0 TCP/IP overview

From these assumptions comes the fundamental structure of the Internet: a packet switched communications facility in which a number of distinguishable networks are connected together using packet communications processors called gateways which implement a store and forward packet forwarding algorithm.

David D. Clark

0.1 The Internet

The Internet is a global information system consisting of millions of computer networks around the world. Users of the Internet can exchange email, access to the resources on a remote computer, browse web pages, stream live video or audio, and publish information for other users. With the evolution of *e-commerce*, many companies are providing services over the Internet, such as on-line banking, financial transactions, shopping, and online auctions. In parallel with the expansion in services provided, there has been an exponential increase in the size of the Internet. In addition, various types of electronic devices are being connected to the Internet, such as cell phones, personal digital assistants (PDA), and even TVs and refrigerators.

Today's Internet evolved from the ARPANET sponsored by the Advanced Research Projects Agency (ARPA) in the late 1960s with only four nodes. The Transmission Control Protocol/Internet Protocol (TCP/IP) protocol suite, first proposed by Cerf and Kahn in [1], was adopted for the ARPANET in 1983. In 1984, NSF funded a TCP/IP based backbone network, called NSFNET, which became the successor of the ARPANET. The Internet became completely commercial in 1995. The term "Internet" is now used to refer to the global computer network loosely connected together using packet switching technology and based on the TCP/IP protocol suite.

The Internet is administered by a number of groups. These groups control the TCP/IP protocols, develop and approve new standards, and assign Internet addresses and other resources. Some of the groups are listed here.

- Internet Society (ISOC). This is a professional membership organization of Internet experts that comments on policies and practices, and oversees a number of other boards and task forces dealing with network policy issues.
- Internet Architecture Board (IAB). The IAB is responsible for defining the overall architecture of the Internet, providing guidance and broad direction to the IETF (see below).
- Internet Engineering Task Force (IETF). The IETF is responsible for protocol engineering and development.
- Internet Research Task Force (IRTF). The IRTF is responsible for focused, long-term research.
- Internet Corporation for Assigned Names and Numbers (ICANN). The ICANN has responsibility for Internet Protocol (IP) address space allocation, protocol identifier assignment, generic and country code Top-Level Domain name system management, and root server system management functions. These services were originally performed by the Internet Assigned Numbers Authority (IANA) and other entities. ICANN now performs the IANA function.
- Internet Network Information Center (InterNIC). The InterNIC is operated by ICANN to provide information regarding Internet domain name registration services.

The Internet standards are published as *Request for Comments* (RFC), in order to emphasize the point that “the basic ground rules were that anyone could say anything and that nothing was official” [2]. All RFCs are available at the IETF’s website `http://www.ietf.org/`. Usually, a new technology is first proposed as an *Internet Draft*, which expires in six months. If the Internet Draft gains continuous interest and support from ISOC or the industry, it will be promoted to a RFC, then to a *Proposed Standard*, and then a *Draft Standard*. Finally, if the proposal passes all the tests, it will be published as an *Internet Standard* by IAB.

0.2 TCP/IP protocols

The task of information exchange between computers consists of various functions and has tremendous complexity. It is impractical, if not

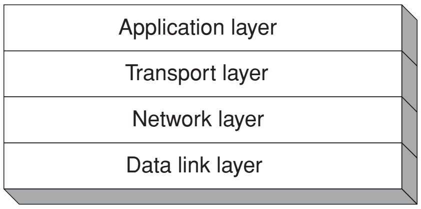


Figure 0.1. The TCP/IP protocol stack.

impossible, to implement all these functions in a single module. Instead, a *divide-and-conquer* approach was adopted. The communication task is broken up into subtasks and organized in a hierarchical way according to their dependencies to each other. More specifically, the subtasks, each of which is responsible for a facet of communication, are organized into different layers. Each higher layer uses the service provided by its lower layers, and provides service to the layers above it. The service is provided to the higher layer transparently, while heterogeneity and details are hidden from the higher layers. A protocol is used for communication between entities in different systems, which typically defines the operation of a subtask within a layer.

TCP/IP protocols, also known more formally as the *Internet Protocol Suite*, facilitates communications across interconnected, heterogeneous computer networks. It is a combination of different protocols, which are normally organized into four layers as shown in Fig. 0.1. The responsibility and relevant protocols at each layer are now given.

- The *application layer* consists of a wide variety of applications, among which are the following.
  - Hypertext Transfer Protocol (HTTP). Provides the World Wide Web (WWW) service.
  - Telnet. Used for remote access to a computer.
  - Domain Name System (DNS). Distributed service that translates between domain names and IP addresses.
  - Simple Network Management Protocol (SNMP). A protocol used for managing network devices, locally or remotely.
  - Dynamic Host Configuration Protocol (DHCP). A protocol automating the configuration of network interfaces.
- The *transport layer* provides data transport for the application layer, including the following.
  - Transmission Control Protocol (TCP). Provides *reliable* data transmission by means of *connection-oriented* data delivery over an IP network.

  - User Datagram Protocol (UDP). A *connectionless* protocol, which is simpler than TCP and does not guarantee reliability.
- The *network layer* handles routing of packets across the networks, including the following.
  - Internet Protocol (IP). The "workhorse" of the TCP/IP protocol stack, which provides *unreliable* and *connectionless* service.
  - Internet Control Message Protocol (ICMP). Used for error and control messages.
  - Internet Group Management Protocol (IGMP). Used for *multicast* membership management.
- The *link layer* handles all the hardware details to provide data transmission for the network layer. Network layer protocols can be supported by various link layer technologies, such as those listed here.
  - Ethernet. A popular *multiple access* local area network protocol.
  - Wireless LAN. A wireless multiple access local area network based the IEEE 802.11 standards.
  - Point to Point Protocol (PPP). A *point-to-point* protocol connecting pairs of hosts.
  - Address Resolution Protocol (ARP). Responsible for resolving network layer addresses.

Figure 0.2 shows the relationship among protocols in different layers. We will discuss these protocols in more detail in later chapters.

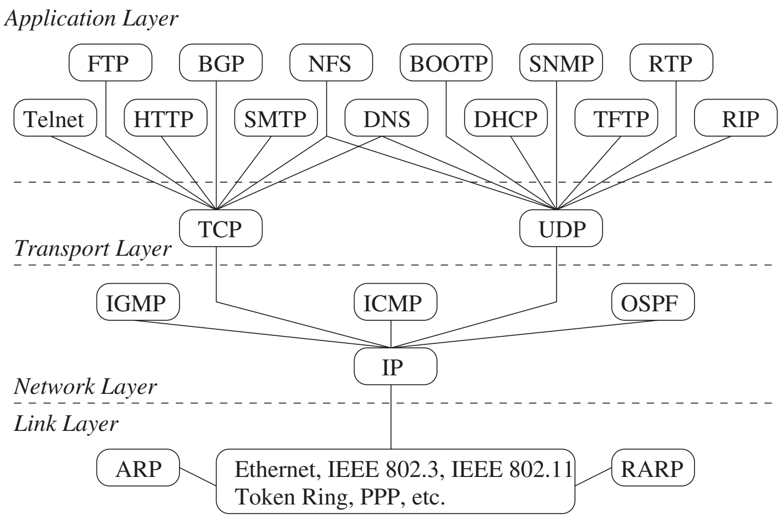


Figure 0.2. The TCP/IP protocols.

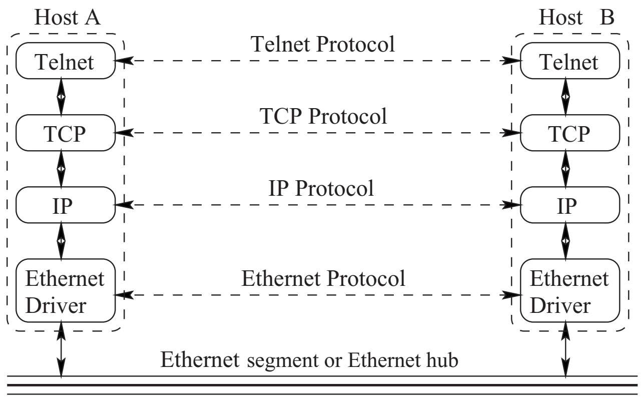


Figure 0.3. An illustration of the layers involved when two hosts communicate over the same Ethernet segment or over an Ethernet hub.

0.3 Internetworking devices

The Internet is a collection of computers connected by internetworking devices. According to their functionality and the layers at which they are operating, such devices can be classified as *hubs*, *bridges*, *switches*, and *routers*.

Hubs are physical layer devices, used to connect multiple hosts. A hub simply copies frames received from a port to all other ports, thus emulating a broadcast medium. Bridges, sometimes called *layer two switches*,[1] are link layer devices. They do not examine upper layer information, and can therefore forward traffic rapidly. Bridges can be used to connect distant stations and thus extend the effective size of a network. Bridges are further discussed in Chapter 3.

Routers, also called *layer three switches*, are network layer devices incorporating the routing function. Each router maintains a *routing table*, each entry of which contains a destination address and a next-hop address. None of the routers has information for the complete route to a destination. When a packet arrives, the router checks its routing table for an entry that matches the destination address, and then forwards the packet to the next-hop address. Routing is further discussed in Chapter 4.

Figure 0.3 shows the layers involved in communication between two hosts when they are connected by an Ethernet hub. The hosts can directly

[1] The industry, confusingly, also uses the term *smart hubs* for switches.

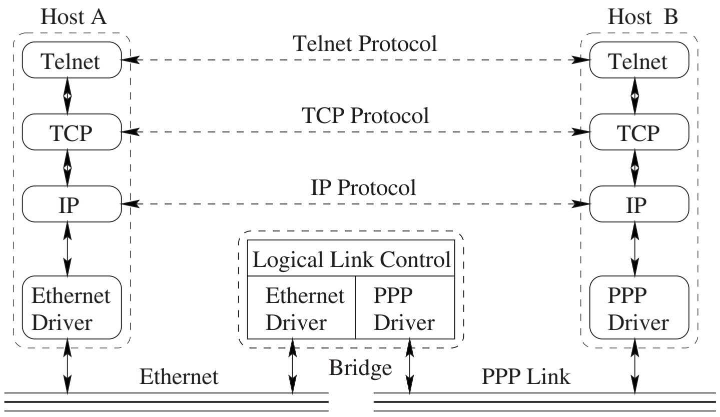


Figure 0.4. An illustration of the layers involved when two hosts communicate through a bridge.

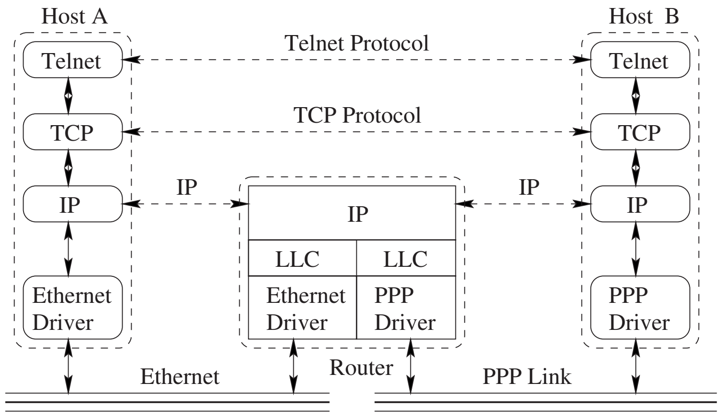


Figure 0.5. An illustration of the layers involved when two hosts communicate through a router.

communicate with each other since the same link layer protocol is used. Figure 0.4 shows how two different network segments using different link layer technologies are interconnected using a bridge, which interfaces between the link layer protocols and performs frame forwarding. Figure 0.5 shows how two networks are interconnected by a router, which not only performs the layer two functions as in Fig. 0.4, but also handles routing and packet forwarding, which are the major functions of the network layer.

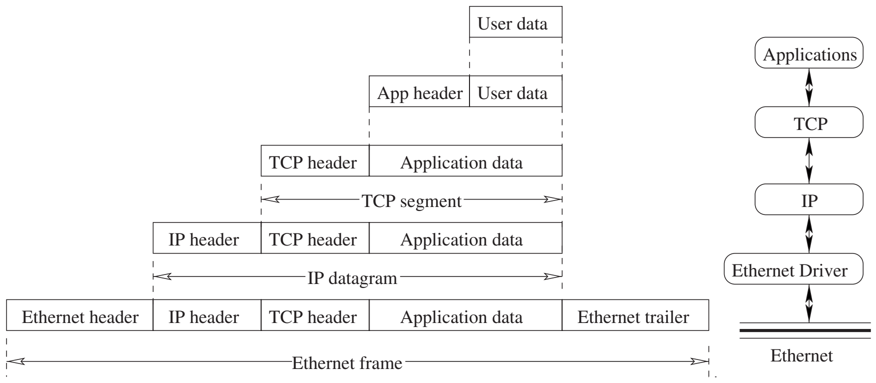


Figure 0.6. Encapsulation of user data through the layers.

As shown in the examples above, a single network segment is formed using hubs. A number of network segments are interconnected by bridges and switches to construct an extended local area network associated with typically a corporate or other institutional networks. Wide Area Networks (WAN) are constructed by connecting the routers of different enterprise networks using high-speed, point-to-point connections. These connections are usually set up over an SDH/SONET circuit-switched network.

0.4 Encapsulation and multiplexing

In a source host, the application data is sent down through the layers in the protocol stack, where each layer adds a header (and maybe a trailer) to the data received from its higher layer (called the *protocol data unit* (PDU)). The header contains information used for the control functions that are defined and implemented in this layer. This *encapsulation* process is shown in Fig. 0.6. When the packet arrives at the destination, it is sent up through the same protocol stack. At each layer, the corresponding header and/or trailer are stripped and processed. Then, the recovered higher layer data is delivered to the upper layer.

As explained in Section 0.2, one of the advantages of the layered structure is the great flexibility it provides for network design and management. For example, different higher layer protocols can use the service provided by the same lower layer protocol, and the same higher layer protocol can use the service provided by different lower layer protocols. In the first

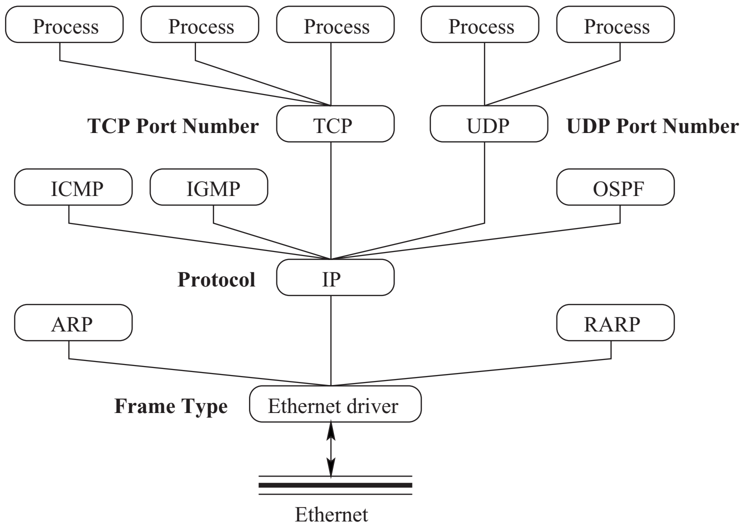


Figure 0.7. Multiplexing/demultiplexing in the layers.

case, each packet sent down to the lower layer should have an identifier indicating which higher layer module it belongs to. As is shown in Fig. 0.7, multiplexing and demultiplexing is performed at different layers using the information carried in the packet headers. For example, a communication process running in a host is assigned a unique *port number*, which is carried by all the packets generated by or destined to this process. Transport layer protocols such as TCP or UDP determine whether a packet is destined for this process by checking the port number field in the transport layer header. In the IP case, each protocol using IP is assigned a unique *protocol number*, which is carried in the `Protocol` IP header field in every packet generated by the protocol. By examining the value of this field of an incoming IP datagram, the type of payload can be determined. A field called *Frame Type* in the Ethernet header is used for multiplexing and demultiplexing at this level.

0.5 Naming and addressing

In order to enable the processes in different computers to communicate with each other, naming and addressing is used to uniquely identify them. As discussed in the previous section, a process running in a host can be

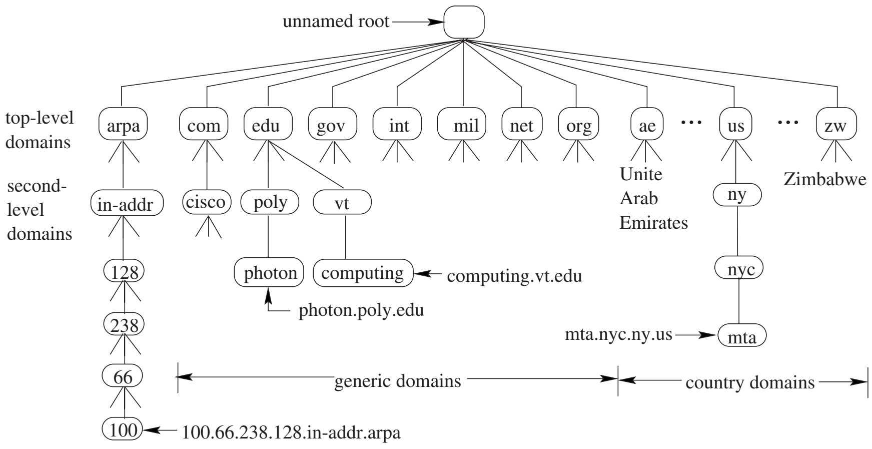


Figure 0.8. The organization of the domain name space.

identified by its port number. Furthermore, a host is identified by a *domain name*, while each network interface is assigned a unique IP address and a *physical*, or MAC, address.

0.5.1 Domain name

In the application layer, an alphanumeric *domain name* is used to identify a host. Since this layer directly interacts with users, a domain name is more user friendly than numeric addressing schemes, i.e., it is easier to remember and less prone to errors in typing.

Domain names are hierarchically organized, as shown in Fig. 0.8. In the tree structure, the root node has a null label, while each nonroot node has a label of up to 63 characters. As shown in Fig. 0.8, there are three types of domains. The `arpa` domain is mainly used for mapping an IP address to the corresponding domain name. The following seven domains are called *generic* domains with three-character labels, one for each of these special type of organization. The classification of the generic domains are given in Table 0.1. The remaining domains are two-character labeled *country* domains, one for each country, e.g., `ca` for Canada and `us` for the United States of America. The domain name of a node is the list of labels written as a text string, starting at the node and ending at the root node. Examples of domain names are `photon.poly.edu` and `mta.nyc.ny.us`, as shown in Fig. 0.8. In addition to the domain names shown in Fig. 0.8, seven new

Table 0.1. *Classification of the generic domains*

domain	Description
`com`	Commercial organizations
`edu`	Educational institutions
`gov`	other US government institutions
`int`	International organizations
`mil`	U.S. military groups
`net`	Major network support centers
`org`	Other organizations

top-level domains, .aero, .biz, .coop, .info, .museum, .name, and .pro, were added to the Internet's domain name system by ICANN in 2000.

Since the TCP/IP programs only recognize numbers, the domain name system (DNS) is used to resolve, i.e., translate, a domain name to the corresponding IP address. Then the resolved IP address, rather than the domain name, is used in the TCP/IP kernel. DNS is a client/server type of service. Since the entire database of domain names and IP addresses is too large for any single server, it is implemented as distributed databases maintained by a large number of DNS servers (usually host computers running the DNS server program). Thus each DNS server only maintains a portion of the domain name database shown in Fig. 0.8. A host can query the DNS servers for the IP address associated with a domain name, or for the domain name associated with an IP address. If the DNS server being queried does not have the target entry in its database, it may contact other DNS servers for assistance. Or, it may returns a list of other DNS servers that may contain the information. Thus the client can query these servers *iteratively*.

It is inefficient to perform name resolution for the same domain name every time its IP address is requested. Instead, DNS servers and clients use *name caching* to reduce the number of such queries. A DNS server or client maintains a cache for the names and corresponding IP addresses which have been recently resolved. If the requested domain name is in the cache, then there is no need to send a DNS query to resolve it. In addition, each cached entry is associated with a Time-to-Live timer. The value of this timer, which is usually set to the number of seconds in two days when the entry is first cached, is determined by the server that returns the DNS reply. The entry will be removed from the cache when the timer expires.

0.5.2 Port number

Port numbers are used as addresses for application layer user processes. The value of the `Port Number` field in the TCP or UDP header is used to decide which application process the data belongs to.

Most network applications are implemented in a client–server architecture, where a server provides a service to the network users, and a client requests the service from the server. The server is always running and uses a *well-known* port number. Well-known port numbers from 1 to 255 are used for Internet-wide services (e.g., **telnet** uses 23 and **ssh** uses port 22), while those from 256 to 1023 are preserved for Unix specific services (e.g., **rlogin** uses 513). On the other hand, a client runs for a period of time associated with the time needed to fullfil its request. It starts up, sends requests to the server, receives service from the server, and then terminates. Therefore clients use *ephemeral* port numbers which are randomly chosen and are larger than 1023.

0.5.3 IP address

Each host interface in the Internet has a unique IP address. A host with multiple interfaces and hence multiple IP addresses is called a *multi-homed* host. An IP address is a 32-bit number written in the *dotted-decimal* notation, i.e., as four decimal numbers, one for each byte, separated by three periods.

The global IP address space is divided into five classes, as shown in Table 0.2. Each IP address has two parts, a *network ID*, which is common for all the IP addresses in the same network, and a *host ID*, which is unique among all hosts in the same network. Figure 0.9 shows the IP address formats for the classes, where all class A IP addresses start with "0", all

Table 0.2. *Ranges of different classes of IP addresses*

Class	From	To
A	0.0.0.0	127.255.255.255
B	128.0.0.0	191.255.255.255
C	192.0.0.0	223.255.255.255
D	224.0.0.0	239.255.255.255
E	240.0.0.0	255.255.255.255

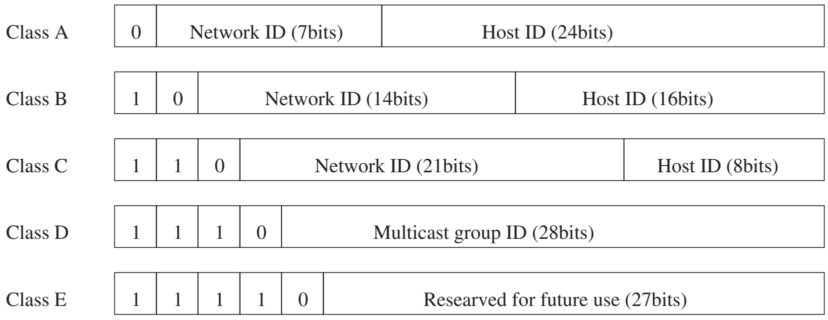


Figure 0.9. The format of IP addresses of different classes.

class B IP addresses start with “10”, so on and so forth. The class of an IP address can thus be easily determined by the first number of its dotted-decimal representation. An IP address consisting of all zero bits or all one bits for the host ID field is invalid for a host IP address.

As shown in Fig. 0.9, a class A (or class B) address uses 24 bits (or 16 bits) as the host ID. Institutions assigned with a class A or B network address usually do not have that many hosts in a single network, resulting in a waste of IP addresses and inconvenience in network administration and operation. In order to provide the flexibility in network administration and operation, the *subnetting* technique was introduced, where an IP address is further divided into three levels: a network ID, a *subnet ID*, and a host ID. With subnetting, IP addresses can be assigned using a finer granularity, e.g., a small organization can be assigned a subnet address that just satisfies its requirement. In addition, with subnetting, an organization can divide its assigned network space into a number of subnets, and assign a subnet to each department. The subnets can be interconnected by routers (see Section 0.3), resulting in better performance, stronger security, and easier management.

By using Table 0.2 and Fig. 0.9, it is possible to determine the network ID of an IP address. In order to determine the subnet ID and host ID, a *subnet mask* is used to indicate how many bits are used for the host ID. A subnet mask is a 32-bit word with “1” bits for the bit positions used by the network ID and subnet ID, and “0” bits for bit positions used by the host ID. By using a subnet mask, a class A, class B or even class C network address can be subnetted based on how many subnets and how many hosts per subnet are needed.

Figure 0.10 shows how, for the same class B IP address, two different subnet masks result in two different class B arrangements. In both examples, the network ID consists of the first 16 bits since it is a class B network

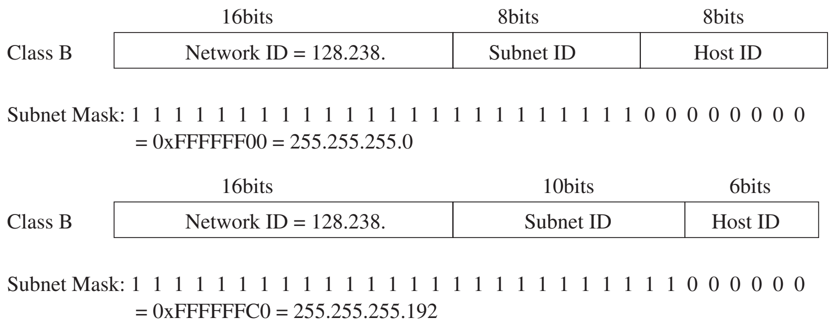

16bits
8bits
8bits
Class B
Network ID = 128.238.
Subnet ID
Host ID
Subnet Mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
= 0xFFFFFF00 = 255.255.255.0
16bits
10bits
6bits
Class B
Network ID = 128.238.
Subnet ID
Host ID
Subnet Mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
= 0xFFFFFFC0 = 255.255.255.192

Figure 0.10. An example of subnet masks for two different class B subnet design.

address. The first example uses a 24-bit subnet mask, resulting in a 8-bit subnet ID and a 8-bit host ID. Therefore, there could be $2^8 = 256$ subnets and $2^8 - 2 = 254$ hosts[2] in each subnet with this subnetting scheme. In the second example, a 26-bit subnet mask is used, resulting in a 10-bit subnet ID and a 6-bit host ID. Therefore, there could be $2^{10} = 1024$ subnets and $2^6 - 2$ hosts in each subnet with this subnetting scheme. Given a network address, the administrator can flexibly trade off the number of bits needed for the subnet ID and for the host ID, to find a subnetting arrangement best suited for the administrative and operative requirements.

The network ID is often referred to as the *network-prefix*. When subnetting is used, the combination of the network ID and subnet ID is called the *extended-network-prefix*. In addition to using the IP address and network mask pair, a *slash-notation* is often used by network engineers, where an IP address is followed by a "/" and the number of 1's in the subnet mask. For example, the class B address arrangements in Fig. 0.10 can be expressed as 128.238.66.101/24 and 128.238.66.101/26, respectively.

With the combination of an IP address and a port number, a process running in a host is uniquely identified in the global Internet, since the IP address is unique in the Internet and the port number is unique within the host. The combination of an IP address and a port number is called a *socket*.

0.5.4 IP version 6

Since it was born, the Internet has been growing exponentially. Every new host computer being connected needs a unique IP address. The recent trends of *pervasive computing* that connects laptop computers, personal digital

[2] Host IDs are not allowed to be all 1's or all 0's.

assistants (PDA), and cell phones to the Internet, and *home networking* that connects consumer electronic devices and home appliances to the Internet require yet more IP addresses.

However, when the current version of IP (IPv4) was designed, it was never imagined that the size of the Internet would be so huge. According to [3], the 32-bit IPv4 addresses will be depleted between 2005 and 2015. Some short-term solutions have been proposed to slow down the depletion of IPv4 addresses, including the following.

- Subnetting. As discussed in the previous subsection, this technique uses network prefixes with IP addresses. Thus IP addresses can be assigned in a finer granularity than "classful" addressing, which improves the efficiency of IPv4 addressing.
- Network Address Translator (NAT). With this technique, a section of IP addresses can be reused by different private networks.

A long-term solution to the above problem is to change the engine of the Internet, i.e., introduce a new, improved version of IP. The next version of IP, IPv6, uses 128-bit addresses, which is four times the size of an IPv4 address. Theoretically, there could be 3.4×10^{38} different IPv6 addresses. Thus, IPv6 provides plenty of IP addresses for all devices that need an IP address, eliminating the need to conserve address space.

In addition to an enlarged IP address space, the IPv6 design keeps the good features of IPv4, while eliminating minor flaws and obsolete functions. Some major enhancements are listed.

- A simpler header format. IPv6 uses a 40-byte fixed length header format. Some fields in the IPv4 header that are not frequently used are removed. Options are now supported by extension headers that follow the 40-byte IPv6 header, and are used only when needed.
- Automatic configuration mechanisms. IPv6 has mechanisms that greatly simplify the network configuration of host computers. An IPv6 host can be used in a "plug-and-play" mode, i.e., without manual configuration. Network management and administration are greatly simplified.
- Security. IPv6 has extensions for authentication and privacy, including encryption of packets and authentication of the sender of packets. IPsec (Chapter 9) is an IPv6 protocol suite requirement.
- Realtime service support. IPv6 provides the *flow labeling* mechanism for realtime services. With the flow label, intermediate routers can easily identify the flow to which a packet belongs, allowing for differentiated service of packets from different flows. For example, IP datagrams

corresponding to a delay-sensitive application like a voice conversation can be served on a priority basis.

0.5.5 Medium access control address

The medium access control (MAC) address, also called the *hardware address*, is used in the link layer to uniquely identify a network interface. MAC addresses contain no location information. Since the MAC address is burned in, network interfaces can be used in *plug-and-play* mode. An IP address, on the other hand, contains information on the location of the network interface and is used to route packets to or from the interface. An IP address usually needs to be configured manually, or by the Dynamic Host Configuration Procotol (DHCP), which will be discussed in Chapter 8.

Different link layer protocols use different MAC addresses. The Ethernet MAC address is 48 bits long and is globally unique. The first 24 bits of an Ethernet address is called the *vendor component*, while the remaining 24 bits is called the *group identifier*. An Ethernet interface card vendor is assigned with a block of Ethernet addresses, starting with a unique vendor component. Each card made by the vendor has a common vendor component, followed by a different group identifier. An example MAC address, using the *hexadecimal* notation, is: 0x8:0:20:87:dd:88.

The ARP protocol is used to translate an IP address to the corresponding MAC address. We will discuss ARP in Section 2.2.4 and Ethernet addresses further in Section 7.2.1.

0.6 Multiple access

The simplest way of interconnecting two computer hosts is using a point-to-point link with a host on each end. As the number of hosts increases, this approach may be inadequate, since there needs to be a large number of links (i.e., $N(N-1)/2$) to fully connect N hosts. In this case, a *broadcast* network, where all the hosts share a common transmission medium, is more efficient.

In order to share the common medium (e.g. a cable or a wireless channel) efficiently, all hosts must follow a set of rules to access the medium. For example, at any time, there may be only one host allowed to transmit data. Otherwise, the data from two or more transmitting users may collide with each other and be corrupted. Hosts should be able to check the availability of the medium and to resolve a collision. In addition, since the total bandwidth of the medium is limited, it is desirable to share it efficiently in terms of the aggregate throughput of all the hosts. Furthermore, each host should have a fair chance to access the medium and should not be allowed to take it forever.

The sharing-rules are defined as *medium access control* (MAC) protocols. Two examples are: Carrier Sense Multiple Access/Collision Detection (CSMA/CD, used in Ethernet), and Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA, used in wireless LANs). MAC protocols are implemented in the link layer. We will discuss CSMA/CD and CSMA/CA in Chapter 2.

0.7 Routing and forwarding

Various networks can be classified as *circuit-switched* networks and *packet-switched* networks. In a circuit switching network, an end-to-end circuit is set up by *circuit switches* along the path. A user communication session is guaranteed with a fixed amount of bandwidth, which is useful for many applications with *quality of service* (QoS) requirements. However, the bandwidth will be wasted if the users have no data to send, since the circuit is not shared by other users. On the other hand, the bandwidth of a network link is shared by all the users in a packet switching network. As the name suggests, user data is partitioned and stored in a sequence of packets and sent through the network. In such networks, *packet switches* route the packets, hop by hop, to the destination using information stored in the packet headers and information learned about the network topology.

Another dimension of classifying networks is defined by how the packets belonging to the same session are treated. In a *connectionless* network, every packet is self-contained, i.e., with sufficient routing information, and is treated independently, while in a *connection-oriented* network, an end-to-end connection is first set up and each packet belonging to the same session is treated consistently. Table 0.3 gives examples of how current networks fall in this classification scheme.

Routing and forwarding are the main functions of the network layer. The IP modules in the hosts and the internet routers are responsible for delivering packets from their sources to their destinations. Routing and

Table 0.3. *Classification of networks*

	Packet switching	Circuit switching
Connectionless	The Internet	–
Connection-oriented	Asynchronous Transfer Mode (ATM) networks	Plain Old Telephone Service (POTS)

forwarding consist of two closely related parts: maintaining network topology information and forwarding packets. Hosts and routers must learn the network topology in order to know where the destinations are, by exchanging information on connectivity and the quality of network links. The learned information is stored in a data structure called *routing tables* in hosts and routers. Routing tables are created or maintained either manually or by dynamic routing protocols. When there is a packet to deliver, a host or a router consults the routing table on where to route the packet. An end-to-end path consists of multiple routers. Each router relays a packet to the next-hop router which brings it closer to its destination. We will examine routing and forwarding in the Internet in Chapter 4.

0.8 Congestion control and flow control

Internet routers forward packets using the *store-and-forward* technique, i.e., an incoming packet is first stored in an input buffer, and then forwarded to the output port buffer, queued for transmission over the next link. Usually the buffer in a router is shared by many data flows belonging to different source-destination pairs. If, in a short period, a large number of packets arrive, the output port may be busy for a while and the buffer may be fully occupied by packets waiting for their turn to be forwarded (i.e., the router is *congested*). A similiar situation may occur at a destination host, which may be receiving packets from multiple sources. The packets received are first stored in a buffer, and then sent to the application processes. If the packet arriving rate is higher than the rate at which the packets are removed from the buffer, the receiving buffer may be fully occupied by packets waiting to be processed. In addition, hosts and routers are heterogeneous in terms of their processing capability and network bandwidth. In the case of a fast transmitter and a slow receiver, the receiver's buffer may get full. When the buffer,

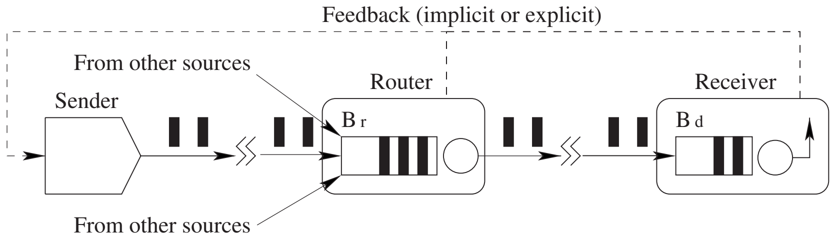


Figure 0.11. An illustration of flow control and congestion control in the Internet.

either at the receiver or at an intermediate router, is full, arriving packets have to be dropped since there is no space left to store them. Packet losses are undesirable since they degrade the quality of the communication session.

In the Internet, congestion control and flow control are used to cope with these problems. The basic idea is to let the source be adaptive to the buffer occupancies in the routers and the receiver (see Fig. 0.11, where the router has a finite buffer size B_r and the receiver has a finite buffer size B_d.). For example, the receiver may notify the sender how much data it can receive without a buffer overflow. Then the sender will not send more data than the amount allowed by the receiver. In the router case, the sender may be explicitly notified about the congestion in the router, or infer congestion from received feedback. Then the source will reduce its sending rate until the congestion is eased. TCP uses *slow start* and *congestion avoidance* to react to congestion in the routers, and to avoid receiver buffer overflow. We will discuss TCP congestion control and flow control in Chapter 6.

0.9 Error detection and control

When a packet is forwarded along its route, it may be corrupted by transmission errors. Many TCP/IP protocols use the *checksum* algorithm (or *parity check*) to detect bit errors in the header of a received packet. Suppose the `checksum` header field is K bits long (e.g., $K = 16$ in IP, UDP, and TCP). The value of the field is first set to 0. Then, the K-bit *one's complement sum* of the header is computed, by considering the header as a sequence of K-bit words. The K-bit one's complement of the sum is stored in the `checksum` field and sent to the receiver. The receiver, after receiving the packet, calculates the checksum over the header (including the `checksum` field) using the same algorithm. The result would be all ones if the header is error free. Otherwise, the header is corrupted and the received packet is discarded. IP, ICMP, IGMP, UDP and TCP use this algorithm to detect errors in the headers.

Ethernet, on the other hand, uses the *cyclic redundancy check* (CRC) technique to detect errors in the entire frame. With CRC, the entire frame is treated as a single number, and is divided by a predefined constant, called the CRC *generator*. The remainder of the division operation is appended to the frame (as the trailer) and sent to the receiver. After receiving the frame, the receiver performs the same division and compares the remainder with the received one. If the two are identical, there is no error in the frame. Otherwise, the frame is corrupted and should be discarded.

In addition to bit errors in a received packet, packets may be lost if there is congestion in the network, or if an incorrect route is used. *Sequence numbers* can be used to detect this type of error. With this technique, the sender and the receiver first negotiate an *initial sequence number*. Then the sender assigns a unique sequence number to each packet sent, starting from the initial sequence number and increased by one for each packet sent. The receiver can detect which packets are lost by ordering the received sequence numbers and looking for gaps in them.

When a packet loss is detected, the receiver may notify the sender, and request for a retransmission of the lost packet. In addition, the sender can use other error control schemes, such as forward error correction (FEC), in the application layer for better protection of the application data. We will examine TCP error control in Chapter 6.

0.10 Header formats of the protocols

The basic control functions discussed in the previous sections are implemented in different layers, while the information used by the control functions are carried in the packet headers. In this section, we examine the header formats of Ethernet, IP, UDP and TCP, which will be frequently used in discussions and data analysis in the following chapters.

0.10.1 Ethernet frame format

The laboratory experiments in this book are all based on Ethernet LANs. Fig. 0.12 shows the Ethernet frame format. The first 6 bytes give the `Destination Ethernet (MAC) Address`, while the next 6 bytes give the `Source Ethernet Address`. Next comes the 2-byte `Frame Type` field which is used to identify the payload of the Ethernet frame. For

Destination Address	Source Address	Frame Type	Data	CRC
6 bytes	6 bytes	2 bytes	46–1500 bytes	4 bytes

Figure 0.12. Ethernet frame format.

Version	Hdr Len	Differentiated Services	Total Length	
Identification			Flags	Fragment Offset
Time to Live		Protocol	Header Checksum	
Source IP Address				
Destination IP Address				
Options (if any, <= 40 bytes)				
Data				

Figure 0.13. IP header format.

example, this field is set to 0x0800 for IP datagrams, 0x0806 for ARP requests and replies, and 0x0835 for RARP requests and replies. The 4-byte trailer is the CRC bits used for error control.

0.10.2 IP header format

The format of the IP header is given in Fig. 0.13. If no option is present, the size of the IP header is 20 bytes. Some of the fields are introduced below, other fields will be explained in later chapters.

- `Version`: 4 bits. The version of IP used, which is four for IPv4.
- `Header Length`: 4 bits. The header length in 32-bit words.
- `Differentiated Services`: 8 bits. Specifies how the upper layer protocol wants the current datagram to be handled. Six bits of this field are used as a differential service code point (DSCP) and a two-bit currently unused (CU) field is reserved.
- `Total Length`: 16 bits. The IP datagram length in bytes, including the IP header.
- `Identification`: 16 bits. Contains an integer that identifies the current datagram.
- `Flags`: 3 bits. Consists of a 3-bit field of which the lower two bits control *fragmentation*. The highest order bit is not used.
- `Fragment Offset`: 13 bits. Indicates the position of the fragment's data relative to the beginning of the data in the original datagram. It allows the destination IP process to properly reconstruct the original datagram.

Source Port Number	Destination Port Number
Length	Checksum

Figure 0.14. UDP header format.

32-bit Source IP Address		
32-bit Destination IP Address		
0x00	8-bit Protocol (0x17)	16-bit UDP Length

Figure 0.15. The pseudo-header used in UDP checksum computation.

- `Time to Live`: 8 bits. A counter that is decremented by one each time the datagram is forwarded. A datagram with 0 in this field is discarded.
- `Protocol`: 8 bits. The upper layer protocol that is the source or destination of the data. The protocol field values for several higher layer protocols are: 1 for ICMP, 2 for IGMP, 6 for TCP, and 17 for UDP.
- `Header Checksum`: 16 bits. Calculated over the IP header to verify its correctness.
- `Source IP Address`: 32 bits. The IP address of the sending host.
- `Destination IP Address`: 32 bits. The IP address of the receiving host.

0.10.3 UDP header format

The UDP header format is shown in Fig. 0.14. The `Port Number` fields identify sending and receiving applications (processes). Given their 16-bit length, the maximum port number is $2^{16} - 1 = 65,535$. The 16-bit `Length`, measured in bytes, ranges from 8 bytes (i.e., data field can be empty) to $2^{16} - 1 = 65,535$ bytes. The 16-bit `Checksum` is computed using the UDP header, UDP data, and a pseudo-header consisting of several IP header fields, as shown in Fig. 0.15. Using the checksum is optional and this field can be set to 0x0000 if it is not used.

0.10.4 TCP header format

The TCP header format is shown in Fig. 0.16. The fields are explained below. A more detailed discussion of TCP can be found in Chapter 6.

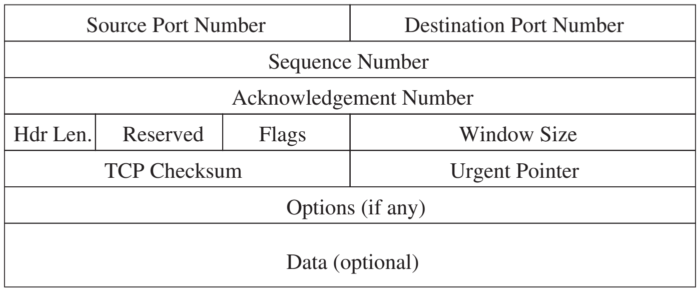


Figure 0.16. TCP header format.

- `Source Port Number`: 16 bits. The port number of the source process.
- `Destination Port Number`: 16 bits. The port number of the process running in the destination host.
- `Sequence Number`: 32 bits. Identifies the byte in the stream of data from the sending TCP to the receiving TCP. It is the sequence number of the first byte of data in this segment represents.
- `Acknowledgement Number`: 32 bits. Contains the next sequence number that the destination host wants to receive.
- `Header Length`: 4 bits. The length of the header in 32-bit words.
- `Reserved`: 6 bits. Reserved for future use.
- `Flags`: There are 6 bits for flags in the TCP header, each is used as follows.
  - `URG`: If the first bit is set, an urgent message is being carried.
  - `ACK`: If the second bit is set, the acknowledgement number is valid.
  - `PSH`: If the third bit is set, it is a notification from the sender to the receiver that the receiver should pass all the data received to the application as soon as possible.
  - `RST`: If the fourth bit is set, it signals a request to reset the TCP connection.
  - `SYN`: The fifth bit of the flag field of the packet is set when initiating a connection.
  - `FIN`: The sixth bit is set to terminate a connection.
- `Window Size`: 16 bits. The maximum number of bytes that a receiver can accept.
- `TCP Checksum`: 16 bits. Covers both the TCP header and TCP data.
- `Urgent Pointer`: 16 bits. If the URG flag is set, the pointer points to the last byte of the urgent message in the TCP payload. More specifically, the last byte of the urgent message is identified by adding the urgent pointer value to the sequence number in the TCP header.

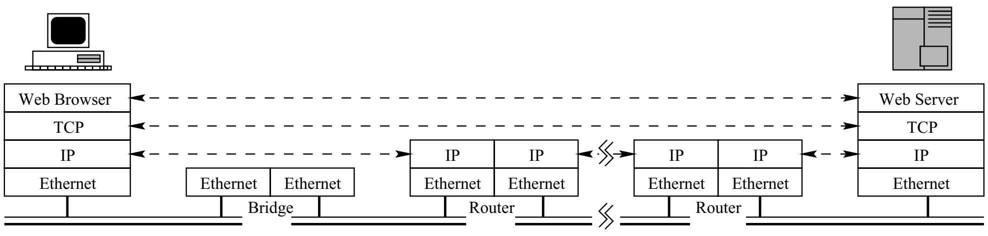


Figure 0.17. An example.

0.11 An example: how TCP/IP protocols work together

In this section, we show how a packet is forwarded from the source to the destination. As shown in Fig. 0.17, assume a user, named Bob, wants to book an air ticket from the website: `http://www.expedia.com`. Here is what happens in the system kernel and in the network.

First, Bob needs to know the domain name `www.expedia.com`, e.g., from a TV commercial or a web advertisement. If he happens to know the IP address corresponding to this domain name, he can use the IP address instead.

The remote computer with the domain name `www.expedia.com` is a *web server*, which is always running and provides the *web service*. Bob can use a web browser, which is a *web client*, to request and receive web service, i.e., to browse a web page. The HyperText Transfer Protocol (HTTP) is used by the web server and web browser. Most of the network services are provided using such a client–server architecture. We will discuss the client–server architecture in Chapter 5, and we will examine a web server in Chapter 8.

Bob starts a web browser, e.g., `Mozilla`, in his computer. Then he types `http://www.expedia.com/index.html` in the `Location` input area. The prefix `http` indicates the application layer protocol for this transaction, followed by the domain name of the web server, `www.expedia.com`, and the target file, `index.html`, in the server.

Next, the web browser needs to translate the domain name to an IP address, since domain names are not recognizable by the TCP/IP kernel. This is done via a query–response process using a protocol called the Domain Name System (DNS). The web browser invokes a function in the TCP/IP kernel called `gethostbyname()`, to send a DNS query which in essence asks "what is the IP address of '`www.expedia.com`'?" The query is sent to the host's DNS server, which is preconfigured in a file in the host, or is obtained dynamically using a protocol called Dynamic Host Configuration Protocol (DHCP) every time when the host bootstraps. A DNS server is a host maintaining a database of domain names and IP addresses. When the server receives a DNS query, it searches its database and sends a response to the querying host with the corresponding IP address. If the DNS server does not know the IP address of `www.expedia.com`, it may further query other DNS servers.

After receiving the DNS reply, the client tries to establish a *TCP connection* to the web server, since TCP is the transport layer protocol used by HTTP. The TCP/IP code is in the system kernel, but an application process can call the `socket` *application programming interface* (API) for TCP/IP services. Each application process invoking the socket API will be assigned a unique port number. The port number is carried in all the packets sent by and destined to this process. When the TCP connection is set up, the application data can be transmitted. The initial application data is a `HTTP request` message for the `index.html` file from the web server. It is sent down to the TCP layer and encapsulated in a *TCP segment*. The TCP header consists of the fields used for end-to-end flow control, congestion control, and error control, which are essential to providing an end-to-end stream-based reliable service. We will examine the use of port numbers and the concept of multiplexing in Chapter 1, study TCP in Chapter 6, and study socket API in Chapter 8.

Next, the TCP segment will be sent down to the IP layer and encapsulated in an IP *datagram*. The IP layer is responsible for forwarding the IP datagram to its destination. In order to deliver a packet to a remote host, each host or router maintains a routing table storing routing information. Only the next-hop IP address to a destination is stored. When a host has an IP datagram to sent, or when a router receives a datagram to forward, it searches the routing table to find the next-hop router, and forwards the datagram to that router. The routing table can be set manually, or dynamically by routing protocols. We will examine IP routing and configure a commercial router in Chapter 4.

In this example, the IP module of Bob's host finds the next-hop router in its routing table, and sends the IP datagram and the next-hop router's IP address down to the MAC layer. This host uses an Ethernet card and the IP datagram is further encapsulated within an Ethernet frame. The Ethernet driver is responsible for delivering the Ethernet frame to the interface of the next-hop router. Before sending the Ethernet frame out, the device driver has to resolve the next-hop IP address, since it only recognizes Ethernet MAC addresses. An ARP request is broadcast, querying the MAC address associated with the target IP address. When the router interface receives this ARP request, it responses with an ARP reply containing its MAC address. Then, the frame is sent on the medium after the ARP reply is received and the destination MAC address is learned. Note that whenever the host sends a frame, it uses the CSMA/CD multiple access algorithm to access the channel and may *backoff* if collision occurs. We will examine the operation and configuration of an Ethernet interface in Chapter 2.

Bob's local network consists of several LAN segments. Several IEEE 802.1d bridges, which are *self-configuring* and *transparent*, are used to connect the LAN segments. The *spanning tree* algorithm is running in the bridges to avoid loops in the local network. In this example, the Ethernet frame is first transmitted on the host's LAN segment, and then forwarded to the router interface by an intermediate bridge. We will examine bridges and the spanning tree protocol in Chapter 3.

Subsequently, the IP datagram is forwarded hop-by-hop by the intermediate routers along the route towards its destination. Some of the routers may be connected by point-to-point long-haul connections running the SDH/SONET protocol. Finally, the remote host's MAC module receives the Ethernet frame. The packet is delivered to the upper layers. At each layer, the corresponding header is stripped and examined. The information carried in the headers is used for such functions such as routing and forwarding, error control, flow control, and congestion control. In addition, the information is also used to identify which higher layer module the payload data belong to. When the Web server at the application layer receives the `HTTP request` message, it assembles an `HTTP response` message containing the requested file, and sends the response to the client. The response message is forwarded back to Bob's host, through a similar procedure. Finally, Bob can see the homepage of `www.expedia.com` in his web browser.

1 Linux and TCP/IP networking

The Linux philosophy is 'Laugh in the face of danger'. Oops. Wrong One. 'Do it yourself'. Yes, that's it. Linus Torvalds

1.1 Objectives

- Getting acquainted with the lab environment.
- Getting acquainted with the Linux operating system.
- Preview of some TCP/IP diagnostic tools.
- Capturing and analyzing the link layer, IP, and TCP headers.
- Understanding the concept of encapsulation.
- Understanding the concept of multiplexing using port numbers, the IP *protocol* field, and the Ethernet *frame type* field.
- Understanding the client–server architecture.

1.2 Linux and TCP/IP Implementations

1.2.1 TCP/IP Implementations

The TCP/IP protocol architecture was first proposed in the Cerf and Kahn paper [1]. Since then, the TCP/IP protocol family has evolved over time into a number of different versions and implementations. The first widely available release of TCP/IP implementation is the 4.2 Berkeley Software Distribution (BSD) from the Computer Systems Research Group at the University of California at Berkeley. Many implementations of TCP/IP protocols are based on the public domain BSD source code, both for Unix and non-Unix systems, as well as public domain implementations and implementations from various vendors.

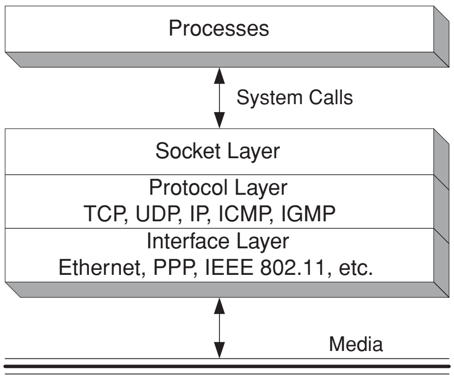


Figure 1.1. Organization of the networking code.

Solaris and FreeBSD are two examples of Unix TCP/IP implementations. Solaris is an operating system developed by Sun Microsystems. It supports both the SPARC platform and the x86 platform. FreeBSD is a Unix operating system derived from BSD. It was developed and is maintained by a large team of individuals. FreeBSD also supports multiple platforms and is available free of charge. Linux is a popular Unix-type operating system. It was originally created by Linus Torvalds and further improved by developers all over the world. Linux is developed under the *GNU General Public License*. The Linux source code is available in the public domain and the system kernel is recompilable. Linux can also be embedded in small devices, such as cellphones and PDAs. These features make Linux very popular in the computer and networking research communities. In addition, Linux is gaining support from major computer vendors, such as IBM, Oracle, and Dell.

From an implementation point of view, the networking code can be organized into four layers, as illustrated in Fig. 1.1. Most applications are implemented as *user space* processes, while protocols in the lower three layers (i.e., the transport layer, network layer, and data link layer) are implemented in the *system kernel*.[1] A user space process can obtain services provided by the kernel by invoking *system calls*. In the system kernel, the networking code is organized into three layers, namely the *socket layer*, the *protocol layer*, and the *interface layer*. The socket layer

[1] The core of an operating system, implementing critical system functions, e.g., managing memory and file systems, loading and executing other programs, and scheduling processes.

Table 1.1. *A few lines in the `/etc/services` file*

```
. . .
ftp stream tcp nowait root /usr/sbin/tcpd in.ftpd ftpd
telnet stream tcp nowait root /usr/sbin/tcpd in.telnetd
. . .
#finger stream tcp nowait root /usr/sbin/tcpd in.fingerd
. . .
```

is protocol independent. It provides a common interface to the user processes and hides the protocol specific details from them. The protocol layer contains the implementation of TCP/IP protocols, while the interface layer consists of *device drivers* which communicate with the network devices [4].

1.2.2 Network daemons and services

A daemon is a process running in the background of the system. Many TCP/IP services (e.g., Telnet) are handled by a daemon called **inetd**. Rather than running several network-related daemons, the **inetd** daemon works as a dispatcher and starts the necessary server processes when requests arrive. When a client wants a particular service from a remote server, the client contacts the **inetd** daemon through the server's well-known port number, which prompts **inetd** to start the corresponding server process.

The network daemons managed by **inetd** are specified in a configuration file called `/etc/inetd.conf`. Each service has a line in the file defining the network daemon that provides the service and its configuration parameters. Table 1.1 shows three lines in the `/etc/inetd.conf` file, which correspond to Ftp, Telnet, and Finger[2] services. One can comment a line, i.e., insert a # at the beginning of the line, to disable the corresponding service. For example, the Finger service in the following example is disabled. Note that there are some stand-alone network daemons that are not managed by **inetd**. For example, web service is provided by the **httpd** daemon, and DNS service is provided by the **named** daemon.

[2] Used to display information about a user.

In Red Hat Linux 9, **xinetd** replaces **inetd**, adding stronger security and more functionality. **xinetd** uses a simple common configuration file `/etc/xinetd.conf`. In addition, each service managed by **xinetd** uses an individual configuration file in the `/etc/xinetd.d` directory. The following is the Echo service configuration file `/etc/xinetd.d/echo`. It can be seen that the Echo service is enabled and uses TCP in the transport layer.

```
# default: off
# description: An echo server. This is the tcp \
# version.
service echo
{
        disable = no
        type = INTERNAL
        id = echo-stream
        socket_type = stream
        protocol = tcp
        user = root
        wait = no
}
```

Well-known port numbers are defined in the `/etc/services` file. A server can handle multiple clients for a service at the same time through the same well-known port number, while a client uses an ephemeral port number. The uniqueness of a communication session between two hosts is preserved by means of the port number and IP address pairs of the server and client hosts.

1.2.3 Network configurations files

When a host is configured to boot locally, certain TCP/IP configuration parameters are stored in appropriate local disk files. When the system boots up, these parameters are read from the files and used to configure the daemons and the network interfaces. A parameter may be changed by editing the corresponding configuration file.

In addition to `/etc/services` and `/etc/inetd.conf` discussed above, we now list other network configuration files.

`/etc/hosts`	Stores the host name of this machine and other machines.
`/etc/sysconfig/network`	Stores the host name and the default gateway IP address.
`/etc/sysconfig/network-scripts/ifcfg-eth0`	Stores the IP address of the first Ethernet interface.
`/etc/default-route`	Stores a default gateway, i.e., the IP address or the domain name of the default router.
`/etc/resolv.conf`	Stores the IP addresses of the DNS servers.
`/etc/nsswitch.conf`	Configures the means by which host names are resolved.

Solaris uses the following network configuration files stored in the `/etc` directory.

`nodename`	Host name of the machine.
`hostname.interface`	Interface IP address or the interface name.
`inet/hosts`	Stores IP addresses of the interfaces of the machine, the corresponding host name for each interface, IP addresses of the file server, and IP address and name of the default router.
`defaultdomain`	The host's fully qualified domain name.
`defaultrouter`	The name for the network interface that functions as this host's default router.
`inet/netmasks`	The network ID and the netmask if the network is subnetted.
`inet/networks`	Associates network names with network numbers, enabling applications to use and display names rather than numbers.
`nsswitch.conf`	Specifies name service to use for a particular machine.

1.3 Linux commands and tools

1.3.1 Basic Linux commands

The basic Linux commands are summarized below. See the manual pages for a list of options for each command.

- **man** *command_name*: Gets online help for *command_name*.
- **passwd**: Sets (changes) the password.
- **pwd** : Displays the current working directory.
- **ls** : Lists the contents of a directory.
- **more** *file_name*: Scrolls through a file.
  - To list the next page, press the space bar.
  - To go backwards, press `b`.
  - To quit from **more**, press `q`.
- **mv** *old_file_name new_file_name*: Renames a file.
  mv *file_name directory_name*: Moves a file to a directory.
  mv *old_directory_name new_directory_name*: Renames a directory.
- **rm** *file_name*: Deletes(removes) a file.
- **mkdir** *directory_name*: Creates a directory.
- **rmdir** *directory_name*: Removes a directory.
- **cd** *directory_name*: Changes the current working directory to *directory_name*. If *directory_name* is omitted, the shell is moved to your home directory.
- **cp** *file_name new_file_name*: Copies a file.
  cp *file_name directory_name*: Copies a file into *directory_name*.
- **chmod** *who op-code permission file_or_directory_name*: Changes the file access permissions.
  who: **u** user, **g** group, **o** other users, **a** all;
  op-code: $+$ add permission, $-$ remove permission;
  permission: **r** read, **w** write, **x** execute.
- **ps**: Process status report.
- **kill** *PID*: Terminates the process with a process ID *PID*.
- `Ctrl`**-c** : Terminates a command before it is finished.
- **cmp** *file1 file2*: Compares *file1* and *file2* byte by byte.
- **grep** *keyword file(s)*: Search the file(s) and outputs the lines containing the keyword.

Most of the above commands accept input from the system's *standard input* device (e.g., the keyboard) and send an output to the system's *standard output* device (e.g., the screen). Sometimes it is convenient to direct the output to another process as input for further processing, or to a file for storage. The *redirect* operator ">" directs the output to a file, as:

command > *file_name*.

With the *pipe* operator "|", two commands can be concatenated as:

command1 | **command2**,

where the output of **command1** is redirected as the input of **command2**.

1.3.2 Text editor

The vi Editor

The **vi** editor is one of the most popular text editors. It is the default text editor of most Linux and Unix systems.

To start **vi**, enter **vi** *file_name* at the command line. If no such file exists yet, it will be created. **vi** can be in one of the two modes, the *command mode* and the *text entry mode*. The command mode allows a user to use a number of commands to modify text. Text is inserted and modified in the text entry mode. Initially, **vi** enters the command mode and awaits instructions. To enter text, switch to the text entry mode by typing one of the following keys.

i:	Text is inserted to the left of the cursor.
a:	Text is appended after the cursor.
o:	Text is added after the current line.
O:	Text is added before the current line.

To switch back to the command mode, press the `Esc` key.

In the command mode, the user may use **vi**'s several editing features, such as cursor movement, text deletion, text replacement, and search operation. Some of the basic features are listed here.

h:	Moves the cursor one space to the left.
j:	Moves the cursor down one line.
k:	Moves the cursor up one line.
l:	Moves the cursor one space to the right.
`Ctrl`-**f**:	Scrolls down one full screen.
`Ctrl`-**b**:	Scrolls up one full screen.

`Ctrl`-**d**:	Scrolls down a half page.
`Ctrl`-**u**:	Scrolls up a half page.

To delete text, place the cursor over the target position, and type the following commands.

x:	Delete the next single character.
dw:	Delete the current word.
dd:	Delete the current line.

To search for a special string, e.g. *foo*, in the text file, type */foo* in the command mode. The cursor will jump to the nearest matching position in the file. To repeat the last search, type **n** in the command mode.

To save the file and quit **vi**, press `Esc` (even if in the command mode, it doesn't hurt), and type `:wq`. To quit **vi** without saving changes to the file, use the command `:q!`

Other text editors

In addition to **vi**, Red Hat Linux provides a number of graphical text editors that are intuitive and easy to use. They are especially convenient for users who are used to a windows-based interface. Examples of such graphical text editors are given below.

Emacs	A free text editor preinstalled in most Linux operating systems. It can be invoked by running **emacs**, or by choosing the system menu item: `Programming/Emacs`. The system menu pops up when the Red Hat icon at the lower-left corner of the workspace is clicked.
gedit	An analog to `Notepad` in Microsoft Windows. It can be invoked by running **gedit**, or by choosing the sytem menu item: `Accessories/Text Editor`.
OpenOffice.org	An analog to the `Microsoft Office` package. The `Writer` component of `OpenOffice.org` provides functions similar to that offered by `Microsoft Word`. A file created by `OpenOffice.org Writer` can be ported to the `Microsoft Word` format.

There is a graphical text editor available with the Solaris OpenWindows or Common Desktop Environment (CDE). It can be invoked by running **/usr/openwin/bin/textedit**, or from the system program menu. The system program menu can be found by clicking the right mouse-key on the background of the workspace. A new texteditor is started when the `Text Editor` menu item is chosen from the menu.

1.3.3 Window dump

The windows on the screen can be dumped into a graphic file. To dump the entire desktop area, you can simply type the `PrintScreen` key.[3] To dump a specific window (e.g., a command console), first click in the window and then type `Alt-PrintScreen`. The window is then dumped and the user is prompted for a file name to store it. The window dump is saved in the Portable Network Graphics (PNG) format. It can be opened, edited, and converted to other formats (e.g., PostScript, JPEG, or GIF) using the GIMP graphic editor supplied with Red Hat Linux.

In Solaris, a user may use **xwd** and an additional tool called **xpr** to dump a window. The pipe operator is used to conveniently redirect the output of **xwd** as the input of **xpr**, as:

xwd | xpr -device ps -output *file_name*

After the shape of the mouse pointer changes, click the mouse in the target window. The keyboard bell rings once at the beginning of the dump and twice when the dump is completed. The dumped file may be examined with **Image tools** found in the **Programs** menu.

1.3.4 Using floppy disks

To format an MS-DOS formatted disk, insert the disk into the floppy disk drive and execute: **fdformat /dev/fd0**. To use an MS-DOS formatted floppy disk, insert the disk and use the **mount /dev/fd0**[4] command to mount the floppy disk to the `/mnt/floppy` directory. A new icon called "floppy" will

[3] Different keyboards may have different names for this key. For example, `Prnt Scrn` or `PrtSc`.

[4] `/dev/fd0` is the device name of the floppy driver. The device names and their corresponding mount points are defined in the `/etc/fstab` file. For example, the CDROM has a device name `/dev/cdrom` and is mounted to the `/mnt/cdrom` directory.

appear on the desktop after the floppy is mounted. A double-click on this icon will start a file manager which opens the floppy disk.

To manipulate the files in the floppy disk, use the same Linux commands as usual. For example, to copy a file into the floppy disk, use

cp *file_name* `/mnt/floppy`.

To delete a file from the floppy disk, use

rm `/mnt/floppy/`*file_name*.

The floppy disk can be unmounted by the **umount /dev/fd0** command. Then the "floppy" icon disappears. The floppy disk can be ejected manually. Note that all the opened files in the floppy disk must be closed. Also, the current directory of any of the command consoles should not be `/mnt/floppy`. Otherwise, the **umount** command will fail with a "device is busy" warning.

In Solaris, the CDROM and floppy drives are controlled by the volume manager, which is a daemon process named **vold**. When a floppy disk is inserted in the drive, **vold** does not automatically recognize it (However, it recognizes a CD automatically). The user should type the **volcheck** command, which mounts the floppy disk under the `/floppy/floppy0` directory. To eject the floppy disk, use the **eject** command. To format a MD-DOS disk, use **fdformat -v -U -d**.

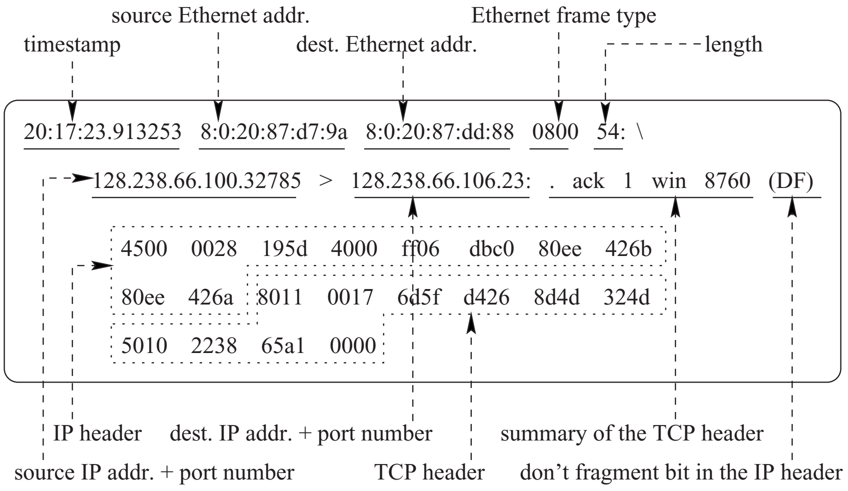
1.4 Diagnostic tools

Diagnostic tools are used to identify problems in the network, as well as to help understand the behavior of network protocols. We will use the following tools extensively in the experiments.

1.4.1 Tcpdump

Tcpdump is a network traffic sniffer built on the packet capture library `libpcap`.[5] While started, it captures and displays packets on the LAN segment. By analyzing the traffic flows and the packet header fields, a

[5] A public domain packet capture library written by the Network Research Group (NRG) of the Information and Computing Sciences Division (ICSD) of the Lawrence Berkeley National Laboratory (LBNL) in Berkeley, California.



source Ethernet addr.
Ethernet frame type
timestamp
dest. Ethernet addr.
length
20:17:23.913253 8:0:20:87:d7:9a 8:0:20:87:dd:88 0800 54: \
128.238.66.100.32785 > 128.238.66.106.23: . ack 1 win 8760 (DF)
4500 0028 195d 4000 ff06 dbc0 80ee 426b
80ee 426a 8011 0017 6d5f d426 8d4d 324d
5010 2238 65a1 0000
IP header
dest. IP addr. + port number
summary of the TCP header
source IP addr. + port number
TCP header
don't fragment bit in the IP header

Figure 1.2. A typical tcpdump output.

great deal of information can be gained about the behavior of the protocols and their operation within the network. Problems in the network can also be identified. A packet filter can be defined in the command line with different options to obtain a desired output.

A typical output of **tcpdump** running on a 128.238.66.0 subnet is shown in Fig. 1.2. The first line of the output gives a summary of the link/IP/TCP headers, while the following data block contains the raw bits of the IP datagram.

1.4.2 Ethereal

Ethereal is a network protocol analyzer built on the packet capture library `pcap`. In addition to capturing network packets as in Tcpdump, Ethereal provides a user friendly graphical interface, and supports additional application layer protocols. Ethereal can also import pre-captured data files from other network monitoring tools, such as Tcpdump and Sniffer.

In the following experiments, we use Ethereal to analyze a packet trace captured by Tcpdump, since generally Ethereal does not allow a normal user to capture packets (see Section A.5).

1.5 Exercises with Linux commands

We start with a simple single segment network, where all eight computers are connected in one Ethernet segment (see Fig. 1.3). The host IP

Table 1.2. *The IP addresses of the hosts in Fig. 1.3*

Host	IP Address	Subnet Mask
`shakti`	128.238.66.100	255.255.255.0
`vayu`	128.238.66.101	255.255.255.0
`agni`	128.238.66.102	255.255.255.0
`apah`	128.238.66.103	255.255.255.0
`yachi`	128.238.66.104	255.255.255.0
`fenchi`	128.238.66.105	255.255.255.0
`kenchi`	128.238.66.106	255.255.255.0
`guchi`	128.238.66.107	255.255.255.0

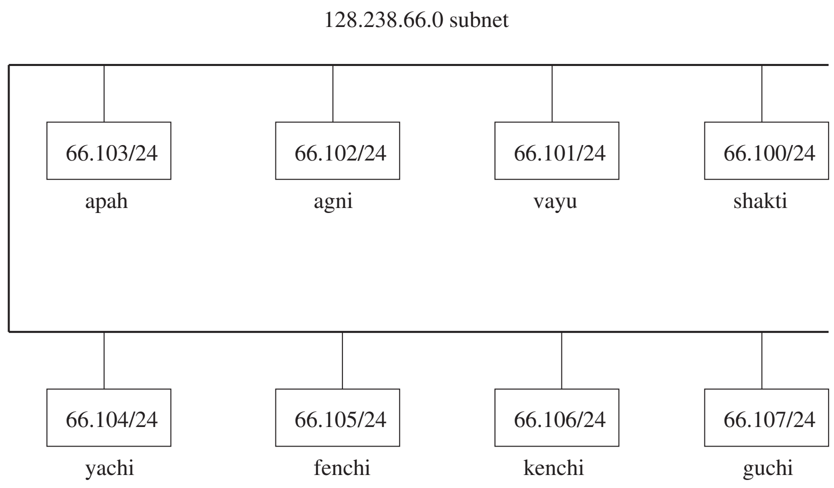


Figure 1.3. A single segment network.

addresses are given in Table 1.2. Note that the *slash-notation* is used, where "128.238.66.100/24" means an IP address of "128.238.66.100" with a subnetmask of "255.255.255.0".

Exercise 1 Login to the system. The login ID is **guest**, and the login password is **guest1**. Get acquainted with the `Gnome` environment, the Linux commands, text editors, and the man pages.

Exercise 2 After logging in, open a command window if one is not opened automatically, by clicking the right mouse-key on the background and choosing the `New Terminal` item in the menu.

In Solaris OpenWindows environment, click the right mouse key to invoke the workspace menu. Then choose `Programs/Command Tool ...` to bring up a new command window. In Solaris CDE, click the right mouse key to get the Workspace Menu. Then choose `Tools/Terminal` to bring up a new command window.

Show your login ID by typing **whoami** in the console.

Create a directory of your own, using **mkdir** *name_of_your_directory*. Change to your directory, using **cd** *name_of_your_directory*. You can save your data files for all your laboratory experiments here.

Open another command window. Run **pwd** in this and the previously opened consoles. Save the outputs in both consoles.

LAB REPORT What is the default directory when you open a new command window? What is your working directory?

Exercise 3 Run **ps -e** to list the processes running in your host. After starting a new process by running **telnet** in another command window, execute **ps -e** again in a third window to see if there is any change in its output.

Find the process id of the **telnet** process you started, by:

ps -e | **grep telnet**.

Then use **kill** *process_id_of_telnet* to terminate the **telnet** process.

LAB REPORT Is the Internet service daemon, **xinetd**, started in your system? Is **inetd** started in your system? Why?

Exercise 4 Display the file `/etc/services` on your screen, using:

more /etc/services.

Then in another console, use the redirect operator to redirect the **more** output to a file using **more /etc/services** > **ser_more**. Compare the file `ser_more` with the original **more** output in the other command window.

Copy `/etc/services` file to a local file named `ser_cp` in your working directory, using **cp /etc/services ser_cp**. Compare files `ser_more` and `ser_cp`, using **cmp ser_more ser_cp**. Are these two files identical?

Concatenate these two files using **cat ser_more ser_cp** > **ser_cat**.

Display the file sizes using **ls -l ser***. Save the output. What are the sizes of files `ser_more, ser_cp`, and `ser_cat`?

LAB REPORT Submit the **ls** output you saved in this exercise and answer the above questions.

1.6 Exercises with diagnostic tools

Exercise 5 Read the **man** pages for the following programs:

arp	**arping**	**ifconfig**	**tcpdump**
ping	**netstat**	**route**	**ethereal**

The **arping** command is not provided in Solaris 8.0.

Study the different options associated with each command. Throughout this lab you will use these commands rather extensively.

LAB REPORT Explain the above commands briefly. Two or three sentences per command would be adequate.

Exercise 6 In this exercise, we will use **tcpdump** to capture a packet containing the link, IP, and TCP headers and use **ethereal** to analyze this packet.

First, run **tcpdump -enx -w exe6.out**. You will not see any **tcpdump** output, since the **-w** option is used to write the output to the `exe6.out` file.

Then, you may want to run **telnet** *remote_host*[6] to generate some TCP traffic. After you login the remote machine, terminate the **telnet** session and terminate the **tcpdump** program.

Next, you will use **ethereal** to open the packet trace captured by **tcpdump** and analyze the captured packets. To do this, run **ethereal -r exe6.out &**. The **ethereal** Graphical User Interface (GUI) will pop up and the packets captured by **tcpdump** will be displayed.

For your report, you need to save any one of the packets that contain the link, IP, and TCP headers. Carry out the following instructions.

1. Click on a TCP packet from the list of captured packets in the **ethereal** window. Then go to the `Edit` menu and choose `Mark Frame`.
2. Go to the `File` menu and choose `Print`. In the `Ethereal:Print` dialog that pops up, check `File`, `Plain Text`, `Expand all levels`, `Print detail`,

[6] We use *remote_host* to denote the IP address of a remote host, i.e., a machine other than the one you are using.

and `Suppress unmarked frames`. Then, enter the output text file name, e.g., headers.txt, and click the `OK` button. The marked packet is now dumped into the text file, with a detailed list of the name and value of every field in all the three headers.

LAB REPORT Draw the format of the packet you saved, including the link, IP, and TCP headers (See Figs 0.12, 0.13, and 0.16 in Chapter 0 of this guide), and identify the value of each field in these headers. Express the values in the decimal format.

LAB REPORT What is the value of the `protocol` field in the IP header of the packet you saved? What is the use of the `protocol` field?

Exercise 7 In a manner similar to the previous exercise, we will run **tcpdump** to capture an ARP request and an ARP reply,[7] and then use **ethereal** to analyze the frames.

Run **tcpdump -enx -w exe7.out** to capture all the packets on the LAN segment.

If there is no arp requests and replies in the network, generate some using **arping** *remote_machine*.

When Solaris 8.0 is used, you can generate an ARP request and an ARP reply by running **telnet** to a remote machine. Note this remote machine should be a different machine from the one you used in Exercise 6.

After you see several ARP replies in the **arping** output, terminate the **arping** and the **tcpdump** program. Open the **tcpdump** trace using **ethereal -r exe7.out &**. Print one ARP request and one ARP reply using **ethereal**.

LAB REPORT What is the value of the `frame type` field in an Ethernet frame carrying an ARP request and in an Ethernet frame carrying an ARP reply, respectively?

What is the value of the `frame type` field in an Ethernet frame carrying an IP datagram captured in the previous exercise?

What is the use of the `frame type` field?

Exercise 8 Using the **tcpdump** utility, capture any packet on the LAN and see the output format for different command-line options. Study the various expressions for selecting which packets to be dumped.

[7] We will examine the Address Resolution Protocol (ARP) in the next Chapter. For this exercise, the purpose is to examine the use of the `frame type` field in an Ethernet frame.

For this experiment, use the **man** page for **tcpdump** to find out the options and expressions that can be used.

If there is no traffic on the network, you may generate traffic with some applications (e.g. **telnet**, **ping**, etc.).

LAB REPORT Explain briefly the purposes of the following **tcpdump** expressions.

tcpdump udp port 520
tcpdump -x -s 120 ip proto 89
tcpdump -x -s 70 host *ip_addr1* **and (***ip_addr2* **or** *ip_addr3***)**
tcpdump -x -s 70 host *ip_addr1* **and not** *ip_addr2*

1.7 Exercises on port numbers

Exercise 9 Start **tcpdump** in a command window to capture packets between your machine and a remote host using:

tcpdump -n -nn host *your_host* **and** *remote_host*[8].

Execute a TCP utility, **telnet** for example, in another command window.

When you see a TCP packet in the **tcpdump** output, terminate **tcpdump** and save its output.

LAB REPORT What are the port numbers used by the remote and the local computer? Which machine's port number matches the port number listed for **telnet** in the `/etc/services` file?

Exercise 10 Start **tcpdump** in one command window using:

tcpdump -n -nn host *your_host* **and** *remote_host*.

Then, **telnet** to the remote host from a second command window by typing **telnet** *remote_host*. Again issue the same **telnet** *remote_host* command from a third command window. Now you are opening two **telnet** sessions to the same remote host simultaneously, from two different command windows.

Check the port numbers being used on both sides of the two connections from the output in the **tcpdump** window. Save a TCP packet from each of the connections.

LAB REPORT When you have two **telnet** sessions with your machine, what port number is used on the remote machine?

Are both sessions connected to the same port number on the remote machine?

[8] For some older versions of **tcpdump**, the `-n -nn` options is combined into one single `-n` option.

What port numbers are used in your machine for the first and second **telnet**, respectively?

LAB REPORT What is the range of Internet-wide well-known port numbers? What is the range of well-known port numbers for Unix/Linux specific service? What is the range for a client port number? Compare your answer to the well-known port numbers defined in the `/etc/services` file. Are they consistent?

LAB REPORT Explain briefly what a `socket` is.

2 A single segment network

Metcalfe's Law: "The value of a network grows as the square of the number of its users."
Robert Metcalfe

2.1 Objectives

- Network interfaces and interface configuration.
- Network load and statistics.
- The Address Resolution Protocol and its operations.
- ICMP messages and Ping.
- Concept of subnetting.
- Duplicate IP addresses and incorrect subnet masks.

2.2 Local area networks

Generally there are two types of networks: point-to-point networks or broadcast networks. A point-to-point network consists of two end hosts connected by a link, whereas in a broadcast network, a number of stations share a common transmission medium. Usually, a point-to-point network is used for long-distance connections, e.g., dialup connections and SONET/SDH links. Local area networks are almost all broadcast networks, e.g., Ethernet or wireless local area networks (LANs).

2.2.1 Point-to-Point networks

The Point-to-Point Protocol (PPP) is a data link protocol for PPP LANs. The main purpose of PPP is encapsulation and transmission of IP datagrams,

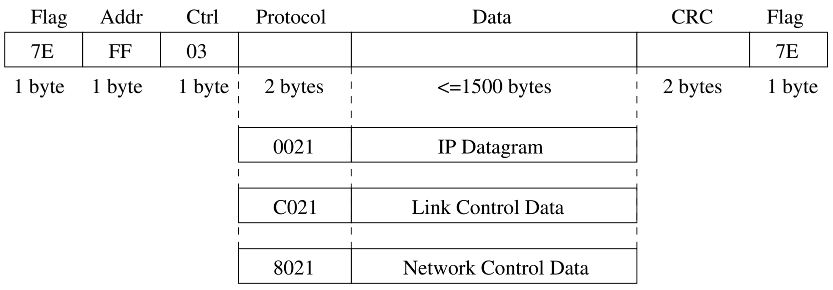


Figure 2.1. PPP frame format.

or other network layer protocol data, over a serial link. Currently, most dial-up Internet access services are provided using PPP.

PPP consists of two types of protocols. The Link Control Protocol (LCP) of PPP is responsible for establishing, configuring, and negotiating the data-link connection, while for each network layer protocol supported by PPP, there is a Network Control Protocol (NCP). For example, the IP Control Protocol (IPCP) is used for transmitting IP datagrams over a PPP link. Once the link is successfully established, the network layer data, i.e., IP datagrams, are encapsulate in PPP frames and transmitted over the serial link.

The PPP frame format is shown in Fig. 2.1. The two `Flag` fields mark the beginning and end points of a PPP frame. The `Protocol` field is used to multiplex different protocol data in the same PPP frame format. Since there are only two end hosts in a PPP LAN, neither an addressing scheme nor medium access control are needed.

2.2.2 Ethernet LANs

In a broadcast network where a number of hosts share a transmission medium, a set of rules, or protocols, are needed in order to resolve collisions and share the medium fairly and efficiently. Such protocols are called medium access control (MAC) protocols. Examples of MAC protocols proposed for various networks are: Aloha, Carrier Sense Multiple Access/Collision Detection (CSMA/CD), and Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA).

Ethernet has been an industry standard since 1982 and is based on the first implementation of CSMA/CD by Xerox. In an Ethernet LAN, all of the hosts are connected to a common channel. When a host has a frame to

Table 2.1. *The exponential backoff algorithm used in CSMA/CD*

1. Set a "time slot" to $2a$.
2. After the i-th collision, the random transmission time is uniformly chosen from a range of 0 to $2^i - 1$ time slots.
3. Do not increase the random time range if $i \geq 10$.
4. Give up after 16 collisions and drop the frame.

send, it first senses the channel to see if there is any transmission going on. If the channel is busy, the host will wait until the channel becomes idle. Otherwise, the host begins transmission if the channel is idle. Assume the maximum end-to-end propagation delay is a seconds. After the first bit is transmitted, the host keeps on sensing the channel for $2a$ seconds. If there is no collision detected during this period, the entire frame is assumed to be transmitted successfully. This is because it takes at most a seconds for all the hosts to hear this transmission, and another a seconds to hear any possible collision with another transmission. When a collision is detected, all hosts involved in the collision stop transmitting data and start to *backoff*, i.e., wait a random amount of time before attempting to transmit again. The random time is determined by the *exponential backoff* algorithm given in Table 2.1.

In addition to attaching all hosts to a common cable or a hub, an Ethernet LAN can be built using *Ethernet switches* with a star topology. Ethernet switches, also called *switched hubs*, are MAC layer devices that switch frames between different ports. An Ethernet switch offers guaranteed bandwidth for the LAN segments connected to each port and separates a LAN into collision domains. If each Ethernet switch port is connected to a single host only, CSMA/CD operation is not required. However, in order for the switch to deal with traffic congestion, the switch may generates a false collision signal (backpressure) to make the transmitting host back off.

2.2.3 IEEE 802.11 wireless LANs

In addition to PPP and Ethernet LANs, wireless LANs (WLANs) using the IEEE 802.11 protocols have rapidly gained popularity in recent years. In a WLAN, computers share a wireless channel. Thus there is no need to install cables, and the computers can be mobile.

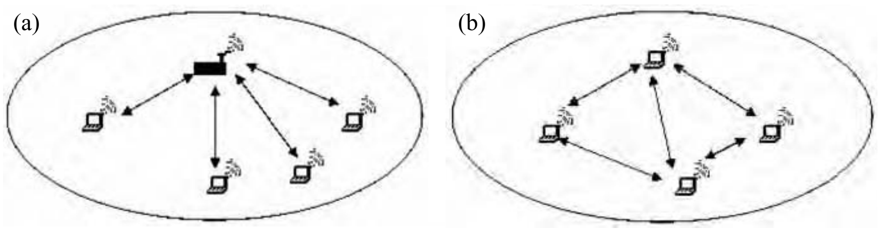


Figure 2.2. Different operation modes of IEEE 802.11 WLANs. (a) The infrastructure mode. (b) The ad-hoc mode.

An IEEE 802.11 WLAN can be configured to work in two modes: the *infrastructure* mode and the *ad-hoc* mode. In the infrastructure mode shown in Fig. 2.2(a), fixed *access points* are used. These access points are connected to the wireline network. Each access point communicates with hosts within its transmission range and serves as a gateway for the hosts. When an active mobile host moves from one access point to another, *handoff* techniques can be applied to switch the connection from the original access point to the new access point without an interruption. In addition, multiple access points can be configured to work together to provide extended coverage. In the ad hoc mode shown in Fig. 2.2(b), there is no need for access points. Host computers can communicate with each other as long as they are in each other's transmission range.

In WLANs, CSMA/CD is inadequate because collision detection cannot be performed effectively in a wireless channel. Rather, CSMA/CA is used for medium access control. In CSMA/CA, a host first senses the medium when it has a frame to send. If the medium remains free for a certain period of time (called the Distributed Coordination Function (DCF) Inter-Frame Space (DIFS)), the host begins transmitting data. When the transmission is over, it waits for an acknowledgement from the receiving host. If no acknowledgement received, it assumes that a collision occurred and prepares to retransmit. On the other hand, if the medium is busy, the host waits for the end of the current frame transmission plus a DIFS, and then begins a backoff procedure like in the case of CSMA/CD protocol. Backoff is performed as follows. The host first chooses a random number within a certain range as a backoff time, and then listens to the wireless channel to determine if it is free or busy. The backoff time is decremented by one if the medium is free in a time slot. However, the host stops decrementing the backoff time if the medium is busy during a time slot, and resumes decrementing it only when the medium becomes free again. When the backoff time becomes 0 and the channel is idle, the host attemps a transmission.

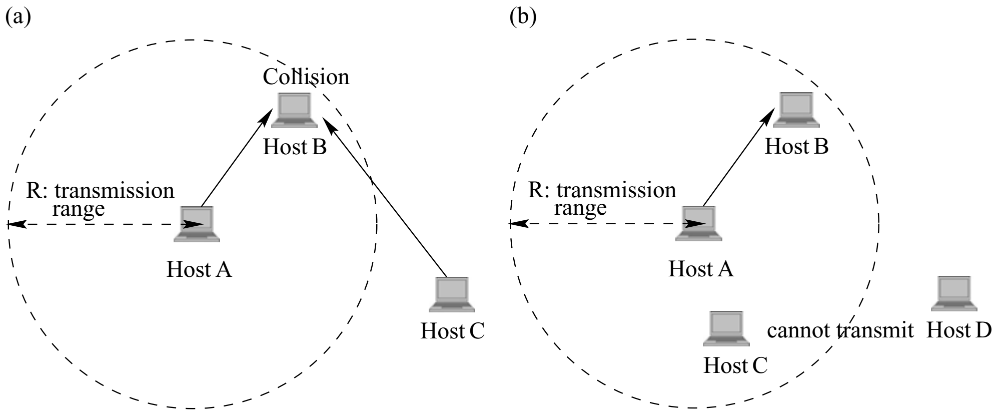


Figure 2.3. The hidden terminal and exposed terminal problems of IEEE 802.11 WLANs. (a) The hidden terminal problem. (b) The exposed terminal problem.

The frame will be dropped if the maximum number of retransmissions is reached.

WLAN has the *hidden terminal* and *exposed terminal* problems inherent from the use of wireless channels. Consider the scenario shown in Fig. 2.3(a), where `Host A` and `C` are far away and cannot hear each other. `Host B` is somewhere in between and can hear both `Host A` and `C`. `Host A` is transmitting data to `Host B`, and `Host C` also has data to send to `B`. If CSMA/CA is used, `Host C` senses an idle channel because it cannot hear `Host A's` transmission. `Host C` therefore begins transmitting data to `Host B` and collision occurs at `Host B`. Neither `Host A` nor `C` can detect the collision in this case. This is called the hidden terminal problem. Now let us consider a different scenario, as shown in Fig. 2.3(b). There are four hosts in the system. `Host A` and `D` are far from each other, and `Host B` and `C` are in between. `Host A` is transmitting data to `Host B`, while `Host C` has data for `Host D`. If `Host D` is out of the tranmission range of `Host A` and `Host B` is out of the transmission range of `Host C`, `Host C` can start transmitting without causing or collision at `Host B and D`. However, if CSMA/CA is used, `Host C` detects a busy channel and will wait till the current transmission is over, resulting in a waste of bandwidth.

In WLAN, the hidden terminal problem is solved by sending request-to-send (RTS) and clear-to-send (CTS) messages before the data transmission. When a host wants to send a data frame, it first sends a RTS carrying the time needed to transmit the frame. The receiving host, if it is free, responds with a CTS. All other hosts that hear the RTS or CTS will mark the medium as busy for the duration of the requested transmission. In the above example, `Host`

`C` first sends a RTS to `Host B`. Since `Host B` is engaged in the transmission from `Host A`, it will not return a CTS. `Host C` cannot transmit without a CTS, and the collision is avoided. However, the exposed terminal problem illustrated in Fig. 2.3(b) is not solved by this mechanism.

IEEE 802.11 is also popularly known as *Wi-Fi* (Wireless Fidelity). The Wi-Fi Alliance, a nonprofit international association, certifies interoperability of WLAN products based on the IEEE 802.11 standard (`http://www.wi-fi.org`).

2.2.4 The Address Resolution Protocol

ARP and RARP

As shown in Fig. 1.1, the protocol layer uses the service provided by the interface layer to send and receive IP datagrams. However, IP addresses used in the protocol layer are not recognizable in the interface layer where physical addresses (or MAC addresses) are used. Furthermore, different kinds of physical networks use different addressing schemes. In order to run TCP/IP over different kinds of physical transmission media, the link layer provides the function that maps an IP address to a physical network address. The protocol that performs this translation is the address resolution protocol (ARP). When a mapping from MAC address to IP address is needed, the reverse address resolution protocol (RARP) is used. Since each type of physical network has different protocol details, there are different ARP RFCs for Ethernet, Fiber-Distributed Data Interface (FDDI), Asynchronos Transfer Mode (ATM), Fiber Channel, and other types of physical networks. In this section, we focus on Ethernet ARP.

When the device driver receives an IP datagram from the IP layer, it first broadcasts an *ARP request* asking for the MAC address corresponding to the destination IP address. After receiving the ARP request, the destination host (with the target IP address) will return an *ARP reply*, telling the sender its MAC address. After this *question-and-answer* process, the source device driver can assemble an Ethernet frame, with the received MAC address as destination MAC address and with the IP datagram as the payload, and then transmit it on the physical medium. Obviously, it is inefficient to have an ARP request/reply exchange for each IP datagram that is sent. Instead, each host maintains an ARP cache, which contains recently resolved IP addresses. When a host has an IP datagram to send to another host, it first checks its ARP cache. If an entry for the destination IP is found, the

Hardware Type	Protocol Type	Hardware Size	Protocol Size	Operation Field	Sender Eth. Addr.	Sender IP Addr.	Target Eth. Addr.	Target IP Addr.
2	2	1	1	2	6	4	6	4 bytes

Figure 2.4. ARP packet format.

corresponding MAC address found in the cache is used and no ARP request and reply will be sent.

Figure 2.4 shows the format of an ARP message, which is 28 bytes long. An ARP request or reply is encapsulated in an Ethernet frame, with the `Protocol Type` field set to 0x0806. An 18-byte padding is needed since the minimum length of an Ethernet frame is 64 bytes.

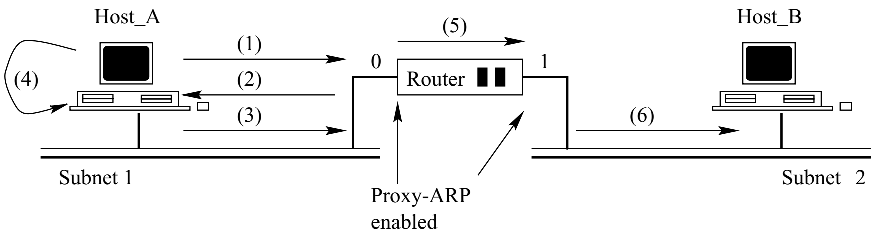
In Fig. 2.4, the first four fields define the types of the addresses to be resolved. `Hardware Type` specifies the type of physical address used, and `Protocol Type` specifies the type of the network protocol address. The next two fields give the length of these two types of addresses. The `Operation` field specifies whether it is an ARP request (with a value of 1), ARP reply (2), RARP request (3), or RARP reply (4). The following four fields are the MAC and IP addresses of the sender and the targeted receiver, respectively.

PPP networks do not use ARP. In this case, hosts must know the IP address at the end of the PPP link. Usually DHCP is used over a PPP link where the IP address of one end host is assigned automatically by the other end host.

Proxy ARP and gratuitous ARO

Proxy ARP and gratuitous ARP are two interesting scenarios of using ARP. Usually proxy ARP is used to hide the physical networks from each other. With proxy ARP, a router answers ARP requests targeted for a host. An example is shown in Fig. 2.5. There are several interesting observations to make in this example.

1. `Host_A` and `Host_B` are in two different subnets connected by a router. By setting their network masks appropriately, they are logically in the same subnet (directly connected) (see the example in Fig. 2.9 and Section 0.5.)
2. In the ARP table of `Host_A`, all the IP addresses in the `Host_B` subnet are mapped to the same MAC address, i.e., Router Port 0's MAC address.
3. All packets from the `Host_A` subnet to the `Host_B` subnet are sent to the router first, and forwarded by the router to the destination.



(1): Host_A broadcasts an ARP request for Host_B

(2): Router Port 0 replies for Host_B

(3): Host_A sends the frame to Router Port 0

(4): Host_A inserts a new entry in its ARP cache: {(Host_B's IP) at (Router Port 0's MAC)}

(5): Router forwards the frame to port 1

(6): Router port 1 sends the frame to Host_B

Figure 2.5. A proxy ARP example.

Gratuitous ARP occurs when a host broadcasts an ARP request resolving its own IP address. This usually happens when the interface is configured at bootstrap time. The interface uses gratuitous ARP to determine if there are other hosts using the same IP address. It also advertises the sender's IP and MAC address, and other hosts in the network will insert this mapping into their ARP table.

Manipulating the ARP table

An entry in the ARP table has three elements: (1) an IP address, (2) the MAC address associated with this IP address, and (3) flags. A normal entry expires 20 min after it is created or the last time it is referred. If an entry is manually entered, it has a *permanent* flag and will never time out. If an entry is manually entered with the **pub** key word, it has an additional *published* flag which means the host will respond to ARP requests on the IP address in this entry. If a host sends an ARP request but gets no reply, an *incomplete* entry will be inserted into the ARP table. The ARP table can be manipulated using the **arp** command. Some options of **arp** are given here.

- **arp -a**: Displays all entries in the ARP table.
- **arp -d**: Deletes an entry in the ARP table.
- **arp -s**: Inserts an entry into the ARP table.

2.3 Network interface

2.3.1 Operations of a network interface

Figure 2.6 illustrates the operations of an Ethernet card with two interfaces. Most TCP/IP implementations have a loopback interface with the IP address

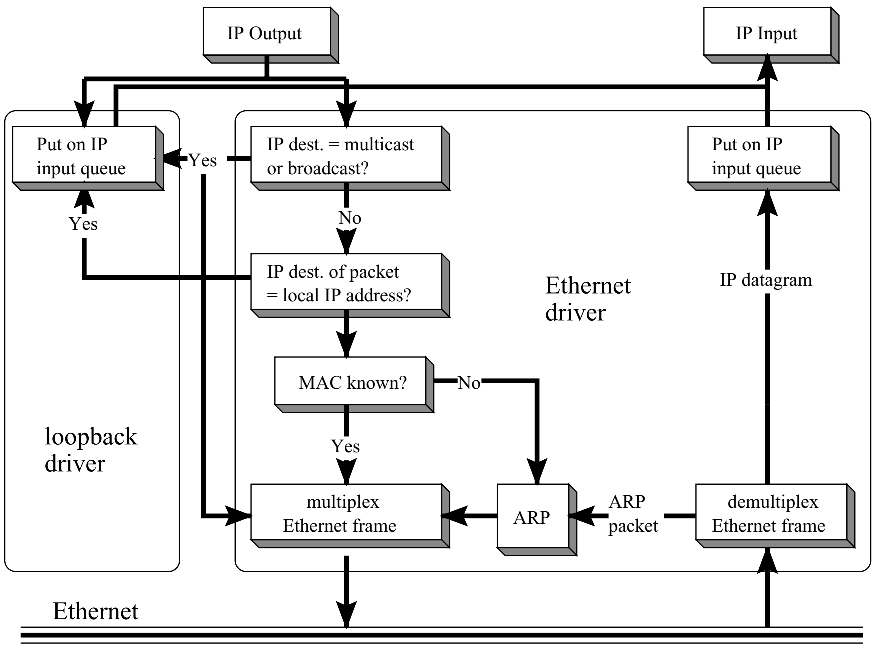


Figure 2.6. Functional diagram of an Ethernet interface card.

127.0.0.1 and domain name `localhost`. It behaves as a separate data link interface, but packets sent to it will be put in the IP input queue and will not be transmitted on the medium. The loopback interface is used for debugging.

The Ethernet interface reads and sends Ethernet frames from or to the medium. When it receives a frame, it reads the `Frame Type` field. If `Frame Type` is 0x0806, the ARP message carried in the frame is extracted and sent to the ARP module. If `Frame Type` is 0x0800, an IP datagram is extracted and sent to the IP input queue. When the IP layer has a datagram to send, the device driver first checks if it is destined for itself. In this case (i.e., multicast, broadcast, destination IP address is its own IP address, or the destination IP address is 127.0.0.1), a copy of the datagram is sent to the loopback interface. Otherwise, the device driver will search the ARP table for the destination MAC address. If there is no hit in the table lookup, an ARP request is broadcast. When the sending host gets the destination MAC address, an Ethernet frame is assembled and the device driver begins to compete for the channel, attempting to send the frame on the medium.

There is a limit on the frame size of each data link layer protocol. This limit is determined by the reliability of the physical medium and the nature of the MAC scheme used. It translates itself to a limit on the size of the IP datagram that can be encapsulated in a link layer frame, which is called the maximum transmission unit (MTU). Examples of MTUs are: 1500 bytes for Ethernet and 4352 bytes for FDDI. If an IP datagram to be sent is longer than the MTU of the interface, the IP datagram will be fragmented and carried in several data link layer frames. We will further discuss MTU and IP fragmentation in Chapter 5.

2.3.2 Configuring a network interface

The **netstat** command can be used to display the configuration information and statistics on a network interface. The same command is also used to display the host routing table. We list several **netstat** options below that will be used frequently in the following experiments.

- **netstat -a**: Shows the state of all sockets, routing table entries, and interfaces.
- **netstat -r**: Displays the routing table.
- **netstat -i**: Displays the interface information.
- **netstat -n**: Displays numbers instead of names.
- **netstat -s**: Displays per-protocol statistics.

The **ifconfig** command is used to configure a network interface. The following options are used for the reconfiguration of the IP address and network mask.

- **ifconfig -a**: Shows the states of all interfaces in the system.
- **ifconfig** *interface_name* **down**: Disables the network interface,[1] where *interface_name* is the name of the Ethernet interface.[2]
- **ifconfig** *interface_name new_IP_address* **up**: Assigns a new IP address to the interface and brings it up.
- **ifconfig** *interface_name* **netmask** *new_netmask*: Assigns a new network mask for the interface.

2.4 The Internet Control Message Protocol

ICMP is a protocol in the network layer that communicates error messages and other conditions that require attention, as well as routing information.

[1] It is recommended that the interface be disabled first, before changing its settings.
[2] Different systems may give different names for the interfaces, e.g., `le0` for Sun Sparc 4 with SunOS 5.5.1 and `hme0` for Sun Ultra 5 with Solaris 8. You can find the name by typing **netstat -i** or **ifconfig -a**.

Table 2.2. *Types and codes of ICMP messages*

Type	Code	Description	–
0	0	echo reply	query
3	0–15	destination unreachable	error
5	0–3	redirect	error
8	0	echo request	query
9	0	router advertisement	query
10	0	router solicitation	query
11	0–1	time exceeded	error

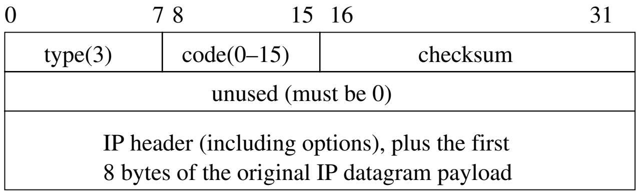

0
7 8
15 16
31
type(3)
code(0–15)
checksum
unused (must be 0)
IP header (including options), plus the first
8 bytes of the original IP datagram payload

Figure 2.7. Format of an ICMP error message.

An ICMP message is encapsulated in an IP datagram, with the `Protocol Type` value 0x01.

There are many different ICMP messages, each of which is used for a specific task. These ICMP messages have a 4-byte common header, as shown in Fig. 2.7 and Fig. 2.8. The Type and Code fields in the common header define the function of the ICMP message. Some frequently used ICMP messages are given in Table 2.2.

Figure 2.7 displays the format of an ICMP error message. The IP header and the first 8 bytes of the payload of the original IP datagram are carried in the ICMP error message and returned to the source. The sender can analyze the returned header and data to identify the cause of the error. Note that the first 8 bytes of the original IP payload contain the source and destination port numbers in the UDP or TCP header.

Figure 2.8 gives the format of an ICMP echo request or reply message. These messages are used by the **ping** program to determine if a remote host is accessible. **ping** sends ICMP echo requests to the target IP address. The target host will respond with an ICMP echo reply for each ICMP request correctly received. The round trip time for each request/reply pair may be reported in the **ping** console. The fact that no ICMP echo reply is received means either that there is no path available to the remote host, or the remote host is not alive. When there are multiple **ping**s running on a host, each of

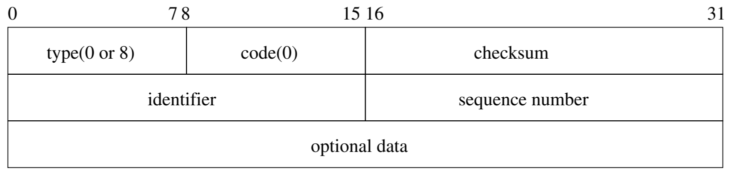


Figure 2.8. Format of an ICMP echo request or echo reply message.

them is assigned a unique identifier. The `sequence number` field is used to match reply to request.

2.5 The Sock traffic generator

Sock is a test program that can be run either as a client or as a server, using UDP or TCP. It also provides a means to set various socket options. Sock operates in one of the following four modes [5].

1. Interactive client: connects to a server, and copies the *standard input*, i.e., keys a user typed, to the server and copies everything received from the server to the *standard output*, i.e., the screen.
2. Interactive server: waits for a connection request from a Sock client, and then copies the standard input to the client and copies everything received from the client to the standard output.
3. Source client: sends packets to a specified server.
4. Sink server: receives packets from a client and discards the received data.

2.6 Network interface exercises

The following exercises use the single segment network topology shown in Fig. 1.3.

Exercise 1 Use the **ifconfig -a** command to display information about the network interfaces on your host. Find the IP address and the net mask of your machine.

LAB REPORT How many interfaces does the host have? List all the interfaces found, give their names, and explain their functions briefly.

LAB REPORT What are the MTUs of the interfaces on your host?

LAB REPORT Is network subnetted? What is the reasoning for your answer? What the experimental are the reasons for subnetting?

Exercise 2 While **tcpdump host** *your_host* is running in one command window, run **ping 127.0.0.1** from another command window.

LAB REPORT From the **ping** output, is the 127.0.0.1 interface on? Can you see any ICMP message sent from your host in the **tcpdump** output? Why?

Exercise 3 By using **netstat -in** command, collect the statistics from all the hosts on the network. Since we use the same login name and password, we can **telnet** to other workstations and run **netstat -in** there.[3]

Save the **netstat -in** outputs.

If you don't see a significant amount of output packets in the **netstat** output, the machine was probably restarted recently. You may do this experiment later, or use the following **sock** command to generate some network traffic:

sock -u -i -n200 *remote_host* **echo**.

LAB REPORT Calculate the average collision rate over all the hosts for the set of statistics you collected in this exercise.

2.7 ARP exercises

In the following experiment, we shall examine the host ARP table and the ARP operation, including two interesting cases: proxy ARP and gratuitous ARP. You may need to ask the lab instructor for the MAC addresses of the host and router interfaces, and record these MAC addresses in Table A.1 and Table A.2 in the appendix. You need these MAC addresses for the exercises and lab report.

Exercise 4 Use **arp -a** to see the entire ARP table. Observe that all the IP addresses displayed are on the same subnet.

If you find that all the remote hosts are in your host's ARP table, you need to delete a remote host (not your workstation) from the table, using,

arp -d *remote_host*.[4]

Save the ARP table for your lab report.

While **tcpdump -enx -w exe2.out** is running, **ping** a remote host that has no entry in your host ARP table. Then terminate the **tcpdump** program.

[3] After you are done with a remote host, you should exit the **telnet** session before you **telnet** to another remote host. Recursive **telnet** will generate unnecessary data in the **tcpdump** output and cause confusion.

[4] If you deleted your workstation's IP address from the ARP table by mistake, you must add the entry back in the table. See the **arp** manual page to add. Note that, in order for your workstation to reply to the ARP requests, the ARP entry of your workstation must have the **P** flag in the ARP table.

Next, run **ethereal -r exe2.out&** to load the **tcpdump** trace file.

Observe the first few lines of the packet trace to see how ARP is used to resolve an IP address.

Run **arp -a** to see a new line added in your host's ARP table. Save the new ARP table for your lab report.

Mark the ARP request packet and the ARP reply packet in the **ethereal** window. Then go to menu `File/Print ...` to print the marked packets for your lab report (See Exercise 6 of Chapter 1).

LAB REPORT From the saved **tcpdump** output, explain how ARP operates. Draw the format of a captured, ARP request and reply including each field and the value.

Your report should include the answers for the following questions.

- What is the target IP address in the ARP request?
- At the MAC layer, what is the destination Ethernet address of the frame carrying the ARP request?
- What is the `frame type` field in the Ethernet frame?
- Who sends the ARP reply?

Exercise 5 While **tcpdump host** *your_host* is running to capture traffic from your machine, execute **telnet 128.238.66.200**. Note there is no host with this IP address in the current configuration of the lab network.

Save the **tcpdump** output of the first few packets for the lab report.

After getting the necessary output, terminate the **telnet** session.

LAB REPORT From the saved **tcpdump** output, describe how the ARP timeout and retransmission were performed. How many attemps were made to resolve a non-existing IP address?

Exercise 6 The network topology for this proxy ARP exercise is shown in Fig. 2.9. We will divide the group into two subnets interconnected by a router. The IP addresses and network masks for the hosts are also given in Fig. 2.9. Change the IP address and network mask of your host accordingly (see Section 2.3.2). The IP addresses and network masks of the `Router4` interfaces are the same as their default settings. Note that the network mask of the hosts in the 128.238.65.0 network is 255.255.0.0.

Next we will enable the proxy ARP function on the `ethernet1` interface of `Router4`.

1. **telnet** to `Router4` from `shakti`: **telnet 128.238.64.4**. The login password is `el537`.[5]

[5] Check with your lab instructor for the password of the router your are using, which may be different from `el537`.

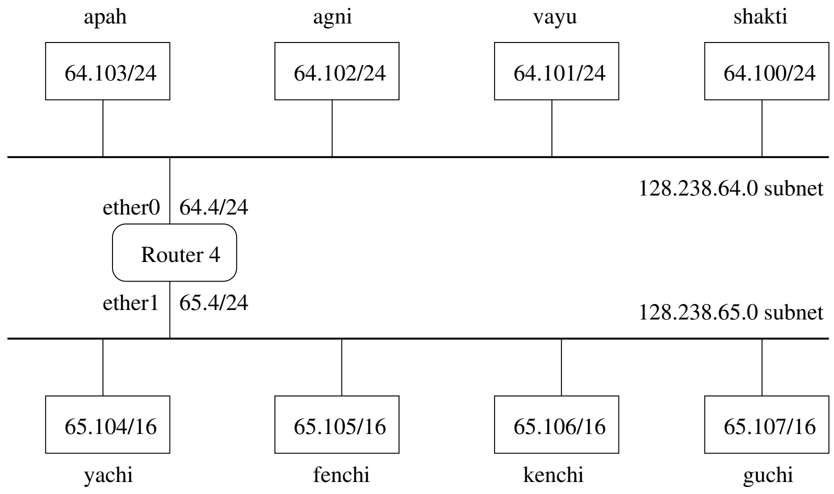


Figure 2.9. Network configuration for the proxy ARP experiment.

2. Log in to the router, type **enable** to enter the *Privileged EXEC* mode.[6] The password is again `el537`.
3. Enter the *Global Configuration* mode by typing **config term**.
4. Then type the following lines:

   interface ethernet 1[7]

   ip proxy-arp

   `Ctrl-Z`
5. Type **exit** to terminate the **telnet** session.

Now `Router4`'s `ethernet1` interface can perform proxy ARP for the hosts in the 128.238.64.0 subnet.

Run **tcpdump -enx** on all the hosts.

Then let the hosts in the 128.238.65.0 subnet send UDP datagrams to the hosts in the 128.238.64.0 subnet. For example, on `guchi` type:

sock -i -u -n1 -w1000 *Host_in_64.0_subnet* **echo**.

When you are done with all the hosts in the 128.238.64.0 subnet, save the tcpdump output for the lab report.

Run **arp -a** to display the new ARP table in your host. Save the ARP table for your lab report.

After the lab instructor restores the network into a single subnet (see Fig. 1.3), change the IP address and network mask of your host's interface back to their default values as in Fig. 1.3.

[6] We will discuss bridge and router configuration in Chapter 3.
[7] The name of the router interfaces may be different for various routers. You can find the names by typing **write term** in the *Privilege EXEC* mode.

Exchange your data saved in this exercise with a student working in the other subnet.

LAB REPORT Explain the operation of proxy ARP.

Why can a host in the 128.238.65.0 subnet reach a host in the 128.238.64.0 subnet, even though they have different subnet IDs?

What are the MAC addresses corresponding to hosts in the 128.238.64.0 subnet, in the ARP table of a host in the 128.238.65.0 subnet?

Give one advantage and one disadvantage of using proxy ARP.

Exercise 7 This exercise will be performed by all the students together. While **tcpdump -ex -w exe7.out** is running on all the hosts, reboot host `guchi`.

After `guchi` is started, terminate **tcpdump** and run **ethereal -r exe7.out &** to load the **tcpdump** trace. Print the the gratuitous ARP request for your lab report.

LAB REPORT What is the purpose of gratuitous ARP?

LAB REPORT List the sender IP address, target IP address, sender MAC address, and target MAC address of the gratuitous ARP you saved.

2.8 Exercise with ICMP and Ping

Exercise 8 Use **ping -sv** *remote_host* to test whether the remote host is reachable, while running: **tcpdump -enx host** *your_host* **and** *remote_host*.

Save the **tcpdump** and **ping** output for the future study on **ping**.

LAB REPORT What ICMP messages are used by **ping**?

Exercise 9 While running **tcpdump -x -s 70 host** *your_host* **and** *remote_host*,

execute the following **sock** command to send a UDP datagram to the remote host: **sock -i -u -n1 -w1000** *remote_host* **88888**.

Save the **tcpdump** output for the lab report.

LAB REPORT Study the saved ICMP port unreachable error message (see Fig. 2.7). Why are the first 8 bytes of the original IP datagram payload included in the ICMP message?

Exercise 10 While **tcpdump** is running to capture the ICMP messages, **ping** a host with IP address 128.238.60.100. Save the **ping** output.

Table 2.3. *Host IP addresses and network masks for exercise 11*

Group	Name	IP address	Subnet mask
Group A	`shakti`	128.238.66.100	255.255.255.0
	`vayu`	128.238.66.100	255.255.255.0
	`agni`	128.238.66.102	255.255.255.0
	`apah`	128.238.66.103	255.255.255.0
Group B	`yachi`	128.238.66.104	255.255.255.0
	`fenchi`	128.238.66.104	255.255.255.0
	`kenchi`	128.238.66.106	255.255.255.0
	`guchi`	128.238.66.107	255.255.255.0

LAB REPORT Can you see any traffic sent on the network? Why? Explain what happened from the **ping** output.

LAB REPORT List the different ICMP messages you captured in Exercises 8, 9, and 10 (if any). Give the values of the `type` and `code` fields.

2.9 Exercises with IP address and subnet mask

In this section, we will observe what happens when the same IP address is assigned to two different hosts. We will also set an incorrect subnet mask for hosts and see what are the consequences. For the next two exercises, we split the current single segment network into two segments, Group A and Group B as shown in Table 2.3, so that they will not interfere with each other.

Exercise 11 Change the IP address of your workstation as shown in Table 2.3.

Delete the entries for all hosts other than your own workstation from your workstation's ARP table.

Run **tcpdump -enx** on all the hosts. Then, do the following three experiments.

1. Execute **telnet** from one of two hosts with the duplicate IP address to a host with unique IP address (e.g. `shakti` –> `agni` in Group A and `yachi` –> `kenchi` in Group B).
   Now, from the other host with the duplicate IP address, execute **telnet** command to the same host (`vayu` –> `agni` or `fenchi` –> `kenchi`).
   Observe what happens and save the **tcpdump** output and the ARP tables in all the hosts in your group.

Table 2.4. *Host IP addresses and network masks for exercise 12*

Group	Name	IP address	Subnet mask
Group A	`shakti`	128.238.66.100	255.255.255.240
	`vayu`	128.238.66.101	255.255.255.0
	`agni`	128.238.66.102	255.255.255.0
	`apah`	128.238.66.120	255.255.255.240
Group B	`yachi`	128.238.66.104	255.255.255.240
	`fenchi`	128.238.66.105	255.255.255.0
	`kenchi`	128.238.66.106	255.255.255.0
	`guchi`	128.238.66.121	255.255.255.240

2. Execute **telnet 128.238.66.100** (or **128.238.66.104**) from `agni` (or `kenchi`). Which host provides the telnet connection? Why?
3. Execute **telnet 128.238.66.100** (or **128.238.66.104**) from `apah` (or `guchi`). Which host is connected to `apah` (or `guchi`)? Why?

LAB REPORT Explain what happened in the first case and why. Answer the questions for the second and third cases.

Exercise 12 Change the host IP addresses and the subnet masks as shown in Table 2.4. Since we still have two separate segments, Groups A and B can do the exercise independently. Note that two hosts in each group (`shakti` and `apah` in Group A, or `yachi` and `guchi` in Group B) are assigned an incorrect subnet mask.

Capture the packets with **tcpdump -e** for the following cases.

1. When `shakti` (`yachi`) **ping**s one of the hosts that have the correct subnet mask.
2. When `apah` (`guchi`) **ping**s one of the hosts that have the correct subnet mask. Now, copy the output displayed from the **ping** window in `apah` (`guchi`). Share the saved output message with other students.
3. When a host with the correct subnet mask **ping**s `shakti` (`yachi`).
4. When a host with the correct subnet mask **ping**s `apah` (`guchi`).

To avoid confusion, only one machine in each group should generate traffic in each case. Clearly, this exercise has to be performed as a team.

LAB REPORT Explain what happened in each case according to the **tcpdump** outputs saved. Explain why `apah` (or `guchi` in Group B) could not be reached from other hosts, whereas `shakti` (or `yachi` in Group B), which has the same incorrect subnet mask, could communicate with the other hosts.