANs that uses a spanning tree to avoid loops. In this section we explain an algorithm, developed by R. C. Prim, for constructing a spanning tree whose sum of link lengths is minimum.

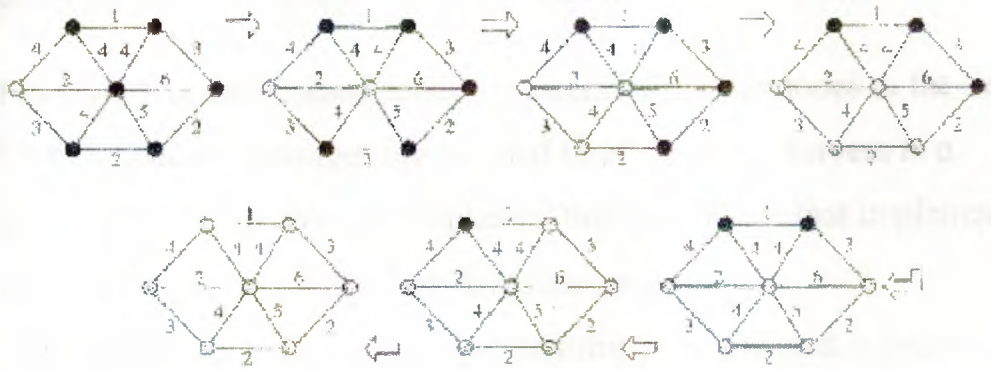
Description

One is given a set $ of N nodes and a set of link lengths between these nodes: **L** := *{L(i, j), i, j* E **S}.** By convention, *L(i, j)* =infinity if there is no link from *i* to j. We assume that *L (i, j) = L (j, i)* for all *i, j* . In a typical application, the links are full-duplex and *L (i , j) = L (j, i)* = 1 if there is a link between *i* and j whereas *L (i, j) = L (j, i) =infinity* otherwise.
The nodes want to discover *a minimum spanning tree.* Recall that a spanning tree in a graph **{S, L}** is a subgraph that spans all the nodes in **S** and has no loop. A minimum spanning tree is a spanning tree with the smallest sum of link lengths. In general a minimum spanning tree is not unique. Prim's algorithm finds one of the minimum spanning trees:
The algorithm starts by selecting an arbitrary node io. At each subsequent step, the algorithm attaches the closest node to the partial tree that it has built so far. Figure 3.22 illustrates the algorithm.

Analysis

We prove that Prim's algorithm constructs a minimum spanning tree. We

first show that a spanning tree T is minimum if and only if it contains the shortest link of every cutset. A cutset is a collection of links whose removal separates the set of nodes into two disconnected subsets. Since, by construction, Prim's tree satisfies that çondition, it will follow that it is a minimum spanning tree.



Prim's minimum spanning tree algorithm.

figure 1.19

To see why the condition is necessary, note that, because it is a tree, **T** contains at least one link in every cutset. If it does not contain the shortest link in some cutset, then one can make the tree shorter by replacing a link of **T** in that cutset by the shortest one and obtain a sorter spanning tree. To see why the condition is sufficient, we argue that it should not be possible for another tree T' to be the minimum spanning tree when **T** satisfies that condition. To see this, consider a link L in T' that is not in **T.** By removing that link we separate the set **S** of nodes into two subsets **A** and **B** such that all the nodes in **A** are still connected by T' after having deleted that link, and so are all the nodes in **B.** All the links thatjoin a node of **A** and a node of **B** form a cutset. By assumption, **T** contains the shortest link L' in that cutset. If L' is shorter than L, then replacing L by L' in T'

forms a shorter tree, which is not possible if T' is a minimum spanning tree. · IPv4 has poor control of the quality of service it provides. · IPv4 has no

4.9 Multicast Routing

A multicast routing algorithm sends packets from a source S to a set **G** of destinations. We explain a few multicast routing algorithms.

Flooding Flooding means sending a packet to all the nodes in the network. USENET news messages are flooded to all the news servers in a distribution group. We also explained that the routers that implement OSPF use flooding to distribute their link state information.
To flood the network, each router transmits every packet it receives once on all its links other than the incoming link of the packet. To prevent multiple transmissions of the same packet, either the router maintains a list of identification numbers of packets it has received previously (as done by OSPF), or the packet itself contains a list of the routers it has already visited (in USENET news).

4.10 Spanning Tree Routing

The switches in interconnected LANs use a spanning tree to prevent the transmission of multiple copies of packets. Routers in a network can use this method to transmit one copy of a packet to all the hosts.

Reverse-Path Forwarding

This algorithm first constructs the spanning tree of shortest paths to the source S from all the nodes with the reversed link metrics, that is, with *L(i, j)* replaced by *L(j, i)*. The algorithm then prunes back the tree so that it reaches only the nodes in the multicast group **G.** The routers can use the Bellman-Ford **(BF)** algorithm to construct : the tree of shortest paths to the

source. The MBONE (multicast backbone) routers use this combination of BF tree with reverse metrics and pruning in the "distance vector multicast routing protocol:' In the Internet, only a subset of the routers can implement multicast routing and duplicate the packets. These multicast routers send packets to each other through nonmulticast routers along paths that network administrators set up explicitly (a procedure called *tunneling).*

The multicast routers along the tree duplicate the packets as needed. Note that this algorithm does not minimize the total length traveled by the copies of a packet. However, this algorithm is easy to implement. One method for pruning back the tree is to send the first packet from the source to all the routers. If a router does not have any group member in its domain, it sends back a "prune" message to the router upstream in the tree. A router that gets a prune message on each downstream tree link sends one "prune" message on the upstream tree link. Each router maintains a table with one entry for each source and each multicast group. The prune messages have a time to live, and packets are again forwarded when the prune messages have expired, possibly triggering new prune messages.

Core-Based Trees

The core-based tree routing was designed to avoid having to send packets periodically to all the nodes and also having the routers maintain potentially large tables. In this algorithm, a router (called the core) in the network is associated with a multicast group. The users who want to receive the multicast send a "join" request toward the core. A router that receives a "join" request marks the incoming link of the packet as belonging to the tree of that multicast group. If the router is not yet part of the tree, it forwards the "join" request toward the core. Eventually, a tree gets built by this procedure and the tree reaches only the hosts that want to

be part of the group. Unfortunately, the tree that this algorithm constructs is not minimum, unlike that of the reverse-path forwarding algorithm.

MOSPF

Multicast OSPF is designed by an IETF working group to provide efficient multicasting within an autonomous system. MOSPF uses the network map that the routers maintain to run the OSPF algorithm (Dijkstra's) and calculates the tree of shortest paths from the source to all the nodes. MOSPF then prunes back the tree to the group members. By convention, all border routers are considered to be part of all multicast groups for the backbone routing. Note that MOSPF requires running OSPF for every multicast source and every multicast group.

4.11 PIM

The protocol-independent multicast (PIM) routing algorithm is an alternative algorithm proposed by the IETF working group. There are two versions of PIM. The first version is to be used when the group members are "dense" and the other version when they are "sparse:'

The dense version of PIM assumes that the links are symmetric and that the routers use some unicast routing algorithm. A router R that receives a message from source S for the multicast group G checks that the packet arrived on the link from R to S that the routing algorithm prefers. If that is not the case, R drops the packet and sends "prune (S, G)" message on the incoming link. If that is the case, R forwards a copy of the message on all the links from which R has not yet received a prune (S, G) message. If R has already received such a message from all its links, R drops the message and sends a prune (S, G) message on the incoming link of the message. The algorithm resolves ties to determine the "preferred" path to S by

preferring the neighbor with the largest IP address. If router R is attached to a broadcast network El (e.g., an Ethernet) to routers
Rı and RZ and sends a message from S to G on that network, then it may be that Rl is attached to a member of G and that RZ is not. In that case, RZ will send back a prune (S, G) message on the broadcast network to R. If R believes this message, it will stop transmitting these messages to Rl. The solution implemented by PIM is that R should defer acting on the prune (S, G) message it gets on the broadcast network and that R2, when it sees this message, should send a "join (S, G)" message back to R. That message voids the prune message. If another router, say R3, is attached to the same broadcast network and has group members, that router cancels its join messages when it sees one already sent on the broadcast network. PIM must deal with one last difficulty. Imagine that the two routers Rl and RZ are both attached to El as before and also to some other broadcast network EZ to which some member M of G is connected. When R sends a message *(S,* G) on El, both Rl and RZ copy that message on EZ and *M* gets two copies. To prevent these duplications, PIM proposes that Rl and RZ should notice the duplication and that they should send an "assert (S; G)" message on E2 which indicates their distance to the source S. The router with the largest distance should remove its interface on El as a preferred path to S. The routers cache the prune messages with a time to live of, say, 1 minute. The sparse mode of PIM, designed for use when group members are few avoids broad- casting a packet from S every minute when the cached entries expire. Instead, a receiver joins a group G by sending a "join G" message to a rendezvous point RP associated with G, using PIM as the routing algorithm. The routers that carry the join message cache that information in a routing table for messages for G. The source S sends its first message for G to RP and when it gets that message, *RP* sends a join

message to *S* and appends that link to S to the routing tree for G. Using this procedure, a tree is constructed for G that goes through RP. Once they get messages from S, members of G can send ajoin G message directly to S together with a prune (S, G) message along the link of their original join G message to RP. The tree is then modified partially. If all the group members were to send join messages to the sources, the routing would become RPF.

CONCLUSION

· The Internet is expanding rapidly and is having a profound impact on many aspects of society. In this chapter we studied the main protocols of Internet.

· To find the Internet protocol (IP) address of a destination, a sending computer uses the domain name system. This system is a hierachical arrangement of name servers.

· The IP calculates a route to reach every network in the Internet. The routing is organized in a two-level hierachy: BGP is used to find a path across different autonomous systems and OSPF to find a path inside an autonomous system.

· The transmission control protocol (TCP) supervises end-to-end transmission by controlling errors and the flow. TCP uses a retransmission protocol whose window is adjusted to use the links' efficiency without congesting them.

· The complements provide details on the transmission of IP packets on a point-to-point link and on the routing algorithms.

REFERENCES

1.Jeank Warlank.(1998).Communication Network.USA:Mc Graw Hill

2.Douglas E. Comer.(1997).The Internet Book.New Jersey:Prentice-Hall.

3.Comer,et.Al.(1997).Computer Networks and Internet.New jersey:Prentice- Hall.

4.Stallings.(1997).Data and Commuter Communications.New Jersey:Prentice- Hall.

5.Routing.Retrived.June 2,2000.From The World Wide Web.