CHAPTER 2

SINGLE SEGMENT NETWORKS

Chapter 2 focuses on topics related to the transmission of IP datagrams over a single Ethernet segment. The first section gives an overview of Ethernet. The second sections discusses the Address Resolution Protocol (ARP), and its role in resolving IP addresses. Section 4 and Section 5, respectively, provide an overview of the Internet Protocol (IP) and the Internet Control Message Protocol (ICMP). The last section discusses network configuration and traffic analysis tools used in Lab 2.

TABLE OF CONTENTS

1. Ethernet Networks

Ethernet networks are the dominant local area network (LAN) technology. The broadcast mode of transmissions in Ethernet makes it easy to observe traffic between hosts and routers, and makes Ethernet very attractive for use in the Internet Lab. This section presents an overview of Ethernet networks, including its Medium Access Control (MAC) protocol, the Ethernet frame format, and physical layer issues.

Ethernet networks were invented in the mid-1970s by Bob Metcalfe and D.R. Boggs at the Xerox Palo Alto Research Center. Ethernet was first standardized in 1980 by a consortium of DEC, Intel, and Xerox. This standard is known as Ethernet II or DIX. In 1983, the Institute of Electrical and Electronics Engineers (IEEE) adopted a revised, but compatible, version of Ethernet II in the IEEE 802.3 CSMA/CD standard. The IEEE 802.3 committee has repeatedly revised the 802.3 standard, to integrate new advances in transmission technology. Currently, standards are available for data rates at 10 Mbps, 100 Mbps, 1 Gbps, and 10 Gbps, running over a wide variety of media, ranging from phone wires and optical fiber cables.

The first generation of Ethernet networks was designed for a bus network topology. A single coaxial cable, with a length of up to 500 m, provides the transmission media, Ethernet devices are directly connected to the coaxial cable, as shown in Figure 2.1. The coaxial cable provides a shared broadcast medium, meaning that any transmission can be received by all devices connected to the coaxial cable. A single Ethernet LAN, as shown in Figure 1.1, is also called an *Ethernet segment.*

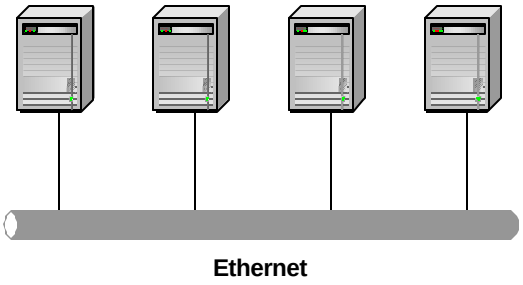


Figure 2.1. Ethernet segment with a bus topology.

Ethernet segments can be interconnected to create larger networks using devices such as hubs, switches, or routers. Ethernet hubs and Ethernet switches are devices that operate at the data link layer, whereas routers operate at the network layer. The issues involved in interconnecting Ethernet segments will be revisited in Chapter 5.

1.1. The Ethernet MAC protocol

The main abstraction of an Ethernet network is that of a broadcast network with a shared communication channel. Broadcast networks are common in local area networks, and are not limited to Ethernet. An important issue with broadcast networks is that only one network device should transmit at any time. If two or more devices transmit at the same time, the transmissions are corrupted. This is referred to as a *collision*. Protocols that arbitrate access to a shared medium are called *Multiple Access Control (MAC) protocols*. The MAC protocol of a local area network is generally implemented in hardware on the network interface card of a host or router.

The MAC protocol of Ethernet networks is called *CSMA/CD (Carrier Sense Multiple Access with Collision Detections)* and works as follows:

1. A device that has data ready for transmission listens to the broadcast channel for an ongoing transmission. If the channel is idle, the device transmits. Otherwise, the device waits until the channel becomes idle. This is the "carrier sense" portion of CSMA/CD, sometimes also called `listen before talking'.

2. While a device is transmitting data, it continues to listen to the broadcast channel to determine if there is a collision. This is the "collision detection" part of the MAC protocol, also referred to as "listen while talking". If a collision is detected, the device stops its transmission and sends a jamming signal that notifies other devices about the collision. Then, the device waits for a random period of time and makes a retransmission attempt.

3. When a retransmission results in a collision, the device waits again a random time and retransmits. This time, however, the station doubles the interval from which the random time is selected (backoff interval). If another collision occurs, the backoff interval is doubled once again. Doubling the backoff interval intends to reduce the likelihood of repeated collisions.

Compared to MAC protocols of other local area networks, CSMA/CD is a relatively simple protocol. The MAC protocol of Ethernet has remained unchanged over the years. Most versions of Ethernet run the CSMA/CD MAC protocol. An exception are full-duplex Ethernet versions, that can transmit and receive concurrently without ever encountering collisions. In these versions of Ethernet, a device can transmit data at any time.

1.2. Ethernet Frame Format

Ethernet frames in transmission have a structure as shown in Figure 2.2. Each Ethernet frame is preceded by a 64 bit preamble that consists of a bit pattern of alternating 1's and 0's, and has the last two bits set to "11". This bit pattern is used by the receiver to synchronize with the bit timing of the frame. The last two bits indicate the beginning of the Ethernet frame. Consecutive frames must be separated by a gap of 96 bits, which corresponds to 96 nanoseconds on a 100 Mbps Ethernet network.

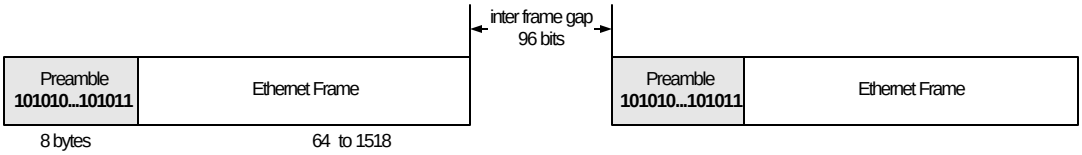


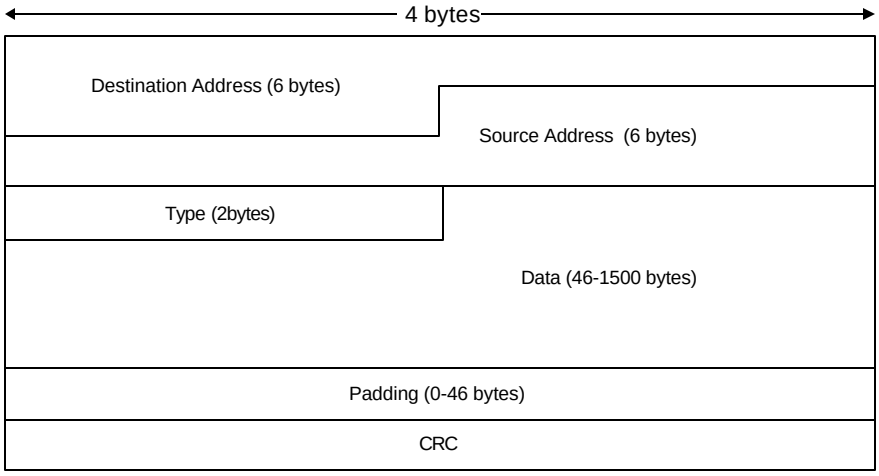
Figure 2.2. Ethernet Frames.

The frame itself has a minimum size of 64 bytes and a maximum size of 1518 bytes. The requirement for a minimum frame size is needed for the collision detection part of the Ethernet MAC protocol. If frames are shorter than 64 bytes, Ethernet stations may not be able to detect a collision of a frame. The collision detection component of Ethernet requires that a transmitting station becomes aware of a collision occurs before the transmission of the frame is completed. The longest time to detect a collision occurs, when two stations with the maximum distance between them start to transmit a frame at the same time. In this case, the time to detect a collision is twice the propagation delay between the transmitter and the receiver. In the original Ethernet specification, the maximum length of an Ethernet network, if extended by signal repeating devices, is 2500 meters, resulting in a delay of 51.2 microseconds, which translates into 512 bits or 64 bytes for a 10 Mbps network.

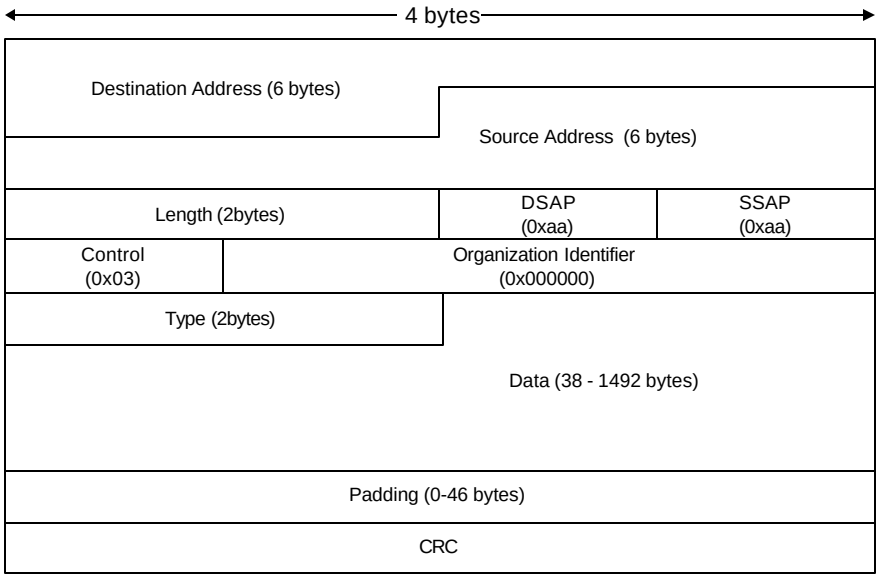
The format of an Ethernet frame is somewhat different in the Ethernet II and IEEE 802.3 versions. Following a widely used convention, we will refer to an Ethernet II frame as an Ethernet frame and an IEEE 802.3 frame as an 802.3 frame. Both versions have a 14 byte long Media Access Control (MAC) header and a 4 byte long trailer. The 802.3 frame has an additional 8 bytes of header information. Both types of frames may be observed on the same Ethernet network, and most Ethernet devices recognize both formats. However, Ethernet frames are much more common.

The Ethernet frame format is shown in Figure 2.3(a). The frame header has a length of 14 bytes. The first two fields are the destination and source MAC addresses, using the format that was discussed in Chapter 0. The next field is the protocol type field. This field serves as the demultiplexing field and identifies the protocol used in the payload of the Ethernet frame. If the payload of the frame is an IP datagram, the protocol field is set to 0x8000. The payload itself has a length between 46 and 1500 bytes. If an IP datagram is smaller than 46 bytes, then

additional zeros are added as padding to satisfy the minimum length requirement. Following the payload is a four byte long Cyclic Redundancy Check (CRC) field, which serves as a checksum for error detection purposes.



(a) Ethernet Encapsulation



(b) IEEE 802.3 Encapsulation

Figure 2.3. Ethernet Frame Formats.

The 802.3 frame format, shown in Figure 2.3, also has a MAC header with 14 bytes. The first two fields are identical to Ethernet, but the third field is interpreted differently. This field contains the length of the Ethernet payload in bytes. Following the type field is what is called a LLC/SNAP header. It consists of an IEEE 802.2 Logical Link Control (LLC) header which has a length of three bytes. The fields of the LLC header are a Destination Service Access Point (DSAP) and a Source Service Access Point (SSAP) with a length of each 1 byte, followed by a 1 byte control field. The IEEE 802.2 Subnetwork Access Protocol (SNAP) header is five bytes long and contains an organization unique identifier (OUI) and a type field. In 802.3 frames the fields are always set to the same value, and, we therefore, skip a lengthy discussion of their role. The DSAP and SSAP fields are always set to 0xaa, the control field is always set to 0x03, the OUI field is always set to 0x000000, and the type field has the same interpretation as the last byte in the MAC header of the Ethernet frame.

Since the same Ethernet network can have both Ethernet and 802.3 frames, how can a host distinguish the type of a frame? The determination is done with the content of bytes 13 and 14. In Ethernet II, these bytes are designated as the type field, and in 802.3 as the length field. Since the length of an Ethernet frame is less than 1500, a host can distinguish 802.3 and Ethernet II frames, as long as all types are indicated by a value larger than 1500. Thus, a frame is identified as 802.3 frame if the value of bytes 13 and 14 does not exceed 1500, and otherwise as an Ethernet frame.

1.3. The Ethernet Physical Layer

The MAC protocol and the frame format of Ethernet have changed little since the early 1980s. On the other hand, the physical layer of Ethernet has evolved significantly. Over the years, close to 20 different physical layers have been specified for Ethernet. The physical layer specifications are generally referred to by acronyms, such as 10Base5, 100BaseT, 1000BaseFX, and so on. The acronyms specify the data rate in Megabits per second, the signaling method, and the transmission media. The data rate is given by the first number “10”, “100”, or “1000”. With one exception, physical layer specifications for Ethernet use digital signaling, and indicate this with a “Base” in the acronym. Only one physical layer, the now obsolete 10Broad36, defines analog signaling over coaxial cables. The transmission media is denoted by a letter or a number at the end of the acronym. The convention is that “5” and “2” specify different types of coaxial cables (10Base2, 10Base5). If the last part of the acronym starts with a “T” (as in 10BaseT, 100BaseTX, 100BaseT4) the physical layer uses unshielded twisted pair (UTP) cables. If the last part of the acronym contains an “X” (100BaseCX, 1000BaseFX, 100BaseX) the physical layer probably uses a fiber optic cable.

The physical layer specifications 10Base5 and 10Base2, also known as thick Ethernet and thin Ethernet, respectively, use a coaxial cable in a bus topology, as seen in Figure 2.1, and support a data rate of 10 Mbps. Both versions of Ethernet have become obsolete.

Today, the most widely used Ethernet physical layers are 10BaseT and 100BaseTX, which have a data rate of 10 Mbps and 100 Mbps, respectively. Both technologies UTP cables with RJ-45 connectors. These Ethernet networks are not configured in a bus topology. Instead, the network is configured as a star, where stations connect to an Ethernet hub or an Ethernet switch[1]. In Figure 2.4 we show an Ethernet segment with one hub and a few hosts. Many Ethernet hubs and Ethernet interface cards are dual-speed, meaning that they can support both 10BaseT and 100BaseTX. All Ethernet networks in the Internet Lab use 10BaseT or 100BaseTX.

In principle, Ethernet hubs preserve the broadcast property of Ethernet LANs. This means that a hub transmits every incoming packet on all of its ports. Increasingly, however, hubs forward a packet on a port only if the packet is marked as a broadcast packet, or if the packet is directed to the host connected to the port. The technical term for such a hub is *switched hub*.

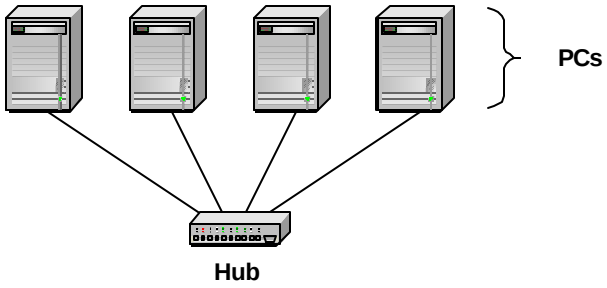


Figure 2.4. Ethernet Segment with a Hub in a Star Topology

The IEEE 802 Committee

The IEEE 802 LAN/MAN standards committee within the Institute of Electrical and Electronics Engineers (IEEE) is devoted to the development of local area network and metropolitan area network standards. Since the 1980s, IEEE 802 has specified standards for a variety of technologies including CSMA/CD (IEEE 802.3), Token Bus (IEEE 802.4) and Token Ring (IEEE 802.5), Wireless LAN (IEEE 802.11), and several more. The IEEE 802 committee has defined more than a dozen different MAC layer protocols and, generally, many different physical layer protocols for each MAC layer protocol. Each of these specifications describes a MAC layer and generally multiple physical layers for that MAC layer. The MAC layer and the physical layer in the 802 standards are very different, however, the frame formats are very similar. Also, most standards use the MAC addresses that we saw in Ethernet.

The 802 committee has defined a complete protocol architecture for LANs and MANs, as illustrated in Figure 2.5. The protocol architecture specifies that all LAN and MAN specifications have a common upper layer interface to the IEEE 802.2 Logical Link Control (LLC) layer. The IEEE 802.1 standard describes the the overall IEEE 802 architecture, addresses interoperability issues between different types of LANs and MANs, and discusses how to interface with layers above the LLC layer.

The modular design of the IEEE 802 protocol architecture, with the common LLC layer makes it relatively easy to introduce new LAN and MAN protocols. For protocols above the MAC layer, including the entire TCP/IP protocol suite, the transition to a new LAN standard is almost transparent.

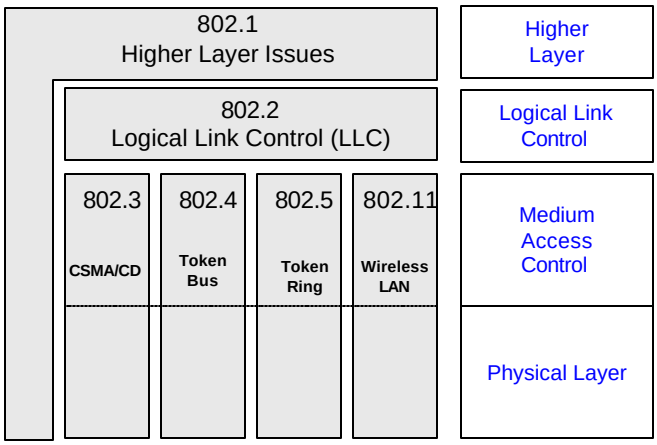


Figure 2.5. IEEE 802 LAN/MAN Protocol Architecture.

1.4. Processing of Ethernet Frames in Linux

The operations at the physical layer and the MAC layer are generally implemented in hardware on the network interface card. The operations at the data link layer, such as the construction and demultiplexing of frames, are performed in the device drivers of the network interface cards. Recall from Chapter 1, that each network interface is associated with a device driver. A device driver is responsible for the data exchange between the Linux operating system and the network interface card, generally via memory mapped I/O.

In Figure 2.6 we show the operations of an device driver for an Ethernet card when it processes an IP datagram. The figure also includes the driver of the loopback interface. Recall that the loopback interface is a virtual interface, which is present on any Linux system and which is generally assigned the IP address 127.0.0.1. If a host sends a packet to the loopback interface,

the packet is immediately returned to the network layer of the host. Thus, the loopback interface permits a host to send packets to itself.

When an IP module on a Linux host transmits an IP datagram on an Ethernet card, the device driver first tests if the sending host should also receive the IP datagram, and if so, passes the IP datagram to the loopback driver. The sender of a datagram should receive the IP datagram, when the IP destination address is a multicast or broadcast address or when the IP destination address matches one of the IP addresses of sending host. If this is not the case, the Ethernet driver tests if it knows the MAC address for the destination IP address. If not, the ARP protocol, which is discussed in the next section, is invoked to resolve the MAC address and the datagram is kept in a buffer until the MAC address becomes available. If the MAC address is known, the Ethernet driver constructs the Ethernet frame with the format shown in Figure 2.3. Once the frame is constructed, the Ethernet device driver passes the frame to the Ethernet card. The MAC and physical layer are handled by the hardware on the Ethernet card, and are not part of the Linux operating system.

When an Ethernet frame arrives at a network interface card, the interface cards assembles the frame, writes the frame into a memory location, and issues a signal to the Linux operating system. Then, the device driver for the appropriate network interface reads and demultiplexes the frame. As shown in Figure 2.3, if the packet is an ARP message the payload of the frame is passed to ARP, and if it is an IP datagram, the payload is passed to the IP module.

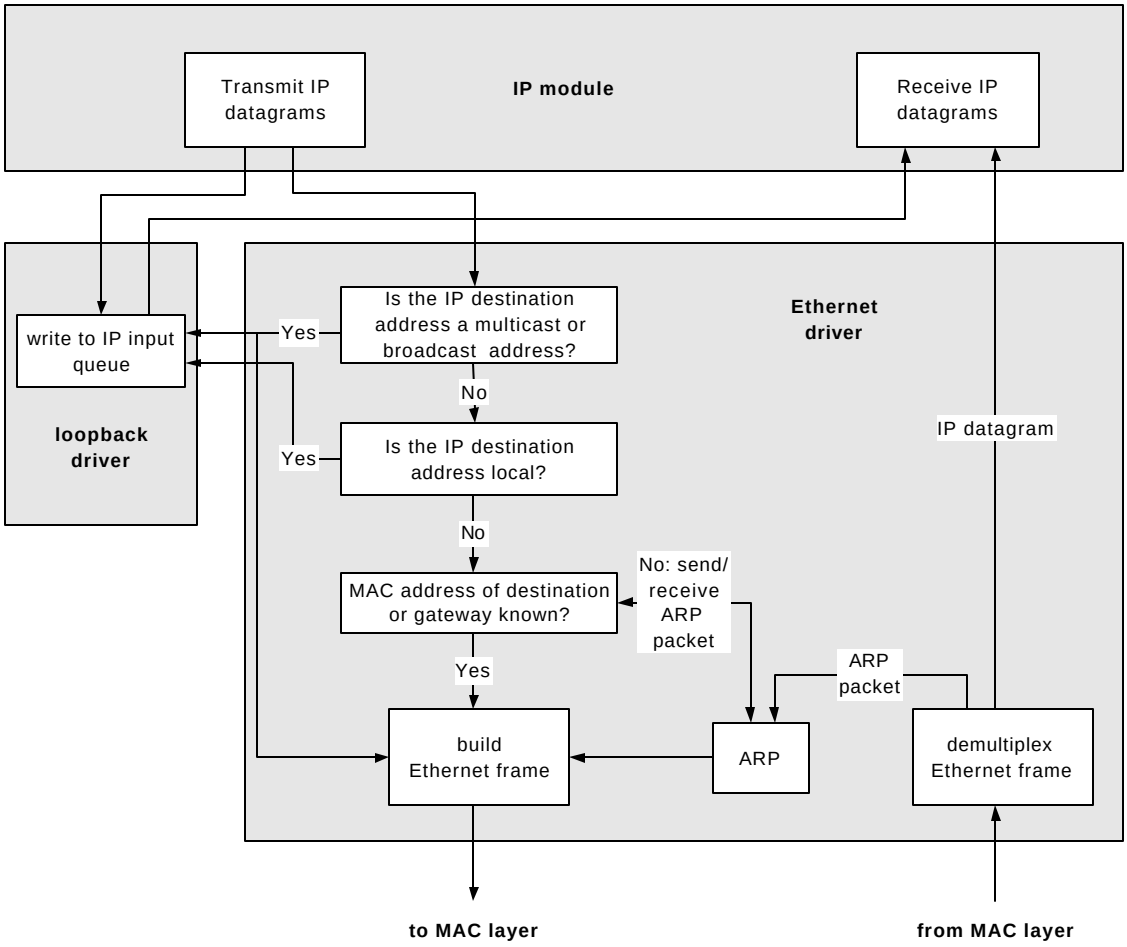


Figure 2.6. Ethernet and loopback device drivers (from W. Stevens: TCP/IP Illustrated, Volume 1).

When a datagram is transmitted it can be processed by either, the loopback driver, or the Ethernet driver, depending on the IP address of the outgoing datagram. If the IP address is not the local loopback address, the datagram is filtered one more time to determine if the address is a broadcast or multicast address, in which case it is sent directly to the Ethernet network with the appropriate hardware address and is also placed on the device's input queue for local processing. If the IP address is none of the above, the driver checks the ARP cache to determine if the hardware address is available. If the ARP cache does not contain the destination IP address, the driver calls the ARP module to send out an ARP query. Once the ARP response comes back, it is processed and the outgoing packet is transmitted. All incoming packets are filtered to separate ARP frames from IP datagrams. The former are used to update the ARP cache and the latter are passed on to the input queue.

2. The Address Resolution Protocol (ARP)

Whenever a system transmits an IP datagram over an Ethernet network, it encapsulates the IP datagram in an Ethernet frame. To send the frame, the sender must include the MAC address of the receiver in the frame. The sender of the frame has the IP address of the receiver, either the destination IP address or the IP address of a router, but may not know the MAC address. This is where the Address Resolution Protocol (ARP) comes in. ARP performs a lookup service that finds a MAC address for a given IP address. The basic operation of ARP is simple. A system that needs a MAC address for a given IP address broadcasts a query which contains the IP address to all systems on the network. If a system receives the query and the IP address in the message matches its own IP address, it sends its MAC address to the sender of the query.

The address resolution service of ARP is not limited to MAC addresses and IP addresses. By design, ARP can be used for a variety of network layer addresses and data link layer addresses. In the context of ARP, the former are called protocol addresses and the latter are called hardware addresses. However, the most common use of ARP, and the only use of ARP in the Internet Lab, is concerned with finding a MAC address for a given IP address.

In principle, one can think of three methods to translate IP addresses and MAC addresses. A direct mapping, a table lookup, and a message based solution. ARP implements the third option. Let us briefly review why this choice is made.

In a direct mapping, the MAC address of a system is derived from its IP address. For example, *Argon*'s IP address, 128.143.137.144, could be directly mapped to the MAC address 00:00:80: 8F:89:90, that is, the first two bytes of the MAC address are set to zero, and the last 4 bytes are directly translated. This solution assumes that the MAC address of a network interface card can be set to an arbitrary value. However, as we discussed in Chapter 0, for MAC addresses, most network interface cards have preconfigured MAC addresses, which are permanently assigned during the manufacturing process. Therefore, for most Ethernet networks and most other LAN networks, a direct mapping is not viable.

Another solution to the address translation problem is a table lookup. Here, each system maintains a table with entries of the form (IP address, MAC address) for each system on the local area network. Then, whenever the host looks for an IP address, it simply performs a lookup in its table. The drawback of this solution is that every change to an IP address or a MAC address in the local area network requires that the table are updated on each system in the network, and keeping the tables current at all systems requires a significant amount of human intervention.

As already indicated, ARP uses a third solution, which is based on a query and reply messages. The system looking for a MAC address sends a query message and waits for a response. When ARP is used on local area networks, the query is broadcast to all systems on the local area network and all systems process the query. However, only the system whose IP address matches

the IP address in the query message responds by sending its own hardware address to the host that issues the query.

As an alternative to broadcasting queries, one could think of a solution where ARP messages are sent to a central server that keeps an updated copy of all (IP address, MAC address) entries. An argument against a server solution are that all systems must know the hardware address, thus requiring yet another network configuration parameter. Also this solution is sensitive to the failure of the server. However, in local networks that do not support broadcast messages at the data link layer, a server based solution to ARP is sensible.

2.1. Operations of ARP

Each system maintains a table, called the *ARP cache*, that temporarily stores the results from previous address resolutions. The ARP cache contains entries of the form (IP address, MAC address). The role of the ARP cache is to reduce the amount of ARP messages being sent. If the cache has an entry for the receiver of the frame, the address resolution is completed. The ARP cache, however, is not a static table. In fact, each ARP entry is associated with a timer, and is deleted when the timer expires. The timers are usually set to a value between 1 and 10 minutes. The timer for an entry in the ARP cache is reset, each time the system looks up this entry. In some implementations of ARP, systems send out a queries for an IP address for which it already has an entry in its cache to verify that the mapping is still valid.

When a host holds an IP datagram and the destination address is not found in the ARP cache, the host issues an ARP Request, and holds the IP datagram in a queue until the MAC address has been found. The ARP Request is sent to all systems on the network using a broadcast message. In Ethernet networks, a frame is broadcast when the destination MAC address is set to the broadcast address ff:ff:ff:ff:ff:ff. A broadcast frame is received and processed by all hosts on the network. If a system receives the ARP request and the IP address in the message matches its own IP address, it issues an ARP Reply message to the sender of the query. The reply message is not a broadcast message.

In Figure 2.7(a), we illustrate the ARP Request from the example in Chapter 0, where *Argon* tries to find the MAC address of *Router137*. The ARP Request contains the IP address needs to be resolved, and the IP address of the sender. When *Router137* processes the request, it notices that the request refers to its own IP address, *Router137* generates an ARP Reply and transmits its IP address and MAC address to *Argon*. The reply message is not broadcast to all hosts, but is sent directly to *Argon*. This is shown in Figure 2.7(b).

When a system sends an ARP Request for a not existing IP address, there will not be a response. In this case, the ARP Request is repeated. The waiting time until the next ARP Request is sent depends on the implementation of ARP. In Linux, one ARP Request is sent every second, until a response is received. In other implementations, the sender of an ARP

request increases the time interval between transmission of ARP Request, and may eventually give up.

ARP Request transmissions are used by all systems on the network to update their ARP caches. Every host that sees an ARP Request verifies its ARP cache if there is an entry for the sender of the ARP Request. If such an entry exists, it updates the MAC address with the address in the ARP Request. Since ARP Requests are broadcast message, these updates are made by all systems each time an ARP Request is transmitted on the network. This feature is exploited in a concept that is called *gratuitous ARP*. A gratuitous ARP is an ARP Request sent by a host for its own its own IP address. Gratuitous ARP messages can be used in several scenarios. For example, when the IP address of a host is changed, a gratuitous ARP can be used to verify if the new IP address is already in use. When some other host responds to the gratuitous ARP, the IP address is already in use. Also, if the MAC address of a host has changed, a gratuitous ARP by this systems forces an update of all ARP caches on the network. Gratuitous ARP messages are sometimes used by backup servers. When the main server fails, the backup server takes over the IP address of the failed server by issuing a gratuitous ARP.

ARP is quite vulnerable to attacks by malicious users. As shown in the scenario with the backup server, a gratuitous ARP packet can be used to redirect traffic directed to a certain IP address to any machine on the network, thereby effectively hijacking an IP address. Another vulnerability is that broadcasting ARP Replies with invalid MAC addresses inserts incorrect entries in ARP caches.

ARP works well in networks where the mapping of IP address and MAC addresses of system changes frequently. Such a change can occur for a number of reasons. When the network interface card of a host is replaced, the MAC address changes even though the IP address does not change. When a host is moved to a different subnetwork, the IP address of the host changes, but the MAC address remains the same. As we will see in Chapter 8, networks may assign IP addresses on an on-demand basis to hosts (via the Dynamic Host Configuration Protocol or DHCP). In this case, the IP address of a host may change frequently. In all of these situations, ARP is designed to automatically adapt to the new mappings without requiring changes to the network configuration.

Sometimes a host needs to perform an address resolution service as provided by ARP, but in the opposite direction. That is, a host may want to find the IP address that corresponds to a given MAC address. One such protocol is the Reverse Address Resolution Protocol (RARP). RARP is intended for hosts that have a MAC address and want to be assigned an IP address. This scenario applies to hosts that do not have a disk to store their IP configuration, or to networks were IP addresses are dynamically assigned by a server when a system is started. As in ARP, a RARP Request is broadcast and the RARP Reply is sent as a unicast message to the querying hosts. RARP assumes that there is always a system on the network that responds to RARP requests. Today, even though dynamic assignment of IP addresses is quite common, RARP is

rarely used. Dynamic assignment of IP addresses is generally performed by other protocols, such as BOOTP and DHCP.

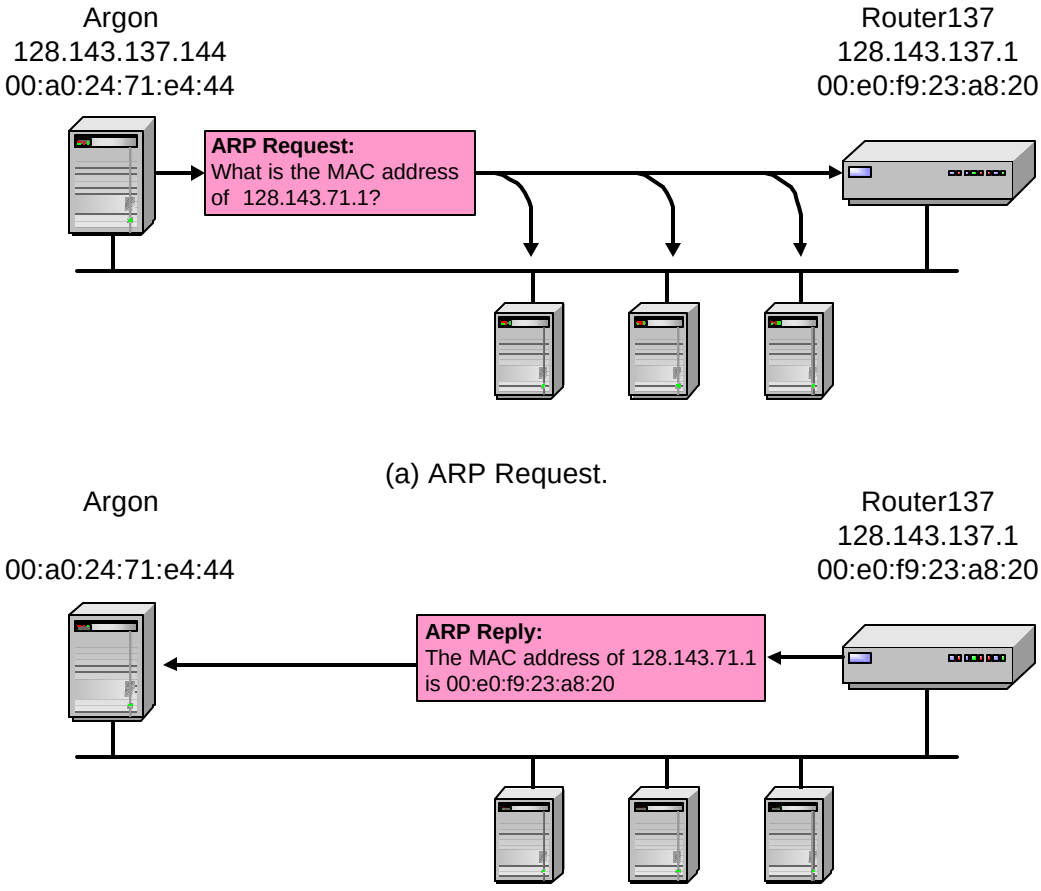


Figure 2.7. ARP Request and Reply.

2.2. ARP Packet Formats

ARP messages are carried in the payload of Ethernet frames. In Figure 2.8, we show the encapsulation of an ARP message in an Ethernet frame. In ARP messages, the type field of the Ethernet frame is set to 0x8060. As we already indicated, ARP can be used to map addresses for a variety of data link protocols and network layer protocols, not only for IEEE 802 MAC addresses and IP addresses. The length of an ARP message is determined by the size of these addresses, ARP packets can have a different length. With 48-bit MAC addresses and 32-bit IP addresses, ARP Request and ARP Reply messages have a length of 28 bytes.

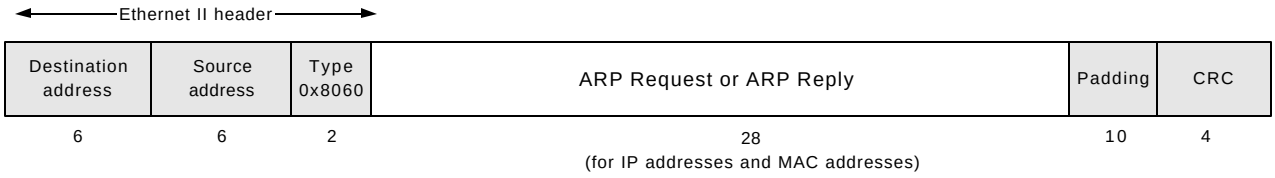


Figure 2.8. Encapsulation of ARP messages in an Ethernet frame.

Hardware type (2 bytes)		Protocol type (2 bytes)
Hardware address length (1 byte)	Protocol address length (1 byte)	Operation code (2 bytes)
Source hardware address*		
Source protocol address*		
Target hardware address*		
Target protocol address*		

* Note: The length of the address fields is determined by the corresponding address length fields

Figure 2.9. Format of ARP packets.

We show the format of an ARP packet in Figure 2.9. The first field, called hardware type, identifies the data link layer protocol to be used, and the second field, called protocol type, identifies the network layer protocol that request the hardware address. We already pointed out that, in principle, ARP can perform address resolution for many data link layer and network layer protocols. In practice, however, ARP is generally resolves IP addresses to 48-bit MAC addresses. For Ethernet, the hardware type is 0x0001 and for IEEE 802 addresses, the hardware type is set to 0x0006. The protocol type field is set to 0x8000 when ARP is used to resolve IP addresses. Note that the value 0x800 is the same value as the type field in Ethernet frames that indicate an IP datagram payload. The operation code is set to 0x0001 for ARP Requests and 0x0002 for ARP Replies. There are several other operation codes defined for ARP, but they do not play a role in the Internet Lab.

The fields for the hardware address length and protocol address length specify the number of bytes of the address types. Since we have 48-bit long MAC addresses the hardware address length is set to six, and for IP addresses the protocol address length is set to four.

The next four fields contain the hardware address and the network address of the sender and the intended receiver of the ARP packet. The former is referred to as the source and the latter is referred to as the target. In the ARP Request in Figure 2.7(a), *Argon* is the source and *Router137* is the target, and the fields in the ARP Request are set as follows:

ARP Request from Argon:

Source hardware address:	`00:a0:24:71:e4:44`
Source protocol address:	`128.143.137.144`
Target hardware address:	`00:00:00:00:00:00`
Target protocol address:	`128.143.137.1`

The target hardware address is set to zero since this is the address that *Argon*, the sender of the ARP Request, is looking for. (In some ARP implementations, including Linux implementations of ARP, one observes that hosts periodically send ARP Requests for IP addresses listed in the ARP cache, even if no IP datagram is sent to these IP addresses. In these requests, the requesting host includes the hardware address of the target from the ARP cache in the target hardware address field of the ARP request.). When *Router137* sends its ARP Reply to *Argon*, as shown in Figure 2.7(a), *Router137* is the source and *Argon* is the target, and the addresses in the ARP Reply packet are set as follows:

ARP Reply from Router137:

Source hardware address:	`00:e0:f9:23:a8:20`
Source protocol address:	`128.143.137.1`
Target hardware address:	`00:a0:24:71:e4:44`
Target protocol address:	`128.143.137.144`

The MAC address needed by *Argon* is contained in the source hardware address. Note that inserting the source addresses in the ARP Reply is superfluous, since they do not play any role in the address resolution.

Finally, in a gratuitous ARP Request transmitted by *Argon*, the addresses in the source and the target fields are identical.

Gratuitous ARP Request from Argon:

Source hardware address:	`00:a0:24:71:e4:44`
Source protocol address:	`128.143.137.144`
Target hardware address:	`00:a0:24:71:e4:44`
Target protocol address:	`128.143.137.144`

2.3. Proxy ARP

Proxy ARP is a configuration option for IP routers, where an IP router responds to ARP Request that arrive from one of its connected networks for a host that is on another of its connected networks. Without Proxy ARP enabled, an ARP Request for a host on a different network is unsuccessful, since routers do not forward ARP packets to another network.

A scenario where Proxy ARP is extremely useful is shown in Figure 2.10. The depicted network configuration is almost identical to the configuration in Figure 0.4, but with one subtle difference: The network interface of *Argon* and the left interface of *Router137* are configured with a 16-bit extended network prefix, or equivalently, with subnetmasks set to 255.255.0.0.

Recalling from Chapter 0, with this network configuration, when *Argon* with IP address 128.143.137.144 sends an IP datagram to *Neon* with IP address 128.143.71.21, the two address match on the bits that correspond to the extended network prefix. Therefore, *Argon* will attempt to deliver the IP datagram directly, and will not forward the IP datagram to Router 137.

Without Proxy ARP, *Argon* sends an ARP Request and wait for a response. However, since *Neon* is behind the router, it does not receive the ARP Request, and the address is not resolved. Now suppose that Proxy ARP is enabled on *Router137*. Then, when *Router137* receives the ARP Request from *Argon*, it checks if the IP address 128.143.71.21 can be reached from one of its connected interfaces. The router determines that 128.143.71.21 is directly reachable via its interface 128.143.71.1. In this case, Router 137 issues an ARP Reply on behalf of *Neon*. In the ARP Reply the source network address is set to 128.143.71.21 and the source hardware address to 00:e0:f9:23:a8:20. This indicates that "IP address 128.143.71.21 has MAC address 00:e0:f9:23:a8:20". In other words, *Router137* sends an ARP Reply to *Argon* as if it was *Neon*. After *Argon* receives the ARP Reply, it sends a frame with IP destination address set to 128.143.71.21 and MAC destination address set to 00:e0:f9:23:a8:20. Note that this is the same frame that *Argon* sent in Figure 0.5, where *Argon* has a 24-bit extended network prefix. When *Router137* receives the frame, it forwards the IP datagram to subnetwork 128.143.71/24. Thus, the IP datagram is correctly delivered to *Neon*, and passed through a router, even though *Argon* believes that it has delivered the IP datagram directly.

As a final note, when *Neon* sends an IP datagram to *Argon*, it determines that *Neon* and *Argon* are on different subnetworks and it sends the datagram to the router without relying on Proxy ARP.

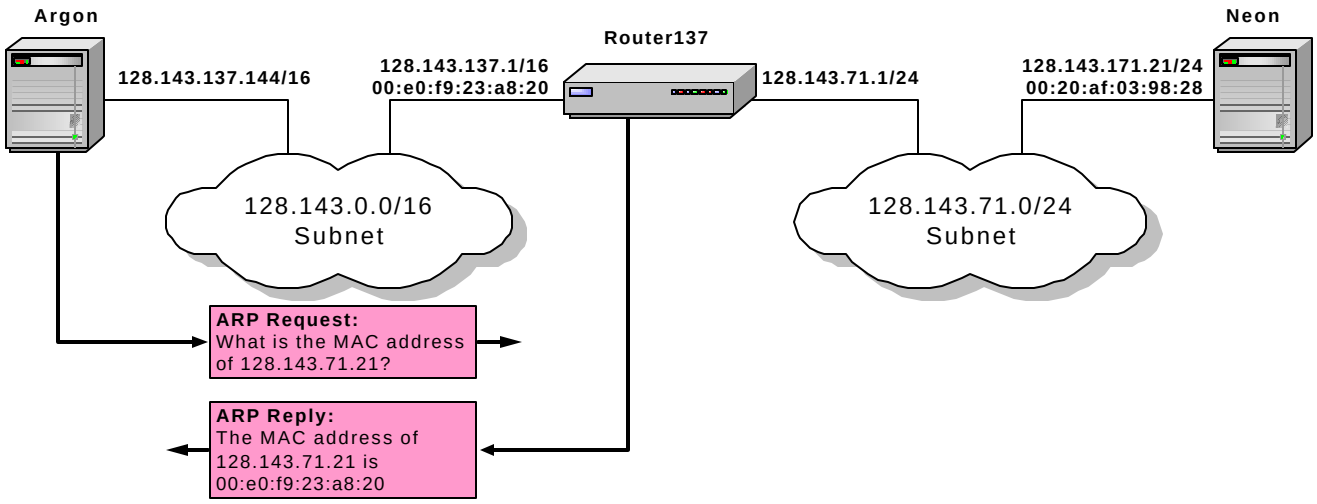


Figure 2.10. Proxy ARP.

3. The Internet Protocol (IP)

The Internet Protocol (IP) carries IP datagrams from a source to a destination across a path of routers. As shown in Figure 2.11, IP is the highest layer protocol that is running on both hosts and routers. At the sending host, IP receives a protocol data unit from a transport layer protocol, TCP or UDP, encapsulates that data in an IP datagram, and then sends the IP datagram to the destination host or to an IP router. For the delivery of an IP datagram, IP requests the services of a data link layer protocol. When an IP router receives an IP datagram, it forwards the datagram to another router or to the destination host. At the destination host, the IP layer demultiplexes an IP datagram and passes the payload of the datagram to a transport layer protocol.

IP is the only network layer protocol that performs data transport in the Internet. For this reason, IP has been called the *waist of the hourglass of the TCP/IP protocol suite.* As illustrated in Figure 2.12, there is a variety of protocols above IP, and a variety of data link layer protocols below IP, but there is no alternative to IP at the network layer. A single network layer protocol makes it easy to ensure interoperability between different types of hosts and routers. As long as a host or router speaks the language of IP, it can communicate with all other systems on the Internet. A drawback of having no alternative to IP at the network layer is that all Internet applications, however different their needs, receive the same one-size-fits-all service of IP.

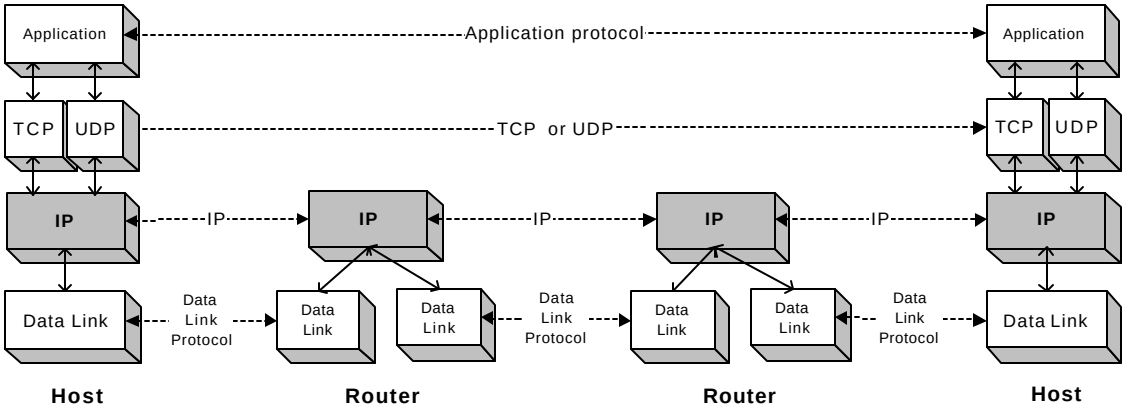


Figure 2.11. Protocols involved in data exchange between two hosts.

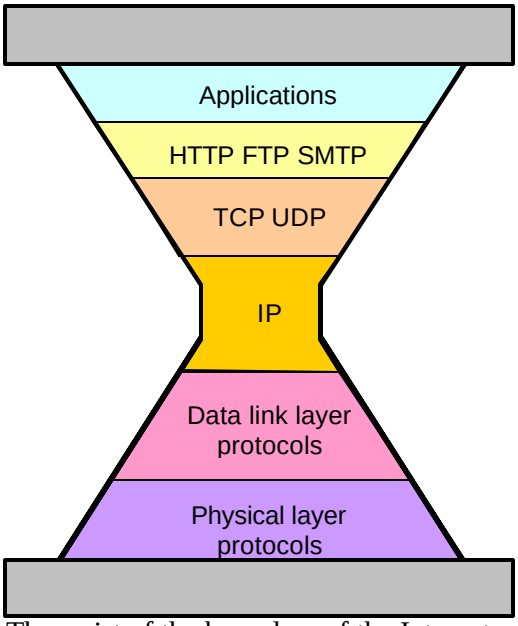


Figure 2.12. The waist of the hourglass of the Internet protocol suite.

The current version of the Internet Protocol is IP version 4 (IPv4) which is in use since 20 years, without incurring any significant revision. In the early 1990s, motivated by a growing concern about the rapid decrease of availalbe IP addresses, the IETF initiated a process to define a new version of IP. The result of this process was the Internet Protocol version 6 (IPv6).[2] [3] Today,

[2] RFC 1883.

[3] RFC 2460.

IPv6 implementations are available on most computers and commercial routers, but IPv6 does not yet play a significant role in operational IP networks. Therefore, in the Internet Lab, we only work with IP version 4.

As a note on the version numbers of IP, the version numbers 1, 2, and 3, were allocated for prototype implementations of IPv4. Version number 5 was allocated for an experimental connection-oriented protocol for the Internet, called Stream Protocol (ST).

3.1. The IP Service

IP provides a very minimal delivery service. IP does not ensure that a transmitted IP datagram actually reaches its destination host. If an IP datagram gets dropped, due to a transmission error, or a buffer overflow at a router, IP does not attempt to recover the datagram. This is called an *unreliable service*.

Also, IP does not recognize sequences of packets that are sent between a sending and receiving application. In IP, each IP datagram is handled completely independent from any other IP datagram. Even if an IP router processes two back-to-back datagrams from the same source to the same destination, the router does not realize that the two IP datagrams belong together. This is called a *connectionless service.* A consequence of a connectionless service is that two back-to-back IP datagrams that are sent between two hosts may traverse the network on a different route and may arrive in a different order than they were sent. In other words, IP does not guarantee an in-sequence delivery of datagrams. Another consequence of a connectionless service is that each IP datagram must contain the IP address of the destination hosts.

By default, IP does not give service guarantees in terms of a bounds on delay, losses, and the data rate. This is called a *best-effort service.* The term best effort points out that IP will make a good faith effort to give an IP datagram the best possible service, but without committing itself to specific guarantees. In IP, a host can indicate that it wishes to receive a service with low delay, high bandwidth or bw loss service for its transmitted traffic. This is done by setting appropriate bits in the IP header. In addition, the IP header can be used to identify that an IP datagram belongs to a certain traffic class, and IP routers can give different priorities to traffic from different classes, resulting in a differential treatment of IP datagrams.

IP has three different delivery modes: unicast, multicast, and broadcast. In a unicast mode, an IP datagram is sent to a single receiver, as identified in the destination IP address. The vast majority of IP datagrams are transmitted in this mode. In a multicast mode, an IP datagram is sent to a set of receivers, called a multicast group. The multicast group is identified by a multicast address in the destination IP address in the IP header. If a host wants to receive packets sent to a multicast address, it *joins* the multicast group for this address. Multicast will be covered in detail in Chapter 7. With broadcast, an IP datagram is transmitted to all hosts with respect to a given scope. Broadcast addresses can be recognized by having all bits in the host portion of an IP address set to 1. Since hosts that receive a broadcast IP datagrams must process

the datagram, it is important to limit the amount of broadcast traffic. Broadcast addresses are often employed to stage Denial of Service (DDoS) attacks in the Internet. Therefore, IP routers often disable forwarding of IP datagrams with broadcast addresses.

3.2. IP Datagram Format

Each IP datagram has a header of at least 20 bytes and at most 60 bytes. The maximum size of an IP datagram, including the header, is 65,535 bytes, however, the actual size is typically much smaller, due to restrictions imposed by the data link layer protocol. For example, with Ethernet at the data link layer, IP datagrams cannot exceed a size of 1500 bytes.

The format of an IP datagram is shown in Figure 2.13. When transmitted on a link, the order of transmissions of an IP datagram is row-by-row, and in each row, from left to right.

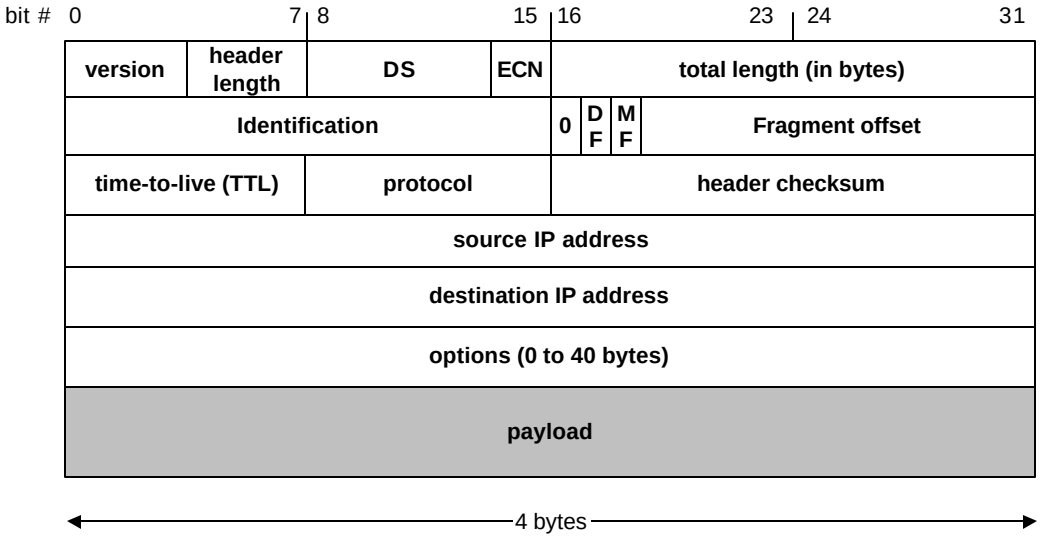


Figure 2.13. The IP Datagram format.

The first field of the header is the *version number* of IP. For IPv4 datagrams, the version field is set to value 4. The next field is the *header length* field, which states the length of the IP header in multiples of four bytes. The header length field is four bits long. Therefore, the maximum size of the IP header is 15*4=60 bytes. The header length field is needed because of the variable length IP header. Without the field IP could not detect the boundary between the IP header and the beginning of the payload.

The second byte in the IP header contains the Differentiated Services (DS) field and the Explicit Congestion Notification (ECN) fields. Until recently, these fields were known as the Type-of-Service (TOS) field. In Figure 2.14 we show the old and the new interpretation of this field.

Figure 2.14(a) shows the format of the TOS field. The first three bits of the TOS field denote the precedence level of an IP datagram. A higher precedence level indicates that the IP datagram should be treated with a higher priority. The next four bits indicate the desired type of service: low delay, high throughput, high reliability, or low cost. Only one of these bits can be set in an IP header. The last two bits are always set to zero. For many Internet application there are recommended values how to set the value of the TOS byte. For example, for Telnet traffic, the low delay bit should be turned on, and for FTP traffic, the high throughput bit should be turned on during a file transfer. In practice, however, most routers ignored the TOS field and gave all IP datagrams the same (best effort) service.

As part of an effort to enhance the ability of the Internet to give differential service to different classes of IP traffic, the TOS field in the IP header has been renamed as DS field. The format of the DS field is shown in Figure 2.14. The DS field is set by the sending host or by the first IP router, and can be modified when an IP datagram crosses a network boundary, e.g., between a regional and a backbone network. The marking of the DS field is interpreted as a code, called Differentiated Services Codepoint (DSCP) or, simply, codepoint. For certain codepoint values there are a set of rules, called Per-Hop-Behaviors (PHBs), which define how an IP datagram that is marked with a given codepoint should be handled by IP routers. For example, the codepoint 101110 is associated with a PHB that provides a rate guarantee to IP datagrams with that marking. All IP datagrams with the same codepoint are treated as a group and receive the same service. The DS field is, to a limited degree, backward compatible to the TOS field. For example, if the first three bits of the DSCP are set to 000 then bits 3 through 5 are associated with a PHB that gives, respectively, low delay, high throughput, and high reliability. In other words, the codepoint 000100 defines a PHB for a low delay service. The codepoint 000000 indicates the default best effort service. At present only a few codepoints and corresponding PHBs have been defined.

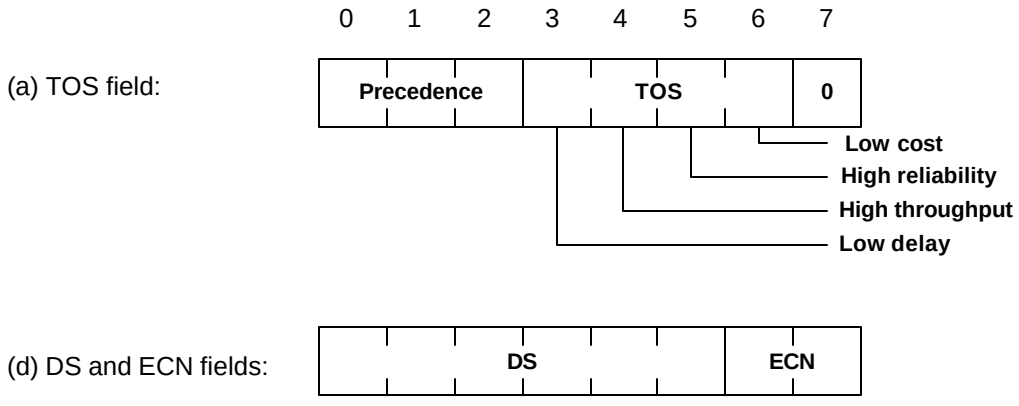


Figure 2.14. Interpretation of the TOS field and DS field.

Following the DSCP, is the Explicit Congestion Notification (ECN) field[4]. This two bit long field is used in a recently proposed algorithm where an overloaded router can mark IP headers when it is congested. The marking in the headers can be used by higher layer protocols such as TCP to reduce the rate of data transmission. The sender of an IP datagram sets the ECN field to 00 if it does not run an ECN algorithm, and to 01 or 10, otherwise. A backlogged router can set the ECN field to 11, when it is experiencing congestion. If the destination receives an IP datagram where the ECN field is set to 11, it notifies the sender in a separate message that there is a congested router on the path. When the sender receives such a notification, it reduces its rate of data transmission.

Continuing with the header fields in Figure 2.13, the *total length* field identifies the total number of bytes of the IP datagram, including IP header and payload. This field has a length of 16 bits. Therefore, the length of an IP datagram is limited to 2^{16}-1 = 65536 bytes. One may wonder why the total length field is needed. Since data link layer protocols pass the entire payload of a frame to IP as a single chunk of data, the end of the payload of a frame should be obvious to IP. However, it may happen that the IP datagram is shorter than the data link layer payload of a frame. For example, Ethernet needs to have a minimum payload of 46 bytes, and if an IP datagram is shorter than that, Ethernet pads the frame to reach the minimum length. In these situations, the total length field is needed, otherwise IP cannot recognize and discard bytes that have been padded.

We mentioned that most data link layer protocols impose a limit on the maximum size of an IP datagram, which is often much smaller than the maximum of 65536 bytes. This limit, called the Maximum Transmission Unit (MTU), is specified during the configuration of a network interface. If an IP datagram must be transmitted over an interface, but its size is larger than the MTU for that interface, the IP datagram is split into multiple IP datagrams with a smaller payload. This process is called fragmentation. The process of fragmentation is relatively complex, and requires several fields in the IP header. In fact, all fields in the second row of the IP header in Figure 2.13 are needed to support fragmentation. The *identification* field is a 16-bit long identifier for a datagram. The identifier is assigned by the sending host, and is incremented by one for each transmitted datagram. This field plays a role when a fragmented IP datagram is reassembled. The next three bits in the IP header contain bit flags. The first bit is always set to 0. The second bit is the *Don't fragment (DF)* bit. If this bit is set by the sending host, the IP datagram will not be fragmented, but simply discarded, if its size exceeds that of the MTU. The third bit is the *more fragments (MF)* bit. This bit indicates if the IP datagram is not the last in a sequence of fragments of a previously fragmented IP datagram. The 13-bit *fragmentation offset* field is used to indicate the position of a fragment in the original datagram payload. The offset is given in multiples of 8 bytes. Fragmentation is discussed in more detail in the next subsection.

[4] RFC 3260

The *time-to-live* (TTL) field is used to limit the lifetime of an IP datagram, in situations when the routing tables of IP routers have become inconsistent and result in a routing loop. As an example of a routing loop, consider three routers A, B, and C, where A forwards a packet to B, B forwards a packet to C, and C forwards a packet to A. The TTL field does not prevent routing loops from occurring. It simply prevents IP datagrams from getting caught forever in routing loops. The sender of an IP datagram writes a number in the TTL field, and each router that processes the IP datagram decrements the TTL field by one. When the value of the TTL field reaches zero, a router drops the IP datagram. Routers decrement the TTL field before they process an IP datagram. Therefore, when an IP datagram with a TTL of one arrives at a router, the router decrements the TTL field, realizes that the TTL has reached zero, and then drops the datagram. The initial value of the TTL field that is set by the source should be large than the longest route in the Internet, otherwise, an IP datagram may be dropped before it has reached its destination. The initial TTL value should be set to 64 or higher.

The *protocol* field is the demultiplexing field in the IP header. It identifies the protocol data unit in the IP payload, that is, if it is an UDP datagram or a TCP segment. The protocol is identified by a protocol number, Table 2.1 lists protocol numbers of frequently used protocols. When the protocol field is set to value four, then the payload of the IP datagram is another IP datagram, complete with its own header. This is referred to as IP-in-IP encapsulation or IP tunneling.

1	ICMP, Internet Control Message Protocol.
2	IGMP, Internet Group Management Protocol.
4	IP-in-IP encapsulation.
6	TCP, Transmission Control Protocol.
17	UDP, User Datagram Protocol.
41	IPv6-over-IPv4
89	OSPF, Open Shortest Path First Routing Protocol.
103	PIM, Protocol Independent Multicast.

Table 2.1. Protocol field values in the IP header (selection).

The header *checksum* field in Figure 2.13 protects the integrity of the IP header. It contains a checksum that is computed over the IP header. The checksum does not cover the payload of the IP datagram. When IP detects an error in the header, it discards the datagram. To compute the checksum, the IP header is divided into a sequence of 16-bit sections, and the sections are added up. The checksum is the one's complement of the last 16 bits of the sum of the sections. The receiver of an IP datagram also divides the header into 16-bit sections, including the checksum

field, and adds the sections up. If the IP header does not contain an error, then the sum should be all ones on the last 16 bits. If the result contains a zero, the receiver assumes that there is an error and discards the IP datagram. Note that the header checksum must be recomputed at each router, since each router changes the TTL field in the IP header.

Following the IP checksum are the 32-bit source and destination IP addresses. The last fields in the IP header are variable length *options*. Options are rarely used, and are not supported by all hosts and routers. They are used for debugging and for special services, such as sending a timestamp or determining the MTU of a route. Interesting IP options are source route and record route. In the source route options, the sender of an IP datagram can specify the route of an IP datagram by writing the IP addresses of routers in the IP header. Up to nine IP addresses can be specified in the source route options. With the record route options, IP routers that handle a datagram with this option, add their IP address to the header. Due to restrictions of the total length of the IP header, not more than nine entries can be recorded.

3.3. IP Fragmentation

As described in the previous section, data link layers generally impose an upper bound on the length of a frame, and, thereby, on the length of an IP datagram that can be encapsulated in the frame. For each network interface, the MTU specifies the maximum length of an IP datagram that can be transmitted over a given data link layer protocol. Ethernet II and IEEE 802.3 networks have an MTU of 1500 bytes and 1492 bytes, respectively. Some protocols set the MTU to the largest datagram size of 65536 bytes. The MTU for any data link layer protocol must be at least 576 bytes.[5] If an IP datagram exceeds the MTU size, the IP datagram is fragmented into multiple IP datagrams, or, if the DF flag is set in the IP header, the IP datagram is discarded.

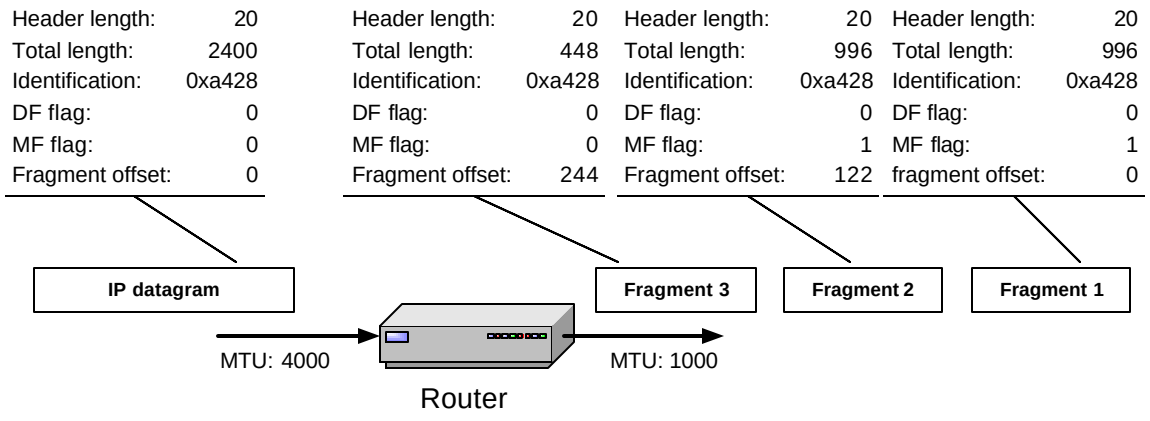
When an IP datagram is fragmented, its payload is split into multiple IP datagrams, each satisfying the limit imposed by the MTU. Each fragment is an independent IP datagram, and is routed in the network independently from the other fragments. Fragmentation can occur at the sending host or at an intermediate router. It is even possible that an IP datagram is fragmented multiple times, e.g., an IP datagram may be transmitted on a network with an MTU of 4000 bytes, then forwarded to an network with an MTU of 2000 bytes, and then to a network with an MTU of 1000 bytes. Fragments are reassembled only at the destination hosts. If a host receives fragments of a larger IP datagram it holds the fragments until the original IP datagram has been fully restored. Fragments do not have to be received in the correct order. The destination host can use the fragment offset field to place each fragment in the right position. IP assumes that a fragment is lost if no new fragment have been received for a timeout period, which is generally

[5] RFC 1009 (STD 3) and RFC 1121 (STD 4).

set to 60–120 seconds.[6] If such a timeout occurs, all fragments of the original datagram that have been received so far are discarded.

Fragmentation of IP datagrams involves the following fields in the IP header: *total length, identification, DF and MF flags,* and *fragment offset.* We illustrate the process of fragmenting an IP datagram in an example shown in Figure 2.15. The fields that are relevant during fragmentation are included in the figure. In the figure an IP datagram with a length of 2400 bytes is transmitted on a network with an MTU of 1000. We assume that the IP header of the datagram has the minimum size of 20 bytes. Since the DF flag is not set in the original IP datagram on the left of Figure 2.15, the IP datagram is now split into three fragments. All fragments are given the same identification as the original IP datagram. The destination host uses the identification field when reassembling the original IP datagram. The first and second IP datagram have the MF flag set, indicating to the destination host that there are more fragments to come. Without this flag, the receiver of fragments cannot determine if it has received the last fragment.

To determine the size of the fragments we recall that, since there are only 13 bits available for the fragment offset, the offset is given as a multiple of eight bytes. As a result, the first and second fragment have a size of 996 bytes (and not 1000 bytes). This number is chosen since 976 is the largest number smaller than 1000–20= 980 that is divisible by eight. The payload for the first and second fragments is 976 bytes long, with bytes 0 through 975 of the original IP payload in the first fragment, and bytes 976 through 1951 in the second fragment. The payload of the third fragment has the remaining 428 bytes, from byte 1952 through 2379. With these considerations, we can determine the values of the fragment offset, which are 0, 976 / 8 = 122, and 1952 / 8 = 244, respectively, for the first, second and third fragment.



[6] RFC 1122

Figure 2.15. Example of IP Fragmentation.

Even though IP fragmentation provides flexibility that can deal effectively with heterogeneity at the data link layer, and can hide this heterogeneity to the transport layer, it is has considerable drawbacks. For one, fragmentation involves significant processing overhead. Also, if a single fragment of an IP datagram is lost, the entire IP datagram needs to be retransmitted (by a transport protocol). To avoid fragmentation, TCP tries to set the maximum size of TCP segments to conform to the smallest MTU on the path, thereby avoiding fragmentation. Likewise, applications that send UDP datagrams often avoid fragmentation by limiting the size of UDP datagrams to 512 bytes, thereby, ensuring that the IP datagrams is smaller than the minimum MTU of 576 bytes.

4. The Internet Control Message Protocol (ICMP)[7]

The Internet Control Message Protocol (ICMP) is a helper protocol for IP that provides IP with a facility for reporting error conditions. For example, when an IP router discards an IP datagram, the router sends an ICMP message to the source of the datagram which explains why the datagram was dropped. ICMP also provides the capability to issue queries to hosts for diagnosing network conditions. The ping program discussed in Chapter 0 Section 3.3 is an example of such a query.

Since ICMP error messages are triggered by certain events, there is a danger that a network gets flooded with ICMP messages. To limit the number of ICMP messages, only one ICMP messages is issued for the same event, and ICMP error messages are never generated in response to a ICMP error message, a broadcast or multicast datagram, and to more than one fragment of an IP datagram.

4.1. ICMP Message Formats

An ICMP message is encapsulated in the payload of an IP datagram as shown in Figure 2.16.

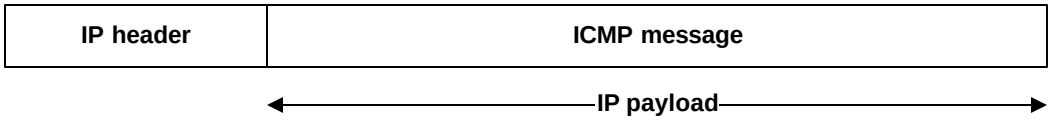


Figure 2.16. Encapsulation of an ICMP message.

[7] RFC 792.

The format of an ICMP message is shown in Figure 2.17. Each ICMP message is at least 8 bytes long. We refer to these 8 bytes as the ICMP header. The first 4 bytes contain the mandatory fields *type*, *code* and *checksum*. The type and code fields identify the particular ICMP message. There are more than thirty different ICMP message types, and some types are further qualified by a code field. The type of an ICMP message determines if there are additional fields following the ICMP header. The checksum is 16 bits long and is computed using the same algorithm as for the IP header checksum. Different from the IP header, the checksum is computed for the entire ICMP message. Following the checksum may be additional fields, that are defined for certain type fields. If there are no additional fields following the checksum, then the ICMP field has four bytes of zeros.

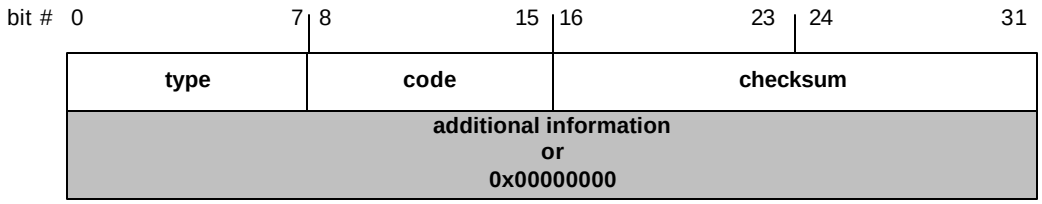


Figure 2.17. ICMP header format.

4.2. ICMP query messages

ICMP messages can be classified in two groups: ICMP queries and ICMP error messages. ICMP queries consist of a pair of message, as illustrated in Figure 2.18. The first message is an ICMP Request that is sent to an IP address, and the second message is an ICMP Reply that is sent in response to the request message. The transmission of a request message is usually triggered by a user-level command, such as the *ping* command. When the reply message arrives at the host that issued the request, ICMP delivers information to the user-level command.

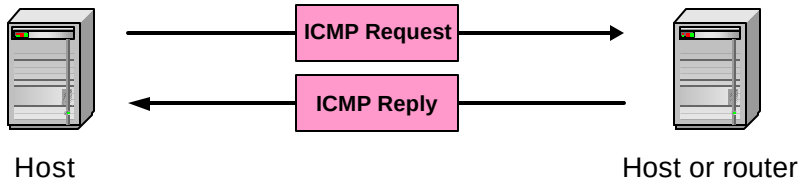


Figure 2.18. ICMP query.

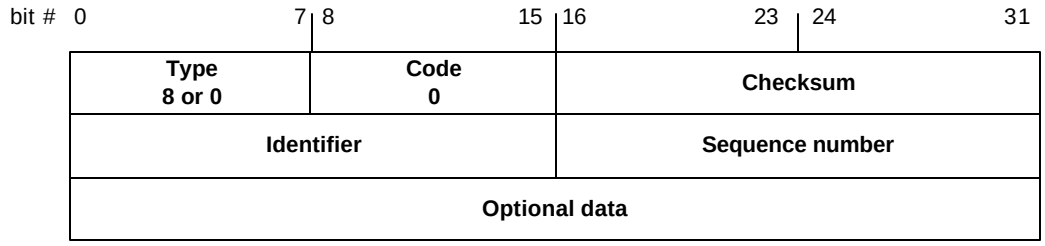


Figure 2.19. Format of ICMP Echo Request and Reply messages.

ICMP Queries			
Type	Code	Description	
0 8	0 0	Echo Reply Echo Request	Tests if an IP address is reachable on the Internet
13 14	0 0	Timestamp Request Timestamp Reply	Used to synchronize clocks and determine the roundtrip time between systems
17 18	0 0	Addressmask Request Addressmask Reply	Queries the subnetmask of a system
10 9	0 0	Router Solicitation Router Advertisement	Used for router discovery. Routers periodically send advertisements as multicast messages. Hosts may send solicitations for advertisements
37 28	0 0	Domain Name Request Domain Name Reply	Queries the hostname from a system.

Table 2.2. ICMP Query Messages.

The different available ICMP queries are shown in Table 2.2.[8] In Figure 2.19, we show the format of an ICMP Echo Request and Reply message. Other ICMP query messages have a similar format. All query messages have a 16-bit identifier field and a 16-bit sequence number field that follows the first four bytes. These fields are used by the sender of a query to match a request with a reply. In Linux systems, the identifier field is set to the process identifier of the process that generated the reply. When a host or router sends a reply, it sets the identifier and sequence number fields to the same values as in the corresponding request.

In the exercises of the Internet Lab, we will mostly see ICMP Echo Request and Echo Reply messages, which are used to test if a host is reachable at the IP layer. A receiver of an ICMP Echo Request immediately responds with an ICMP Echo Reply.

The only other ICMP query that we will encounter in the Internet Lab are ICMP Router Solicitation and ICMP Router Advertisement messages. These messages make up what is

[8] RFC 792, RFC 950, RFC 1256, RFC 1393, RFC 1788.

sometimes referred to as the Router Discovery Protocol[9], and provide hosts and routers a mechanisms to announce and learn the presence of IP routers. A host sends ICMP Router Solicitation messages to a multicast or broadcast address. When an IP router that is on the same subnetwork receives the solicitation, it responds by sending an ICMP Router Advertisement messages. The reply message contains the IP address of the router, and possibly the IP addresses of other routers on the same subnetwork. In addition, IP routers periodically send, about every five to ten minutes, send an advertisements without having received a solicitation. When a host receives a router advertisement it updates its routing table.

4.3. ICMP Error messages

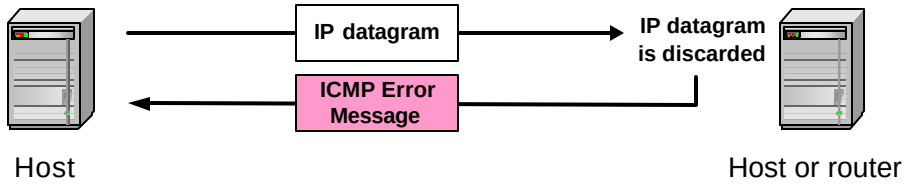


Figure 2.20. ICMP Error message

ICMP error messages report an error condition to the sender of an IP datagram. As shown in Figure 2.20, the transmission of an ICMP error message is usually triggered when an IP datagram is discarded. Often, when a host receives an ICMP error message, the error message is reported to the application program that sent the IP datagram that was discarded. In this way, a user program is made aware of the error. It is import to not that ICMP is simply a reporting mechanism, and does not take any correcting actions.

[9] RFC 1256

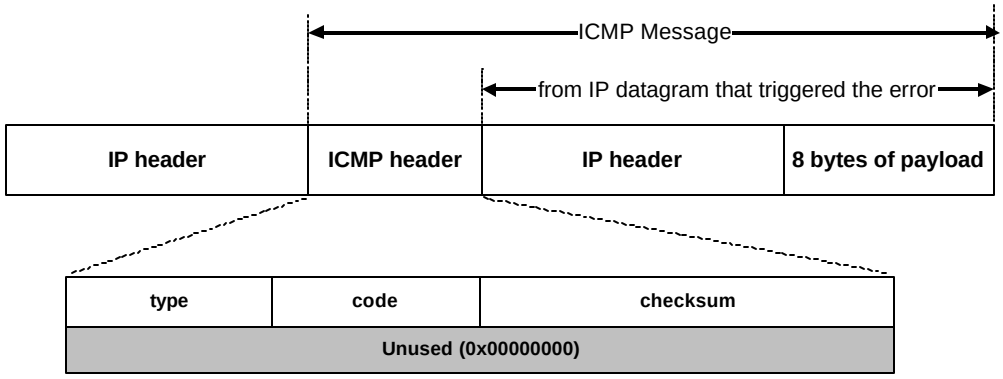


Figure 2.21. Format of an ICMP error message.

The general format of an ICMP error message is shown in Figure 2.21. The ICMP message consists of an ICMP header which contains the type, code and checksum fields, followed by four bytes of zeros. Following the ICMP header is the IP header and the first eight bytes of the IP payload from the IP datagram that triggered the error. The first 8 bytes of the IP payload are significant because they contain the source port number and the destination port number in TCP segments and UDP datagrams. With this information a host that receives an ICMP error message can identify the user-level process that needs to receive the error notification.

The different types of ICMP error message are listed in Table 2.3. The table is not comprehensive but contains the most frequently seen error messages.[10] In the Internet Lab, we will encounter ICMP error messages of Type 3 (Destination unreachable), Type 5 (Redirect), and Type 11 (Time Exceeded). ICMP Redirect messages are discussed in more detail in Chapter 3. The ICMP Time Exceeded message is sent with Code 0 by an IP router when the TTL value of an IP datagram has reached zero, and with Code 1 by a host when there is a timeout for reassembling the fragments of an IP datagram.

ICMP Error Messages			
Type	Code	Description	
3	0–15	Destination unreachable	Notification that an IP datagram could not be forwarded and was dropped. The code field contains an explanation.

[10] RFC 792, RFC 1475, RFC 2521.

5	0–3	Redirect	Informs about an alternative route for the datagram and should result in a routing table update. The code field explains the reason for the route change.
11	0, 1	Time exceeded	Sent when the TTL field has reached zero (Code 0) or when there is a timeout for the reassembly of segments (Code 1)
12	0, 1	Parameter problem	Sent when the IP header is invalid (Code 0) or when an IP header option is missing (Code 1)

Table 2.3. ICMP Error Messages.

ICMP Destination Unreachable messages are generated by a host or by a router when they discard an IP datagram because they cannot deliver the datagram to the next host or router, or to an application program. The code field in the ICMP message explains why a datagram could not be delivered. The different values of the code field and their interpretation are listed in Table 2.4. Some ICMP Destination Unreachable messages indicate that the network configuration is faulty. For example, when the network address not reachable, the routing tables are not properly configured. The error *host unreachable* indicates that the routing tables are properly configured, but the router on the destination network does not receive an ARP Reply for the destination IP address. This indicates that the host IP address of the host is incorrect, or the host is currently not operational.

The error *port unreachable* is raised by the destination host if the transport layer protocol cannot deliver the data to an application program. This error occurs when a client program attempts to connect to a server program that does not exist. This error message is not needed when the transport protocol is TCP, since a TCP server responds to connection requests to non-existing server programs by closing the connection. However, when the transport protocol is UDP, the ICMP port unreachable error is needed, otherwise, a client of an application program that uses UDP cannot determine if the server is down, unless a port unreachable message is sent. Therefore, this error is seen only when trying to connect to a UDP server program, such as a TFTP server.

Code	Description	Reason for Sending
0	Network Unreachable	No routing table entry is available for the destination network.
1	Host Unreachable	Destination host should be directly reachable on one of the interfaces, but does not respond to ARP Requests.
2	Protocol Unreachable	The protocol in the protocol field of the IP header is not

		supported at the destination.
3	Port Unreachable	The transport protocol at the destination host cannot pass the datagram to an application. Typically issued when an UDP datagram is sent to an destination port and no application program is attached to the port.
4	Fragmentation Needed and DF Bit Set	IP datagram must be fragmented, but the DF bit in the IP header is set.
5	Source Route Failed	IP header has source route option set, but the IP datagram cannot be forward to the next router in the list.
11	Network Unreachable For Type Of Service	No routing table entry is available for the destination network with the default TOS value or the TOS value requested in the IP header.
12	Host Unreachable For Type Of Service	The host is unreachable for IP datagrams with the given Type-Of-Service field.
13	Communication Administratively Prohibited	A router does not forward the IP datagram due to an administrative policy
14	Host Precedence Violation	Indicates that the requested precedence setting in the precedence bits of the TOS field of the IP header is not permitted for this IP datagram.
15	Precedence cutoff in effect	The precedence level in precedence bits of the TOS field of the IP header is too low.

Table 2.4. Codes in ICMP Destination Unreachable messages.

5. IP version 6

IPv6[11] is a revised version of the Internet Protocol. The designers of IPv6 did not introduce drastic changes, and were conservative with adding new features to the protocol. As its predecessor, IPv6 offers an connectionless unreliable best-effort delivery service for IP datagrams.

Other than expanding the size of the IP address from 32 bits to 128 bits, IPv6 introduces a number of changes. The IP header format has been simplified. Security mechanisms, such as

[11] RFC 2460

authentication and encryption of IP datagrams, have been integrated into the IPv6 design. IPv6 introduces a new type of addresses, called anycast addresses. Just like a multicast addresse, an anycast address designates a group of hosts. Different from multicast, where a packet is sent to all hosts in a group, a packet with an anycast destination address is sent to only one, generally the closest, member in the group.

There also a new version of the Internet Control Message Protocol, called ICMPv6. ICMPv6 streamlines the previous version of ICMP by eliminating several ICMP message types. ICMPv6 has been expanded to perform the functions of ARP and IGMP, and makes the two protocols obsolete.

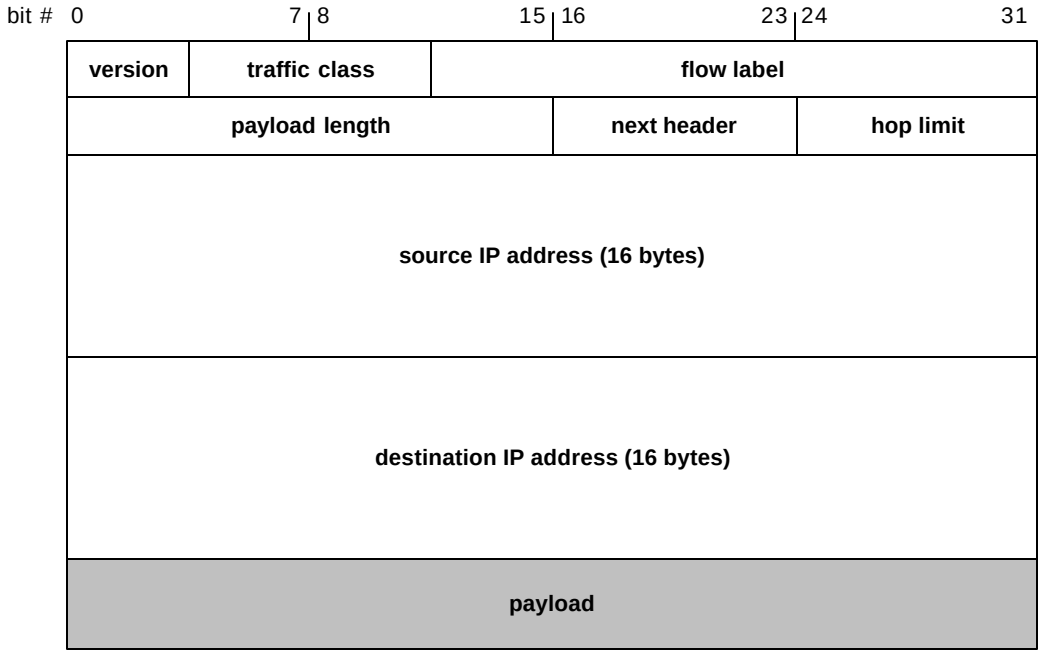


Figure 2.22. IPv6 header format.

The format of an IPv6 datagram is shown in Figure 2.22. In comparison to IPv4, the new IP header is much simplified. The IPv6 header has a fixed length of 40 bytes. It is possible to add an arbitrary number of extension headers to the IPv6 header.

The *version* field is, as can be expected, set to six. The eight-bit *traffic class* field plays a similar role as the DS (previously TOS) field, and is used to give differential service to traffic classes. The 20-bit long *flow label* is a new field which identifies a flow, defined to be a sequence of packets sent from a source to a destination for which the source requests special handling. The flow label can be used to provision service guarantees to the traffic from individual applications, e.g., give rate and delay guarantees to a live video transmission. The payload length contains the

total number of bytes of the IP datagram that follow the 40-byte IPv6 header. The *next header* field has two functions. First, if there is an extension header, the next header field identifies the type of the extension header. Second, if there is no extension header, the next header field is the demultiplexing field of the IPv6 header, and plays the same role as the protocol field in IPv4, even using the using the same values as the protocol field in IPv4 (see Table 2.1).

The hop limit field plays an identical role as the Time-to-live field in IPv4. The value is initialized by the source host, and decremented at each router. If the hop limit reaches zero the IP datagram is discarded. The last two fields of the fixed header are the 128 bit long source and destination addresses.

As we can see, some fields from the IPv4 header are missing in IPv6. With a fixed-length header, there is no need for a header length field. Also, IPv6 removes the ability for IP fragmentation. As a result, the identification and the fragment offset fields, as well as the flags from the IPv4 header have been removed. Finally, there is no header checksum. Since data link layer protocols generally provide a cyclic redundancy checksum (CRC), which is very effective in detecting bit errors, the designers of IPv6 felt that adding a checksum to the IP header does not provide significant added protection and eliminated the field altogether.

IPv6 replaces the IP header options in IPv4 with the elegant concept of extension headers. An example of an extension headers is a routing header that records the route of an IP datagram. Another example is an authentication header which can carry a cryptographic checksum for authentication purposes. IPv6 allows concatenating an arbitrary number of extension headers. In Figure 2.23, we show how headers are concatenated. Figure 2.23(a) shows a header without an extensions. Here, the next header field identifies the protocol type of the payload, TCP in this case. The datagram in Figure 2.23(b) has one extension header, and the datagram in Figure 2.23(c) has two extension headers.

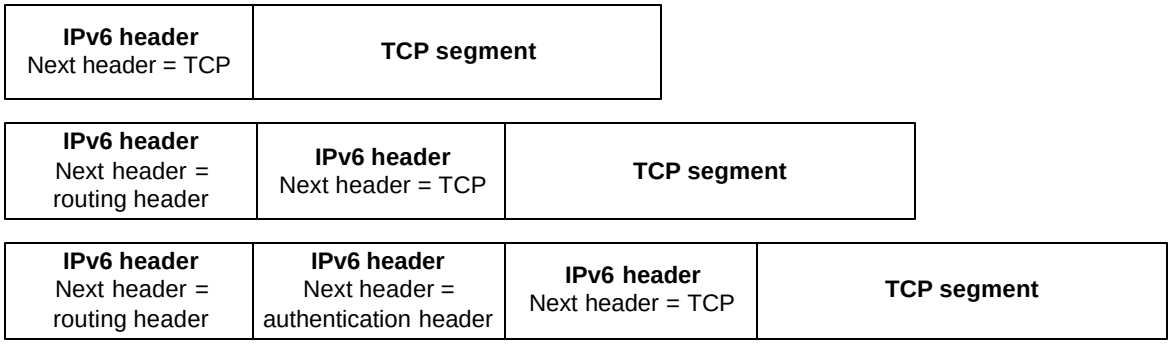


Figure 2.23. Extension headers in IPv6.

The transition of the Internet to the new version of IP is a tremendous challenge. Eventually, every host and every router on the Internet has to switch from IPv4 to IPv6. With several

hundred million hosts and routers, the transition to IPv6 is approached as a process rather than a single event. The design of IPv6 allows that both versions of the Internet Protocol can coexist during a transition period.

Today, it is difficult to predict if and when IPv6 will fully replace IPv4. The conversion to version 6 has been slower than anticipated. A possible explanation is that the increasing use of private networks, IP masquerading, and dynamic assignment of IPv4 addresses (see Chapter 8), have made the depletion of IP addresses a less urgent problem, thereby removing a main incentive for converting to IPv6.

6. Networking Tools and Utilities

In this section we discuss the software tools that are introduced in Lab 2 for configuring and displaying the network configuration of a Linux PC. We also discuss commandsto manipulate the ARP cache at a Linux PC. Finally, we discuss the use of filters in the *tcpdump* and *ethereal* commands.

6.1. The *netstat* command

The Linux command **netstat** displays information on the network configuration and activity of a Linux system, including network connections, routing tables, interface statistics, masquerade connections, and multicast memberships. The following list shows four important uses of the *netstat* command.

netstat –i

Displays a table with statistics of the currently configured network interfaces.

netstat –rn

Displays the kernel routing table. The **-n** option forces **netstat** to print the IP addresses. Without this option, **netstat** attempts to display the hostnames.

netstat –an
netstat –tan
netstat –uan

Displays the active network connections. The **–a** option displays all active network connections, the **–ta** option displays only information on TCP connections, and the **–tu** option displays only information on UDP traffic. Omitting the **–n** options prints hostnames and names of servers, instead of IP addresses and ports numbers.

netstat –s

Displays summary statistics for each protocol that is currently running on the host.

Invoking *netstat* with the *–in* option produces the following output.

```
%netstat –in

Kernel Interface table
Iface   MTU  Met    RX-OK  RX-ERR RX-DRP RX-OVR  TX-OK   TX-ERR TX-DRP TX-OVR   Flg
eth0   1500   0     1353        0      0      0   1608        0      0      0   BRU
eth1   1500   0     944         0      0      0   1176        0      0      0   BRU
lo    16436   0     1625        0      0      0   1625        0      0      0   LRU
```

The output shows that the host has interfaces *eth0* and *lo* (loopback) enabled. For each interface, a number of configuration parameters and statistics are displayed. *MTU* is the maximum transmission unit for that interface, *Met* is a link metric that can be used by a routing protocol. The RX and TX columns show how many packets were received or transmitted without an error (RX-OK and TX-OK), how many had bit errors (RX-ERR and TX-ERR), how many have dropped packets (RX-DRP and TX-DRP), and how many had an overrun (RX-OVR and TX-OVR). These statistics are reset every time the host is rebooted. The last column contains a set of flags for the interface which are interpreted as follows: B indicates that a broadcast address has been set for this interface, L indicates the loopback interface, M is set if the interface is set to promiscuous mode, O shows that ARP is disabled for this interface, P indicates a point-to-point link (as opposed to a broadcast network), R indicates that the interface is running, and means that the interface is enabled.

When *netstat* is issues with the *–rn* option it displays the content of the routing table of the Linux kernel. The output of the command could be as follows.

```
% netstat –rn
Kernel IP routing table
Destination     Gateway      Genmask         Flags   MSS Window  irtt Iface
10.0.1.4        0.0.0.0     255.255.255.255  UH      40  0       0     eth0
10.0.3.0        0.0.0.0     255.255.255.0    U       40  0       0     eth1
10.0.2.0        0.0.0.0     255.255.255.0    UG      40  0       0     eth0
10.0.5.0        10.0.2.1    255.255.255.0    UG      40  0       0     eth0
127.0.0.0       0.0.0.0     255.0.0.0        UH      40  0       0     lo
0.0.0.0         10.0.3.1    0.0.0.0          UG      40  0       0     eth1
```

We defer some of the discussion of the routing table to Chapter 3, when we discuss routing tables in detail. The first column contains the destination, which can be a network address or a host address. The third column (Genmask) provides the netmask for the destination address. Thus, the routing table entries are given for destination addresses 10.0.1.4/32, 10.0.3.0/24, 10.0.2.0/24, 127.0.0.0/8. Destination 127.0.0.0/24 is the entry for the loopback, and 0.0.0.0 indicates, by convention, the default route. The second column (Gateway) identifies the next

hop router where matching IP datagrams are transmitted, and the last column (`Iface`) identifies the network interface where the packet with a matching entry is sent. Entries that are set to 0.0.0.0 in the second column should be considered empty. From here we can derive that there are two IP directly connected networks: 10.0.2.0/24 via interface `eth0`, and 10.0.3.0/24 via interface `eth1`. The columns MSS, Window, and itt are parameters used by the TCP protocol, and indicate respectively, maximum segment size, the advertised window, and initial round-trip time for TCP connections over this link. A value of 0 in these columns means that the default values are used.

The flags qualify the type of route: *G* indicates that the next hop is a gateway, *H* indicates a host route, *D* means that the route is dynamically created using a routing protocol or an ICMP redirect, *M* tells that the entry was modified by an ICMP redirect message, and *!* indicates that this route is rejected and the datagrams with matching destinations will be dropped.

We can list the TCP and UDP activity at a host by invoking *netstat* with the *–ta* and the *–ua* options.

```
% netstat –tan
Proto Recv-Q Send-Q Local Address           Foreign Address      State
tcp        0      0 0.0.0.0:32768           0.0.0.0:*            LISTEN
tcp        0      0 0.0.0.0:32769           0.0.0.0:*            LISTEN
tcp        0      0 0.0.0.0:515             0.0.0.0:*            LISTEN
tcp        0      0 0.0.0.0:611             0.0.0.0:*            LISTEN
tcp        0      0 0.0.0.0:905             0.0.0.0:*            LISTEN
tcp        0      0 0.0.0.0:139             0.0.0.0:*            LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*            LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*            LISTEN
tcp        0      0 128.143.71.29:139       128.143.71.37:1078   ESTABLISHED
tcp        0      0 128.143.71.29:22        128.143.71.37:4385   ESTABLISHED
```

Each row shows the information of one TCP connection. The columns Recv-Q and Send-Q address display the amount of received and sent user level data, measured in bytes. The local address and foreign address are the local and remote IP addresses and port numbers. A * or 0.0.0.0 for the IP address indicates the local host, and an entry *:* or 0.0.0.0:* for the foreign address indicates that no connection is established. The last column lists the state of the TCP connection. The states of TCP will be discussed in detail in Chapter 6. Here, `LISTEN` presents a TCP server that is waiting for connection requests, and `ESTABLISHED` means that a TCP connection is in place. By identifying the well-known port numbers, we can identify that this host has servers running for. The well-known port numbers are replaced with the name of the application, when the *–n* option is omitted in the command.

Next we invoke netstat to display the active UDP ports.

```
% netstat –uan
Proto Recv-Q Send-Q Local Address           Foreign Address         State
```

```
udp        0      0 0.0.0.0:32768           0.0.0.0:*
udp        0      0 0.0.0.0:2049            0.0.0.0:*
udp        0      0 0.0.0.0:32769           0.0.0.0:*
udp        0      0 0.0.0.0:32770           0.0.0.0:*
udp        0      0 0.0.0.0:902             0.0.0.0:*
udp        0      0 128.143.71.29:137       0.0.0.0:*
```

The output for UDP is interpreted similarly as for TCP. The main difference is that UDP is a connectionless protocol. Therefore, the column with connection state information is empty.

As a last example, we show the output *netstat –s*. The output is quite long, and we only show a subset of the displayed information. The results are self-explanatory.

```
% netstat –s
Ip:
    2917 total packets received
    0 forwarded
    0 incoming packets discarded
    2088 incoming packets delivered
    3171 requests sent out
Icmp:
    692 ICMP messages received
    1 input ICMP message failed.
    ICMP input histogram:
        destination unreachable: 291
        echo requests: 27
        echo replies: 374
    318 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 291
        echo replies: 27
Tcp:
    547 active connections openings
    0 passive connection openings
    1 failed connection attempts
    0 connection resets received
    0 connections established
    1930 segments received
    2235 segments send out
    1 segments retransmited
    0 bad segments received.
    511 resets sent
Udp:
    6 packets received
    289 packets to unknown port received.
    0 packet receive errors
    295 packets sent
```

6.2. The *ifconfig* command

The **ifconfig** command is used to configure or query the status of network interfaces in Linux. The following list shows how *ifconfig* is used to query the status of network interfaces.

ifconfig

Displays the configuration parameters of all active interfaces.

ifconfig –a

Displays the configuration parameters of all network interfaces, including the inactive interfaces.

ifconfig *interface*

Displays the configuration parameters of a single interface. For example, *ifconfig eth0* displays information on interface *eth0*.

The output a query for interface *eth0* is as follows.

```
%ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:00:C0:88:51:00
          inet addr:10.0.1.11  Bcast:10.0.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1574 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1906 errors:0 dropped:0 overruns:0 carrier:0
          collisions:11 txqueuelen:100
          Interrupt:11 Base address:0xf880
```

The output in the first two lines is the name of the interface, the link layer protocol, the MAC address, the Internet address, the broadcast address, and the extended network prefix expresses as a subnetmask. The third line contains a set of flags. The `UP` and `RUNNING` flag means that the interface is enabled. In a disabled interface the two entries are replaced by a `DOWN` flag. `BROADCAST` indicates that a broadcast address has been configured, and `MULTICAST` tells that the interface is capable of sending and receiving multicast traffic. The `MTU` value shows the maximum transmission unit for the interface, and `Metric` is a link metric which is used by a routing protocol. The fourth and following lines display statistics for the interface.

There are numerous options for configuring a network interface with *ifconfig*. The following example shows how to enable and disable an interface and how to change the IP configuration.

ifconfig *eth0* down

ifconfig *eth0* up

Enables the *eth0* interface.

ifconfig *eth0 10.0.1.8* netmask *255.255.255.0* broadcast *10.0.1.255*

Assigns interface *eth0* the IP address of *10.0.1.8/24* and a broadcast address of *10.0.1.255*. the interface should be disabled before an IP address is assigned, and should be enabled after the IP address has been modified.

ifconfig *eth0* down *10.0.1.8* netmask *255.255.255.0* broadcast *10.0.1.255* up

Performs all three commands above in sequence. Interface *eth0* is disabled, an IP address and a broadcast address are assigned, and the interface is enabled.

ifconfig *eth0* mtu *500*

Sets the MTU of interface *eth0* to 500 bytes.

6.3. The *arp* command

The **arp** command displays and manipulates the ARP cache in the Linux kernel. When an ARP entry is manually added, this entry is permanent, and is not overwritten by ARP messages. When an ARP entry is deleted, the entry is added again if a corresponding ARP message arrives.

arp –a

Displays the content of the ARP cache

arp –d *IPAddress*

Deletes the entry with IP address *IPAddress* from the ARP table.

arp –s *IPaddress MAC_Address*

Manually adds a permanent entry to the ARP cache. Such an added entry is not overwritten by an ARP message. The MAC address is entered as six hexadecimal bytes separated by colons.
Example: *arp –s 10.0.1.2 00:02:2D:0D:68:C1*

Displaying the ARP cache results in the following output.

```
%arp –a

? (10.0.1.11) at 00:00:C0:AD:50:00 [ether] on eth0
? (10.0.1.11) at 00:02:2D:0D:68:C1 [ether] PERM on eth0
```

The output shows the IP address and its corresponding hardware address. The entry "`? (10.0.1.11)`" shows the hostname and the IP address. A '?' indicates that the hostname has not been resolved. The output also indicates the link layer encapsulation (`[ether]`) and the interface (`eth0`) where the entry was resolved. The entry `PERM` indicates a permanent entry that has been manually added.

6.4. Setting filters in *tcpdump*

When the *tcpdump* tool is started with the command

`%tcpdump –n eth0`,

it displays all packets that are captured on network interface *eth0*. Instead of capturing all traffic, and then searching through the output for the data of interest, one can limit the amount of traffic captured by *tcpdump* by specifying a filter expression in the command line. With a filter expression, only the traffic that matches the filter expression is captured and displayed. For example, the command

```
%tcpdump –n eth0 host 10.0.1.12
```

captures IP datagrams from or to IP address 10.0.1.12, and ignores traffic with different IP addresses and non-IP addresses.. A list of filter expressions which may be useful in the exercises of the Internet Lab is shown in Table 2.5. The filter expressions can be combined using negation (*not*), concatenation (*and*), or alternation (*or*) to form more complex filter expressions. In filter expressions with multiple operators, negation has the highest precedence. Concatenation and alternation have equal precedence and are interpreted from left to right. For example, the command

```
%tcpdump –n eth0 not \icmp or src host 10.0.1.12 and ip multicast
```

displays IP datagrams that are not ICMP messages or come from host 10.0.1.12 and, in addition, do not have an IP multicast destination address. A different precedence of the operators can be enforced with parentheses. For example, each of the following three filter expressions yields a different result:

```
not \icmp or host 10.0.1.2  and \tcp
not (\icmp or host 10.0.1.2)  and \tcp
not \icmp or (host 10.0.1.2  and \tcp).
```

If an address or number is not specified by a keyword (*host, proto, net*), then the most recent keyword is assumed. For example,

```
host 10.0.1.2  and 10.0.1.3
```

is short for

```
host 10.0.1.2  and  host 10.0.1.2 .
```

It is possible to access specific fields in protocol headers, and select packets based on the values of protocol header fields. This is done with expressions of the form *proto[offset : size]* which select bytes *offset+1, offset+2, …, offset+size* from the header of protocol *proto.* For example, *ip[2:2]* selects the third and fourth byte in the IP header the total length field. Therefore, the expression *ip[2:2]>576* selects IP datagrams that are longer than 576 bytes. The *tcpdump* expression that displays these IP datagrams is

```
%tcpdump –n ‘ip[2:2]>576’.
```

Note that the selection is put in quotes (‘ ’). If a selection specifies a protocol header, packets that do not have such a protocol header are simply ignored. Table 2.6 shows examples for selecting packets based on the contents of protocol headers. Single bits can be tested using a *bitwise and* operator (&) and a comparison operator *(*>, <, >=, <=, =, *!=*). For example *ip[0] & 0x80 > 0* selects packets where the first bit of the IP header is set. Also, a selection can be combined with any other filter expression. For example,

```
%tcpdump –n ‘ip[2:2]>576’ and not host 10.0.1.2
```

selects all IP datagrams longer than 576 bytes that do not have IP address *10.0.1.2* as source or destination IP address.

`dst host 10.0.1.2`	IP destination address field is 10.0.1.2
`src host 10.0.1.2`	IP source address field is 10.0.1.2
`host 10.0.1.2`	IP source or destination address field is 10.0.1.2
`src net 10.0.1.0/24`	IP source address matches the network address 10.0.1.0/24
`dst net 10.0.1.0/24`	IP destination address matches the network address 10.0.1.0/24
`net 10.0.1.0/24`	IP source or destination address matches the network

	address 10.0.1.0/24
`dst port 80`	Destination port is 80 in TCP segment or UDP datagram
`src port 80`	Source port is 80 in TCP segment or UDP datagram
`port 80`	Destination or source port is 80 in TCP segment or UDP datagram
`src and dst port 80`	Destination and source port is 80 in TCP segment or UDP datagram
`tcp port 80` `udp port 80`	Destination or source port is 80 in TCP segment Destination or source port is 80 in UDP datagram
`len <= 200`	Packet size is not longer than 200 bytes

`ip proto \icmp` *or* `icmp`
`ip proto \tcp` *or* `tcp`
`ip proto \udp` *or* `udp`
`ip proto ospf`
	IP protocol field is set to the number of ICMP or TCP or UDP or OSPF. (Since icmp, tcp, and udp are keywords in *tcpdump* there an escape character (`\') must be placed in front of these keywords.)

`ip proto 17`	IP protocol number is set to 17. The protocol numbers are given in Table 2.1.
`broadcast`	Ethernet broadcast packet
`ip broadcast`	IP broadcast packet
`multicast`	Ethernet `multicast` packet
`ip multicast`	IP `multicast` packet
`ether proto \ip` *or* `ip` `ether proto \arp` *or* `arp`	Ethernet payload is IP or ARP

Table 2.5. Filter expressions for *tcpdump* filters.

`tcp[0] > 4`	The first byte of the TCP header is greater than 4.
`ip[2] <= 0xf`	The third byte of the IP header does not exceed 15.
`udp[0:2] == 1023`	The first two bytes of the UDP header are equal to 1023.
`ip[0] & 0xf > 5`	IP headers that are longer than 20 bytes, i.e., IP headers with options. The expression *ip[0]* selects the first byte from the IP packet and *ip[0]& 0xf* filters the four lower order bits in the first byte. The expression *ip[0] & 0xf > 5*, selects all IP packets where the IP header length is larger than five. Since the IP header field expresses multiples of four bytes, these are packets where the IP header is longer than 20 bytes.

`ip[6:2] & 0x1fff == 0`	IP packets where the fragment offset field is zero, i.e., unfragmented IP packets or the first fragment of a fragmented IP packet. The expression *ip[6:2]* selects the seventh and eight byte from an IP header, and *ip[6:2] & 0x1fff* filters the last 13 bits from these two bytes.
`tcp[13] & 3 != 0`	TCP headers with the SYN flag or the FIN flag set. The expression *tcp[13] & 3* selects the two least significant bits of the 14-th byte in the TCP header. These bits hold the SYN flag and the FIN flag. The expression is not 0, if at least one of the bits is set.

Table 2.6. Selection of packets based on protocol header contents.

6.5. Setting filters in *ethereal*

Just like *tcpdump*, *ethereal* permits users to set filters for the traffic that is analyzed by the tool. *Ethereal* has two types of filters: *capture filters* and *display filters*. A capture filter specifies the type of traffic that is captured by *ethereal*, similarly to filters in *tcpdump*. In fact, capture filters in *ethereal* are written using the same syntax as *tcpdump* filters. A display filter specifies the type of traffic that is displayed in the main window of *ethereal*, but does not restrict the captured traffic. An advantage of using display filters is that it is possible to change the display filter after packets have been captured. The syntax for setting display filter is completely different from the syntax for setting a capture filter.

Capture Filters: A capture filter can be set in the command line when *ethereal* is started or in the capture window before a traffic capture is initiatedThe following command is used to set a capture filter from the command line

ethereal –i *interface* –f *filter*

where *interface* is a network interface and *filter* is a filter expression. The filter expression is written using the syntax described for *tcpdump* filters. If no capture filter is specified, *ethereal* captures all traffic.

Alternatively, the interface and the capture filter can be set from the *Capture Preferences* window of *ethereal*, which is opened by selecting Capture→Start in the main window, and by typing in the interface name and the desired filter expression in the appropriate boxes.

The *Capture Preferences* window of *ethereal* is shown Figure 2.24. Here, the interface is set to *eth0* and the capture filter is set to *host 10.0.1.12*.

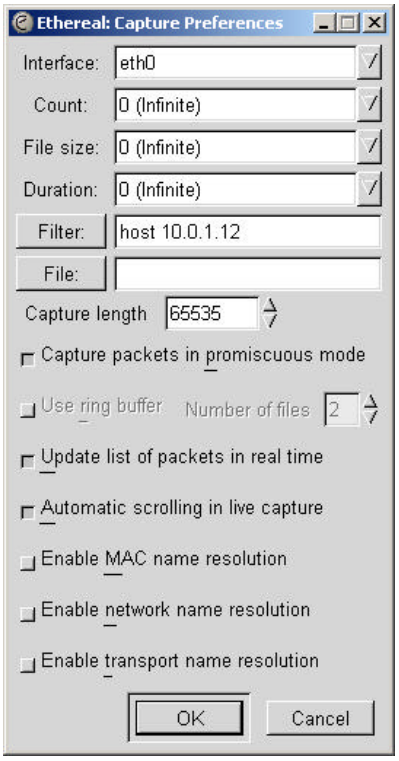


Figure 2.24. Setting a capture filter.

Display Filters: Display filters cannot be set in the command line. A display filter is set by typing a filter expression at the bottom of main window in *ethereal*, next to the label *Filter*. This is illustrated in Figure 2.25. Here the display filter is set to *ip.dst==10.0.1.12*, which selects all IP packets with the destination IP address *10.0.1.12*. When the filter is applied, only those packets that match the filter are displayed in the main window. The *Reset* button next to the *Filter* box removes the filter.

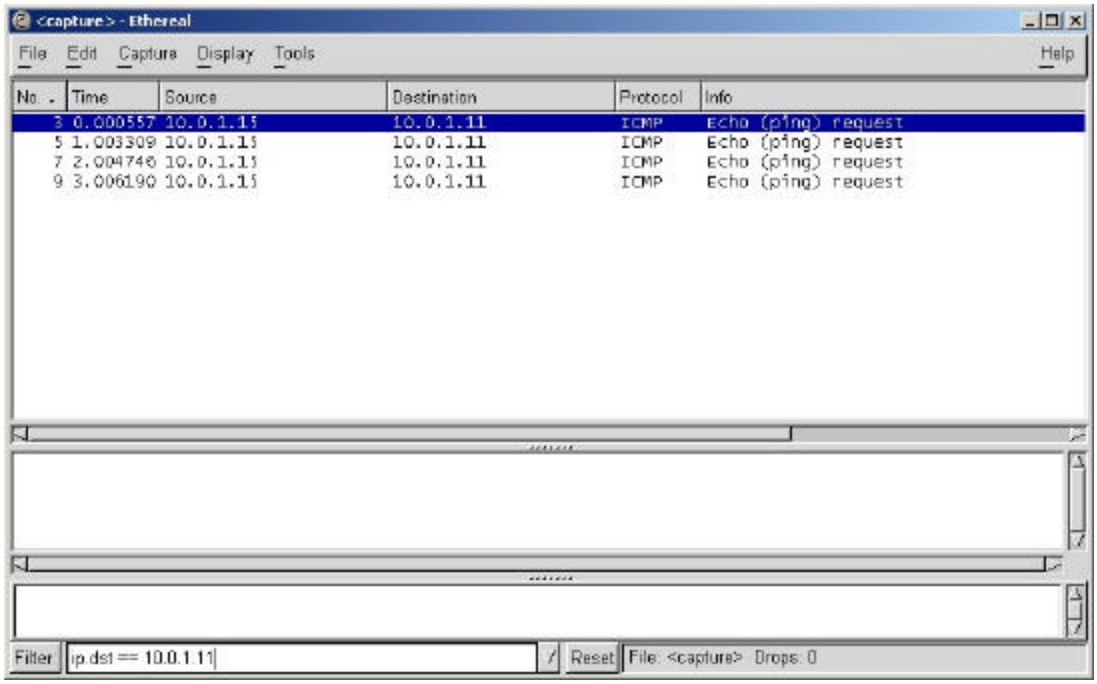


Figure 2.25. Applying a display filter in *ethereal*.

The syntax for display filters is different from that for capture filters. In Table 2.7 and Table 2.8 we show *ethereal* display filters that correspond to the *tcpdump* filters (and *ethereal* capture filters) from Table 2.5 and Table 2.6. Display filters have a separate keyword for each header field of a protocol, and are generally easier to read. The keywords for a particular header field can be obtained from the manual page of *ethereal*.

Ethereal offers interactive help for writing display filters. This is activated by clicking on the *Filter* button in the main window (see Figure 2.25). This pops up the *Display Filter* window (see Figure 2.26). Then, clicking on the *Add Expression* button pops up the *Filter Expression* window (see Figure 2.27). Now, the desired filter expression can be built by selecting a protocol and a protocol field. Once a filter expressions is constructed, it is displayed in the main window of *ethereal*.

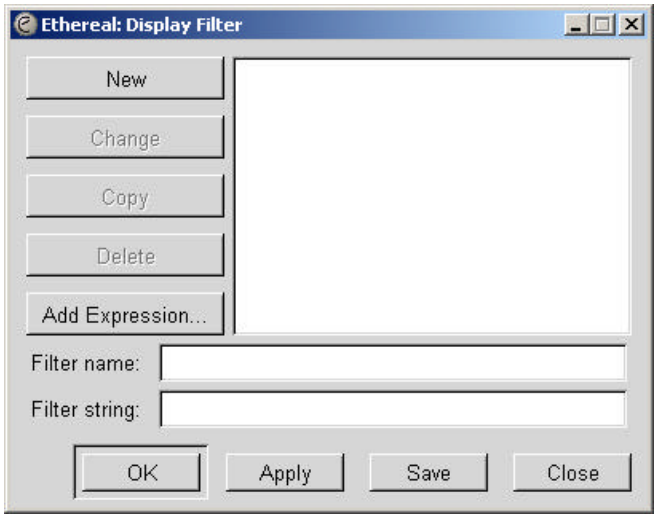


Figure 2.26. Display filter window.

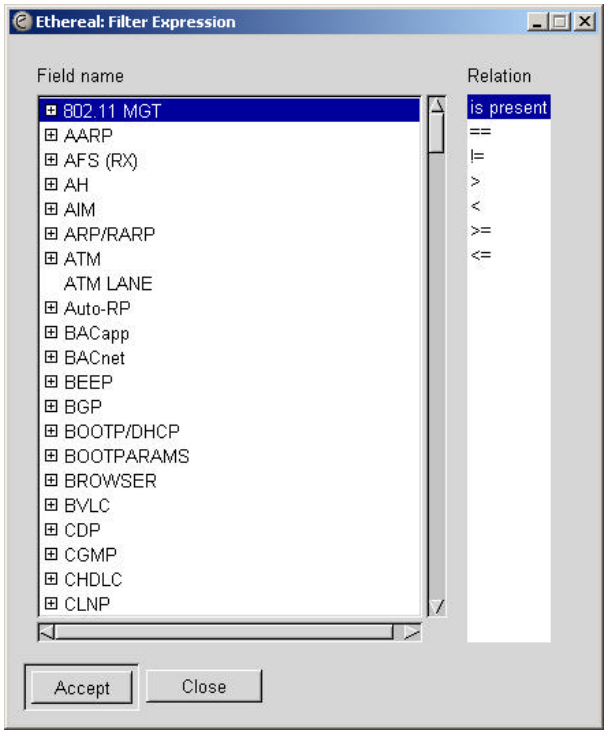


Figure 2.27. Filter expression window

`ip.dst==10.0.1.2`	IP destination address field is 10.0.1.2
`ip.src==10.0.1.2`	IP source address field is 10.0.1.2
`ip.addr==10.0.1.2`	IP source or destination address field is 10.0.1.2
`ip.src==10.0.1.0/24`	IP source address matches the network address 10.0.1.0/24
`ip.dst==10.0.1.0/24`	IP destination address matches the network address 10.0.1.0/24
`ip.addr== 10.0.1.0/24`	IP source or destination address matches the network address 10.0.1.0/24
tcp.dstport == 80 or udp.dstport == 80	Destination port is 80 in TCP segment or UDP datagram
`Tcp.srcport==80 or udp.srcport==80`	Source port is 80 in TCP segment or UDP datagram
`Tcp.port==80 or udp.port==80`	Destination or source port is 80 in TCP segment or UDP datagram
`(tcp.srcport==80 and tcp.dstport==80) or (udp.srcport==80 and udp.dstport==80)`	Destination and source port is 80 in TCP segment or UDP datagram
`tcp.port==80 udp.port==80`	Destination or source port is 80 in TCP segment Destination or source port is 80 in UDP datagram
`eth.len <= 200`	Packet size is not longer than 200 bytes
`Icmp tcp udp ospf`	IP protocol field is set to the number of ICMP or TCP or UDP or OSPF. (Since icmp, tcp, and udp are keywords in *tcpdump* there an escape character (`\') must be placed in front of these keywords.)
`ip.proto==17`	IP protocol number is set to 17. The protocol numbers are given in Table 2.1.
`eth.dst==ff:ff:ff:ff:ff:ff`	Ethernet broadcast packet
`ip.dst[==`	No general expression exists for IP broadcast packet.
`eth.dst[0]==1`	Ethernet `multicast` packet.
`ip.dst==224.0.0.0/4`	IP `multicast` packet.
`Ip arp`	Ethernet payload is IP or ARP

Table 2.7. Display filter expressions for *ethereal* (compare with Table 2.5).

`tcp[0] > 4`	The first byte of the TCP header is greater than 4.
`ip[2] <= f`	The third byte of the IP header does not exceed 15.
`udp[0:2] == 3:ff`	The first two bytes of the UDP header are equal to 1023. Note: When selecting bytes from a header, each byte is

	written as a hexadecimal number, and bytes are separated by a colon.
`ip.hdr_len > 20`	IP headers that are longer than 20 bytes, i.e., IP headers with options.
`ip.frag_offset == 0`	IP packets where the fragment offset field is zero, i.e., unfragmented IP packets or the first fragment of a fragmented IP packet.
`tcp.flags.syn==1 or tcp.flags.fin==1`	TCP headers with the SYN flag or the FIN flag set.

Table 2.8. More display filter expressions for *ethereal* (compare with Table 2.6).

RFCs:

[RFC 791] Internet Protocol

[RFC 792] Internet Control Message Protocol

[RFC2212] Specification of Guaranteed Quality of Service.

[RFC 2372] IP Version 6 Addressing Architecture

[RFC 2460] Internet Protocol, Version 6 (IPv6) Specification
Obsoletes: RFC 1883.

[RFC 2461] Neighbor Discovery for IP Version 6 (IPv6).
Obsoletes: RFC 1970.

[RFC 2463] Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification.
Obsoletes: RFC 1885.

[RFC 2474] Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. Obsoletes: RFC 1349, RFC 1455.

[RFC 2475] An Architecture for Differentiated Services.

[RFC 2521] ICMP Security Failures Messages.

[RFC 2893] Transition Mechanisms for IPv6 Hosts and Routers.
Obsoletes: RFC 1933.

[RFC3168] The Addition of Explicit Congestion Notification (ECN) to IP

[RFC 3246] An Expedited Forwarding PHB.

[RFC 3260] New Terminology and Clarification for Diffserv.

Other References:

[Stevens94] W. Richard Stevens, TCP/IP Illustrated, Volume 1. The Protocols. Addison-Wesley Publishing Company, 1994.

[Huitema96] Christian Huitema, IPv6 The New Internet Protocol, Prentice Hall, PTR, 1996.

[Kirch99] Olaf Kirch, Terry Dawson. Linux Network Administrator's Guide. 2nd edition. O'Reilly and Associates, 1999.

[Tcpdump] Van Jacobson, Craig Leres and Steven McCanne. Tcpdump 3.6 Manual page. **http://tcpdump.org, 2001.**

[Ethereal] Gerald Combs et.al. Ethereal 0.9.5 Manual page. **http://www.ethereal.com, June 2002**