TRACEROUTE ALGORITHM

A THESIS SUBMITTED FOR THE AWARD OF THE DEGREE OF
BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE

BY
AMIT GODARA 2K1/COE/008
PIYUSH SRIVASTAVA 2K1/COE/039
SHOBHIT GUPTA 2K1/COE/054

UNDER THE GUIDANCE OF
DR. GOLDIE GABRANI

DEPARTMENT OF COMPUTER ENGINEERING,
DELHI COLLEGE OF ENGINEERING,
UNIVERSITY OF DELHI,
2001-2005

CERTIFICATE

This is to certify that this project work entitled "**Trace route algorithm**", in partial fulfillment of requirement leading to the completion of *Bachelor of Engineering* in Computer Technology and Application submitted by Amit Godara (2K1/COE/039), Piyush Srivastava (2K1/COE/039), Shobhit Gupta (2K1/COE/054) to the *Department of Computer Engineering, Delhi College of Engineering,* is an authentic record of student's own work carried out under the supervision and guidance of *Dr. Goldie Gabrani,* (Asst. Professor) in the Department of Computer Engineering.
The work embodied in the project has not been submitted for the award of any other degree to the best of my knowledge.

Dr. D. Roy Choudhury	**Dr. Goldie Gabrani**
Professor and Head	Asst. Professor
Dept. of Computer Engg.	Dept. of Computer Engg.
Delhi College of Engg.	Delhi College of Engg.
New Delhi	New Delhi

ACKNOWLEDGEMENT

It is a distinct pleasure to express our deep sense of gratitude and indebtedness to our learned supervisor **Dr. Goldie Gabrani,** Asst. Professor in the Department of Computer Engineering, Delhi College of Engineering, for her valuable guidance, encouragement and patient review. Without her help and guidance, this project would have been impossible.

We own a deep sense of gratitude to **Dr. D. Roy Choudhury,** Professor and Head, Department of Computer Science, Delhi College of Engineering, for his constant advice and unflinching encouragement.

We are also indebted to our friends for their constructive feedbacks provide to us at regular interval.

Lastly, we express our respect and regards to our parents, who have been a constant source of inspiration to us.

Amit Godara	**Piyush Srivastava**	**Shobhit Gupta**
2K1/COE/008	**2K1/COE/039**	**2K1/COE/054**
B.E. COE	**B.E. COE**	**B.E. COE**

ABSTRACT

As we all know Internet is actually network of networks. There are millions of machines which are interacting simultaneously with each other through this single service. Client is never connected to the server directly, due to connections of various networks our request for the particular data from the server reach to us through various networks.

Traceroute lets us determine the path that IP datagrams follow from our host to some other destination. It uses the IPv4 TTL field or the IPv6 hop limit field and two ICMP messages. It starts by sending UDP datagram to the destination with a TTL (or hop limit) of 1. This datagram causes the first-hop router to return an ICMP "time exceeded in transit error". The TTL is then increased by one and another UDP datagram is sent, which locates the next router in the path. When the UDP datagram reaches the final destination, the goal is to have that host return an ICMP "port unreachable" error. This is done by sending the UDP datagram to a random port that is (hopefully) not in use on that host.

CONTENTS

1. Introduction
2. What are TCP/IP and the Internet?
    2.1. The Evolution of TCP/IP (and the Internet)
    2.2. Internet Growth
    2.3. Internet Administration
    2.4. Domain Names and IP Addresses (and Politics)
3. The TCP/IP Protocol Architecture
    3.1. The Network Interface Layer
    3.2. The Internet Layer
        3.2.1. IP Addressing and Subnet Masks
    3.3 Networking Devices

    3.4. The Transport Layer Protocols
        3.3.1. Ports
        3.3.2. TCP
        3.3.3. UDP
        3.3.4. ICMP

4.Routing Basics
    4.1 What is routing?
    4.2 Routing Components.
    4.3 Path determination.
    4.4. Switching.
        4.4.1 Switching Types
        4.4.2 Switching Algorithms
    4.5 Routing Algorithms
        4.5.1 Algorithms types.
            4.5.1.1 Static vs Dynamic.
            4.5.1.2 Single path vs Multiple path.
            4.5.1.3 Flat vs Hierarchical
            4.5.1.4 Host intelligent vs Router intelligent.
            4.5.1.5 Intradomain vs Interdomain.
            4.5.1.6 Link state vs Distance vector.
        4.5.2 Routing Metrices.
    4.6 Network protocols.
    4.7 network layer.

5. Trace Route
    5.1. Method 1
    5.2 Method 2
    5.3 VB Code
        5.3.1 VB Code Output
    5.4 C Code

1. Introduction

An increasing number of people are using the Internet and, many for the first time, are using the tools and utilities that at one time were only available on a limited number of computer systems (and only for really intense users!). One sign of this growth in use has been the significant number of TCP/IP and Internet books, articles, courses, and even TV shows that have become available in the last several years; there are so many such books that publishers are reluctant to authorize more because bookstores have reached their limit of shelf space! This memo provides a broad overview of the Internet and TCP/IP, with an emphasis on history, terms, and concepts. It is meant as a brief guide and starting point, referring to many other sources for more detailed information.

2. What are TCP/IP and the Internet?

While the TCP/IP protocols and the Internet *are* different, their histories are most definitely intertwingled! This section will discuss some of the history.

2.1. The Evolution of TCP/IP (and the Internet)

While the Internet today is recognized as a network that is fundamentally changing social, political, and economic structures, and in many ways obviating geographic boundaries, this potential is merely the realization of predictions that go back nearly forty years. In a series of memos dating back to August 1962, J.C.R. Licklider of MIT discussed his "Galactic Network" and how social interactions could be enabled through networking. The Internet certainly provides such a national and global infrastructure and, in fact, interplanetary Internet communication has already been seriously discussed.

Prior to the 1960s, what little computer communication existed comprised simple text and binary data, carried by the most common telecommunications network technology of the day; namely, circuit switching, the technology of the telephone networks for nearly a hundred years. Because most data traffic is bursty in nature (i.e., most of the transmissions occur during a very short period of time), circuit switching results in highly inefficient use of network resources.

The fundamental technology that makes the Internet work is called *packet switching*, a data network in which all components (i.e., hosts and switches) operate independently, eliminating single point-of-failure problems. In addition, network communication resources appear to be dedicated to individual users but, in fact, statistical multiplexing and an upper limit on the size of a transmitted entity result in fast, economical networks.

In the 1960s, packet switching was ready to be discovered. In 1961, Leonard Kleinrock of MIT published the first paper on packet switching theory (and the first book on the subject in 1964). In 1962, Paul Baran of the Rand Corporation described a robust, efficient, store-and-forward data network in a report for the U.S. Air Force. At about the same time, Donald Davies and Roger Scantlebury suggested a similar idea from work at the National Physical Laboratory (NPL) in the U.K. The research at MIT (1961-1967), RAND (1962-1965), and NPL (1964-1967) occurred independently and the principal researchers did not all meet together until the Association for Computing Machinery (ACM) meeting in 1967. The term *packet* was adopted from the work at NPL.

The modern Internet began as a U.S. Department of Defense (DoD) funded experiment to interconnect DoD-funded research sites in the U.S. The 1967 ACM meeting was also where the initial design for the so-called ARPANET — named for the DoD's Advanced Research Projects Agency (ARPA) — was first published by Larry Roberts. In December 1968, ARPA awarded a contract to Bolt Beranek and Newman (BBN) to design and deploy a packet switching network with a proposed line speed of 50 kbps. In September 1969, the first node of the ARPANET was installed at the University of California at Los Angeles (UCLA), followed monthly with nodes at Stanford Research Institute (SRI), the University of California at Santa Barbara (UCSB), and the University of Utah. With four nodes by the end of 1969, the ARPANET spanned the continental U.S. by 1971 and had connections to Europe by 1973.

The original ARPANET gave life to a number of protocols that were new to packet switching. One of the most lasting results of the ARPANET was the development of a user-network protocol that has become the standard interface between users and packet switched networks; namely, ITU-T (formerly CCITT) Recommendation X.25. This "standard" interface encouraged BBN to start Telenet, a commercial packet-switched data service, in 1974; after much renaming, Telenet became a part of Sprint's X.25 service.

The initial host-to-host communications protocol introduced in the ARPANET was called the Network Control Protocol (NCP). Over time, however, NCP proved to be incapable of keeping up with the growing network traffic load. In 1974, a new, more robust suite of communications protocols was proposed and implemented throughout the ARPANET, based upon the Transmission Control Protocol (TCP) for end-to-end network communication. But it seemed like overkill for the intermediate gateways (what we would today call *routers*) to needlessly have to deal with an end-to-end protocol so in 1978 a new design split responsibilities between a pair of protocols; the new Internet Protocol (IP) for routing packets and device-to-device communication (i.e., host-to-gateway or gateway-to-gateway) and TCP for reliable, end-to-end host communication. Since TCP and IP were originally envisioned functionally as a single protocol, the protocol suite, which actually refers to a large collection of protocols and applications, is usually referred to simply as *TCP/IP*.

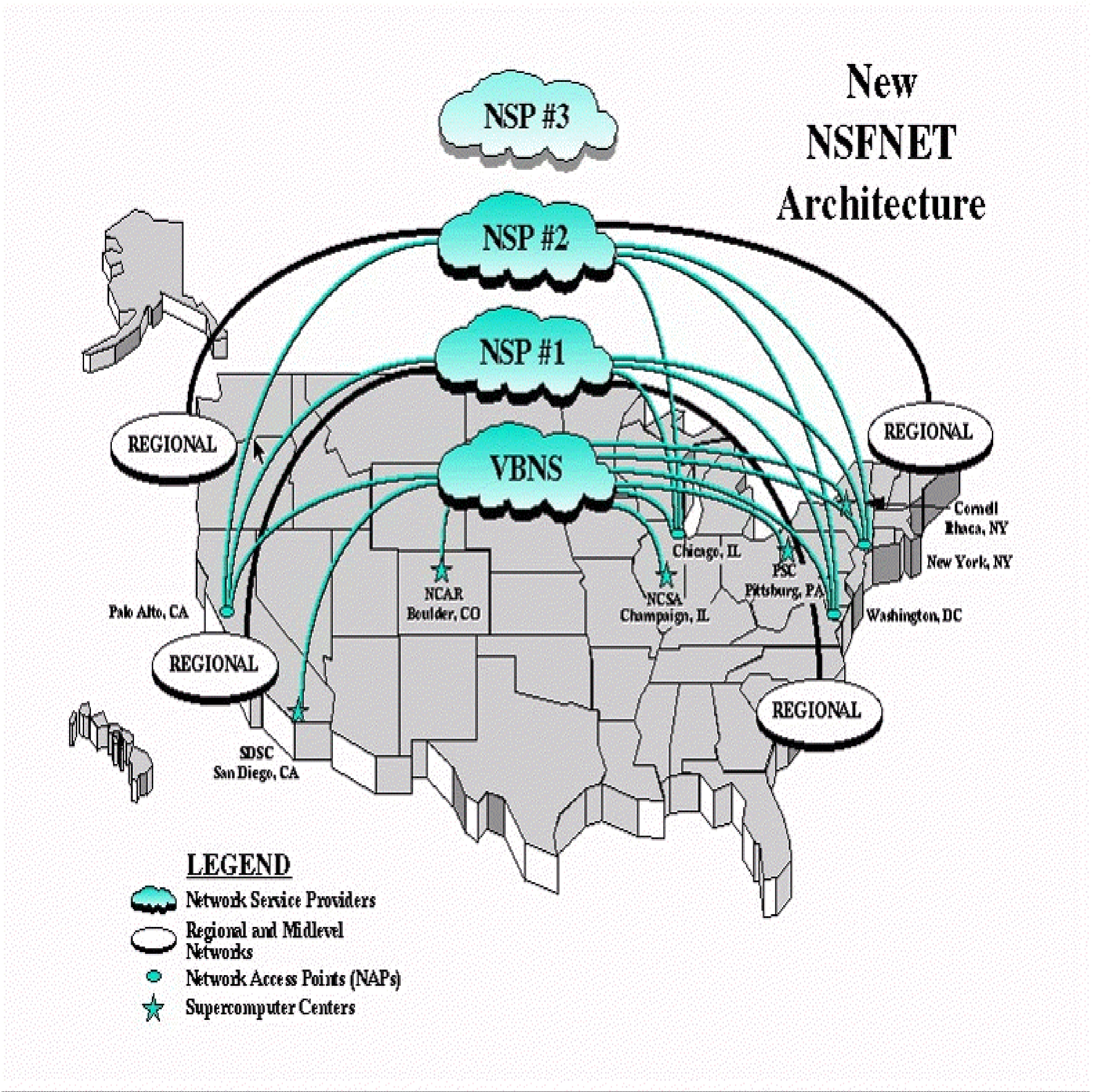
The original versions of both TCP and IP that are in common use today were written in September 1981, although both have had several modifications applied to them (in addition, the IP version 6, or IPv6, specification was released in December 1995). In 1983, the DoD mandated that all of their computer systems would use the TCP/IP protocol suite for long-haul communications, further enhancing the scope and importance of the ARPANET.

In 1983, the ARPANET was split into two components. One component, still called ARPANET, was used to interconnect research/development and academic sites; the other, called MILNET, was used to carry military traffic and became part of the Defense Data Network. That year also saw a huge boost in the popularity of TCP/IP with its inclusion in the communications kernel for the University of California s UNIX implementation, 4.2BSD (Berkeley Software Distribution) UNIX.

In 1986, the National Science Foundation (NSF) built a backbone network to interconnect four NSF-funded regional supercomputer centers and the National Center for Atmospheric Research (NCAR). This network, dubbed the NSFNET, was originally intended as a backbone for other networks, not as an interconnection mechanism for individual systems. Furthermore, the "Appropriate Use Policy" defined by the NSF limited traffic to non-commercial use. The NSFNET continued to grow and provide connectivity between both NSF-funded and non-NSF regional networks, eventually becoming the backbone that we know today as the Internet. Although early NSFNET applications were largely multiprotocol in nature, TCP/IP was employed for interconnectivity (with the ultimate goal of migration to Open Systems Interconnection).

The NSFNET originally comprised 56-kbps links and was completely upgraded to T1 (1.544 Mbps) links in 1989. Migration to a "professionally-managed" network was supervised by a consortium comprising Merit (a Michigan state regional network headquartered at the University of Michigan), IBM, and MCI. Advanced Network & Services, Inc. (ANS), a non-profit company formed by IBM and MCI, was responsible for managing the NSFNET and supervising the transition of the NSFNET backbone to T3 (44.736 Mbps) rates by the end of 1991. During this period of time, the NSF also funded a number of regional Internet service providers (ISPs) to provide local connection points for educational institutions and NSF-funded sites.

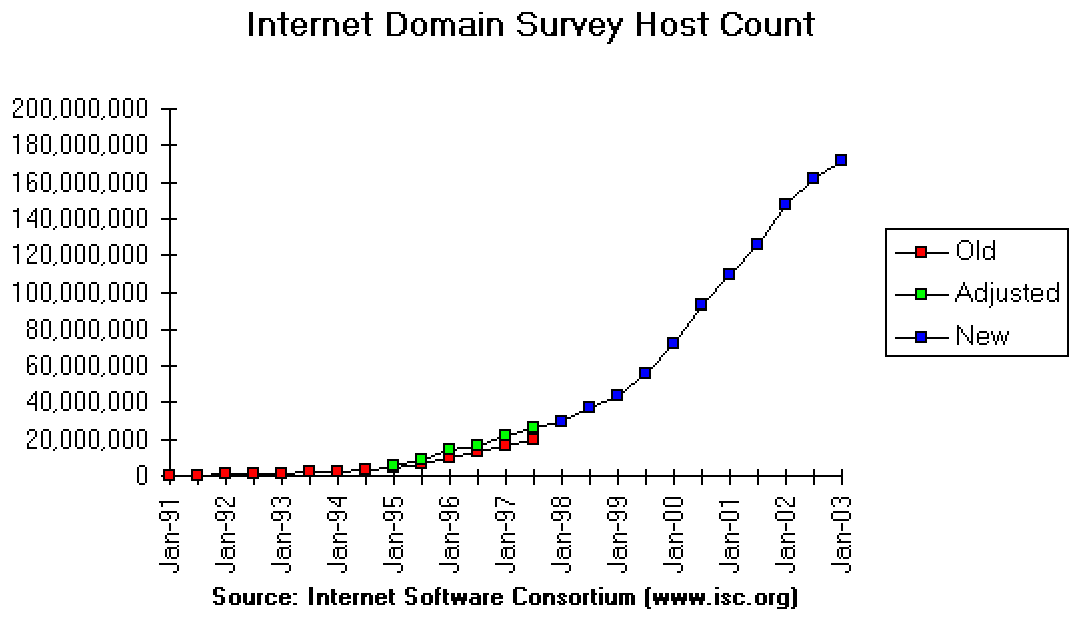
In 1993, the NSF decided that it did not want to be in the business of running and funding networks, but wanted instead to go back to the funding of research in the areas of supercomputing and high-speed communications. In addition, there was increased pressure to commercialize the Internet; in 1989, a trial gateway connected MCI, CompuServe, and Internet mail services, and commercial users were now finding out about all of the capabilities of the Internet that once belonged exclusively to academic and hard-core users! In 1991, the Commercial Internet Exchange (CIX) Association was formed by General Atomics, Performance Systems International (PSI), and UUNET Technologies to promote and provide a commercial Internet backbone service. Nevertheless, there remained intense pressure from non-NSF ISPs to open the network to all users.



2.2. Internet Growth

In Douglas Adams' *The Hitchhiker's Guide to the Galaxy* (Pocket Books, 1979), the hitchhiker describes outer space as being "...big. Really big. ...vastly hugely mind-bogglingly big..." A similar description can be applied to the Internet. To paraphrase the hitchhiker, you may think that your 750 node LAN is big, but that's just peanuts compared to the Internet.

The ARPANET started with four nodes in 1969 and grew to just under 600 nodes before it was split in 1983. The NSFNET also started with a modest number of sites in 1986. After that, the network experienced literally exponential growth.



2.3. Internet Administration

The Internet has no single owner, yet everyone owns (a portion of) the Internet. The Internet has no central operator, yet everyone operates (a portion of) the Internet. The Internet has been compared to anarchy, but some claim that it is not nearly that well organized!

Some central authority is required for the Internet, however, to manage those things that can only be managed centrally, such as addressing, naming, protocol development, standardization, etc. Among the significant Internet authorities are:

- The Internet Society (ISOC), chartered in 1992, is a non-governmental international organization providing coordination for the Internet, and its internetworking technologies and applications. ISOC also provides oversight and communications for the Internet Activities Board.
- The Internet Activities Board (IAB) governs administrative and technical activities on the Internet.
- The Internet Engineering Task Force (IETF) is one of the two primary bodies of the IAB. The IETF's working groups have primary responsibility for the technical

activities of the Internet, including writing specifications and protocols. The impact of these specifications is significant enough that ISO accredited the IETF as an international standards body at the end of 1994. RFCs 2028 and 2031 describe the organizations involved in the IETF standards process and the relationship between the IETF and ISOC, respectively, while RFC 2418 describes the IETF working group guidelines and procedures. The background and history of the IETF and the Internet standards process can be found in "IETF—History, Background, and Role in Today's Internet."

- The Internet Engineering Steering Group (IESG) is the other body of the IAB. The IESG provides direction to the IETF.
- The Internet Research Task Force (IRTF) comprises a number of long-term reassert groups, promoting research of importance to the evolution of the future Internet.
- The Internet Engineering Planning Group (IEPG) coordinates worldwide Internet operations. This group also assists Internet Service Providers (ISPs) to interoperate within the global Internet.
- The Forum of Incident Response and Security Teams is the coordinator of a number of Computer Emergency Response Teams (CERTs) representing many countries, governmental agencies, and ISPs throughout the world. Internet network security is greatly enhanced and facilitated by the FIRST member organizations.
- The World Wide Web Consortium (W3C) is not an Internet administrative body, per se, but since October 1994 has taken a lead role in developing common protocols for the World Wide Web to promote its evolution and ensure its interoperability. W3C has more than 400 Member organizations internationally. The W3C, then, is leading the technical evolution of the Web, having already developed more than 20 technical specifications for the Web's infrastructure.

2.4. Domain Names and IP Addresses (and Politics)

Although not directly related to the administration of the Internet for operational purposes, the assignment of Internet domain names (and IP addresses) is the subject of some controversy and a lot of current activity. Internet hosts use a hierarchical naming structure comprising a top-level domain (TLD), domain and subdomain (optional), and host name. The IP address space, and all TCP/IP-related numbers, have historically been managed by the Internet Assigned Numbers Authority (IANA). Domain names are assigned by the TLD naming authority; until April 1998, the Internet Network Information Center (InterNIC) had overall authority of these names, with NICs around the world handling non-U.S. domains. The InterNIC was also responsible for the overall coordination and management of the Domain Name System (DNS), the distributed database that reconciles host names and IP addresses on the Internet.

The InterNIC is an interesting example of the recent changes in the Internet. Since early 1993, Network Solutions, Inc. (NSI) operated the registry tasks of the InterNIC on behalf of the NSF and had exclusive registration authority for the *.com*, *.org*, *.net*, and *.edu* domains. NSI's contract ran out in April 1998 and was extended several times because no other agency was in place to continue the registration for those domains. In October 1998, it was decided that NSI would remain the sole administrator for those domains but that a plan needed to be put into place so that users could register names in those domains with other firms. In addition, NSI's contract was extended to September 2000, although the registration business was opened to competition in June 1999. Nevertheless, when NSI's original InterNIC contract expired, IP address assignments moved to a new entity called the American Registry for Internet Numbers (ARIN). (And NSI itself was purchased by VeriSign in March 2000.)

The newest body to handle governance of global Top Level Domain (gTLD) registrations is the Internet Corporation for Assigned Names and Numbers (ICANN). Formed in October 1998, ICANN is the organization designated by the U.S. National Telecommunications and Information Administration (NTIA) to administer the DNS. Although surrounded in some early controversy (which is well beyond the scope of this paper!), ICANN has received wide industry support. ICANN has created several Support Organizations (SOs) to create policy for the administration of its areas of responsibility, including domain names (DNSO), IP addresses (ASO), and protocol parameter assignments (PSO).

On April 21, 1999, ICANN announced that five companies had been selected to be part of this new *competitive* Shared Registry System for the .com, .net, and .org domains:

- America Online, Inc. (U.S.)
- CORE (Internet Council of Registrars) (International)
- France Telecom/Oléane (France)
- Melbourne IT (Australia)
- register.com (U.S.)

3. The TCP/IP Protocol Architecture

TCP/IP is most commonly associated with the Unix operating system. While developed separately, they have been historically tied, as mentioned above, since 4.2BSD Unix started bundling TCP/IP protocols with the operating system. Nevertheless, TCP/IP protocols are available for all widely-used operating systems today and native TCP/IP support is provided in OS/2, OS/400, and Windows 9x/NT/2000, as well as most Unix variants.

Figure 2 shows the TCP/IP protocol architecture; this diagram is by no means exhaustive, but shows the major protocol and application components common to most commercial TCP/IP software packages and their relationship.

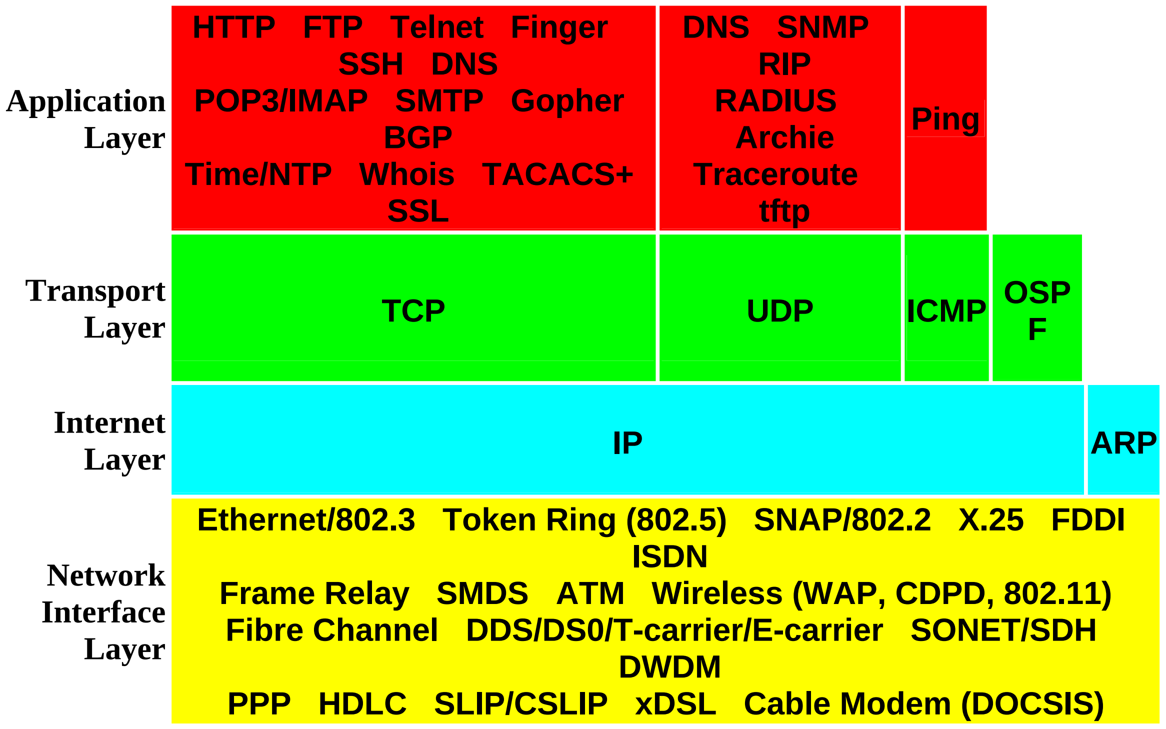

Application Layer
HTTP FTP Telnet Finger SSH DNS POP3/IMAP SMTP Gopher BGP Time/NTP Whois TACACS+ SSL
DNS SNMP RIP RADIUS Archie Traceroute tftp
Ping
Transport Layer
TCP
UDP
ICMP
OSPF
Internet Layer
IP
ARP
Network Interface Layer
Ethernet/802.3 Token Ring (802.5) SNAP/802.2 X.25 FDDI ISDN
Frame Relay SMDS ATM Wireless (WAP, CDPD, 802.11)
Fibre Channel DDS/DS0/T-carrier/E-carrier SONET/SDH DWDM
PPP HDLC SLIP/CSLIP xDSL Cable Modem (DOCSIS)

FIGURE 2. Abbreviated TCP/IP protocol stack.

3.1. The Network Interface Layer

The TCP/IP protocols have been designed to operate over nearly any underlying local or wide area network technology. Although certain accommodations may need to be made, IP messages can be transported over all of the technologies shown in the figure, as well as numerous others. It is beyond the scope of this paper to describe most of these underlying protocols and technologies.

1. Two of the underlying network interface protocols, however, are particularly relevant to TCP/IP. The Serial Line Internet Protocol (SLIP, RFC 1055) and Point-to-Point Protocol (PPP, RFC 1661), respectively, may be used to provide data link layer protocol services where no other underlying data link protocol may be in use, such as in leased line or dial-up environments. Most commercial TCP/IP software packages for PC-class systems include these two protocols. With SLIP or PPP, a remote computer can attach directly to a host server and, therefore, connect to the Internet using IP rather than being limited to an

asynchronous connection.

3.2. The Internet Layer

The Internet Protocol (RFC 791), provides services that are roughly equivalent to the OSI Network Layer. IP provides a datagram (connectionless) transport service across the network. This service is sometimes referred to as *unreliable* because the network does not guarantee delivery nor notify the end host system about packets lost due to errors or network congestion. IP datagrams contain a message, or one fragment of a message, that may be up to 65,535 bytes (octets) in length. IP does not provide a mechanism for flow control.

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |      TOS      |          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|    Fragment Offset      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      TTL      |    Protocol   |        Header Checksum        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Source Address                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Destination Address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Options....                              (Padding)    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Data...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```
FIGURE 4. IP packet (datagram) header format.

The basic IP packet header format is shown in Figure 4. The format of the diagram is consistent with the RFC; bits are numbered from left-to-right, starting at 0. Each row represents a single *32-bit word*; note that an IP header will be at least 5 words (20 bytes) in length. The fields contained in the header, and their functions, are:

- *Version:* Specifies the IP version of the packet. The current version of IP is version 4, so this field will contain the binary value `0100`. [NOTE: Actually, many IP version numbers have been assigned besides 4 and 6; see the IANA's list of IP Version Numbers.]
- *Internet Header Length (IHL):* Indicates the length of the datagram header in 32 bit (4 octet) words. A minimum-length header is 20 octets, so this field always has a value of at least 5 (`0101`) Since the maximum value of this field is 15, the IP Header can be no longer than 60 octets.

- *Type of Service (TOS):* Allows an originating host to request different classes of service for packets it transmits. Although not generally supported today in IPv4, the TOS field can be set by the originating host in response to service requests across the Transport Layer/Internet Layer service interface, and can specify a service priority (0-7) or can request that the route be optimized for either cost, delay, throughput, or reliability.
- *Total Length:* Indicates the length (in bytes, or octets) of the entire packet, including both header and data. Given the size of this field, the maximum size of an IP packet is 64 KB, or 65,535 bytes. In practice, packet sizes are limited to the maximum transmission unit (MTU).
- *Identification:* Used when a packet is fragmented into smaller pieces while traversing the Internet, this identifier is assigned by the transmitting host so that different fragments arriving at the destination can be associated with each other for reassembly.
- *Flags:* Also used for fragmentation and reassembly. The first bit is called the More Fragments (MF) bit, and is used to indicate the last fragment of a packet so that the receiver knows that the packet can be reassembled. The second bit is the Don't Fragment (DF) bit, which suppresses fragmentation. The third bit is unused (and always set to 0).
- *Fragment Offset:* Indicates the position of this fragment in the original packet. In the first packet of a fragment stream, the offset will be 0; in subsequent fragments, this field will indicates the offset in increments of 8 bytes.
- *Time-to-Live (TTL):* A value from 0 to 255, indicating the number of hops that this packet is allowed to take before discarded within the network. Every router that sees this packet will decrement the TTL value by one; if it gets to 0, the packet will be discarded.
- *Protocol:* Indicates the higher layer protocol contents of the data carried in the packet; options include ICMP (1), TCP (6), UDP (17), or OSPF (89). A complete list of IP protocol numbers can be found at the IANA's list of Protocol Numbers. An implementation-specific list of supported protocols can be found in the `protocol` file, generally found in the `/etc` (Linux/Unix), `c:\windows` (Windows 9x, ME), or `c:\winnt\system32\drivers\etc` (Windows NT, 2000) directory.
- *Header Checksum:* Carries information to ensure that the received IP header is error-free. Remember that IP provides an *unreliable* service and, therefore, this field only checks the header rather than the entire packet.
- *Source Address:* IP address of the host sending the packet.
- *Destination Address:* IP address of the host intended to receive the packet.
- *Options:* A set of options which may be applied to any given packet, such as sender-specified source routing or security indication. The option list may use up to 40 bytes (10 words), and will be padded to a word boundary; IP options are taken from the IANA's list of IP Option Numbers.

3.2.1. IP Addresses

IP addresses are 32 bits in length (Figure 5). They are typically written as a sequence of four numbers, representing the decimal value of each of the address bytes. Since the values are separated by periods, the notation is referred to as *dotted decimal*. A sample IP address is 208.162.106.17.

```
                            1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
            0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
           --+-------------+-------------------------------------------------
 Class A   |0|     NET_ID  |                         HOST_ID               |
           |-+-+-----------+---------------+-------------------------------|
 Class B   |1|0|    NET_ID                 |            HOST_ID            |
           |-+-+-+-------------------------+---------------+---------------|
 Class C   |1|1|0|                     NET_ID              |    HOST_ID    |
           |-+-+-+-+---------------------------------------+---------------|
 Class D   |1|1|1|0|                            MULTICAST_ID               |
           |-+-+-+-+-------------------------------------------------------|
 Class E   |1|1|1|1|                      EXPERIMENTAL_ID                  |
           --+-+-+-+---------------------------------------------------------
```

FIGURE 5. IP Address Format.

IP addresses are hierarchical for routing purposes and are subdivided into two subfields. The Network Identifier (NET_ID) subfield identifies the TCP/IP subnetwork connected to the Internet. The NET_ID is used for high-level routing between networks, much the same way as the country code, city code, or area code is used in the telephone network. The Host Identifier (HOST_ID) subfield indicates the specific host within a subnetwork.

To accommodate different size networks, IP defines several *address classes*. Classes A, B, and C are used for host addressing and the only difference between the classes is the length of the NET_ID subfield:

- A Class A address has an 8-bit NET_ID and 24-bit HOST_ID. Class A addresses are intended for very large networks and can address up to 16,777,214 (2^{24}-2) hosts per network. The first bit of a Class A address is a 0 and the NETID occupies the first byte, so there are only 128 (2^{7}) possible Class A NETIDs. In fact, the first digit of a Class A address will be between 1 and 126, and only about 90 or so Class A addresses have been assigned.
- A Class B address has a 16-bit NET_ID and 16-bit HOST_ID. Class B addresses are intended for moderate sized networks and can address up to 65,534 (2^{16}-2) hosts per network. The first two bits of a Class B address are 10 so that the first digit of a Class B address will be a number between 128 and 191; there are 16,384

(2^{14}) possible Class B NETIDs. The Class B address space has long been threatened with being used up and it is has been very difficult to get a new Class B address for some time.

- A Class C address has a 24-bit NET_ID and 8-bit HOST_ID. These addresses are intended for small networks and can address only up to 254 (2^8-2) hosts per network. The first three bits of a Class C address are 110 so that the first digit of a Class C address will be a number between 192 and 223. There are 2,097,152 (2^{21}) possible Class C NETIDs and most addresses assigned to networks today are Class C (or sub-Class C!).

The remaining two address classes are used for special functions only and are not commonly assigned to individual hosts. Class D addresses may begin with a value between 224 and 239 (the first 4 bits are 1110), and are used for IP multicasting (i.e., sending a single datagram to multiple hosts); the IANA maintains a list of Internet Multicast Addresses. Class E addresses begin with a value between 240 and 255 (the first 4 bits are 1111), and are reserved for experimental use.

Several address values are reserved and/or have special meaning. A HOST_ID of 0 (as used above) is a dummy value reserved as a place holder when referring to an entire subnetwork; the address 208.162.106.0, then, refers to the Class C address with a NET_ID of 208.162.106. A HOST_ID of all ones (usually written "255" when referring to an all-ones byte, but also denoted as "-1") is a broadcast address and refers to all hosts on a network. A NET_ID value of 127 is used for loopback testing and the specific host address 127.0.0.1 refers to the *localhost*.

Several NET_IDs have been reserved in RFC 1918 for private network addresses and packets will not be routed over the Internet to these networks. Reserved NET_IDs are the Class A address 10.0.0.0 (formerly assigned to ARPANET), the sixteen Class B addresses 172.16.0.0-172.31.0.0, and the 256 Class C addresses 192.168.0.0-192.168.255.0.

An additional addressing tool is the *subnet mask*. Subnet masks are used to indicate the portion of the address that identifies the network (and/or subnetwork) for routing purposes. The subnet mask is written in dotted decimal and the number of 1s indicates the significant NET_ID bits. For "classful" IP addresses, the subnet mask and number of significant address bits for the NET_ID are:

Class	Subnet Mask	Number of Bits
A	255.0.0.0	8
B	255.255.0.0	16
C	255.255.255.0	24

Depending upon the context and literature, subnet masks may be written in dotted decimal form or just as a number representing the number of significant address bits for the NET_ID. Thus, `208.162.106.17 255.255.255.0` and

208.162.106.17/24 both refer to a Class C NET_ID of 208.162.106. Some, in fact, might refer to this 24-bit NET_ID as a "slash-24."

Subnet masks can also be used to subdivide a large address space into subnetworks or to combine multiple small address spaces. In the former case, a network may subdivide their address space to define multiple logical networks by segmenting the HOST_ID subfield into a Subnetwork Identifier (SUBNET_ID) and (smaller) HOST_ID. For example, user assigned the Class B address space 172.16.0.0 could segment this into a 16-bit NET_ID, 4-bit SUBNET_ID, and 12-bit HOST_ID. In this case, the subnet mask for Internet routing purposes would be 255.255.0.0 (or "/16"), while the mask for routing to individual subnets within the larger Class B address space would be 255.255.240.0 (or "/20").

But how a subnet mask work? To determine the subnet portion of the address, we simply perform a bit-by-bit logical AND of the IP address and the mask. Consider the following example: suppose we have a host with the IP address 172.20.134.164 and a subnet mask 255.255.0.0. We write out the address and mask in decimal and binary as follows:

```
    172.020.134.164     10101100.00010100.10000110.10100100
AND 255.255.000.000     11111111.11111111.00000000.00000000
    ---------------     -----------------------------------
    172.020.000.000     10101100.00010100.00000000.00000000
```

From this we can easily find the NET_ID 172.20.0.0 (and can also infer the HOST_ID 134.164).

As an aside, most ISPs use a /30 address for the WAN links between the network and the customer. The router on the customer's network will generally have two IP addresses; one on the LAN interface using an address from the customer's public IP address space and one on the WAN interface leading back to the ISP. Since the ISP would like to be able to ping both sides of the router for testing and maintenance, having an IP address for each router port is a good idea.

By using a /30 address, a single Class C address can be broken up into 64 smaller addresses. Here's an example. Suppose an ISP assigns a particular customer the address 24.48.165.130 and a subnet mask 255.255.255.252. That would look like the following:

```
    024.048.165.130     00011000.00110000.10100101.10000010
AND 255.255.255.252     11111111.11111111.11111111.11111100
    ---------------     -----------------------------------
    024.048.165.128     00011000.00110000.10100101.10000000
```

So we find the NET_ID to be 24.48.165.128. Since there's a 30-bit NET_ID, we are left with a 2-bit HOST_ID; thus, there are four possible host addresses in this subnet: 24.48.165.128 (00), .129 (01), .130 (10), and .131 (11). The .128 address isn't used because it is all-zeroes; .131 isn't used because it is all-ones. That leave .129 and .130, which is ok since we only have two ends on the WAN link! So, in this case, the customer's router might be assigned 24.48.165.130/30 and the ISP's end of the link might get 24.48.165.129/30. Use of this subnet mask is very common today (so common that there is a proposal to allow the definition of 2-address NET_IDs specifically for point-to-point WAN links).

3.3. The Transport Layer Protocols

The TCP/IP protocol suite comprises two protocols that correspond roughly to the OSI Transport and Session Layers; these protocols are called the Transmission Control Protocol and the User Datagram Protocol (UDP). One can argue that it is a misnomer to refer to "TCP/IP applications," as most such applications actually run over TCP or UDP, as shown in Figure 2.

3.3.1. Ports

Higher-layer applications are referred to by a port identifier in TCP/UDP messages. The port identifier and IP address together form a *socket*, and the end-to-end communication between two hosts is uniquely identified on the Internet by the four-tuple (source port, source address, destination port, destination address).

Port numbers are specified by a 16-bit number. Port numbers in the range 0-1023 are called *Well Known Ports*. These port numbers are assigned to the server side of an application and, on most systems, can only be used by processes with a high level of privilege (such as root or administrator). Port numbers in the range 1024-49151 are called *Registered Ports*, and these are numbers that have been publicly defined as a convenience for the Internet community to avoid vendor conflicts. Server or client applications can use the port numbers in this range. The remaining port numbers, in the range 49152-65535, are called *Dynamic and/or Private Ports* and can be used freely by any client or server.

Some well-known port numbers include:

Port #	Common Protocol	Service	Port #	Common Protocol	Service
7	TCP	echo	80	TCP	http
9	TCP	discard	110	TCP	pop3
13	TCP	daytime	111	TCP	sunrpc
19	TCP	chargen	119	TCP	nntp
20	TCP	ftp-control	123	UDP	ntp
21	TCP	ftp-data	137	UDP	netbios-ns
23	TCP	telnet	138	UDP	netbios-dgm

25	TCP	smtp	139	TCP	netbios-ssn
37	UDP	time	143	TCP	imap
43	TCP	whois	161	UDP	snmp
53	TCP/UDP	dns	162	UDP	snmp-trap
67	UDP	bootps	179	TCP	bgp
68	UDP	bootpc	443	TCP	https (http/ssl)
69	UDP	tftp	520	UDP	rip
70	TCP	gopher	1080	TCP	socks
79	TCP	finger	33434	UDP	traceroute

A complete list of port numbers that have been assigned can be found in the IANA's list of Port Numbers. An implementation-specific list of supported port numbers and services can be found in the `services` file, generally found in the `/etc` (Linux/Unix), `c:\windows` (Windows 9x, ME), or `c:\winnt\system32\drivers\etc` (Windows NT, 2000) directory.

3.3.2. TCP

TCP, described in RFC 793, provides a virtual circuit (connection-oriented) communication service across the network. TCP includes rules for formatting messages, establishing and terminating virtual circuits, sequencing, flow control, and error correction. Most of the applications in the TCP/IP suite operate over the *reliable* transport service provided by TCP.

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Acknowledgement Number                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Offset |(reserved) |   Flags   |            Window             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Checksum              |       Urgent Pointer          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Options....                              (Padding)   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Data...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

FIGURE 8. TCP segment format.

The TCP data unit is called a *segment*; the name is due to the fact that TCP does not recognize messages, per se, but merely sends a block of bytes from the byte stream between sender and receiver. The fields of the segment (Figure 8) are:

- *Source Port* and *Destination Port:* Identify the source and destination ports to identify the end-to-end connection and higher-layer application.
- *Sequence Number:* Contains the sequence number of this segment's first data byte in the overall connection byte stream; since the sequence number refers to a byte count rather than a segment count, sequence numbers in contiguous TCP segments are not numbered sequentially.
- *Acknowledgment Number:* Used by the sender to acknowledge receipt of data; this field indicates the sequence number of the next byte expected from the receiver.
- *Data Offset:* Points to the first data byte in this segment; this field, then, indicates the segment header length.
- *Control Flags:* A set of flags that control certain aspects of the TCP virtual connection. The flags include:
  - *Urgent Pointer Field Significant (URG):* When set, indicates that the current segment contains urgent (or high-priority) data and that the Urgent Pointer field value is valid.
  - *Acknowledgment Field Significant (ACK):* When set, indicates that the value contained in the Acknowledgment Number field is valid. This bit is usually set, except during the first message during connection establishment.
  - *Push Function (PSH):* Used when the transmitting application wants to force TCP to immediately transmit the data that is currently buffered without waiting for the buffer to fill; useful for transmitting small units of data.
  - *Reset Connection (RST):* When set, immediately terminates the end-to-end TCP connection.
  - *Synchronize Sequence Numbers (SYN):* Set in the initial segments used to establish a connection, indicating that the segments carry the initial sequence number.
  - *Finish (FIN):* Set to request normal termination of the TCP connection in the direction this segment is traveling; completely closing the connection requires one FIN segment in each direction.
- *Window:* Used for flow control, contains the value of the *receive window size* which is the number of transmitted bytes that the sender of this segment is willing to accept from the receiver.
- *Checksum:* Provides rudimentary bit error detection for the segment (including the header and data).
- *Urgent Pointer*: Urgent data is information that has been marked as high-priority by a higher layer application; this data, in turn, usually bypasses normal TCP buffering and is placed in a segment between the header and "normal" data. The Urgent Pointer, valid when the URG flag is set, indicates the position of the first octet of nonexpedited data in the segment.

- *Options:* Used at connection establishment to negotiate a variety of options; maximum segment size (MSS) is the most commonly used option and, if absent, defaults to an MSS of 536. Another option is Selective Acknowledgement (SACK), which allows out-of-sequence segments to be accepted by a receiver. The IANA maintains a list of all TCP Option Numbers.

3.3.3. UDP

UDP, described in RFC 768, provides an end-to-end datagram (connectionless) service. Some applications, such as those that involve a simple query and response, are better suited to the datagram service of UDP because there is no time lost to virtual circuit establishment and termination. UDP's primary function is to add a port number to the IP address to provide a socket for the application.

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Source Port           |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Length              |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Data...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```
FIGURE 9. UDP datagram format.

The fields of a UDP datagram (Figure 9) are:

- *Source Port:* Identifies the UDP port being used by the sender of the datagram; use of this field is optional in UDP and may be set to 0.
- *Destination Port:* Identifies the port used by the datagram receiver.
- *Length:* Indicates the total length of the UDP datagram.
- *Checksum:* Provides rudimentary bit error detection for the datagram (including the header and data).

3.3.4. ICMP

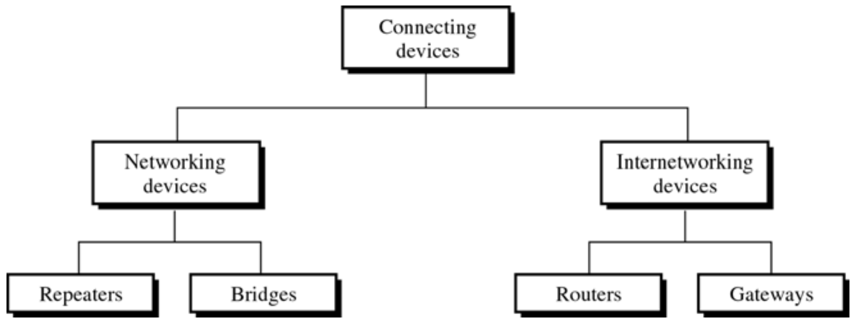
The Internet Control Message Protocol, described in RFC 792, is an adjunct to IP that notifies the sender of IP datagrams about abnormal events. This collateral protocol is particularly important in the connectionless environment of IP. ICMP is not a classic host-to-host protocols like TCP or UDP, but is host-to-host in the sense that one device (e.g., a router or computer) is sending a message to another device (e.g., another router or computer).

The commonly employed ICMP message types include:

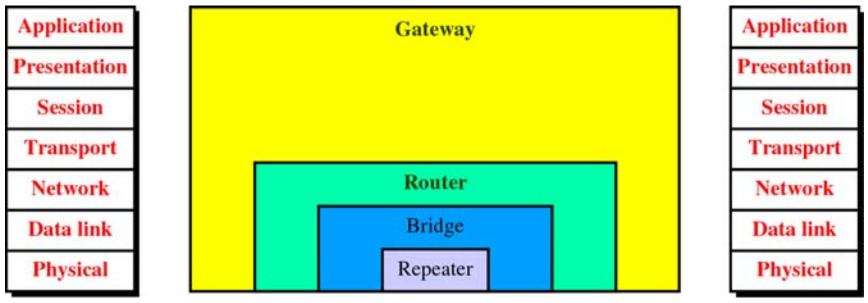
- *Destination Unreachable:* Indicates that a packet cannot be delivered because the destination host cannot be reached. The reason for the non-delivery may be that the host or network is unreachable or unknown, the protocol or port is unknown or unusable, fragmentation is required but not allowed (DF-flag is set), or the network or host is unreachable for this type of service.
- *Echo* and *Echo Reply:* These two messages are used to check whether hosts are reachable on the network. One host sends an Echo message to the other, optionally containing some data, and the receiving host responds with an Echo Reply containing the same data. These messages are the basis for the Ping command.
- *Parameter Problem:* Indicates that a router or host encountered a problem with some aspect of the packet's Header.
- *Redirect:* Used by a host or router to let the sending host know that packets should be forwarded to another address. *For security reasons, Redirect messages should usually be blocked at the firewall.*
- *Source Quench:* Sent by a router to indicate that it is experiencing congestion (usually due to limited buffer space) and is discarding datagrams.
- *TTL Exceeded:* Indicates that a datagram has been discarded because the TTL field reached 0 or because the entire packet was not received before the fragmentation timer expired.
- *Timestamp* and *Timestamp Reply:* These messages are similar to the Echo messages, but place a timestamp (with millisecond granularity) in the message, yielding a measure of how long remote systems spend buffering and processing datagrams, and providing a mechanism so that hosts can synchronize their clocks.

ICMP messages are carried in IP packets. The IANA maintains a complete list of ICMP parameters.

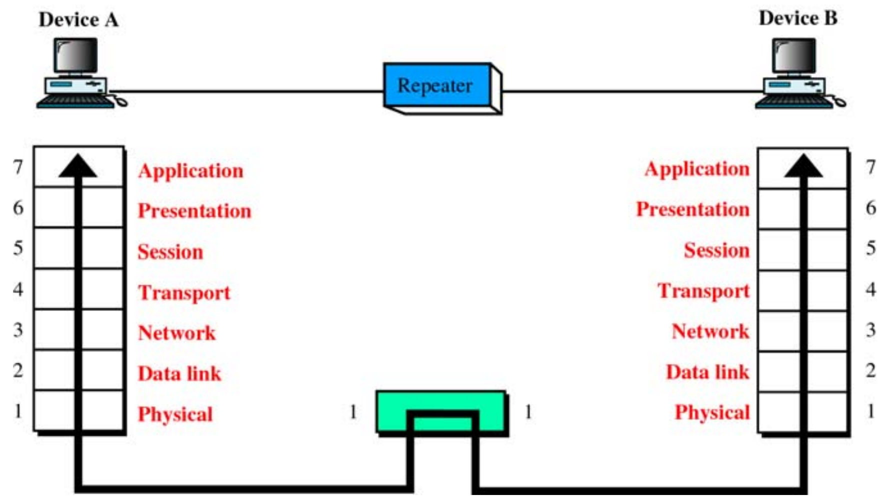
Connecting Devices



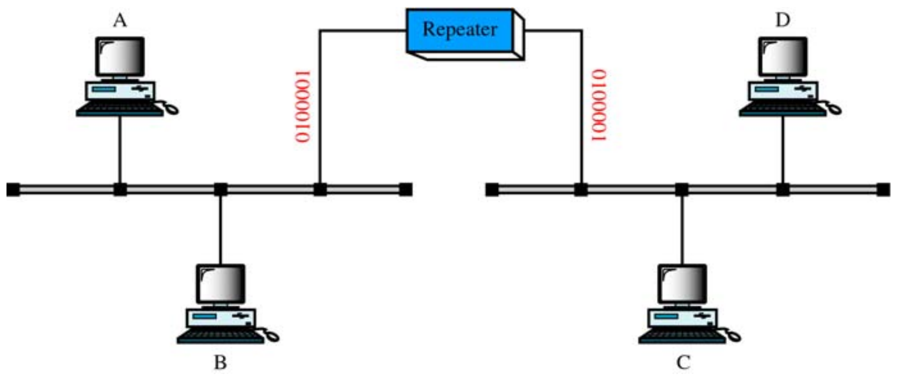
Connecting Devices and the OSI Model



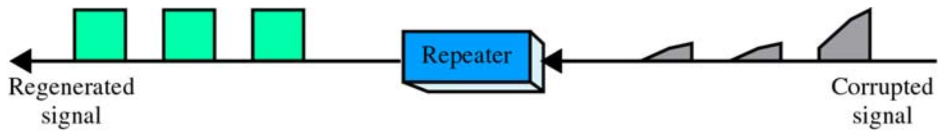
A Repeater in the OSI Model



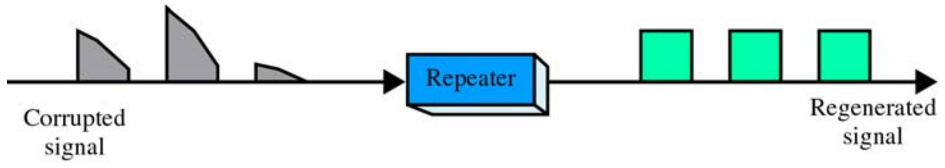
A Repeater



Function of a Repeater

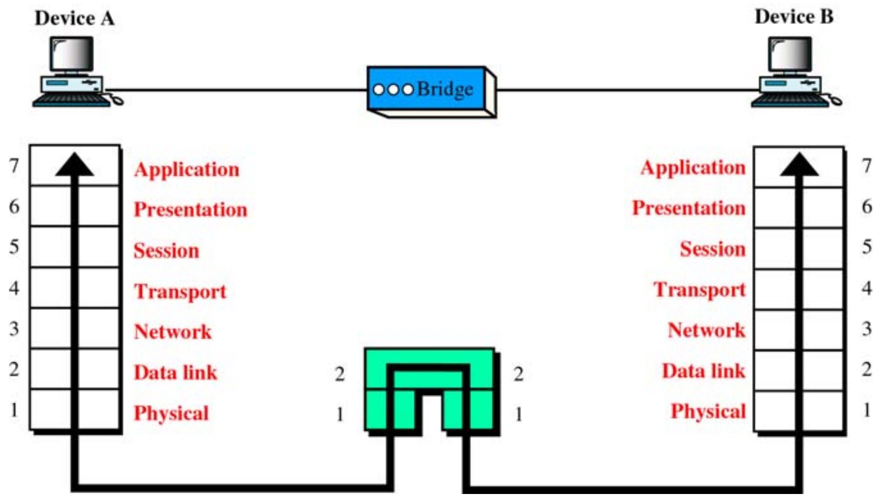


(a) Right-to-left transmission.

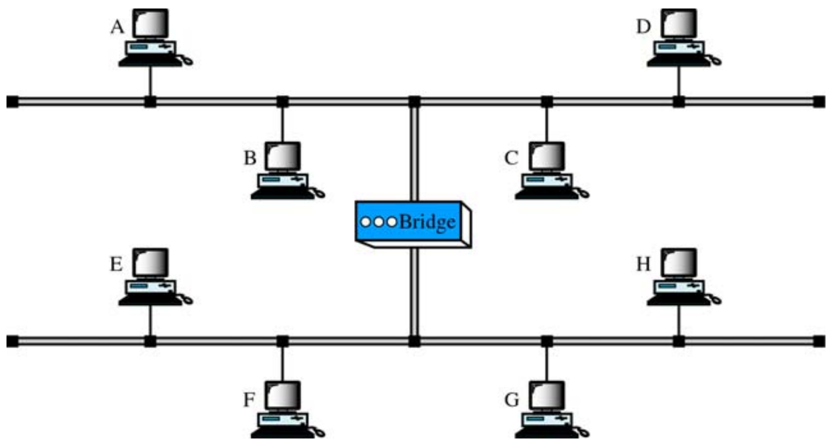


(b) Left-to-right transmission.

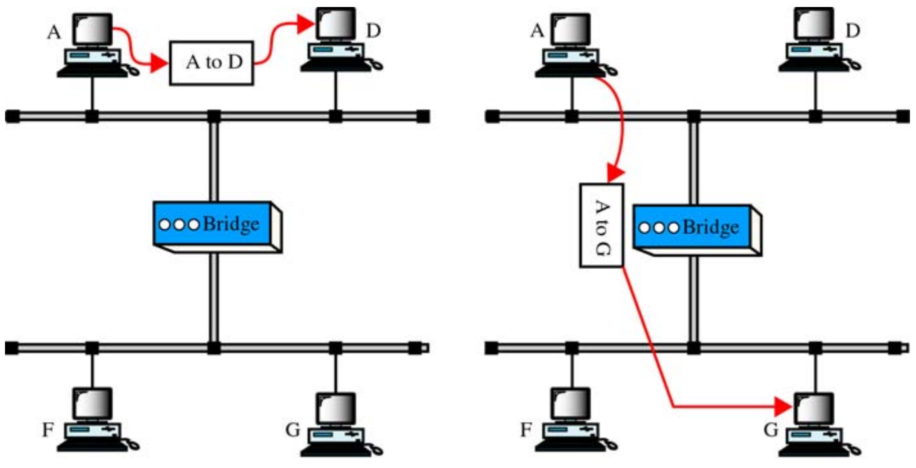
A Bridge in the OSI Model



A Bridge



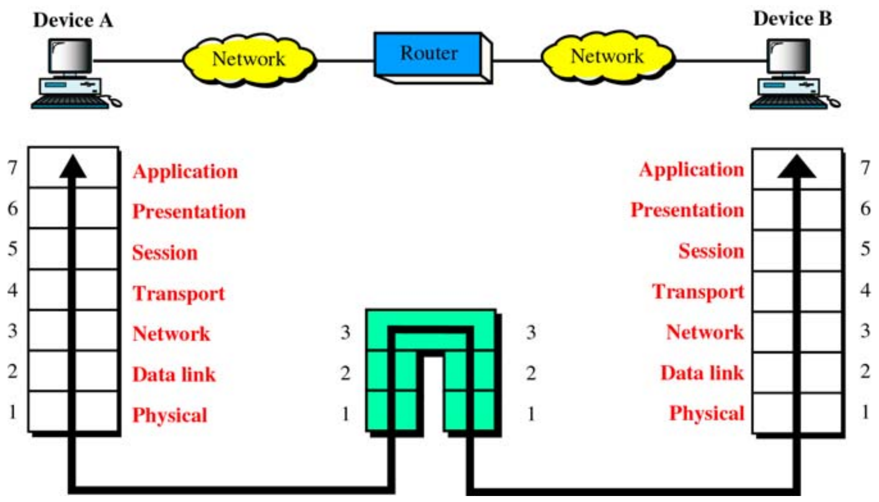
Function of a Bridge



a. A packet from A to D

b. A packet from A to G

A Router in the OSI Model



4. Routing Basics

This chapter introduces the underlying concepts widely used in routing protocols. Topics summarized here include routing protocol components and algorithms. In addition, the role of routing protocols is briefly contrasted with the roles of routed or network protocols.

4.1 What is Routing?

Routing is the act of moving information across an internetwork from a source to a destination. Along the way, at least one intermediate node typically is encountered. Routing is often contrasted with bridging, which might seem to accomplish precisely the same thing to the casual observer. The primary difference between the two is that bridging occurs at Layer 2 (the link layer) of the OSI reference model, whereas routing occurs at Layer 3 (the network layer). This distinction provides routing and bridging with different information to use in the process of moving information from source to destination, so the two functions accomplish their tasks in different ways.

The topic of routing has been covered in computer science literature for more than two decades, but routing achieved commercial popularity as late as the mid-1980s. The primary reason for this time lag is that networks in the 1970s were fairly simple, homogeneous environments. Only relatively recently has large-scale internetworking become popular.

4.2 Routing Components

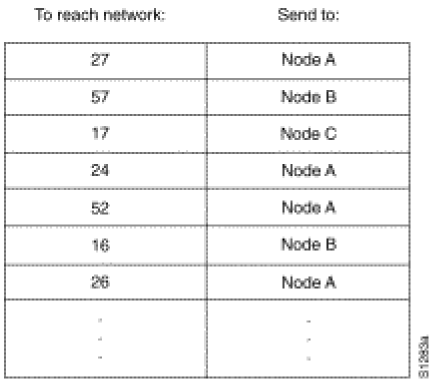
Routing involves two basic activities: determining optimal routing paths and transporting information groups (typically called *packets*) through an internetwork. In the context of the routing process, the latter of these is referred to as *switching*. Although switching is relatively straightforward, path determination can be very complex.

4.3 Path Determination

A *metric* is a standard of measurement, such as path length, that is used by routing algorithms to determine the optimal path to a destination. To aid the process of path determination, routing algorithms initialize and maintain *routing* *tables*, which contain route information. Route information varies depending on the routing algorithm used.

Routing algorithms fill routing tables with a variety of information. Destination/next hop associations tell a router that a particular destination can be gained optimally by sending the packet to a particular router representing the "next hop" on the way to the final destination. When a router receives an incoming packet, it checks the destination address and attempts to associate this address with a next hop. Figure 5-1 depicts a sample destination/next hop routing table.

Figure 4-1: Destination/next hop associations determine the data's optimal path.



To reach network:	Send to:
27	Node A
57	Node B
17	Node C
24	Node A
52	Node A
16	Node B
26	Node A
. . .	. . .

Routing tables also can contain other information, such as data about the desirability of a path. Routers compare metrics to determine optimal routes, and these metrics differ depending on the design of the routing algorithm used. A variety of common metrics will be introduced and described later in this chapter.

Routers communicate with one another and maintain their routing tables through the transmission of a variety of messages. The *routing update* message is one such message that generally consists of all or a portion of a routing table. By analyzing routing updates from all other routers, a router can build a detailed picture of network topology. A *link-state advertisement*, another example of a message sent between routers, informs other routers of the state of the sender's links. Link information also can be used to build a complete picture of topology to enable routers to determine optimal routes to network destinations.

4.4.1 Switching Types

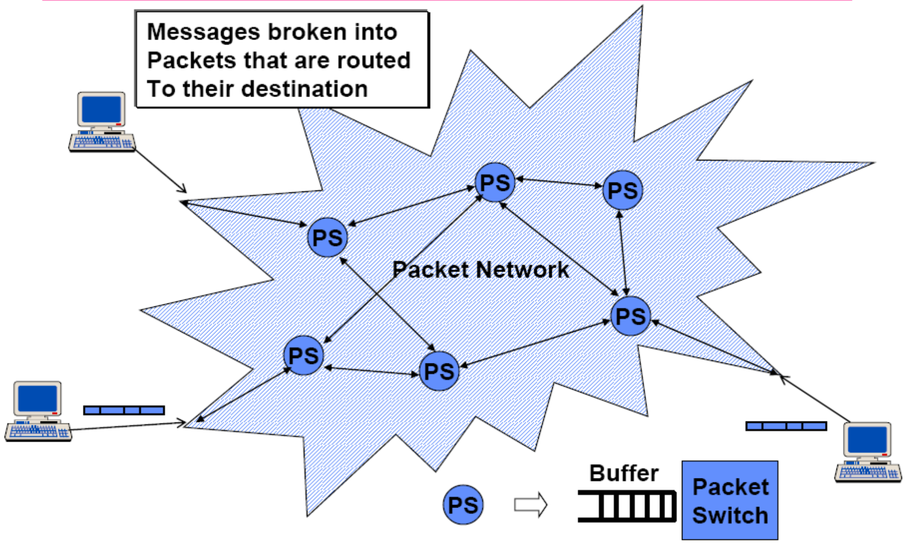
Circuit Switching

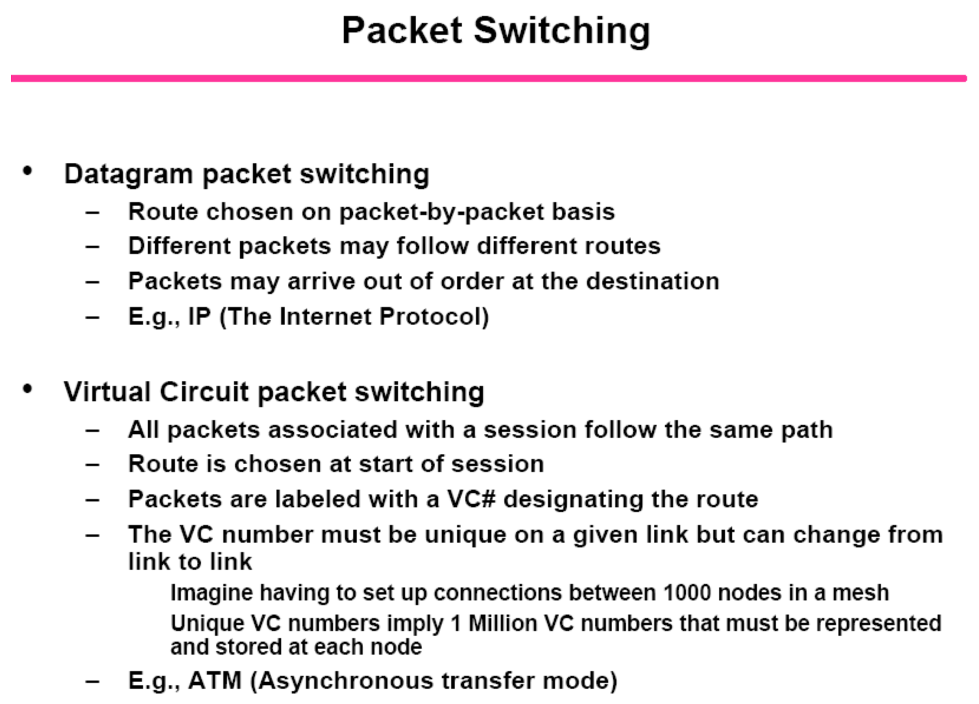
- Each session is allocated a fixed fraction of the capacity on each link along its path
  - Dedicated resources
  - Fixed path
  - If capacity is used, calls are blocked

    E.g., telephone network
- Advantages of circuit switching
  - Fixed delays
  - Guaranteed continuous delivery
- Disadvantages
  - Circuits are not used when session is idle
  - Inefficient for bursty traffic
  - Circuit switching usually done using a fixed rate stream (e.g., 64 Kbps)

    Difficult to support variable data rates

Packet Switched Networks





Packet Switching

- **Datagram packet switching**
  - **Route chosen on packet-by-packet basis**
  - **Different packets may follow different routes**
  - **Packets may arrive out of order at the destination**
  - **E.g., IP (The Internet Protocol)**
- **Virtual Circuit packet switching**
  - **All packets associated with a session follow the same path**
  - **Route is chosen at start of session**
  - **Packets are labeled with a VC# designating the route**
  - **The VC number must be unique on a given link but can change from link to link**
    - Imagine having to set up connections between 1000 nodes in a mesh
    - Unique VC numbers imply 1 Million VC numbers that must be represented and stored at each node
  - **E.g., ATM (Asynchronous transfer mode)**

4.4.2

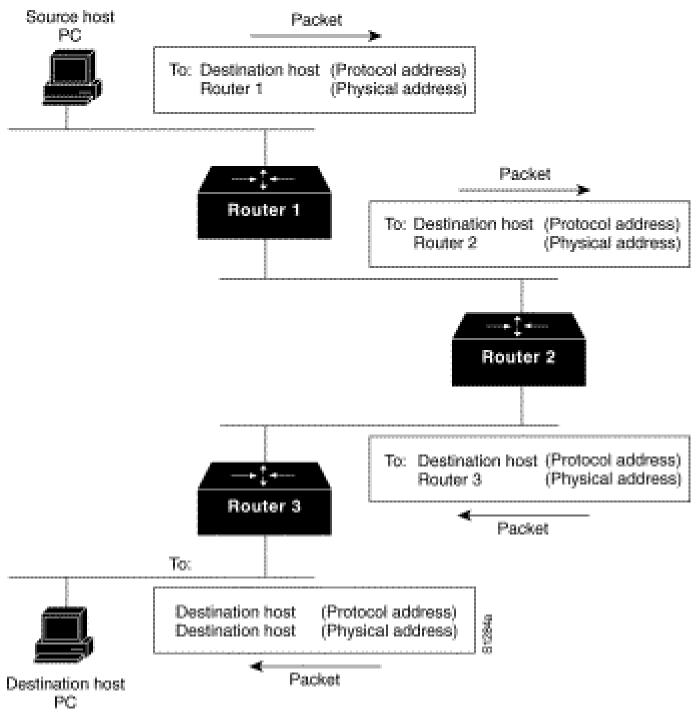
Switching algorithms are relatively simple and are basically the same for most routing protocols. In most cases, a host determines that it must send a packet to another host. Having acquired a router's address by some means, the source host sends a packet addressed specifically to a router's physical (Media Access Control [MAC]-layer) address, this time with the protocol (network- layer) address of the destination host.

As it examines the packet's destination protocol address, the router determines that it either knows or does not know how to forward the packet to the next hop. If the router does not know how to forward the packet, it typically drops the packet. If the router knows how to forward the packet, it changes the destination physical address to that of the next hop and transmits the packet.

The next hop may, in fact, be the ultimate destination host. If not, the next hop is usually another router, which executes the same switching decision process. As the packet moves through the internetwork, its physical address changes, but its protocol address remains constant.

The preceding discussion describes switching between a source and a destination end system. The International Organization for Standardization (ISO) has developed a hierarchical terminology that is useful in describing this process. Using this terminology, network devices without the capability to forward packets between subnetworks are called *end systems* (ESs), whereas network devices with these capabilities are called *intermediate systems* (ISs). ISs are further divided into those that can communicate within routing domains (*intradomain ISs*) and those that communicate both within and between routing domains (*interdomain ISs*). A *routing domain* generally is considered to be a portion of an internetwork under common administrative authority that is regulated by a particular set of administrative guidelines. Routing domains are also called *autonomous systems*. With certain protocols, routing domains can be divided into *routing areas*, but intradomain routing protocols are still used for switching both within and between areas.

Figure 4-2: Numerous routers may come into play during the switching process.


Source host PC
Packet
To: Destination host (Protocol address)
Router 1 (Physical address)
Router 1
Packet
To: Destination host (Protocol address)
Router 2 (Physical address)
Router 2
To: Destination host (Protocol address)
Router 3 (Physical address)
Packet
Router 3
To:
Destination host (Protocol address)
Destination host (Physical address)
Packet
Destination host PC
S1284a

4.5 Routing Algorithms

Routing algorithms can be differentiated based on several key characteristics. First, the particular goals of the algorithm designer affect the operation of the resulting routing protocol. Second, various types of routing algorithms exist, and each algorithm has a different impact on network and router resources. Finally, routing algorithms use a variety of metrics that affect calculation of optimal routes. The following sections analyze these routing algorithm attributes.

Design Goals

Routing algorithms often have one or more of the following design goals:

- Optimality
- Simplicity and low overhead
- Robustness and stability
- Rapid convergence
- Flexibility

Optimality refers to the capability of the routing algorithm to select the best route, which depends on the metrics and metric weightings used to make the calculation. One routing algorithm, for example, may use a number of hops and delays, but may weight delay more heavily in the calculation. Naturally, routing protocols must define their metric calculation algorithms strictly.

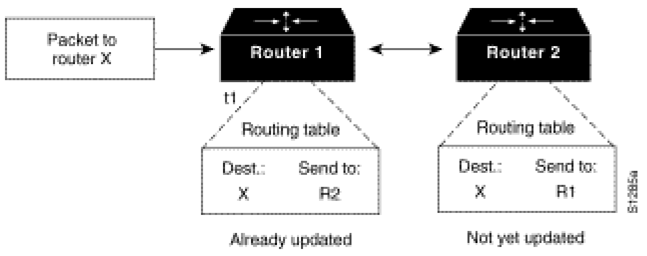
Routing algorithms also are designed to be as simple as possible. In other words, the routing algorithm must offer its functionality efficiently, with a minimum of software and utilization overhead. Efficiency is particularly important when the software implementing the routing algorithm must run on a computer with limited physical resources.

Routing algorithms must be robust, which means that they should perform correctly in the face of unusual or unforeseen circumstances, such as hardware failures, high load conditions, and incorrect implementations. Because routers are located at network junction points, they can cause considerable problems when they fail. The best routing algorithms are often those that have withstood the test of time and have proven stable under a variety of network conditions.

In addition, routing algorithms must converge rapidly. Convergence is the process of agreement, by all routers, on optimal routes. When a network event causes routes either to go down or become available, routers distribute routing update messages that permeate networks, stimulating recalculation of optimal routes and eventually causing all routers to agree on these routes. Routing algorithms that converge slowly can cause routing loops or network outages.

In the routing loop displayed in Figure 5-3, a packet arrives at Router 1 at time t1. Router 1 already has been updated and thus knows that the optimal route to the destination calls for Router 2 to be the next stop. Router 1 therefore forwards the packet to Router 2, but because this router has not yet been updated, it believes that the optimal next hop is Router 1. Router 2 therefore forwards the packet back to Router 1, and the packet continues to bounce back and forth between the two routers until Router 2 receives its routing update or until the packet has been switched the maximum number of times allowed.

Figure 4-3: Slow convergence and routing loops can hinder progress.



Routing algorithms should also be flexible, which means that they should quickly and accurately adapt to a variety of network circumstances. Assume, for example, that a network segment has gone down. As they become aware of the problem, many routing algorithms will quickly select the next-best path for all routes normally using that segment. Routing algorithms can be programmed to adapt to changes in network bandwidth, router queue size, and network delay, among other variables.

4.5.1 Algorithm Types

Routing algorithms can be classified by type. Key differentiators include:

- Static versus dynamic
- Single-path versus multi-path
- Flat versus hierarchical
- Host-intelligent versus router-intelligent
- Intradomain versus interdomain
- Link state versus distance vector

4.5.1.1Static Versus Dynamic

Static routing algorithms are hardly algorithms at all, but are table mappings established by the network administrator prior to the beginning of routing. These mappings do not change unless the network administrator alters them. Algorithms that use static routes are simple to design and work well in environments where network traffic is relatively predictable and where network design is relatively simple.

Because static routing systems cannot react to network changes, they generally are considered unsuitable for today's large, changing networks. Most of the dominant routing algorithms in the 1990s are dynamic routing algorithms, which adjust to changing network circumstances by analyzing incoming routing update messages. If the message indicates that a network change has occurred, the routing software recalculates routes and sends out new routing update messages. These messages permeate the network, stimulating routers to rerun their algorithms and change their routing tables accordingly.

Dynamic routing algorithms can be supplemented with static routes where appropriate. A *router of last resort* (a router to which all unroutable packets are sent), for example, can be designated to act as a repository for all unroutable packets, ensuring that all messages are at least handled in some way.

4.5.1.2 Single-Path Versus Multipath

Some sophisticated routing protocols support multiple paths to the same destination. Unlike single-path algorithms, these multipath algorithms permit traffic multiplexing over multiple lines. The advantages of multipath algorithms are obvious: They can provide substantially better throughput and reliability.

4.5.1.3 Flat Versus Hierarchical

Some routing algorithms operate in a flat space, while others use routing hierarchies. In a flat routing system, the routers are peers of all others. In a hierarchical routing system, some routers form what amounts to a routing backbone. Packets from non-backbone routers travel to the backbone routers, where they are sent through the backbone until they reach the general area of the destination. At this point, they travel from the last backbone router through one or more non-backbone routers to the final destination.

Routing systems often designate logical groups of nodes, called domains, autonomous systems, or areas. In hierarchical systems, some routers in a domain can communicate with routers in other domains, while others can communicate only with routers within their domain. In very large networks, additional hierarchical levels may exist, with routers at the highest hierarchical level forming the routing backbone.

The primary advantage of hierarchical routing is that it mimics the organization of most companies and therefore supports their traffic patterns well. Most network communication occurs within small company groups (domains). Because intradomain routers need to know only about other routers within their domain, their routing algorithms can be simplified, and, depending on the routing algorithm being used, routing update traffic can be reduced accordingly.

4.5.1.4 Host-Intelligent Versus Router-Intelligent

Some routing algorithms assume that the source end-node will determine the entire route. This is usually referred to as *source routing*. In source-routing systems, routers merely act as store-and-forward devices, mindlessly sending the packet to the next stop.

Other algorithms assume that hosts know nothing about routes. In these algorithms, routers determine the path through the internetwork based on their own calculations. In the first system, the hosts have the routing intelligence. In the latter system, routers have the routing intelligence.

The trade-off between host-intelligent and router-intelligent routing is one of path optimality versus traffic overhead. Host-intelligent systems choose the better routes more often, because they typically discover all possible routes to the destination before the packet is actually sent. They then choose the best path based on that particular system's definition of "optimal." The act of determining all routes, however, often requires substantial discovery traffic and a significant amount of time.

4.5.1.5 Intradomain Versus Interdomain

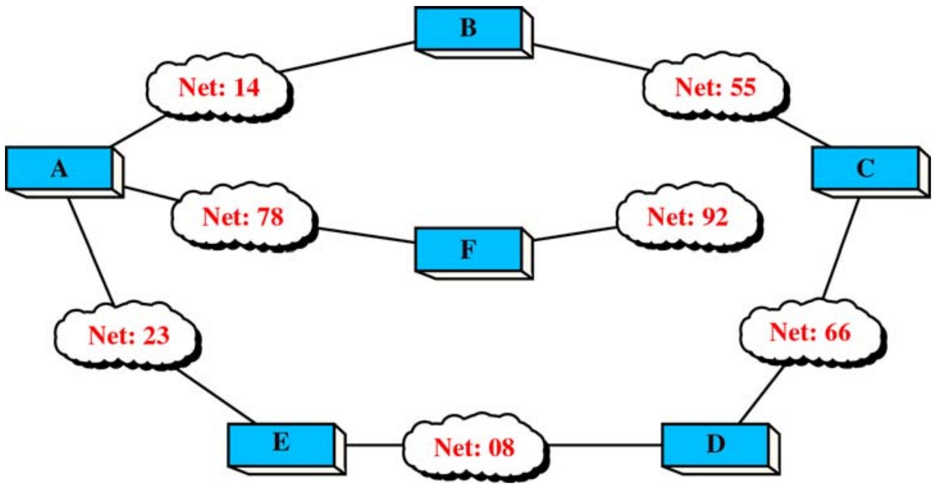
Some routing algorithms work only within domains; others work within and between domains. The nature of these two algorithm types is different. It stands to reason, therefore, that an optimal intradomain- routing algorithm would not necessarily be an optimal interdomain- routing algorithm.

4.5.1.6 Link State Versus Distance Vector

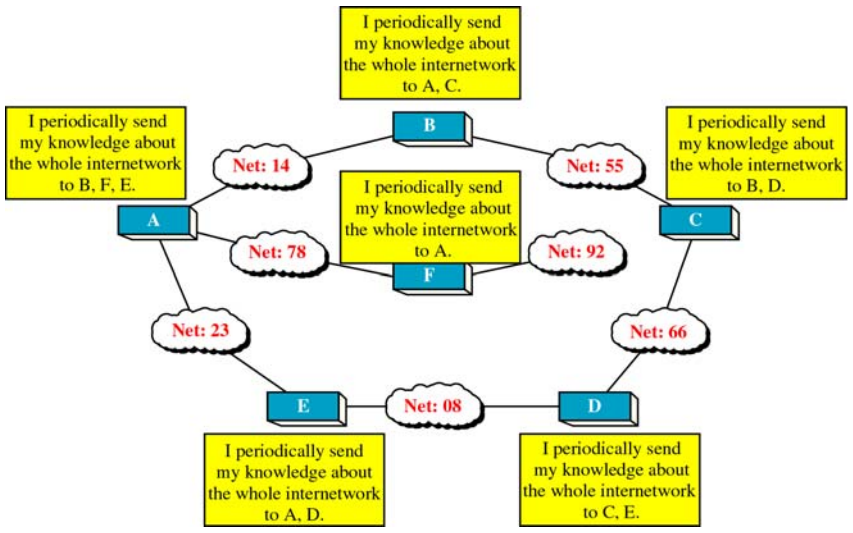
Link- state algorithms (also known as *shortest path first* algorithms) flood routing information to all nodes in the internetwork. Each router, however, sends only the portion of the routing table that describes the state of its own links. Distance-vector algorithms (also known as *Bellman-Ford* algorithms) call for each router to send all or some portion of its routing table, but only to its neighbors. In essence, link- state algorithms send small updates everywhere, while distance- vector algorithms send larger updates only to neighboring routers.

Because they converge more quickly, link- state algorithms are somewhat less prone to routing loops than distance- vector algorithms. On the other hand, link-state algorithms require more CPU power and memory than distance- vector algorithms. Link-state algorithms, therefore, can be more expensive to implement and support. Despite their differences, both algorithm types perform well in most circumstances.

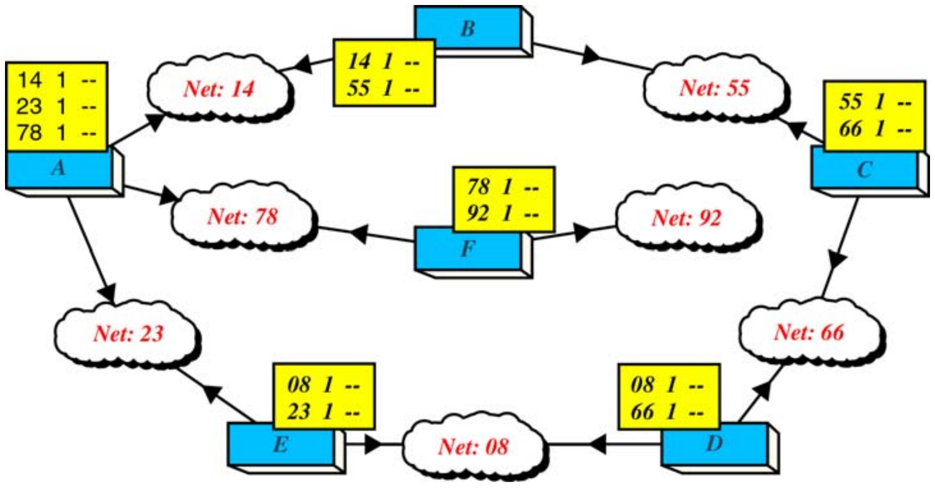
Example of an Internet



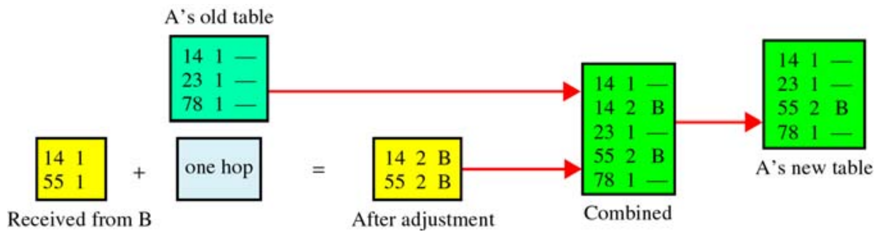
The Concept of Distance Vector Routing


I periodically send my knowledge about the whole internetwork to A, C.
I periodically send my knowledge about the whole internetwork to B, F, E.
I periodically send my knowledge about the whole internetwork to B, D.
I periodically send my knowledge about the whole internetwork to A.
B
Net: 14
Net: 55
A
C
Net: 78
F
Net: 92
Net: 23
Net: 66
E
Net: 08
D
I periodically send my knowledge about the whole internetwork to A, D.
I periodically send my knowledge about the whole internetwork to C, E.

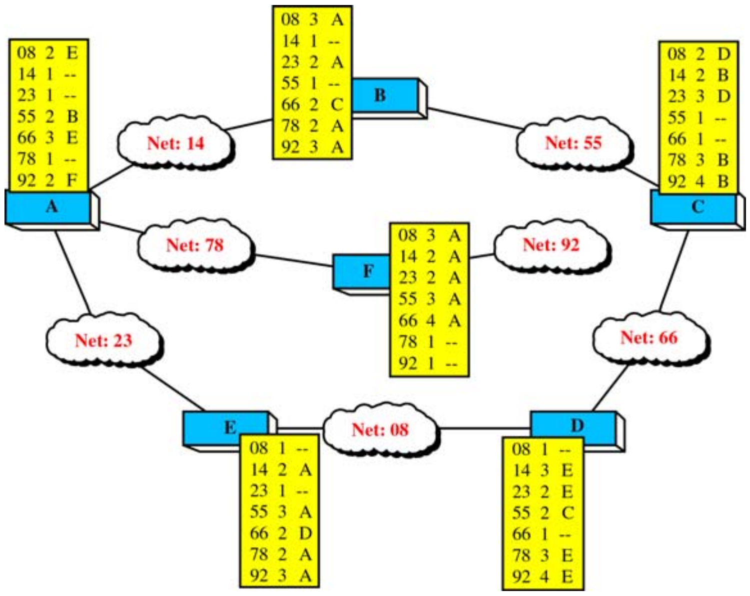
Routing Table Distribution



Updating Routing Table for Router A



Final Routing Tables



08	2	E
14	1	--
23	1	--
55	2	B
66	3	E
78	1	--
92	2	F

08	3	A
14	1	--
23	2	A
55	1	--
66	2	C
78	2	A
92	3	A

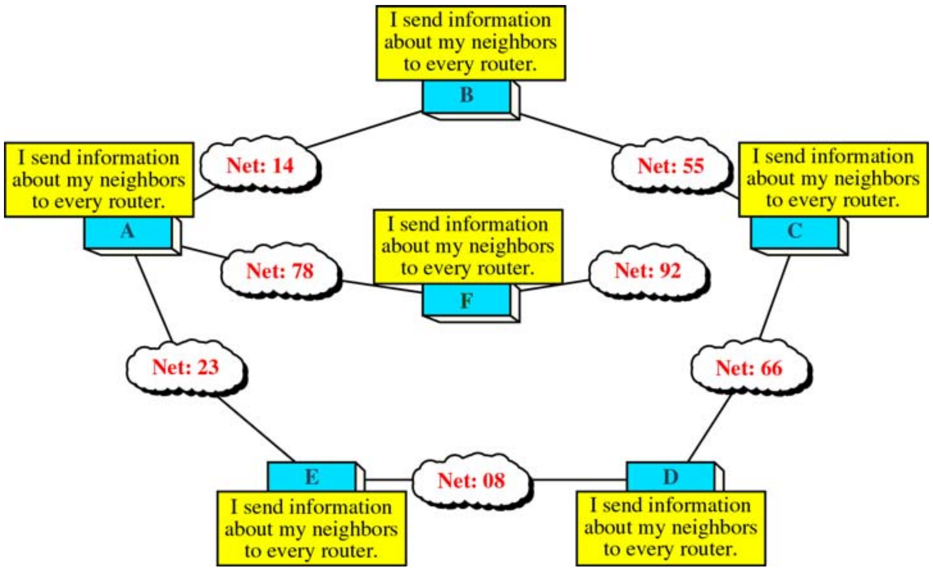
08	2	D
14	2	B
23	3	D
55	1	--
66	1	--
78	3	B
92	4	B

08	3	A
14	2	A
23	2	A
55	3	A
66	4	A
78	1	--
92	1	--

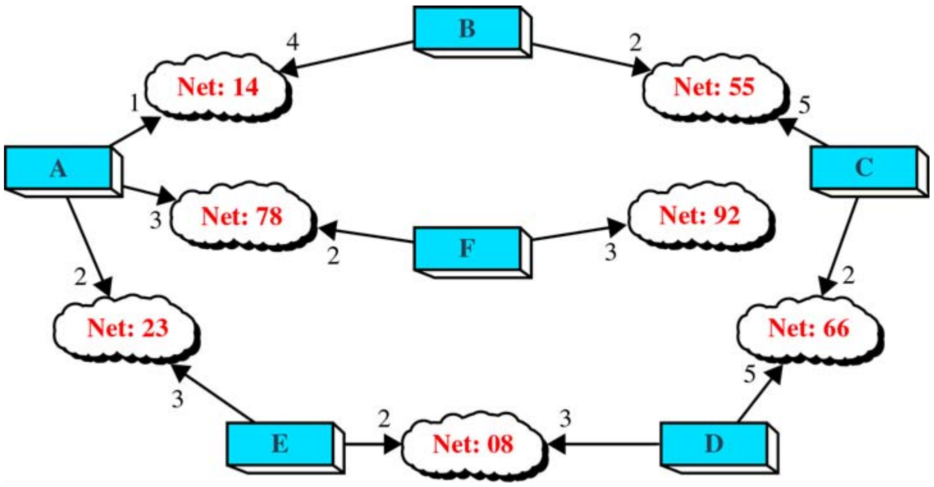
08	1	--
14	2	A
23	1	--
55	3	A
66	2	D
78	2	A
92	3	A

08	1	--
14	3	E
23	2	E
55	2	C
66	1	--
78	3	E
92	4	E

Concept of Link State Routing


I send information about my neighbors to every router.
B
I send information about my neighbors to every router.
A
Net: 14
Net: 55
I send information about my neighbors to every router.
C
I send information about my neighbors to every router.
F
Net: 78
Net: 92
Net: 23
Net: 66
E
Net: 08
D
I send information about my neighbors to every router.
I send information about my neighbors to every router.

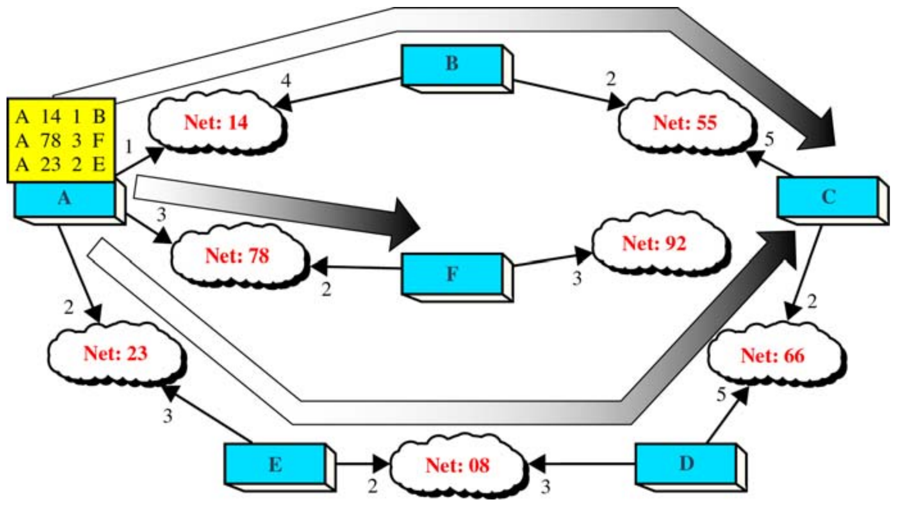
Cost in Link State Routing



Link State Packet

Advertiser	Network	Cost	Neighbor
. . . .	. . . . . .	. . . .	. . . .
. . . .	. . . . . .	. . . .	. . . .
. . . .	. . . . . .	. . . .	. . . .

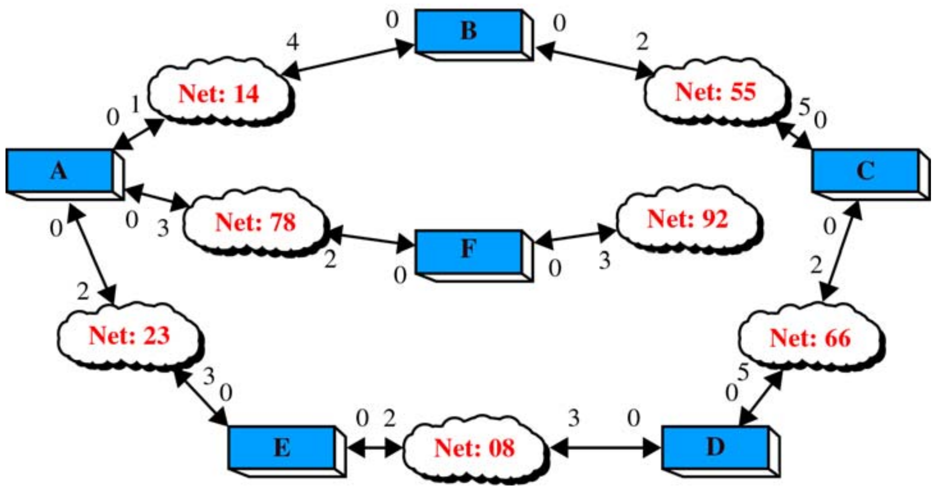
Flooding of A’s LSP



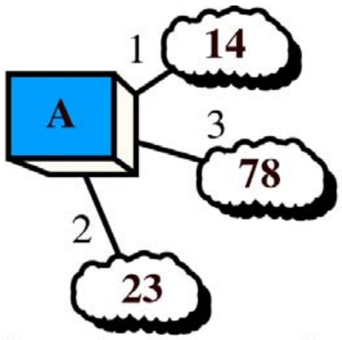
Link State Database

Advertiser	Network	Cost	Neighbor
A	14	1	B
A	78	3	F
A	23	2	E
B	**14**	**4**	**A**
B	**55**	**2**	**C**
C	55	5	B
C	66	2	D
D	**66**	**5**	**C**
D	**08**	**3**	**E**
E	23	3	A
E	08	2	D
F	**78**	**2**	**A**
F	**92**	**3**	**—**

Costs in the Dijkstra Algorithm

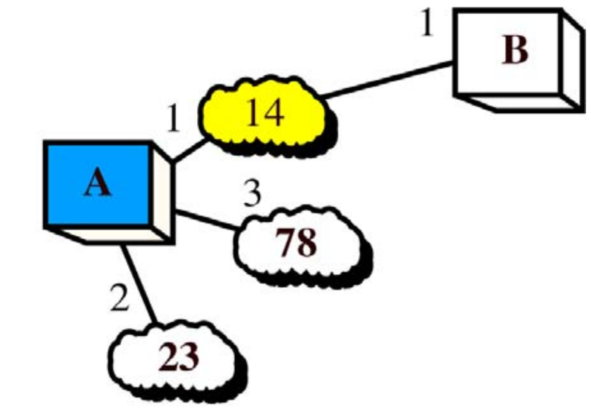


Shortest Path Calculation, Part I



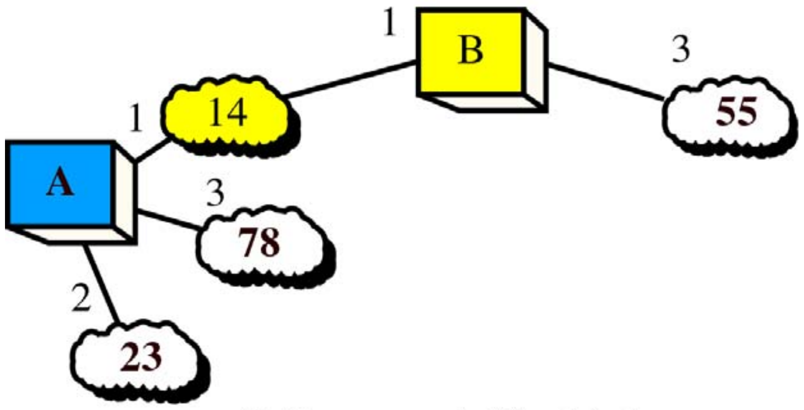
Root is A, networks 14, 78, 23 added

Shortest Path Calculation, Part II



14 permanent, B added

Shortest Path Calculation, Part III



B Permanent, 55 added

4.5.2 Routing Metrics

Routing tables contain information used by switching software to select the best route. But how, specifically, are routing tables built? What is the specific nature of the information they contain? How do routing algorithms determine that one route is preferable to others?

Routing algorithms have used many different metrics to determine the best route. Sophisticated routing algorithms can base route selection on multiple metrics, combining them in a single (hybrid) metric. All the following metrics have been used:

- Path Length
- Reliability
- Delay
- Bandwidth
- Load
- Communication Cost

Path length is the most common routing metric. Some routing protocols allow network administrators to assign arbitrary costs to each network link. In this case, path length is the sum of the costs associated with each link traversed. Other routing protocols define *hop count*, a metric that specifies the number of passes through internetworking products, such as routers, that a packet must take en route from a source to a destination.

Reliability**,** in the context of routing algorithms, refers to the dependability (usually described in terms of the bit-error rate) of each network link. Some network links might go down more often than others. After a network fails, certain network links might be repaired more easily or more quickly than other links. Any reliability factors can be taken into account in the assignment of the reliability ratings, which are arbitrary numeric values usually assigned to network links by network administrators.

Routing delay refers to the length of time required to move a packet from source to destination through the internetwork. Delay depends on many factors, including the bandwidth of intermediate network links, the port queues at each router along the way, network congestion on all intermediate network links, and the physical distance to be travelled. Because delay is a conglomeration of several important variables, it is a common and useful metric.

Bandwidth refers to the available traffic capacity of a link. All other things being equal, a 10-Mbps Ethernet link would be preferable to a 64-kbps leased line. Although bandwidth is a rating of the maximum attainable throughput on a link, routes through links with greater bandwidth do not necessarily provide better routes than routes through slower links. If, for example, a faster link is busier, the actual time required to send a packet to the destination could be greater.

Load refers to the degree to which a network resource, such as a router, is busy. Load can be calculated in a variety of ways, including CPU utilization and packets processed per second. Monitoring these parameters on a continual basis can be resource-intensive itself.

Communication cost is another important metric, especially because some companies may not care about performance as much as they care about operating expenditures. Even though line delay may be longer, they will send packets over their own lines rather than through the public lines that cost money for usage time.

4.6 Network Protocols

Routed protocols are transported by routing protocols across an internetwork. In general, routed protocols in this context also are referred to as *network* protocols. These network protocols perform a variety of functions required for communication between user applications in source and destination devices, and these functions can differ widely among protocol suites. Network protocols occur at the upper four layers of the OSI reference model: the transport layer, the session layer, the presentation layer, and the application layer.

4.7Network Layer

The internet's network layer provides connectionless datagram service rather than the virtual circuit service. When the network layer at the sending host receives a segment from the transport layer, it encapsulates the segment host receiving within an IP datagram , writes the destination host address as well as the other fields in the datagram, and sends the datagram to the first router on the path towards the destination host. The network layer in a datagram oriented network such as the Internet has three major components:-

1) The first component is the network protocol, which defines network layer addressing, the fields in the datagram (that is, the network layer- PDU), and the actions taken by routers and end systems on a datagram based on the values in these fields. The network protocolin the Internet is called the **Internet Protocol**. The current version of IP in use is **IPv4**. a newer

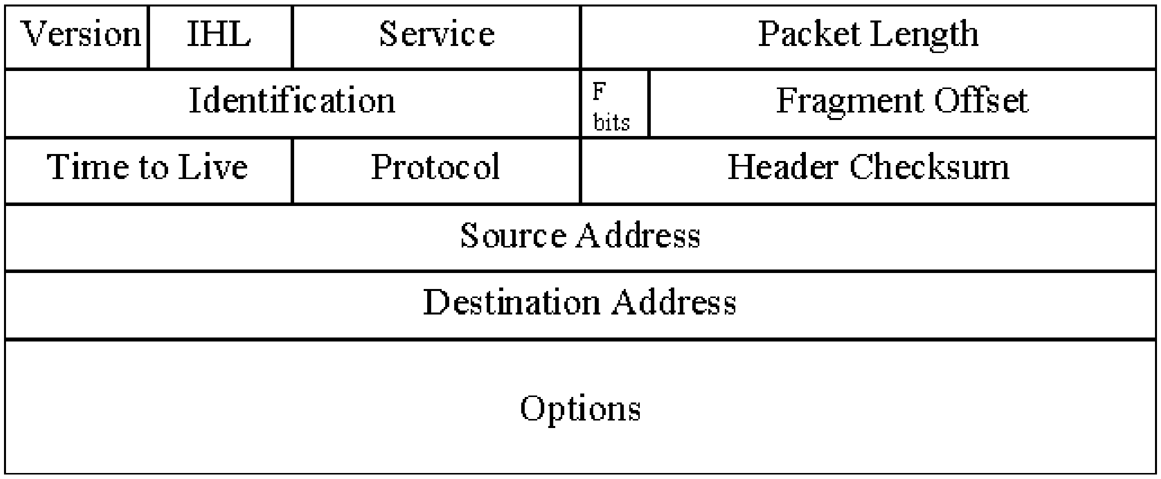
and more effective version of IP, IPv6 has been proposed and will eventually replace IPv4.

2) The second major component of the network layer is the path determination component, it determines the route a datagram follows from source to destination.
3) The final component of the network layer is a facility to report errors in datagrams and respond to requests for certain network layer information. The Internet's network layer eroor reporting protocol is called **Internet Control Message Protocol.**

IPv4 Addressing

The IP address of each host, router etc. on the Internet is 32 bits long (equivalently, four bytes), and there are thus a total of 2^{32} possible IP addresses. These addresses are typically written in so called **dotted-decimail notation**, in which each byte of the address is written in its decimal form and is separated by a period ("dot") from other bytes in the address. But the dotted notation address is not of use for making calculations. We can convert this dotted notation to long int with the help of predefined functions.

Internet Protocol Header



Version 4

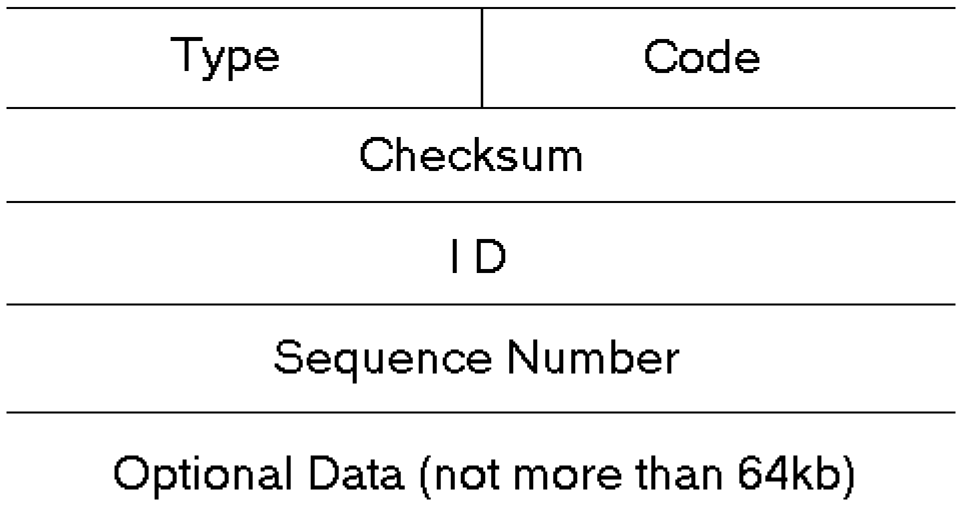
The diagram above shows the Internet Protocol header in rows of 32 bits. The packets are sent from left to right with the high order bit going first.

1) The version field specifies which version of IP the datagram is using.
2) IHL specifies the header length (4<IHL) in 32 bit words.
3) The service field allows the host to specify the required service to the subnet.
4) Packet length specifies the total datagram length in bytes. Maximum 65535.
5) The identification field is essentially the packet number and is used by the fragmentation process. The next three bits (F bits) are: unused, DF (do not fragment) and MF (More Fragments).
6) The fragment offset specifies where the first byte in the fragment was in the original packet. Fragments are multiples of 8 bytes which is the elementary fragmentation unit for IP.
7) The time to live field is a counter which was originally meant to count down in seconds, in practice it is used as a hop counter. When this field reaches zero the packet is discarded as it is assumed to be corrupted or perhaps in a routing loop.
8) The protocol field specifies which process in the transport layer the packet should be handed to (TCP, UDP…).
9) The header checksum is formed by 1’s compliment adding the 16 bit half words before this field and taking the 1’s compliment of the result.

Internet Control Message Protocol

The Internet Control Message Protocol (ICMP) is the network layers error detection and correction protocol. Thus all that ICMP does is perform diagnostic tasks on the Internet, and is not used to carry any data. In this sense, the structure of ICMP is a lot like IP. In fact, ICMP is always carried by the IP or encapsulated within the IP data packets.

The ICMP header looks like :-



The header without optional data is 8 Bytes long.

1) The type field is a single byte field that contains the type of message (11 for expired ICMP response)
2) The code field is also a single byte field that contains the code.
3) The next field is a 2 byte error detection field, Checksum. It is very similar to the Internet Protocol Header Checksum.
4) The next field is a 2 byte field called Identification. Mostly the process number of the process sending the message is put here.
5) The next Field is a 2 Byte field and it contains the sequence number of the packet
6) The last field is an optional data field. The data added will be echoed back so that the reliability of the line can be checked. The optional data MUST NOT be more than 64 KB in size or the machine at the other end may hang. This is known as the **Ping of Death** and it occurs when more than 64kb of data is sent. The extra data sent exceeds the size of the buffers and often ends up overwriting other information in memory. This causes the machine at the other end to do all sorts of strange and interesting things. This programming bug is specific to certain Operating Systems only.

Tracing Route- Method 1.

This is the obvious method that comes to mind. In this method the options of the IP header are used to obtain the path the packet traverses. The routers can be requested to add their IP to the IP header of the packet as it passes through them. Thus is we send just the single packet. We will get the IP’s of all the routers encountered on the way. But the drawbacks of this method are very prominent. Firstly since the maximum size of the IP header can only be 60 bytes in length. And 20 bytes out of these are necessary, only 40 bytes are left for the IP’s encountered on the way to be stored. Each IP will require 4 bytes space to store, thus only **10** such IP’s could be stored on the packet. It may be the better method to use in Local Area Networks where the number of intermediate machines will not exceed 10. But on the Internet it will exhaust the maximum size, more often than not. Also this may be extended to specifying the exact path the packet may take and therefore violate the general consensus among Internet Professionals because it can be too dangerous. Specifying exact packet paths may allow the promiscuous user to avoid firewalls. None the less, an elementary Trace route with an ability of successfully making 10 router hops can be created by this method.

Trace Route- Method 2

Since the method already discussed can only be used upto 10 nodes, another method must be devised that does not have this apparent limitation. The technique is that is used in ping programs, the technique of using **ICMP** (Internet Control Message Protocol) messages to receive the route. The method os implementation is discussed below.

There is a one byte field in the IP header called the Time To Live or TTL. This field holds the largest number of routers that particular packet can meet on the way to it's destination. This field was implemented to make sure that a packet that went AWOL wouldn't end up wandering the Internet forever. So if the value of the TTL is 10, then the packet see's only 9 routers. That's because each router decrements the value in the TTL field and when the nineth router gets the packet, it decrements the TTL by one (1-1=0) and then discards it when it see's that the TTL's zero. When the packet is discarded, the router sends a 'TTL Expired' ICMP error message (the Type Field is 11) to the sender. If we were to set the TTL to 1 and then dispatch a packet to a server, it'll be dropped by the first router in it's path (TTL - 1=0). That router will then send a TTL Expired message back to the sender. From that error message can be discovered the identity (the IP address) of the router and from the IP address the name of that machine can be obtained. The next packet sent will have a TTL of 2 and will be dropped by the second router, which will then dispatch an error message and so on till the packet reaches the server. On reaching the server, we can compare the IP address of the server with the IP address specified in the original message. In this way we can trace the exact route our packets will take to and fro from a certain site. If however for some reason the connection is not working, the final string obtained will be checked for it being = NULL, so that we don’t compare to NULL quantities and conclude that they are equal, which would be the case then, since we wouldn’t obtained the IP address of the domain name we specified. That will be so because the network will be deactivated for the DNS request as well. So the original destination IP and the final node encountered would both be NULL, and thus be equal, yet the connection was never made and no route traced. This would generate an error message by the algorithm.

APPENDIX A

CODE OF PROGRAM

VB CODE

```
Option Explicit

Public Sub Form_Load()

    With Combo1
      .AddItem "www.google.com"
      .AddItem "www.hotmail.com"
      .AddItem "www.microsoft.com"
      .AddItem "www.yahoo.com"
      .ListIndex = 0
   End With

   Text1.Text = ""
   Text4.Text = ""

   ReDim TabArray(0 To 3) As Long

   TabArray(0) = 30
   TabArray(1) = 54
   TabArray(2) = 105
   TabArray(3) = 182

  'Clear existing tabs
  'and set the text tabstops
   Call SendMessage(Text4.hwnd, EM_SETTABSTOPS, 0&, ByVal 0&)
   Call SendMessage(Text4.hwnd, EM_SETTABSTOPS, 4&, TabArray(0))
   Text4.Refresh

End Sub


Public Sub Command1_Click()

   Command1.Enabled = False
   Call TraceRT
   Command1.Enabled = True
```

```
End Sub


Public Function TraceRT()

  Dim ipo As ICMP_OPTIONS
  Dim echo As ICMP_ECHO_REPLY
  Dim ttl As Integer
  Dim ttlAdjust As Integer
  Dim hPort As Long
  Dim nChrsPerPacket As Long
  Dim dwAddress As Long
  Dim sAddress As String
  Dim sHostIP As String

 'set up
  Text1.Text = ""  'the target IP
  Text2.Text = "1" 'force the no of packets = 1 for a tracert
  Text4.Text = ""  'clear the output window
  List1.Clear      'for info/debuging only

 'the chars per packet - can be between 32 and 128
  If IsNumeric(Text3.Text) Then
    If Val(Text3.Text) < 32 Then Text3.Text = "32"
    If Val(Text3.Text) > 128 Then Text3.Text = "128"
  Else
    Text3.Text = "32"
  End If

  nChrsPerPacket = Val(Text3.Text)

  If SocketsInitialize() Then

   'returns the IP Address for the Host in Combo 1
   'ie returns 209.68.48.118 for www.mvps.org
    sAddress = GetIPFromHostName(Combo1.Text)

   'convert the address into an internet address.
   'ie returns 1982874833 when passed 209.68.48.118
    dwAddress = inet_addr(sAddress)

   'open an internet file handle
    hPort = IcmpCreateFile()

    If hPort <> 0 Then
```

```
'update the textboxes
 Text1.Text = sAddress
 Text4.Text = "Tracing Route to " + Combo1.Text + ":" & vbCrLf & vbCrLf

'The heart of the call. See the VBnet
'page description of the TraceRt TTL
'member and its use in performing a
'Trace Route.
 For ttl = 1 To 255

   '--------------------------------
   'for demo/dedbugging only. The
   'list will show each TTL passed
   'to the calls. Duplicate TTL's
   'mean the request timed out, and
   'additional attempts to obtain
   'the route were tried.
    List1.AddItem ttl
   '--------------------------------

   'set the IPO time to live
   'value to the current hop
    ipo.ttl = ttl

   'Call the API.
   '
   'Two items of consequence happen here.
   'First, the return value of the call is
   'assigned to an 'adjustment' variable. If
   'the call was successful, the adjustment
   'is 0, and the Next will increment the TTL
   'to obtain the next hop. If the return value
   'is 1, 1 is subtacted from the TTL value, so
   'when the next increments the TTL counter it
   'will be the same value as the last pass. In
   'doing this, routers that time out are retried
   'to ensure a completed route is determined.
   '(The values in the List1 show the actual
   ' hops/tries that the method made.)
   'i.e. if the TTL = 3 and it times out,
   '     adjust = 1 so ttl - 1 = 2. On
   '     encountering the Next, TTL is
   '     reset to 3 and the route is tried again.

   'The second thing happening concerns the
   'sHostIP member of the call. When the call
```

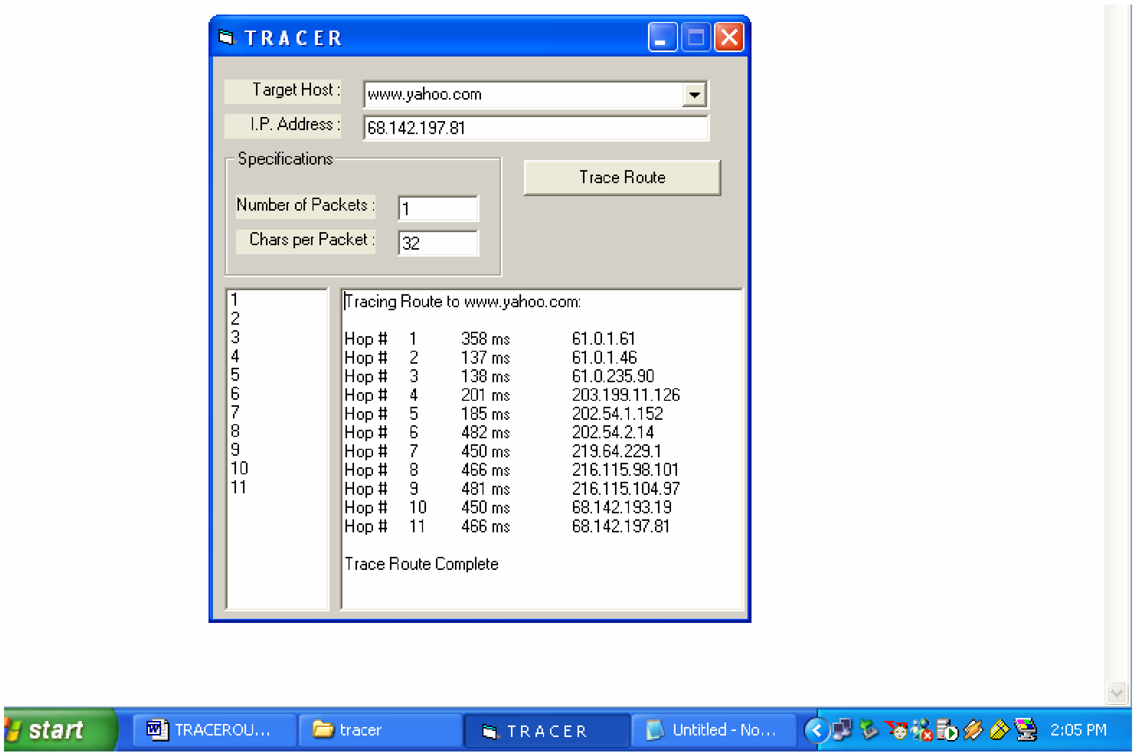
```
        'returns, sHostIP will contain the name
        'of the traced host IP.  If it matches the
        'string initially used to create the address
        '(above) were at the target, so end.
         ttlAdjust = TraceRTSendEcho(hPort, _
                          dwAddress, _
                          nChrsPerPacket, _
                          sHostIP, _
                          echo, _
                          ipo)

         ttl = ttl - ttlAdjust
        'need some processing time
         DoEvents

         If sHostIP = Text1.Text Then

           'we're done
                  Text4.Text = Text4.Text & vbCrLf + "Trace Route Completed By
         Amit Godara , Piyush and Shobhit gupta"
           Exit For

         End If

       Next ttl

      'clean up
       Call IcmpCloseHandle(hPort)

    Else: MsgBox "Unable to Open an Icmp File Handle", vbOKOnly, "VBnet
TraceRT Demo"
    End If  'If hPort

   'clean up
    Call SocketsCleanup

  Else: MsgBox "Unable to initialize the Windows Sockets", vbOKOnly, "VBnet
TraceRT Demo"
  End If  'if SocketsInitialize()

End Function


Private Sub ShowResults(timeToLive As Byte, _
                tripTime As Long, _
                sHostIP As String)
  Dim sTripTime As String
  Dim buff As String
  Dim tmp As String

 'format a string representing
 'the round trip time
  Select Case tripTime
    Case Is < 10:   sTripTime = "<10 ms"
    Case Is > 1200: sTripTime = "*"
    Case Else:      sTripTime = CStr(tripTime) & " ms"
  End Select

 'cache the textbox
  buff = Text4.Text

 'create a new entry
  tmp = "Hop #" & vbTab & _
        CStr(timeToLive) & vbTab & _
        sTripTime & vbTab & _
        sHostIP & vbCrLf

 'update textbox
  Text4.Text = buff & tmp

End Sub


Private Function TraceRTSendEcho(hPort As Long, _
                                 dwAddress As Long, _
                                 nChrsPerPacket As Long, _
                                 sHostIP As String, _
                                 echo As ICMP_ECHO_REPLY, _
                                 ipo As ICMP_OPTIONS) As Integer

  Dim sData As String
  Dim sError As String
  Dim sHostName As String
  Dim ttl As Integer

 'create a buffer to send
  sData = String$(nChrsPerPacket, "a")

  If IcmpSendEcho(hPort, _
                  dwAddress, _
                  sData, _
```

```
                Len(sData), _
                ipo, _
                echo, _
                Len(echo) + 8, _
                2400) = 1 Then

     'a reply was received, so update the display
      sHostIP = GetIPFromAddress(echo.Address)

      ShowResults ipo.ttl, echo.RoundTripTime, sHostIP

     'return 0 to continue with retrieval
      TraceRTSendEcho = 0

   Else

     'a timeout was received, so set the
     'return value to 1. In the TraceRT
     'calling routine, the TTL will be
     'de-incremented by 1, causing the
     'for / next to retry this hop.
      TraceRTSendEcho = 1

   End If

End Function
'--end block--'
```

```
Option Explicit
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Private Const WSADescription_Len As Long = 255  '256, 0-based
Private Const WSASYS_Status_Len As Long = 127   '128, 0-based
Private Const WS_VERSION_REQD As Long = &H101
Private Const SOCKET_ERROR As Long = -1
Private Const AF_INET As Long = 2
Private Const IP_SUCCESS As Long = 0
Private Const MIN_SOCKETS_REQD As Long = 1
Public Const EM_SETTABSTOPS As Long = &HCB

Private Type WSADATA
   wVersion As Integer
   wHighVersion As Integer
   szDescription(0 To WSADescription_Len) As Byte
   szSystemStatus(0 To WSASYS_Status_Len) As Byte
   imaxsockets As Integer
   imaxudp As Integer
   lpszvenderinfo As Long
End Type

Public Type ICMP_OPTIONS
   ttl             As Byte        'Time To Live
   Tos             As Byte        'Timeout
   Flags           As Byte        'option flags
   OptionsSize     As Long        '
   OptionsData     As Long        '
End Type

Public Type ICMP_ECHO_REPLY
   Address         As Long        'replying address
   Status          As Long        'reply status code
   RoundTripTime   As Long        'round-trip time, in milliseconds
   datasize        As Integer     'reply data size. Always an Int.
   Reserved        As Integer     'reserved for future use
   DataPointer     As Long        'pointer to the data in Data below
   Options         As ICMP_OPTIONS 'reply options, used in tracert
   ReturnedData    As String * 256 'the returned data follows the
                                   'reply message. The data string
                                   'must be sufficiently large enough
                                   'to hold the returned data.
End Type

Public Declare Function SendMessage Lib "user32" _
   Alias "SendMessageA" _
  (ByVal hwnd As Long, _
   ByVal wMsg As Long, _
   ByVal wParam As Long, _
   lParam As Any) As Long

Private Declare Function WSAStartup Lib "wsock32" _
  (ByVal VersionReq As Long, _
   WSADataReturn As WSADATA) As Long

Private Declare Function WSACleanup Lib "wsock32" () As Long

Public Declare Function inet_addr Lib "wsock32" _
  (ByVal s As String) As Long

Private Declare Function gethostbyaddr Lib "wsock32" _
  (haddr As Long, _
   ByVal hnlen As Long, _
   ByVal addrtype As Long) As Long

Private Declare Function gethostname Lib "wsock32" _
   (ByVal szHost As String, _
    ByVal dwHostLen As Long) As Long

Private Declare Function gethostbyname Lib "wsock32" _
   (ByVal szHost As String) As Long

Private Declare Sub CopyMemory Lib "kernel32" _
   Alias "RtlMoveMemory" _
  (Dest As Any, _
   Source As Any, _
   ByVal nbytes As Long)

Private Declare Function inet_ntoa Lib "wsock32.dll" _
   (ByVal addr As Long) As Long

Private Declare Function lstrcpyA Lib "kernel32" _
  (ByVal RetVal As String, _
   ByVal Ptr As Long) As Long

Private Declare Function lstrlenA Lib "kernel32" _
  (ByVal Ptr As Any) As Long

Public Declare Function IcmpCreateFile Lib "icmp.dll" () As Long

Public Declare Function IcmpCloseHandle Lib "icmp.dll" _
   (ByVal IcmpHandle As Long) As Long

Public Declare Function IcmpSendEcho Lib "icmp.dll" _
   (ByVal IcmpHandle As Long, _
    ByVal DestinationAddress As Long, _
    ByVal RequestData As String, _
    ByVal RequestSize As Long, _
    RequestOptions As ICMP_OPTIONS, _
    ReplyBuffer As ICMP_ECHO_REPLY, _
    ByVal ReplySize As Long, _
    ByVal Timeout As Long) As Long


Public Function GetIPFromHostName(ByVal sHostName As String) As String

  'converts a host name to an IP address.
```

```
  Dim ptrHosent As Long      'address of hostent structure
  Dim ptrName As Long        'address of name pointer
  Dim ptrAddress As Long     'address of address pointer
  Dim ptrIPAddress As Long   'address of string holding final IP address
  Dim dwAddress As Long      'the final IP address

  ptrHosent = gethostbyname(sHostName & vbNullChar)

  If ptrHosent <> 0 Then

    'assign pointer addresses and offset

    'ptrName is the official name of the host (PC).
    'If using the DNS or similar resolution system,
    'it is the Fully Qualified Domain Name (FQDN)
    'that caused the server to return a reply.
    'If using a local hosts file, it is the first
    'entry after the IP address.
     ptrName = ptrHosent

    'Null-terminated list of addresses for the host.
    'The Address is offset 12 bytes from the start of
    'the HOSENT structure. Addresses are returned
    'in network byte order.
     ptrAddress = ptrHosent + 12

    'get the actual IP address
     CopyMemory ptrAddress, ByVal ptrAddress, 4
     CopyMemory ptrIPAddress, ByVal ptrAddress, 4
     CopyMemory dwAddress, ByVal ptrIPAddress, 4

     GetIPFromHostName = GetIPFromAddress(dwAddress)

  End If

End Function


Public Sub SocketsCleanup()

 'only show error if unable to clean up the sockets
  If WSACleanup() <> 0 Then
     MsgBox "Windows Sockets error occurred during Cleanup.", vbExclamation
  End If

End Sub
```

```
Public Function SocketsInitialize() As Boolean

   Dim WSAD As WSADATA

  'when the socket version returned == version
  'required, return True
   SocketsInitialize = WSAStartup(WS_VERSION_REQD, WSAD) = IP_SUCCESS

End Function


Public Function GetIPFromAddress(Address As Long) As String

   Dim ptrString As Long

   ptrString = inet_ntoa(Address)
   GetIPFromAddress = GetStrFromPtrA(ptrString)

End Function


Public Function GetStrFromPtrA(ByVal lpszA As Long) As String

   GetStrFromPtrA = String$(lstrlenA(ByVal lpszA), 0)
   Call lstrcpyA(ByVal GetStrFromPtrA, ByVal lpszA)

End Function
'--end block--'
```



Target Host : www.yahoo.com

I.P. Address : 68.142.197.81

Number of Packets : 1

Chars per Packet : 32

```
1
2
3
4
5
6
7
8
9
10
11
```

```
Tracing Route to www.yahoo.com:

Hop #   1     358 ms      61.0.1.61
Hop #   2     137 ms      61.0.1.46
Hop #   3     138 ms      61.0.235.90
Hop #   4     201 ms      203.199.11.126
Hop #   5     185 ms      202.54.1.152
Hop #   6     482 ms      202.54.2.14
Hop #   7     450 ms      219.64.229.1
Hop #   8     466 ms      216.115.98.101
Hop #   9     481 ms      216.115.104.97
Hop #   10    450 ms      68.142.193.19
Hop #   11    466 ms      68.142.197.81

Trace Route Complete
```

APPENDIX B
CODE OF PROGRAM

C CODE

```
/***********************HEADER FILE TRACE.H***********************/
#include "unp.h"
#include <netinet/in_systm.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/udp.h>

#define BUFSIZE  1500

struct rec {
        u_short rec_seq;
```

```
            u_short rec_ttl;
            struct timeval rec_tv;
  };

 /*globals*/

char recvbuf[BUFSIZE];
char sendbuf[BUFSIZE];

int datalen;
char *host;
u_short sport,dport;
int nsent;
pid_t pid;
int probe,nprobes;
int sendfd,recvfd;
int ttl, max_ttl;
int verbose;



 /* function prototype */

const char *icmpcode_v4(int);
const char *icmpcode_v6(int);
int  recv_v4(int, struct timeval *);
int  recv_v6(int, struct timeval *);
void sig_alrm(int);
void traceloop(void);
void tv_sub(struct timeval *,struct timeval *);

struct proto {
        const char *(*icmpcode) (int);
        int (*recv) (int,struct timeval *);
        struct sockaddr *sasend;
        struct sockaddr *sarecv;
        struct sockaddr *salast;
        struct sockaddr *sabind;
            socklen_t salen;
            int icmpproto;
            int ttllevel;
            int ttloptname;
  } *pr;

 #ifdef IPV6
```

```
#include <netinet/ip6.h>
#include <netinet/icmp6.h>

#endif
```

/*************************HEADER FILE UNP.H*************************/

```
/* Our own header*/

# ifndef ___unp_h
# define ___unp_h

# include "config.h"

# include <sys/types.h>
# include <sys/socket.h>
# include <sys/time.h>
# include <time.h>
# include <netinet/in.h>
# include <arpa/inet.h>
# include <errno.h>
# include <fcntl.h>
# include <netdb.h>
# include <signal.h>
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <sys/stat.h>
# include <sys/uio.h>
# include <unistd.h>
# include <sys/wait.h>
# include <sys/un.h>

#ifdef HAVE_SYS_SELECT_H
# include <sys/sysctl.h>
#endif

#ifdef HAVE_POLL_H
# include <poll.h>
#endif
```

```
#ifdef HAVE_SYS_EVENT_H
# include <sys/event.h>
#endif

#ifdef HAVE_STRINGS_H
# include <strings.h>
#endif

#ifdef HAVE_SYS_IOCTL_H
# include <sys/ioctl.h>
#endif

#ifdef HAVE_SYS_FILIO_H
# include <sys/filio.h>
#endif

#ifdef HAVE_SYS_SOCKIO_H
# include <sys/sokio.h>
#endif

#ifdef HAVE_PTHREAD_H
# include <pthread.h>
#endif

#ifdef HAVE_NET_IF_DL_H
# include <net/if_dl.h>
#endif

#ifdef HAVE_NETINET_SCTP_H
# include <netinet/sctp.h>
#endif

#ifdef __osf__
#undef recv
#undef send

#define recv(a,b,c,d)    recvfrom(a,b,c,d,0,0)
#define send(a,b,c,d)    sendto(a,b,c,d,0,0)
#endif

#ifndef INADDR_NONE
#define INADDR_NONE  0xffffffff
#endif

#ifndef SHUT_RD
```

```
#define SHUT_RD  0
#define SHUT_WR  1
#define SHUT_RDWR  2
#endif

#ifndef INET_ADDRSTRLEN
#define INET_ADDRSTRLEN  16
#endif

#ifndef INET6_ADDRSTRLEN
#define INET6_ADDRSTRLEN  46
#endif

#ifndef HAVE_BZERO
#define bzero(ptr,n)     memset(ptr,0,n)
#endif

#ifndef HAVE_GETHOSTBYNAME2
#define gethostbyname2(host,family)   gethostbyname((host))
#endif

struct unp_in_pktinfo {
          struct in_addr  ipi_addr;
          int ipi_ifindex;
  };


#ifndef CMSG_LEN
#define CMSG_LEN(size)   (sizeof(struct cmsghdr) + (size) )
#endif

#ifndef CMSG_SPACE
#define CMSG_SPACE(size)   (sizeof(struct cmsghdr) + (size) )
#endif

#ifndef SUN_LEN
#define SUN_LEN(su)  (sizeof(*(su))- sizeof ((su)->sun_path) + strlen((su)->sun_path))
#endif

#ifndef AF_LOCAL
#define AF_LOCAL  AF_UNIX
#endif

#ifndef PF_LOCAL
#define PF_LOCAL  PF_UNIX
#endif
```

```
#ifndef INFTIM
#define INFTIM   (-1)
#ifdef HAVE_POLL_H
#define INFTIM_UNP
#endif
#endif

#define LISTENQ  1024
#define MAXLINE 4096
#define BUFFERSIZE  8192

#define SERV_PORT  9877
#define SERV_PORT_STR  "9877"
#define UNIXSTR_PATH
#define UNIXDG_PATH

#define SA struct sockaddr
//#define HAVE_STRUCT_SOCKADDR_STORAGE
#ifndef HAVE_STRUCT_SOCKADDR_STORAGE

#define  __SS_MAXSIZE   128
#define __SS_ALIGNSIZE  (sizeof(int64_t))
#ifdef HAVE_SOCKADDR_SA_LEN
#define __SS_PAD1SIZE  (__SS_ALIGNSIZE - sizeof(u_char) - sizeof(sa_family_t))
#else
#define __SS_PAD1SIZE  (__SS_ALIGNSIZE - sizeof(sa_family_t))
#endif
#define __SS_PAD2SIZE  (__SS_MAXSIZE - 2*__SS_ALIGNSIZE)

struct sockaddr_storage {
#ifdef HAVE_SOCKADDR_SA_LEN
        u_char ss_len;
#endif
        sa_family_t ss_family;
        char __ss_pad1[__SS_PAD1SIZE];
        int64_t __ss_align;
        char __ss_pad2[__SS_PAD2SIZE];
    };
#endif

#define FILE_MODE (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
#define DIR_MODE (FILE_MODE | S_IXUSR | S_IXGRP | S_IXOTH)

typedef void Sigfunc (int);
```

```
#define min(a,b)   ((a) < (b) ? (a) : (b))
#define max(a,b)   ((a) > (b) ? (a) : (b))

#ifndef HAVE_ADDRINFO_STRUCT
#include "../lib/addrinfo.h"
#endif

#ifndef HAVE_IF_NAMEINDEX_STRUCT
struct if_nameindex {
        unsigned int if_index;
        char *if_name;
  };
#endif

#ifndef HAVE_TIMESPEC_STRUCT
struct timespec {
       time_t tv_sec;
       long tv_nsec;
   };
#endif
```

/*********************HEADER FILE CONFIG.H**********************/

```
#define CPU_VENDOR_OS "sparc64-unknown-freebsd5.1"

#define HAVE_ADDRINFO_STRUCT  1

#define HAVE_ARPA_INET_H  1

#define HAVE_BZERO  1

#define HAVE_ERRNO_H  1

#define HAVE_FCNTL_H  1

#define HAVE_GETADDRINFO  1

#define HAVE_GETADDRINFO_PROTO  1

#define HAVE GETHOSTBYNAME2  1

#define HAVE_GETHOSTNAME  1
```

```
#define HAVE_GETHOSTNAME_PROTO  1

#define HAVE_GETNAMEINFO  1

#define HAVE_GETNAMEINFO_PROTO  1

#define HAVE_GETRUSAGE_PROTO  1

#define HAVE_HSTRERROR  1

#define HAVE_HSTRERROR_PROTO  1

#define HAVE_IF_NAMEINDEX_STRUCT  1

#define HAVE_IF_NAMETOINDEX 1

#define HAVE_IF_NAMETOINDEX_PROTO  1

#define HAVE_INET_PTON  1

#define HAVE_INET_PTON_PROTO  1

#define HAVE_KEVENT  1

#define HAVE_KQUEUE  1

#define HAVE_MKSTEMP  1

#define HAVE_NETCONFIG_H  1

#define HAVE_MSGHDR_MSG_CONTROL  1

#define HAVE_NETDB_H  1

#define HAVE_NETINET_IN_H  1

#define HAVE_NET_IF_DL_H  1

#define HAVE_POLL_H  1

#define HAVE_PSELECT  1

#define HAVE_PSELECT_PROTO  1

#define HAVE_PTHREAD_H  1
```

```
#define HAVE_SIGNAL_H  1

#define HAVE_SPRINTF  1

#define HAVE_SPRINTF_PROTO  1

#define HAVE_SOCKADDR_SA_LEN  1

#define HAVE_SOCKADDR_DL_STRUCT  1

#define HAVE_SOCKATMARK  1

#define HAVE_SOCKATMARK_PROTO  1

#define HAVE_STDIO_H  1

#define HAVE_STDLIB_H  1

#define HAVE_STRINGS_H  1

#define HAVE_STRING_H  1

#define HAVE_STRUCT_IFREQ_IFR_MTU  1

#define HAVE_STRUCT_SOCKADDR_STORAGE  1

#define HAVE_SYS_EVENT_H  1

#define HAVE_SYS_FILIO_H  1

#define HAVE_SYS_IOCTL_H  1

#define HAVE_SYS_SELECT_H  1

#define HAVE_SYS_SOCKET_H  1

#define HAVE_SYS_SOCKIO_H  1

#define HAVE_SYS_STAT_H  1

#define HAVE_SYS_SYSCTL_H  1

#define HAVE_SYS_TIME_H  1

#define HAVE_SYS_TYPES_H  1
```

```
 #define HAVE_SYS_UIO_H  1

 #define HAVE_SYS_UN_H  1

 #define HAVE_SYS_WAIT_H  1

 #define HAVE_TIMESPEC_STRUCT  1

 #define HAVE_TIME_H  1

 #define HAVE_UNISTD_H  1

 #define HAVE_VSNPRINTF  1

 #define IPV4  1

 #define IPv4  1

 #define MCAST  1

 #define STDC_HEADERS  1

 #define TIME_WITH_SYS_TIME  1

 #define UNIXDOMAIN  1

 #define UNIXdomain  1

 #define t_scalar_t int32_t

 #define t_uscalar_t uint32_t

 #define __FAVOR_BSD  1
```

/****************************MAIN.C****************************/

```
 #include "trace.h"

 struct proto proto_v4 = { icmpcode_v4, recv_v4, NULL, NULL, NULL, NULL, 0,
IPPROTO_ICMP, IPPROTO_IP, IP_TTL };

 #ifdef IPV6
```

```
  struct proto proto_v6 = { icmpcode_v6, recv_v6, NULL, NULL, NULL, NULL, 0,
IPPROTO_ICMPV6, IPPROTO_IPV6, IPV6_UNICAST_HOPS };

 #endif

 int datalen = sizeof(struct rec);
 int max_ttl = 30;
 int nprobes = 3;
 u_short dport = 32768 + 666;

 int
 main (int argc, char **argv)
  {
        int c;
        struct addrinfo *ai;
        char *h;

        opterr = 0;
        while ( (c = getopt(argc, argv, "m:v")) != -1) {
                switch (c) {
                        case 'm':
                                if ( (max_ttl = atoi(optarg)) <= 1)
                                        err_quit("invalid -m value");
                                break;

                        case 'v':
                                verbose++;
                                break;

                        case '?':
                                err_quit ("unrecognized option: %c",c);
                           }
        }

 if (optind != argc - 1)
        err_quit("usage: traceroute [ -m <maxttl> -v ] <hostname>");
  host = argv[optind];

  pid = getpid();
  Signal (SIGALRM, sig_alrm);

  ai = Host_serv(host, NULL, 0, 0);

  h = Sock_ntop_host (ai->ai_addr, ai->ai_addrlen);
  printf("traceroute to %s (%s): %d hops max, %d data bytes \n",ai->ai_canonname ? ai-
>ai_canonname : h, h, max_ttl, datalen);
```

```
 /*initialize according to protocol*/

  if (ai->ai_family == AF_INET) {
          pr = &proto_v4;

 #ifdef IPV6
  }

  else if (ai->ai_family == AF_INET6) {
          pr = &proto_v6;
          if (IN6_IS_ADDR_V4MAPPED (&(((struct sockaddr_in6 *) ai->ai_addr)-
>sin6_addr)))
                  err_quit("cannot traceroute IPv4-mapped IPv6 address");
 #endif
   } else
          err_quit("unknown address family %d", ai->ai_family);

          pr->sasend = ai->ai_addr;
          pr->sarecv = Calloc(1, ai->ai_addrlen);
          pr->salast = Calloc(1,ai->ai_addrlen);
          pr->sabind = Calloc(1,ai->ai_addrlen);
          pr->salen = ai->ai_addrlen;

   traceloop();


   exit (0);

  }
```

/************************TRACELOOP.C**************************/

```
 #include "trace.h"

  void
  traceloop(void)
   {
          int seq, code, done;
          double rtt;
          struct rec *rec;
          struct timeval tvrecv;

          recvfd = Socket(pr->sasend->sa_family, SOCK_RAW, pr->icmpproto);
          setuid(getuid());

  #ifdef IPV6
          if (pr->sasend->sa_family == AF_INET6 && verbose == 0)
           {
                  struct icmp6_filter myfilt;
                  ICMP6_FILTER_SETBLOCKALL(&myfilt);
                  ICMP6_FILTER_SETPASS(ICMP6_TIME_EXCEEDED< &myfilt);
                  ICMP6_FILTER_SETPASS(ICMP6_DST_UNREACH, &myfilt);
                  setsockopt(recvfd, IPPROTO_IPV6, ICMP6_FILTER, &myfilt,
sizeof(myfilt));
           }
  #endif

          sendfd = Socket(pr->sasend->sa_family, SOCK_DGRAM, 0);

          pr->sabind->sa_family = pr->sasend->sa_family;
          sport = (getpid() & 0xffff) | 0x8000;    /* our source UDP port# */
          sock_set_port(pr->sabind, pr->salen, htons(sport));
          Bind(sendfd, pr->sabind, pr->salen);

          sig_alrm(SIGALRM);

          seq = 0;
          done = 0;

          for (ttl = 1; ttl <= max_ttl && done == 0; ttl++)
           {
                  Setsockopt(sendfd, pr->ttllevel, pr->ttloptname, &ttl, sizeof(int));
                  bzero(pr->salast, pr->salen);

                  printf("%2d ",ttl);
                  fflush(stdout);

                  for (probe = 0; probe < nprobes; probe++)
                   {
                          rec = (struct rec *) sendbuf;
                          rec->rec_seq = ++seq;
                          rec->rec_ttl = ttl;
                          Gettimeofday(&rec->rec_tv, NULL);

                          sock_set_port(pr->sasend, pr->salen, htons(dport + seq));
                          Sendto(sendfd, sendbuf, datalen, 0, pr->sasend, pr->salen);

                          if ( (code = (*pr->recv) (seq, &tvrecv)) == -3)
                                  printf(" *");     /* timeout no reply */
```

```
                                else {
                                        char str[NI_MAXHOST];

                                        if (sock_cmp_addr(pr->sarecv, pr->salast, pr->salen) != 0)
{
                                                if (getnameinfo(pr->sarecv, pr->salen, str,
sizeof(str), NULL, 0, 0) == 0)
                                                printf(" %s (%s)", str, Sock_ntop_host(pr->sarecv,
pr->salen));
                                        else
                                                printf(" %s", Sock_ntop_host(pr->sarecv, pr-
>salen));
                                        memcpy(pr->salast, pr->sarecv, pr->salen);
                                    }

                                tv_sub(&tvrecv, &rec->rec_tv);
                                rtt = tvrecv.tv_sec * 1000.0 + tvrecv.tv_usec / 1000.0;
                                printf("   %.3f ms",rtt);

                                if (code == -1)  /* port unreachable; at destination */
                                        done++;
                                else if (code >= 0)
                                        printf("  (ICMP %s)", (*pr->icmpcode) (code));
                          }
                        fflush(stdout);
                }
                printf("\n");
        }
  }
```

/**********************ICMPCODE_V6.C**************************/

```
 #include "trace.h"

  const char *
   icmpcode_v6(int code)
    {

     #ifdef IPV6
          static char errbuf[100];
          switch (code)
```

```
                {
                        case ICMP6_DST_UNREACH_NOROUTE:
                                return ("no route to host");
                        case ICMP6_DST_UNREACH_ADMIN:
                                return ("administratively prohibited");
                        case ICMP6_DST_UNREACH_NOTNEIGHBOR:
                                return ("not a neighbor");
                        case ICMP6_DST_UNREACH_ADDR:
                                return ("address unreachable");
                        case ICMP6_DST_UNREACH_NOPORT:
                                return ("port unreachable");
                        default:
                                sprintf(errbuf, "[unknown code %d]", code);
                                return errbuf;
                }

  #endif
 }
```

/***************************RECV_V4.C*****************************/

```
#include "trace.h"

extern int gotalarm;

/*
 * return:    -3 on timeout
                -2 on ICMP time exceeded in transit (caller keeps going)
                -1 on ICMP port unreachable (caller is done)
                >= 0 return value is some other ICMP unreachable code

*/

int
recv_v4 (int seq, struct timeval *tv)
 {
        int hlen1, hlen2, icmplen, ret;
        socklen_t len;
        ssize_t n;
        struct ip *ip, *hip;
        struct icmp *icmp;
        struct udphdr *udp;

        gotalarm = 0;
        alarm(3);
        for ( ; ; )
         {
                if (gotalarm)
                        return (-3);    /* alarm expired */
                len = pr->salen;
                n = recvfrom(recvfd, recvbuf, sizeof(recvbuf), 0, pr->sarecv, &len);
                if (n < 0)
                 {
                        if (errno == EINTR)
                                continue;
                        else
                                err_sys("recvfrom error");
                 }

                ip = (struct ip *) recvbuf;  /* start of IP header */
                hlen1 = ip->ip_hl << 2;      /* length of IP header */
                icmp = (struct icmp *) (recvbuf + hlen1);  /* start of ICMP header */
                if ( (icmplen = n- hlen1) < 8)
                        continue;        /* not enough to look at ICMP header */
                if (icmp->icmp_type == ICMP_TIMXCEED && icmp->icmp_code ==
ICMP_TIMXCEED_INTRANS)
                  {
                        if (icmplen < 8 + sizeof(struct ip))
                                continue;  /* not enough data to look at inner IP */
                        hip = (struct ip *) (recvbuf + hlen1 + 8);
                        hlen2 = hip->ip_hl << 2;
                        if (icmplen < 8 + hlen2 + 4)
                                continue; /* not enough data to look at UDP ports */
                        udp = (struct udphdr *) (recvbuf + hlen1 + 8 + hlen2);
                        if (hip->ip_p == IPPROTO_UDP && udp->uh_sport == htons
(sport) && udp->uh_dport == htons(dport + seq)) {
                                ret = -2;  /* we hit an intermediate router */
                                break;
                           }

                  } else if (icmp->icmp_type == ICMP_UNREACH) {
                                if (icmplen < 8 + sizeof(struct ip))
                                        continue; /* not enough data to look an inner IP */
                                hip = (struct ip *) (recvbuf + hlen1 + 8);
                                hlen2 = hip->ip_hl << 2;
                                if (icmplen < 8 + hlen2 + 4)
                                        continue;   /* not enough data to look an UDP ports
*/

                                udp = (struct udphdr *) (recvbuf + hlen1 + 8 + hlen2);
```

```
                                        if (hip->ip_p == IPPROTO_UDP && udp->uh_sport ==
htons (sport) && udp->uh_dport == htons (dport + seq )) {
                                        if (icmp->icmp_code == ICMP_UNREACH_PORT)
                                                  ret = -1;  /* have reached destination */
                                        else
                                                  ret = icmp->icmp_code;  /* 0,1,2,3.........*/
                                        break;
                                   }
                              }
                              if (verbose)  {
                                                  printf(" (from %s: type = %d, code = %d)\n",
Sock_ntop_host(pr->sarecv, pr->salen), icmp->icmp_type, icmp->icmp_code);
                              }

                              /* some other ICMP error, recvfrom() again */
                    }
          alarm(0);
          Gettimeofday(tv, NULL);   /* get time of packet arrival */
          return (ret);
  }
```

/***************************RECV_V6.C***************************/

```
#include "trace.h"

 extern int gotalarm;

 /*
  * return:     -3 on timeout
                    -2 on ICMP time exceeded in transit (caller keeps going)
                    -1 on ICMP port unreachable (caller is done)
                    >= 0 return value is some other ICMP unreachable code

 */

 int
 recv_v6 (int seq, struct timeval *tv)
  {
  #ifdef IPV6
          int hlen2, icmp6len, ret;
          ssize_t n;
          sock_t len;
          struct ip6_hdr *hip6;
          struct icmp6_hdr *icmp6;
          struct udphdr *udp;
	gotalarm = 0;
	alarm(3);
	for ( ; ; )
	 {
		if (gotalarm)
			return (-3);   /* alarm expired */
		len = pr->salen;
		n = recvfrom(recvfd, recvbuf, sizeof(recvbuf), 0, pr->sarecv, &len);
		if (n < 0)
		 {
			if (errno == EINTR)
				continue;
			else
				err_sys("recvfrom error");
		 }
		icmp6 = (struct icmp6_hdr *) recvbuf;  /* ICMP header */
		if ( (icmp6len = n) < 8)
			continue;    /* not enough to look at ICMP header */
		if (icmp6->icmp6_type == ICMP_TIME_EXCEEDED && icmp6-
>icmp6_code == ICMP_TIME_EXCEED_TRANSIT) {
			if (icmp6len < 8 + sizeof(struct ip6_hdr) + 4)
				continue;  /* not enough data to look at inner header */

			hip6 = (struct ip6_hdr *) (recvbuf + 8);
			hlen2 = sizeof(struct ip6_hdr);
			udp = (struct udphdr *) (recvbuf + 8 + hlen2 );
			if (hip6->ip6_nxt == IPPROTO_UDP && udp->uh_sport ==
htons(sport) && udp->uh_dport == htons (dport + seq))
				ret = -2;
			break;
		   } else if (icmp6->icmp6_type == ICMP_DST_UNREACH)  {
			if (icmp6len < 8 + sizeof(struct ip6_hdr) + 4)
				continue;   /* not enough data to look at inner header */
			hip6 = (struct ip6_hdr *)  (recvbuf + 8);
			hlen2 = sizeof(struct ip6_hdr);
			udp = (struct udphdr *) (recvbuf + 8 + hlen2);
			if (hip->ip6_nxt == IPPROTO_UDP && udp->uh_sport ==
htons(sport) && udp->uh_dport == htons (dport + seq)) {
			if (icmp6->icmp6_code == ICMP6_DST_UNREACH_NOPORT)
				ret = -1;  /* have reached destination */
			else
				ret = icmp6->icmp6_code;   /* 0,1,2....... */
			break;
		     }
		} else if (verbose)  {
```

```
                                        printf("  (from %s: type = %d, code = %d)\n",
Sock_ntop_host(pr->sarecv, pr->salen), icmp6->icmp6_type, icmp6->icmp6_code);
                                        }

              /* some other ICMP error, recvfrom() again */
          }
          alarm(0);   /*dont leave alarm running */
          Gettimeofday(tv, NULL);   /* get time of packet arrival */
          return(ret);
  #endif
 }
```

/**************************SIG_ALRM.C*******************************/

```
  #include "trace.h"

          int gotalarm;

          void
          sig_alrm(int signo)
          {
                    gotalarm = 1;   /* set flag to note that alarm occured */
                    return;         /* and interrupt the recvfrom() */
          }
```