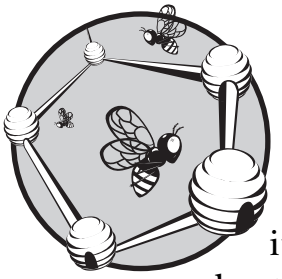
60

TCP/IP BOOTSTRAP PROTOCOL (BOOTP)



Before a device on a TCP/IP network can effectively communicate, it needs to know its IP address. While a conventional network host can read this information from its internal disk, some devices have no storage, so they do not have this luxury. They need help from another device on the network to provide them with an IP address and the other information and/or software they need to become active Internet Protocol (IP) hosts. This problem of getting a new machine up and running is commonly called *bootstrapping*, and to provide this capability to IP hosts, the TCP/IP *Bootstrap Protocol (BOOTP)* was created.

In this chapter, I provide a detailed look at BOOTP. I begin with an overview and history of the protocol and a look at the standards that define it. I then discuss the general client/server nature of BOOTP and how addressing is done in communication between the client and the server. I describe the operation of BOOTP step by step and illustrate the format of BOOTP messages. I conclude with a description of BOOTP vendor extensions, which

are used to allow the information sent in BOOTP messages to be customized, and a discussion of BOOTP relay agents, which allow the protocol to operate even when the BOOTP server and client are on different networks.

RELATED INFORMATION *BOOTP was the predecessor of the Dynamic Host Configuration Protocol (DHCP). DHCP was built to be substantially compatible with BOOTP, and so the two protocols have a fair degree of commonality. To avoid duplication, certain information has been included only in the following chapters about DHCP (with references to this chapter where appropriate). On the other hand, some of the historical background information behind features like vendor information extensions and relay agents, which were first developed for BOOTP and adopted by DHCP, is in this chapter and referenced from the DHCP chapters. If you plan to read about DHCP as well as BOOTP, I recommend reading this section first. If you don't plan to read up on DHCP, you may wish to check the discussion of DHCP/BOOTP interoperability in Chapter 64.*

BOOTP Overview, History, and Standards

The TCP/IP protocol suite has been with us for over two decades, and the problem of how to automate the configuration of parameters on IP hosts has been around almost as long. Back in the early 1980s, networks were small and relatively simple, so manual configuration wasn't that difficult. Automated host configuration was primarily needed because it was the only way to configure devices like diskless workstations.

As I discussed in Chapter 59, without a form of internal storage, a device must rely on someone or something to tell it "who it is" (its address) and how to function each time it is powered up. When a device like this is turned on, it is in a difficult position: It needs to use IP to communicate with another device that will tell it how to communicate using IP! This process, called *bootstrapping* or *booting*, comes from an analogy to a person "pulling himself up using his own bootstraps." You've likely encountered this term before, if at no other time then when some tech support person has told you to "reboot" your computer.

BOOTP: Correcting the Weaknesses of RARP

The Reverse Address Resolution Protocol (RARP) was the first attempt to resolve this bootstrap problem. Created in 1984, RARP is a direct adaptation of the low-level Address Resolution Protocol (ARP) that binds IP addresses to link-layer hardware addresses (see Chapter 13). RARP is capable of providing a diskless device with its IP address, using a simple client/server exchange of a request and reply between a host and an RARP server.

The difficulty with RARP is that it has so many limitations. It operates at a fairly low level using hardware broadcasts, so it requires adjustments for different hardware types. An RARP server is also required on every physical network to respond to layer 2 broadcasts. Each RARP server must have address assignments manually provided by an administrator. And perhaps worst of all, RARP provides only an IP address to a host and none of the other information a host may need. (I describe these issues in detail in Chapter 14.)

RARP clearly wasn't sufficient for the host configuration needs of TCP/IP. To support both diskless hosts and other situations where the benefits of autoconfiguration were required, BOOTP was created. BOOTP was standardized in RFC 951,

published in September 1985. This relatively straightforward protocol was designed specifically to address the shortcomings of RARP:

- BOOTP is still based on a client/server exchange, but is implemented as a higher-layer software protocol, using the User Datagram Protocol (UDP) for message transport (see Chapter 44). It is not dependent on the particular hardware of the network as RARP is.
- It supports sending additional configuration information to a client beyond just an IP address. This extra information can usually be sent in one message for efficiency.
- It can handle having the client and server on different networks of an internetwork. This allows the administration of the server providing IP addresses to be more centralized, saving money as well as administrative time and hassle.

It should be noted that, even though the name of BOOTP implies that it defines everything needed for a storageless device to boot, this isn't really the case. As the BOOTP standard itself describes, bootstrapping generally requires two phases. In the first, the client is provided with an address and other parameters. In the second, the client downloads software, such as an operating system and drivers, that let it function on the network and perform other tasks. BOOTP really deals with only the first of these phases: address assignment and configuration. The second is assumed to take place using a simple file transfer protocol like the Trivial File Transfer Protocol (TFTP, discussed in Chapter 73).

KEY CONCEPT The first widely used host configuration protocol for TCP/IP was the *Boot Protocol (BOOTP)*. It was created specifically to enable host configuration while addressing many of the weaknesses of RARP. BOOTP is intended to be used as the first phase of a two-phase boot procedure for storageless devices. After obtaining an IP address and other configuration parameters using BOOTP, the device employs a protocol such as TFTP to download software necessary to function on the network.

Vendor-Specific Parameters

One smart decision made when BOOTP was created was the inclusion of a *vendor-specific area*. This was intended to provide a place where hardware vendors could define parameters relevant to their own products. As the complexity of TCP/IP increased, it was realized that this field could be used to define a method of communicating certain parameters that were commonly needed by IP hosts, and were in fact vendor-independent. This was first proposed in RFC 1048, "BOOTP Vendor Information Extensions," published in February 1988.

The fact that BOOTP can be used to provide information to a client beyond just an IP address makes it useful even in cases where a device already knows its address. BOOTP can be used to send parameters that the administrator wants all hosts to have, to ensure that they use the network in a consistent manner. Also, in the case of devices that do have local storage (and therefore do not need BOOTP to get an IP address), BOOTP can still be used to let these devices get the name of a boot file for phase two of bootstrapping, in which the client downloads software.

Changes to BOOTP and the Development of DHCP

BOOTP was the TCP/IP host configuration protocol of choice from the mid-1980s through the end of the 1990s. The vendor extensions introduced in RFC 1048 were popular, and over the years, additional vendor extensions were defined. RFC 1048 was replaced by RFCs 1084, 1395, and 1497 in succession.

Some confusion also resulted over the years in how some sections of RFC 951 should be interpreted and how certain features of BOOTP work. RFC 1542, "Clarifications and Extensions for the Bootstrap Protocol," was published in October 1993 to address this and also to make some slight changes to the protocol's operation. (RFC 1542 is actually a correction of the nearly identical RFC 1532, which had some small errors.)

While BOOTP was obviously quite successful, it also had certain weaknesses. One of the most important of these was the lack of support for *dynamic* address assignment. The need for dynamic assignment became much more pronounced when the Internet really started to take off in the late 1990s. This led directly to the development of the Dynamic Host Configuration Protocol (DHCP).

While DHCP replaced BOOTP as the TCP/IP host configuration protocol of choice, it would be inaccurate to say that BOOTP is gone. It is still used to this day in some networks. Furthermore, DHCP was based directly on BOOTP, and they share many attributes, including a common message format. BOOTP vendor extensions were used as the basis for DHCP *options*, which work in the same way but include extra capabilities. In fact, the successor to RFC 1497 is RFC 1533, which officially merges BOOTP vendor extensions and DHCP options into the same standard.

BOOTP Client/Server Messaging and Addressing

While BOOTP can be used for a variety of devices, one of the primary motives behind its creation was to provide a way to automatically configure "dumb" devices that have no storage. Most of these devices are relatively limited in their capabilities, so requiring them to support a fancy boot protocol would not make sense. BOOTP is thus an uncomplicated protocol, which accomplishes host configuration without a lot of complicated concepts or implementation requirements.

Like so many other TCP/IP protocols, BOOTP is client/server in nature. The operation of the protocol consists of a single exchange of messages between a *BOOTP client* and a *BOOTP server*. A BOOTP client can be any type of device that needs to be configured. A BOOTP server is a network device that has been specially set up to respond to BOOTP client requests, and has been programmed with addressing and other information it can provide to clients when required.

The BOOTP server maintains a special set of information about the clients it serves. One key part of this is a table that maps the hardware (layer 2, the data link layer) addresses of each client to an assigned IP address for that device. The client specifies its hardware address in its request, and the server uses that address to look up the client's IP address and return it to the client. (Other techniques can also be used, but a mapping table is most common.)

BOOTP Messaging and Transport

BOOTP messaging uses UDP as its layer 4 transport protocol, for a couple of reasons:

- UDP is a lot less complex than the other layer 4 transport protocol, the Transmission Control Protocol (TCP), and is ideal for simple request/reply protocols like BOOTP.
- Since the client obviously doesn't know the address of a BOOTP server, the request is broadcast on its local network. UDP supports broadcasts; TCP does not.

UDP uses a special well-known (reserved) port number for BOOTP servers: UDP port 67. BOOTP servers listen on port 67 for these broadcast BOOTP requests sent by clients. After processing the request, the server sends a reply back to the client. How this is handled depends on whether or not the client knows its own address.

BOOTP is often used to provide an IP address to a client that doesn't know its address. This is sometimes called a chicken-and-egg problem, because it represents a loop of sorts like the old conundrum of which came first, the chicken or the egg? To resolve this dilemma, the BOOTP server has two choices. If the operating system supports it, the server can use the client's hardware address to create an ARP entry for the device, and then use a layer 2 unicast to deliver the reply. Otherwise, it must send the reply as a broadcast as well on the local network.

However, in the case where the BOOTP client already knows its own address, that address can be used by the BOOTP server to send back its reply directly.

BOOTP Use of Broadcasts and Ports

The fact that BOOTP servers may need to broadcast back to the client necessitates a bit of a change from the way most TCP/IP protocols use client ports. Recall that normally, the client in a client/server transaction using UDP or TCP generates a temporary, or ephemeral, port number that it uses as the source port in its request. The server sends the reply back to the client's IP address using that ephemeral port number.

Ephemeral port numbers must be unique for a particular IP address, but may not necessarily be unique across all the devices on a network. For example, Device A may be using ephemeral port number 1248 for an HTTP request to a web server, while Device B may be using port number 1248 on its TCP/IP stack to send a Domain Name System (DNS) request. Since the server in BOOTP is broadcasting, it is not targeting a particular device with a unicast transmission. This means it cannot safely send to an ephemeral port number. This is because some other device on the network may have selected the same ephemeral port number for some other transaction and may mistake the BOOTP server's response as being intended for itself. To avoid this problem, another well-known port number is used just for BOOTP clients: UDP port 68. Clients listen on this port for broadcast or unicast transmissions; devices that have not sent a BOOTP request will ignore it. This dual-broadcast BOOTP communication process is illustrated in Figure 60-1.

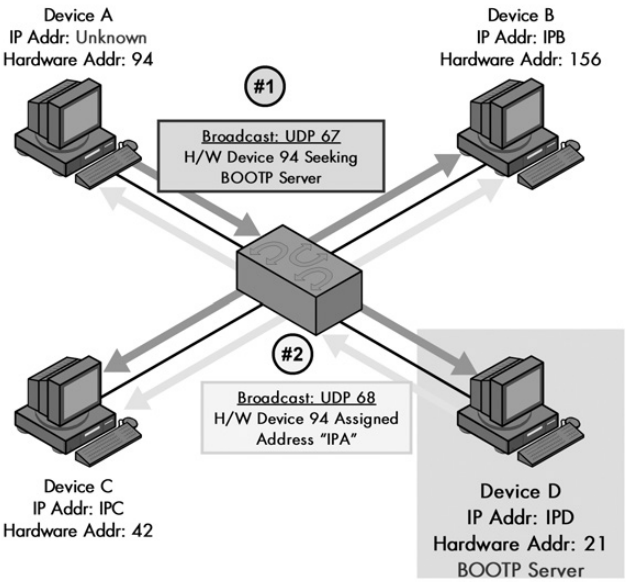


Figure 60-1: *General operation of BOOTP*

KEY CONCEPT BOOTP is a relatively simple client/server protocol that relies on broadcasts to permit communication with devices that do not have an assigned IP address. In this example, Device A is trying to determine its IP address and other parameters. It broadcasts a BOOTP request on the local network using UDP port 67 and then listens for a reply on port 68. Device D is configured as a BOOTP server and listens on this port. When it receives the request, it sends a broadcast on port 68 telling Device A what its IP address is. A BOOTP client uses broadcasts to send its requests to any listening BOOTP server. In most cases, the BOOTP client device does not know its own IP address when it uses the protocol. For this reason, a BOOTP server will also typically use broadcast in sending its reply, to be sure it reaches the client.

Retransmission of Lost Messages

The drawback of the simplicity of using UDP for BOOTP messaging is that UDP is unreliable, which means a BOOTP request might be lost before it gets to the server, or the server's response may not get back to the client. Like many other protocols using UDP, BOOTP clients take care of this by using a retransmission timer. If after a certain period of time the client has not received a response, it resends its request.

However, BOOTP clients must take care in how they implement their retransmission strategy. Consider a scenario where a network with 200 BOOTP clients loses power. These machines are all pretty much the same, so when the power comes back on, they all restart and try to send BOOTP requests at about the same time. Most likely, problems will occur due to all these requests: Some will be lost, or the server may drop some due to overload. If all the clients use the same amount of time for retransmission, then after that time elapses, a whole bunch of machines will again send requests and re-create the original problem.

To avoid retransmission problems, the BOOTP standard recommends using an exponential backoff scheme for retransmissions, starting with a retransmission

interval of 4 seconds and doubling it for successive tries. A randomness element is also added to prevent many devices from overlapping their retransmissions. The idea is very similar to the backoff method used by Ethernet (in fact, the standard even refers to the Ethernet specification). For example, the first retransmission would occur after a random period of time between 0 and 4 seconds (plus or minus a random amount); a second retransmission, if needed, after a random time interval between 0 and 8 seconds, plus or minus, and so forth. This helps reduce the chances of retransmissions being lost and also helps ensure BOOTP traffic doesn't bog down the network.

KEY CONCEPT BOOTP uses UDP for transport, which provides no reliability features. For this reason, the BOOTP client must detect when its requests are lost and, if necessary, retransmit them.

BOOTP Detailed Operation

Now that you have seen how BOOTP messaging works in general terms, let's take a closer look at the detailed operation of the protocol. This will clarify how clients and servers create and process messages, and also help make sense of some of the important fields in the BOOTP message field format. Understanding the basic operation of BOOTP will also be of use when we examine BOOTP relay agents later in this chapter, and even when we discuss DHCP in the following chapters.

BOOTP Bootstrapping Procedure

The following are the basic steps performed by the client and server in a regular BOOTP bootstrapping procedure (see Figure 60-2).

Client Creates Request The client machine begins the procedure by creating a BOOTP request message. In creating this message, it fills in the following information:

- It sets the message type (Op) to the value 1, for a BOOTREQUEST message.
- If it knows its own IP address that it plans to keep using, it specifies it in the CIAddr (Client IP Address) field. Otherwise, it fills this field with zeros. (The CIAddr field is discussed in more detail in the next section.)
- It puts its own layer 2 hardware address in the CHAddr field. This is used by the server to determine the right address and other parameters for the client.
- It generates a random transaction identifier and puts this in the XID field.
- The client may specify a particular server that it wants to send it a reply and put that in the SName field. It may also specify the name of a particular type of boot file that it wants the server to provide in the File field.
- The client may specify vendor-specific information, if programmed to do so.

Client Sends Request The client broadcasts the BOOTREQUEST message by transmitting it to address 255.255.255.255. Alternatively, if it already knows the address of a BOOTP server, it may send the request unicast.

Server Receives Request and Processes It A BOOTP server, listening on UDP port 67, receives the broadcasted request and processes it. If a name of a particular server was specified and this name is different from the name of this server, the server may discard the request. This is especially true if the server knows that the server the client asked for is also on the local network. If no particular server is specified, or this particular server was the one the client wanted, the server will reply.

Server Creates Reply The server creates a reply message by copying the request message and changing several fields:

- It changes the message type (Op) to the value 2, for a BOOTREPLY message.
- It takes the client's specified hardware address from the CHAddr field and uses it in a table lookup to find the matching IP address for this host. It then places this value into the YIAddr (Your IP Address) of the reply.
- It processes the File field and provides the filename type the client requested, or if the field was blank, the default filename.
- It puts its own IP address and name in the SIAddr and SName fields.
- It sets any vendor-specific values in the Vend field.

Server Sends Reply The server sends the reply. The method it uses depends on the contents of the request:

- If the B (Broadcast) flag is set, this indicates that the client can't have the reply sent unicast, so the server will broadcast it.
- If the CIAddr field is nonzero, the server will send the reply unicast back to that CIAddr.
- If the B flag is zero and the CIAddr field is also zero, the server may either use an ARP entry or broadcast, as described earlier.

Client Processes Reply The client receives the server's reply and processes it, storing the information and parameters provided. (See the next section for one important issue related to this processing.)

Client Completes Boot Process Once configured, the client proceeds to phase two of the bootstrapping process, by using a protocol such as TFTP to download its boot file containing operating system software, using the filename the server provided.

Interpretation of the Client IP Address (CIAddr) Field

A complication can arise when a client chooses to specify an IP address in the CIAddr field in its request. The problem is how exactly to interpret this field. Does it mean that the client is already using this IP address? Or is it just the one it used last time it was booted? Then there is the related problem of what to do if the server supplies an address in the YIAddr that is different from the one the client is using. Should the server's provided address override the client's address? Or should the client ignore it? Who makes the decision, the server or the client?

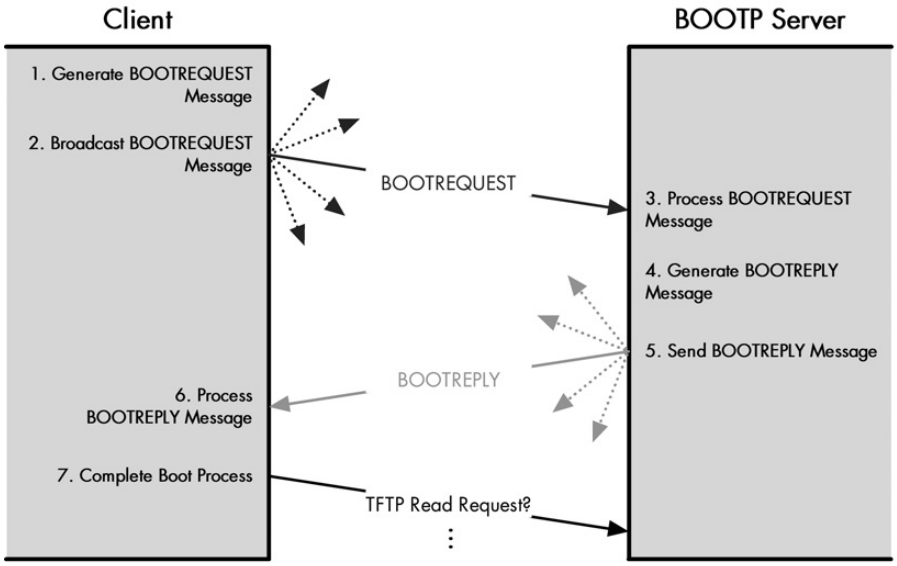


Figure 60-2: BOOTP operation *BOOTP uses a simple two-step message exchange consisting of a broadcast request and broadcast reply. After the client receives configuration information from the BOOTP server, it completes the bootstrapping process using a protocol such as TFTP.*

Much confusion occurred due to the vagueness of the original standard in this regard, and this led to nonuniformity in how different implementations chose to handle this issue. There were even some implementations that used the CIAddr to mean "the client requests this IP address," which was never part of BOOTP functionality. This is an especially bad idea since it could lead to BOOTP replies never reaching the client.

RFC 1542 was written in part to try to clean up this mess. It suggests that the following is the best way to handle the meaning of these fields:

- If a client is willing to accept whatever IP address the server provides, it sets CIAddr to all zeros, even if it knows a previous address.
- If the client fills in a value for the field, it is saying it will use this address, and it must be prepared to receive unicast messages sent to that address.
- If the client specifies an address in CIAddr and receives a different address in the YIAddr field, the server-provided address is ignored.

Note that not all hardware devices may necessarily agree with this interpretation as provided by RFC 1542, so there are still potential interoperability concerns here with older equipment. Then again, RFC 1542 was written in 1993, so this is probably no longer much of an issue!

BOOTP Message Format

The exchange of information in BOOTP takes the form of a request sent by a client and a reply sent back by the server. BOOTP, like a number of other request/reply protocols, uses a common message format for requests and replies. The client starts

by setting aside memory space for the message and clearing it to all zeros. It then fills in the fields of the message and sends the request, as you saw in the previous section. The server creates its reply not from scratch, but by copying the request and changing certain fields.

BOOTP messages contain a considerable number of fields, so the message format is rather large. It is described in Tables 60-1 and 60-2, and illustrated in Figure 60-3.

Table 60-1: BOOTP Message Format

Field Name	Size (Bytes)	Description
Op	1	Operation Code: Specifies the type of message. A value of 1 indicates a request (BOOTREQUEST message). A value of 2 is a reply (BOOTREPLY message).
HType	1	Hardware Type: This field specifies the type of hardware used for the local network and is used in exactly the same way as the equivalent field (HRD) in the ARP message format (see Chapter 13). Some of the most common values for this field are shown in Table 60-2.
HLen	1	Hardware Address Length: Specifies how long hardware addresses are in this message. For Ethernet or other networks using IEEE 802 MAC addresses, the value is 6. This is the same as the field with a similar name (HLN) in the ARP field format.
Hops	1	Hops: Set to 0 by a client before transmitting a request and used by BOOTP relay agents to control the forwarding of BOOTP messages.
XID	4	Transaction Identifier: A 32-bit identification field generated by the client, to allow it to match up the request with replies received from BOOTP servers.
Secs	2	Seconds: According to RFC 951, the client enters into this field the number of seconds "elapsed since [the] client started trying to boot." This is supposed to provide information to BOOTP servers to help them decide which requests to respond to first. Unfortunately, it isn't clear if this meant the amount of time since the machine was powered on or since the first BOOTREQUEST message was sent. In addition, some devices incorrectly implemented this field. As a result, it is not always used.
Flags	2	Flags: In the original BOOTP standard (RFC 951), this was an empty 2-byte field. RFC 1542 changed this to a Flags field, which at present contains only one flag. It has a B (Broadcast) flag subfield, 1 bit in size, which is set to 1 if the client doesn't know its own IP address at the time it sends its BOOTP request. This serves as an immediate indicator to the BOOTP server or relay agent that receives the request that it definitely should send its reply by broadcast. The other subfield is Reserved, which is 15 bits, set to 0, and not used.
CIAddr	4	Client IP Address: If the client has a current IP address that it plans to keep using, it puts it in this field. By filling in this field, the client is committing to responding to unicast IP datagrams sent to this address. Otherwise, it sets this field to all 0s to tell the server it wants an address assigned. (See the previous section in this chapter for important information about this field.)
YIAddr	4	Your IP Address: The IP address that the server is assigning to the client. This may be different than the IP address currently used by the client.
SIAddr	4	Server IP Address: The IP address of the BOOTP server sending a BOOTREPLY message.
GIAddr	4	Gateway IP Address: Used to route BOOTP messages when BOOTP relay agents facilitate the communication of BOOTP requests and replies between a client and a server on different subnets or networks. To understand the name, remember that the old TCP/IP term for *router* is *gateway*; BOOTP relay agents are typically routers. Note that this field is set to 0 by the client and should be ignored by the client when processing a BOOTREPLY. It specifically does not represent the server giving the client the address of a default router address to be used for general IP routing purposes.

(continued)

Table 60-1: BOOTP Message Format (continued)

Field Name	Size (Bytes)	Description
CHAddr	16	Client Hardware Address: The hardware (layer 2) address of the client sending a BOOTREQUEST. It is used to look up a device's assigned IP address and also possibly in delivery of a reply message.
SName	64	Server Name: The server sending a BOOTREPLY may optionally put its name in this field. This can be a simple text nickname or a fully qualified DNS domain name (such as myserver .organization.org). Note that a client may specify a name in this field when it creates its request. If it does so, it is saying that it wants to get a reply only from the BOOTP server with this name. This may be done to ensure that the client is able to access a particular boot file stored on only one server.
File	128	Boot Filename: Contains the full directory path and filename of a boot file that can be downloaded by the client to complete its bootstrapping process. The client may request a particular type of file by entering a text description here, or it may leave the field blank and the server will supply the filename of the default file.
Vend	64	Vendor-Specific Area: Originally created to allow vendors to customize BOOTP to the needs of different types of hardware, this field is now also used to hold additional vendor-independent configuration information. The next section, on BOOTP vendor information extensions, contains much more detail on this field. It may be used by the client and/or the server.

Table 60-2: BOOTP Message HType Values

HType Value	Hardware Type
1	Ethernet (10 Mb)
6	IEEE 802 Networks
7	ARCNet
15	Frame Relay
16	Asynchronous Transfer Mode (ATM)
17	High-Level Data Link Control (HDLC)
18	Fibre Channel
19	ATM
20	Serial Line

As I mentioned earlier in this chapter, both requests and replies are encapsulated into UDP messages for transmission. The BOOTP standard specifies that the use of UDP checksums is optional. Using the checksum provides protection against data-integrity errors and is thus recommended. This may cause unacceptable processing demands on the part of very simple clients, so the checksum can legally be skipped.

Similarly, for simplicity, BOOTP assumes that its messages will not be fragmented. This is to allow BOOTP clients to avoid the complexity of reassembling fragmented messages. Since BOOTP messages are only 300 bytes in length, under the maximum transmission unit (MTU) required for all TCP/IP links, this is not normally an issue.

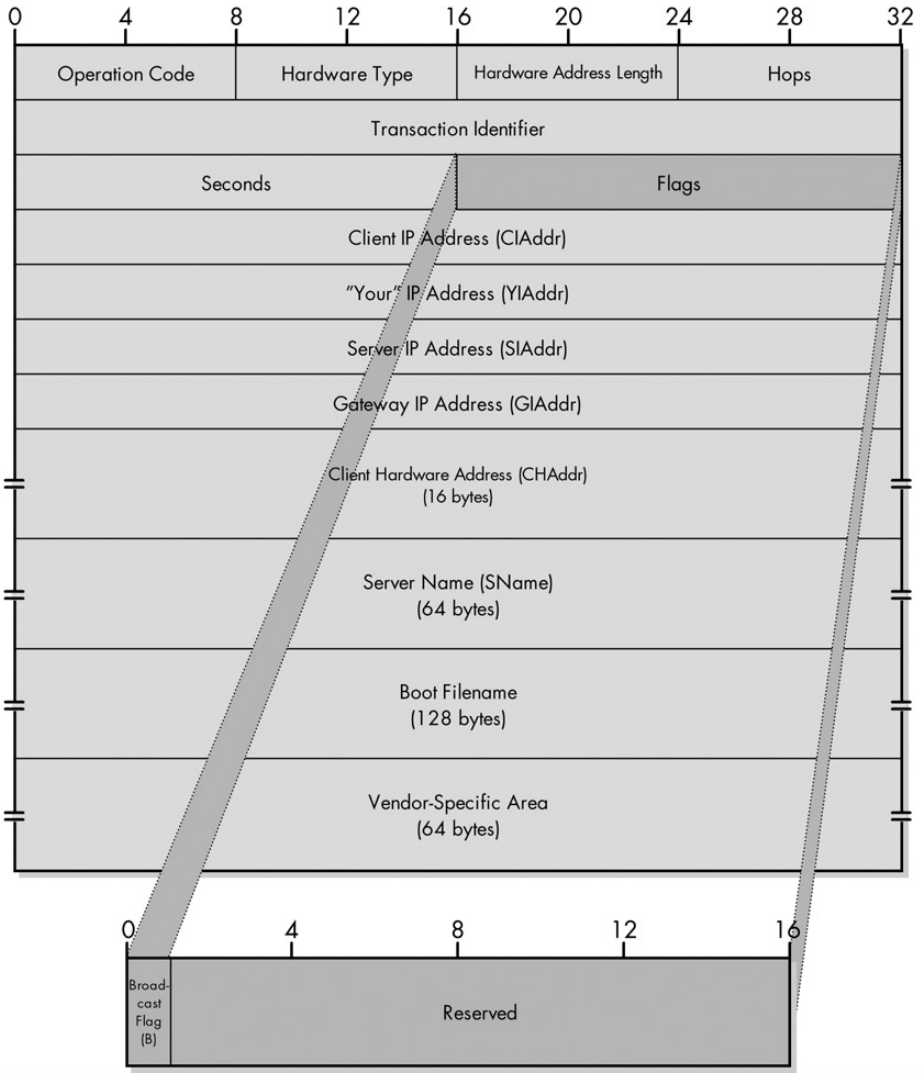


Figure 60-3: *BOOTP message format*

BOOTP Vendor-Specific Area and Vendor Information Extensions

The creators of BOOTP realized that certain types of hardware might require additional information to be passed from the server to the client in order for the client to boot. For this reason, they put into the BOOTP field format the 64-byte Vend field, also called the Vendor-Specific Area. Including this field makes BOOTP flexible, since it allows vendors to decide for themselves how they want to use the protocol and to tailor it to their needs.

A client can use the Vend field by asking for certain types of information in the field when composing its request. The server can then respond to these requests, and it may also include parameters it wants the client to have, even if they were not requested. The original BOOTP protocol does not define any structure for the Vendor-Specific Area, leaving this up to each manufacturer to decide.

Obviously, there is nothing preventing a client made by one manufacturer from trying to send a request to a server made by another one. If each one is expecting the Vend field to contain something different, the results will be less than satisfactory. Thus, for the Vend field to be used properly, both devices must be speaking the same language when it comes to the meaning of this field. This is done by setting the first four bytes of the field to a special value. Each manufacturer chooses its own *magic number*, sometimes called a *magic cookie*, for this four-byte subfield.

NOTE *Why is it called a magic cookie? I'm not sure, to be honest. I have heard that its origin may be the cookie that Alice ate to grow or shrink in the story* Alice in Wonderland.

BOOTP Vendor Information Extensions

Including the Vend field in BOOTP gives the protocol extensibility for vendor-specific information. Unfortunately, the original field format didn't include any way of extending the information sent from a server to a client for *generic*, nonvendor-specific TCP/IP information. This was a significant oversight in the creation of the protocol, because there are many types of information that a TCP/IP host needs when it starts up that really have nothing to do with its vendor. For example, when a host boots, we probably want it to be told the address of a default router, the subnet mask for its local subnet, the address of a local DNS server, the MTU of the local network, and much more. None of these things are vendor-specific, but there is no place to put them in the BOOTP reply message.

Since there was no nonvendor-specific area field in BOOTP, the decision was made to define a way of using the Vendor-Specific Area (Vend field) for communicating this additional generic information. This was first standardized in RFC 1048 and then refined in later RFCs, as I explained in the BOOTP overview earlier in this chapter. This scheme basically represents one particular way of using the Vend field that most TCP/IP BOOTP implementations have chosen to adopt, regardless of their vendor. This enhancement is formally referred to as *BOOTP vendor information extensions*.

KEY CONCEPT The BOOTP message format includes a Vend field that was originally intended for vendor-specific customized fields. It was later changed to a place where additional generic information could be sent from a BOOTP server to a BOOTP client. Each such parameter is carried in a BOOTP *vendor information field*.

To clearly mark that this particular meaning of the Vend field is being used, a special, universal magic cookie value of 99.130.83.99 is inserted into the first four bytes of the field. Then the remaining 60 bytes can contain a sequence of one or more *vendor information fields*. The overall structure of the Vendor-Specific Area when vendor information extensions are used is shown in Figure 60-4.

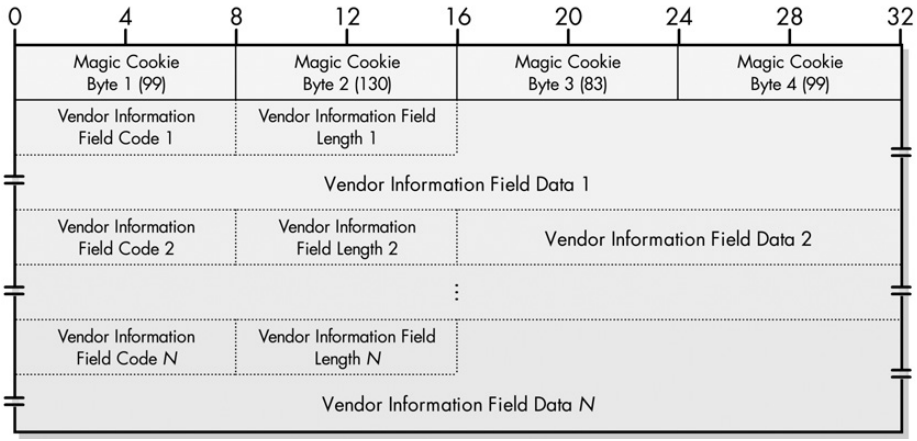


Figure 60-4: *BOOTP Vendor-Specific Area format showing vendor information fields*

NOTE *The BOOTP Vendor-Specific Area begins with the four-byte magic cookie and then contains a number of variable-length vendor information fields, each of which has the format shown above and in Table 60-3. Despite the use of IP dotted-decimal notation to represent the value 99.130.83.99, this is not an IP address. It's just a marker—a magic number that is universally recognized.*

BOOTP Vendor Information Fields

Each vendor information field specifies a particular type of information to be communicated, and it is encoded using a special subfield structure that specifies the field's type, length, and value. This is a common method of specifying options, called *TLV-encoding* (for *type*, *length*, *value*). The same basic method is used for encoding Internet Protocol versions 4 and 6 (IPv4 and IPv6) options. Table 60-3 shows the structure and the common names for the subfields of each vendor information field.

Table 60-3: BOOTP Vendor Information Field Format

Subfield Name	Size (Bytes)	Description
Code	1	Vendor Information Field Code: A single octet that specifies the vendor information field type.
Len	1	Vendor Information Field Length: The number of bytes in this particular vendor information field. This does not include the two bytes for the Code and Len fields.
Data	Variable	Vendor Information Field Data: The data being sent, which has a length indicated by the Len subfield, and which is interpreted based on the Code subfield.

There are two special cases that violate the field format shown in Table 60-3. A Code value of 0 is used as padding when subfields need to be aligned on word boundaries; it contains no information. The value 255 is used to mark the end of the vendor information fields. Both of these codes contain no actual data. To save space, when either is used, just the single Code value is included, and the Len and

Data fields are omitted. A device seeing a Code value of 0 just skips it as filler; a device seeing a Code value of 255 knows it has reached the end of the vendor information fields in this Vend field.

The vendor information extensions of BOOTP have become so popular that the use of this field for sending extra generic information is pretty much standard. In fact, I am not sure if anyone today still uses the Vend field solely for vendor-specific information.

When the vendor information extensions were introduced, one was created that points to a file where vendor-specific information can be found. This lets devices have the best of both worlds—they can use the standard vendor-independent fields and can incorporate vendor-specific fields (through the referenced file) where needed. Later, another field type was created that lets vendor-specific fields be mixed with vendor-independent ones directly in a BOOTP message.

When DHCP was created, the same vendor extension mechanism was maintained and enhanced further, but instead of the field being called vendor information extensions, it was renamed to *Options*. (A much better name!) The BOOTP vendor information fields were retained in DHCP, and new DHCP-specific options were defined. To avoid duplication, I have listed all the BOOTP vendor information fields and DHCP options in a set of tables in Chapter 63, which covers DHCP messaging. This includes a discussion of how vendor-specific and vendor-independent information can be mixed. You may also want to read the section in Chapter 63 that describes DHCP options, which discusses how they were created from BOOTP vendor information extensions.

BOOTP Relay Agents (Forwarding Agents)

One reason why RARP was quickly replaced by BOOTP is that RARP required the client being configured and the server providing it with an IP address to be on the same physical network. This is fine when you run a small organization with ten machines, which are probably all on the same physical network. Larger networks must be divided into multiple physical networks for efficiency, however. RARP would require a separate RARP server for each network, meaning needing to duplicate all the functions of a single server onto multiple machines. Worse yet, all the configuration information would also be duplicated, and any changes would need to be made to all the different servers each time.

BOOTP is designed to allow the BOOTP server and the clients it serves to be on different networks. This centralizes the BOOTP server and greatly reduces the amount of work required of network administrators. However, implementing this feature means increasing the complexity of the protocol. In particular, we need to involve a third-party device in the configuration process.

You might rightly wonder why this would be the case. RARP is a low-level protocol that works at the link layer, so that explains why it would have problems putting the client and server on different physical networks. But wasn't the whole point of making BOOTP a high-level protocol that it was able to use IP? And if BOOTP uses IP, can't we send from one network to another arbitrarily, just like any IP-based messaging protocol?

The answer is that even though we are indeed using IP and UDP, BOOTP still has one of the same issues that RARP had: a reliance on *broadcasts*. The client usually doesn't know the address of a server, so it must send out its request as a broadcast, saying in essence, "Can anyone hear this and give me the information I need?" For efficiency reasons, routers do not route such broadcasts, as they would clog the network. This means that if the server and client are not on the same network, the server can't hear the client's broadcast. Similarly, if the server ever did get the request and broadcast its reply back to the client, the client would never get it anyway. To make this all work, we need something to act as an intermediary between the client and the server: a *BOOTP relay agent*.

The Function of BOOTP Relay Agents

The job of a BOOTP relay agent is to sit on a physical network where BOOTP clients may be located and act as a proxy for the BOOTP server. The agent gets its name because it relays messages between the client and server, and thus enables them to be on different networks.

NOTE *BOOTP relay agents were originally called* forwarding agents. *RFC 1542 changed the name to make explicit the fact that BOOTP relaying was not the same as conventional IP datagram forwarding by regular routers.*

In practice, a BOOTP relay agent is not usually a separate piece of hardware. Rather, it's a software module that runs on an existing piece of hardware that performs other functions. It is common for BOOTP relay agent functionality to be implemented on an IP router. In that case, the router is acting both as a regular router and also playing the role of a BOOTP agent. The forwarding functions required of a BOOTP relay agent are distinct from the normal IP datagram forwarding tasks of a router.

KEY CONCEPT Since BOOTP uses broadcasts, the BOOTP client and BOOTP server must be on the same physical network to be able to hear each other's broadcasted transmissions. For a client and server on different networks to communicate, a third party is required to facilitate the transaction: a *BOOTP relay agent*. This device, which is often a router, listens for transmissions from BOOTP clients and relays them to the BOOTP server. The server responds back to the agent, which then sends the server's response back to the client.

Naturally, the placement of the client and server on different networks and the presence of a relay agent change the normal request/reply process of BOOTP significantly. A couple of specific fields in the BOOTP message format are used to control the process. RFC 951 was rather vague in describing how this process works, so RFC 1542 described it in much more detail.

Normal BOOTP Operation Using a Relay Agent

The following shows, in simplified form, a revised set of BOOTP operation steps when a relay agent is involved. To keep the size of this discussion manageable, I have omitted the details of the basic request/reply process to focus on the relaying functionality, which you can also see graphically in Figure 60-5.

Client Creates Request The client machine creates its request normally. The existence of a relay agent is totally transparent to the client.

Client Sends Request The client broadcasts the BOOTREQUEST message by transmitting it to address 255.255.255.255. (Note that in the case where a client already knows both its own address and the address of a BOOTP server, we don't need the relay agent at all—both the request and reply can be sent unicast over an arbitrary internetwork.)

Relay Agent Receives Request and Processes It The BOOTP relay agent on the physical network where the client is located is listening on UDP port 67 on the server's behalf. It processes the request as follows:

- It checks the value of the Hops field. If the value is less than or equal to 16, it increments it. If the value is greater than 16, it discards the request and does nothing further.
- It examines the contents of the GIAddr field. If this field is all zeros, it knows it is the first relay agent to handle the request and puts its own IP address into this field. (If the agent is a router, it has more than one IP address, so it chooses the one of the interface on which it received the request.)

Relay Agent Relays Request The relay agent sends the BOOTP request to the BOOTP server. If the relay agent knows the server's IP address, it will send it unicast directly to the server. Otherwise, if the agent is a router, it may choose to broadcast the request on a different interface from the one on which it received the request. In the latter case, it is possible that multiple relay agents may be required to convey the request to the server. See the next section for more on this.

Server Receives Request and Processes It The BOOTP server receives the relayed request from the BOOTP relay agent. It processes it as normal.

Server Creates Reply The server creates a reply message as normal.

Server Sends Reply Seeing that the GIAddr field in the request was nonzero, the server knows the request was relayed. Instead of trying to send its reply back to the client that sent the request, it transmits the reply unicast back to the relay agent specified in GIAddr.

Relay Agent Relays Reply The BOOTP relay agent transmits the BOOTREPLY message back to the client. It does this either unicast or broadcast, depending on the value of the CIAddr field and the B (Broadcast) flag, just as a server does in the nonrelay case.

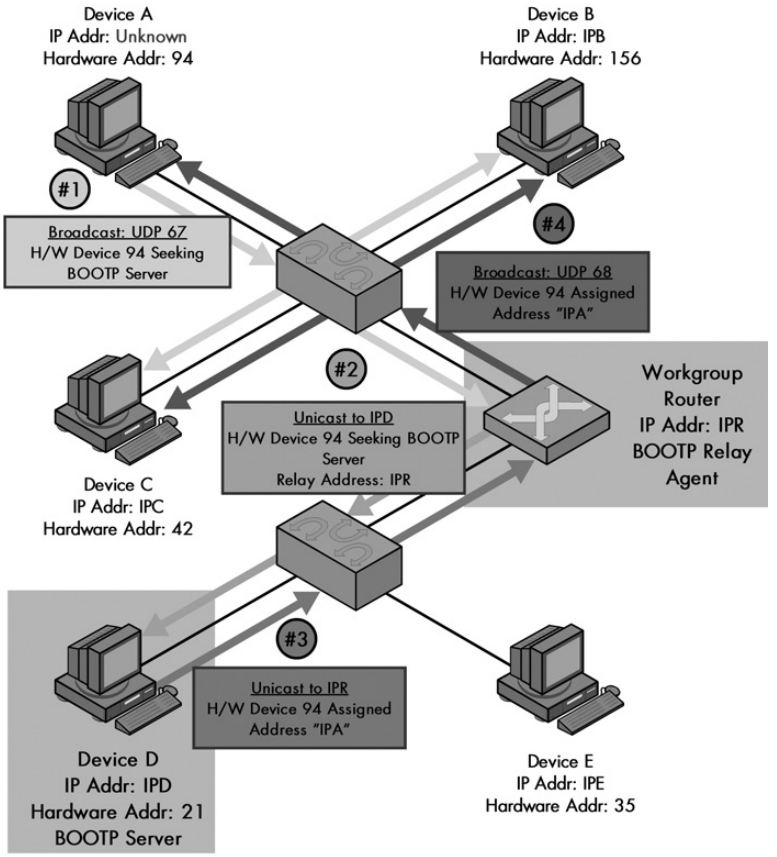


Figure 60-5: BOOTP operation using a relay agent *In this example, Device A is trying to access a BOOTP server, but the only one is on a different network; the two are connected by a workgroup router that is configured to act as a BOOTP relay agent. Device A broadcasts its request, which the router receives. It relays the request to the BOOTP server, Device D, and puts its own IP address (IPR) into the BOOTP GIAddr field. The BOOTP server sends the reply back to the router using address IPR. The router then broadcasts it on Device A's local network so that Device A can receive it.*

Relaying BOOTP Requests Using Broadcasts

The simplest case of relaying is when each network has a relay agent that knows the IP address of the BOOTP server. The relay agent captures the request in step 3 of the procedure described in the preceding section, and sends it directly to the BOOTP server, wherever it may be on the network. The request is relayed as a regular unicast UDP message and routed to the BOOTP server. The BOOTP server's reply is routed back to the BOOTP relay agent, just like any UDP message in an IP datagram, and the relay agent forwards the reply.

It is also possible to set up BOOTP relay agents to relay requests even if they don't know the BOOTP server's address. These agents take requests received on one network and relay them to the next, where they expect another agent to continue the relaying process until a BOOTP server is reached. For example, suppose we have a set of three networks. Network N1 is connected to Network N2 using Router RA, and Network N2 connects to Network N3 using Router RB. Both of these routers function as relay agents but don't know the IP address of the BOOTP server. Here's what would happen if a client on Network N1 sent a request and the server was on Network N3:

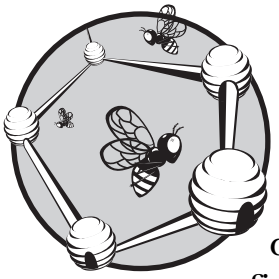
1. The client would send its request.
2. Router RA would capture the request and put its address into GIAddr. It would increment the Hops field to a value of 1 and then broadcast the request out on Network N2.
3. Router RB would capture this request. It would see there is already an address in GIAddr, so it would leave that alone. It would increment the Hops field to 2 and broadcast the request on Network N3.
4. The BOOTP server would receive the request, process it, and return the reply directly back to Router RA.
5. Router RA would relay the reply back to the client.

As you can see, the purpose of the Hops field is to ensure that errant requests don't circle around the network endlessly. Each relay agent increments it, and if the value of 16 is ever exceeded, the request is dropped. You can also see that any relay agents other than the first are involved only for handling the request; the reply is sent unicast back to the agent closest to the client.

Incidentally, if this multiple-step relaying process sounds like IP routing (only using broadcasts), and the Hops field sounds like the Time to Live (TTL) field in an IP datagram, then you've been paying attention. It is essentially the same idea (as explained in Chapter 21).

61

DHCP OVERVIEW AND ADDRESS ALLOCATION CONCEPTS



In some ways, technological advancement can be considered more a journey than a destination. When a particular technology is refined or replaced with a superior one, it's usually only a matter of time before it, too, is replaced with something better. And so it was with the TCP/IP Boot Protocol (BOOTP), described in the previous chapter. While BOOTP was far more capable than the protocol it replaced, Reverse Address Resolution Protocol (RARP), after a number of years BOOTP, itself was replaced with a new TCP/IP configuration protocol: the *Dynamic Host Configuration Protocol (DHCP)*.

Where BOOTP represented a revolutionary change from RARP, DHCP is more of an evolution of BOOTP. It was built using BOOTP as a foundation, with the same basic message format. The most significant addition in DHCP is the ability to *dynamically* assign addresses to clients and to centrally

manage them. It is this capability that makes DHCP so powerful. Today, DHCP is the standard TCP/IP host configuration protocol and is used in everything from single-client home networks to enterprise-class internetworks.

In this first chapter on DHCP, I provide an overview of the protocol and a description of the concepts behind DHCP address assignment and leasing. I take a high-level look at how DHCP address assignment works and give a description of the three DHCP address allocation mechanisms. I then delve into DHCP leases and the policies and techniques used to decide how to implement DHCP leasing. I provide an overview of the lease life cycle from start to finish and describe the two DHCP lease timers that help control the process. Finally, I describe DHCP lease address pools and ranges, and the general concepts behind address management.

RELATED INFORMATION *Since DHCP builds on BOOTP, they have a number of things in common. For example, DHCP makes use of BOOTP relay agent functionality, and DHCP options are basically the same as BOOTP vendor information fields. Since DHCP is the more common of the two protocols, I have tried to be complete in describing the operation of these features here, highlighting especially any differences between how they work for DHCP and in BOOTP. However, I have avoided duplicating the history and reasoning for the existence of many of these features. Since BOOTP came first, I have placed more of the historical information in the previous chapter. In general, if you plan to read about DHCP as well as BOOTP, I recommend reading the chapter on BOOTP first. If you don't plan to read up on BOOTP, you may wish to check the topic on DHCP/BOOTP interoperability in Chapter 64 instead.*

DHCP Overview, History, and Standards

As you learned in the previous chapter, BOOTP represents a significant improvement over RARP because it solves so many of RARP's problems. BOOTP is a higher-layer protocol, not hardware-dependent like RARP. It can support sending extra information beyond an IP address to a client to enable customized configuration. Also, through the use of BOOTP relay agents, it allows a large organization to use just one or two BOOTP servers to handle clients spread out over many physical networks. In so doing, BOOTP effectively solves one of the major classes of problems that administrators have with manual configuration: the "I have to go configure each host myself" issue. It allows "dumb" (storageless) hosts to configure themselves automatically and saves administrators the hassles of needing to trek to each host individually to specify important configuration parameters.

BOOTP normally uses a static method of determining what IP address to assign to a device. When a client sends a request, it includes its hardware address, which the server looks up in a table to determine the IP address for that client. (It is possible for BOOTP to use other methods of determining the relationship between an IP and hardware address, but static mapping is usually used.) This means BOOTP works well in relatively static environments, where changes to the IP addresses assigned to different devices are infrequent. Such networks were basically the norm in the 1980s and early 1990s.

Over time, many networks quickly started to move away from this model, for a number of reasons. As computers became smaller and lighter, it was more common for them to move from one network to another, where they would require a different address using the new network's network ID. Laptop and even palmtop computers could literally move from one network to another many times per day. Another

major issue was the looming exhaustion of the IP address space (see Chapter 17). For many organizations, permanently assigning a static IP address to each and every computer that might connect to their network was a luxury they could not afford.

In many organizations, trying to keep track of constant IP address changes became a daunting task in and of itself. BOOTP, with its static table of mappings between hardware addresses and IP addresses, simply wasn't up to the task. It also offered no way to reuse addresses; once an address had been assigned, a device could keep it forever, even if it were no longer needed.

DHCP: Building on BOOTP's Strengths

A new host configuration protocol was needed to serve modern networks, which would move away from static, permanent IP address assignment. The Internet Engineering Task Force (IETF) supplied this in the form of DHCP, first formalized in RFC 1541, published in October 1993. (Actually, it was really originally specified in RFC 1531 in that same month, but due to minor errors in 1531, the standard was quickly revised and 1541 published.)

Because BOOTP worked well within its limitations and was also already widely deployed, it would not have made sense to start over from scratch with DHCP. This was especially so given that such a decision would have meant dealing with the inevitable painful transition, as well as compatibility problems associated with having both BOOTP and DHCP around for many years.

So, instead of tossing out BOOTP, DHCP was built on it as a foundation. In it simplest form, DHCP consists of two major components: an address allocation mechanism and a protocol that allows clients to request configuration information and servers to provide it. DHCP performs both functions in a manner similar to BOOTP, but with improvements.

Overview of DHCP Features

The most significant differences between BOOTP and DHCP are in the area of address allocation, which is enhanced through the support for *dynamic* address assignment. Rather than using a static table that absolutely maps hardware addresses to IP addresses, a *pool* of IP addresses is used to dynamically allocate addresses. Dynamic addressing allows IP addresses to be efficiently allocated, and even shared among devices. At the same time, DHCP still supports static mapping of addresses for devices where this is needed.

The overall operation and communication between clients and servers are similar to that used by BOOTP, but with changes. The same basic request/reply protocol using UDP was retained for communicating configuration information, but additional message types were created to support DHCP's enhanced capabilities. BOOTP relay agents can be used by DHCP in a manner very similar to how they are used by BOOTP clients and server. The vendor information extensions from BOOTP were retained as well, but were formalized, renamed *DHCP options*, and extended to allow the transmission of much more information.

The result of all of this development effort is a widely accepted, universal host configuration protocol for TCP/IP that retains compatibility with BOOTP while significantly extending its capabilities. Today, DHCP is found on millions of networks

worldwide. It is used for everything from assigning IP addresses to corporate networks with thousands of hosts, to allowing a home Internet access router to automatically providing the correct Internet configuration information to a single user's computer.

KEY CONCEPT The *Dynamic Host Configuration Protocol (DHCP)* is the host configuration protocol currently used on modern TCP/IP internetworks. It was based on BOOTP and is similar to its predecessor in many respects, including the use of request/reply message exchanges and a nearly identical message format. However, DHCP includes added functionality, the most notable of which is *dynamic address assignment*, which allows clients to be assigned IP addresses from a shared pool managed by a DHCP server.

The original DHCP specification was revised in March 1997 with the publishing of RFC 2131, also titled "Dynamic Host Configuration Protocol." This standard defined another new DHCP message (DHCPINFORM) type to allow active IP hosts to request additional configuration information. It also made several other small changes to the protocol. Since that time, numerous other DHCP-related RFCs have been published, most of which either define new DHCP option types (other kinds of information DHCP servers can send to DHCP clients) or slightly refine the way that DHCP is used in particular applications.

DHCP Address Assignment and Allocation Mechanisms

The two main functions of DHCP are to provide a mechanism for assigning addresses to hosts and to provide a method by which clients can request addresses and other configuration data from servers. Both functions are based on the ones implemented in DHCP's predecessor, BOOTP, but the changes are much more significant in the area of address assignment than they are in communication. It makes sense to start our look at DHCP here, since this will naturally lead us into a detailed discussion of defining characteristic of DHCP: *dynamic addressing*.

DHCP Address Allocation

Providing an IP address to a client is the most fundamental configuration task performed by a host configuration protocol. To provide flexibility for configuring addresses on different types of clients, the DHCP standard includes three different address allocation mechanisms: manual, automatic, and dynamic.

I don't really care for the names *automatic* and *dynamic* allocation, because they don't do a good job of clearly conveying the differences between these methods. Both methods can be considered automatic, because in each, the DHCP server assigns an address without requiring any administrator intervention. The real difference between them is only in how long the IP address is retained, and therefore, whether a host's address varies over time. I think better names would be *static* or *permanent* automatic allocation and *dynamic* or *temporary* automatic allocation.

Regardless of what you call them, all three of these methods exist for configuring IP hosts using DHCP. It is not necessary for administrators to choose one over the others. Instead, they will normally combine the methods, using each where it makes the most sense.

DHCP Manual Allocation

With manual allocation, a particular IP address is preallocated to a single device by an administrator. DHCP communicates only the IP address to the device.

Manual allocation is the simplest method, and it is equivalent to the method BOOTP uses for address assignment, described in the previous chapter. Each device has an address that an administrator gives it ahead of time, and all DHCP does is look up the address in a table and send it to the client for which it is intended. This technique makes the most sense for devices that are mainstays of the network, such as servers and routers. It is also appropriate for other devices that must have a stable, permanent IP address.

Okay, now here's a fair question you might have. DHCP acts basically like BOOTP in the case of manual allocation. But BOOTP was created for devices that needed help with configuration. Servers and routers are complex devices with their own internal storage, and they obviously don't need a DHCP server to tell them their IP address as a diskless workstation does, so why bother using DHCP for them at all?

Well, in fact, you could just manually assign the address to the device directly and tell DHCP to ignore those addresses. However, using DHCP for manual assignments yields a different benefit: an administrative one. It keeps all the IP address information centralized in the DHCP address database, instead of requiring an administrator to go from machine to machine checking addresses and ensuring there are no duplicates. Updates can be made in a single place as well.

DHCP Dynamic Allocation

While manual allocation is possible in DHCP, dynamic allocation is its real *raison d'etre*. With dynamic allocation, DHCP assigns an IP address from a pool of addresses for a limited period of time chosen by the server, or until the client tells the DHCP server that it no longer needs the address. An administrator sets up the *pool* (usually a range or set of ranges) of IP addresses that are available for use. Each client that is configured to use DHCP contacts the server when it needs an IP address. The server keeps track of which IP addresses are already assigned, and it *leases* one of the free addresses from the pool to the client. The server decides the amount of time that the lease will last. When the time expires, the client must either request permission to keep using the address (*renew* the lease) or must get a new one.

Dynamic allocation is the method used for most client machines in modern DHCP-enabled IP internetworks. It offers numerous benefits, such as the following:

Automation Each client can be automatically assigned an IP address when it is needed, without any administrator intervention. Administrators do not need to manually decide which address goes with which client.

Centralized Management All the IP addresses are managed by the DHCP server. An administrator can easily look to see which devices are using which addresses and perform other network-wide maintenance tasks.

Address Reuse and Sharing By limiting the amount of time that each device holds an IP address, the DHCP server can ensure that the pool of IP addresses is used only by devices actively using the network. After a period of time, addresses no longer being used are returned to the pool, allowing other devices to use them. This allows an internetwork to support a total number of devices larger than the number of IP addresses available, as long as not all the devices connect to the internetwork at the same time.

Portability and Universality BOOTP (and DHCP manual allocation) both require that the DHCP server know the identity of each client that connects to it, so the server can find the client's assigned address. With dynamic allocation, there are no predefined allocations, so any client can request an IP address. This inherently makes dynamic allocation the ideal choice for supporting mobile devices that travel between networks.

Conflict Avoidance Since IP addresses are all assigned from a pool that is managed by the DHCP server, IP address conflicts are avoided. This, of course, assumes that all the clients use DHCP. The administrator must ensure that the address pool is not used by non-DHCP devices.

DHCP Automatic Allocation

With the automatic allocation method, DHCP automatically assigns an IP address permanently to a device, selecting it from a pool of available addresses. This method can be used in cases where there are enough IP addresses for each device that may connect to the network, but where devices don't really care which IP address they use. Once an address is assigned to a client, that device will keep using it. Automatic allocation can be considered a special case of dynamic allocation: It is essentially dynamic allocation where the time limit on the use of the IP address by a client (the lease length) is forever.

In practice, automatic allocation is not used nearly as much as dynamic allocation, for a simple reason: Automatically assigning an IP address to a device permanently is a risky move. Most administrators feel it is better to use manual allocation for the limited number of machines that really need a permanent IP address assignment and to use dynamic addressing for others.

KEY CONCEPT DHCP defines three basic mechanisms for address assignment. *Dynamic allocation* is the method most often used, and it works by having each client *lease* an address from a DHCP server for a period of time. The server chooses the address dynamically from a shared address pool. *Automatic allocation* is like dynamic allocation, but the address is assigned permanently instead of being leased. *Manual allocation* preassigns an address to a specific device, just as BOOTP does, and is normally used only for servers and other permanent, important hosts.

DHCP Leases

Of the three address allocation methods supported by DHCP, dynamic address allocation is by far the most popular and important. The significance of the change that dynamic addressing represents to how IP addresses are used in TCP/IP can be seen in the semantics of how addresses are treated in DHCP. Where conventionally a host was said to *own* an IP address, when dynamic address allocation is used, hosts are said instead to *lease* an address.

The notion of a lease conveys very accurately the difference between dynamic allocation and the other types. A host no longer is strictly entitled to a particular address, with a server merely telling it what the address is. In DHCP, the server remains the real owner of all the IP addresses in the address pool, and it merely gives permission for a client to use the address for a period of time. The server guarantees that it will not try to use the address for another client only during this time. The client is responsible for taking certain actions if it wants to continue using the address. If it does not successfully reacquire permission for using the address after a period of time, it must stop using it or risk creating an IP address conflict on the network.

KEY CONCEPT DHCP's most significant new feature is dynamic allocation, which changes the way that IP addresses are managed. Where in traditional IP each device owns a particular IP address, in DHCP the server owns all the addresses in the address pool, and each client *leases* an address from the server, usually for only a limited period of time.

DHCP Lease Length Policy

When dynamic address allocation is used, the administrator of the network must provide parameters to the DHCP server to control how leases are assigned and managed. One of the most important decisions to be made is the *lease length policy* of the internetwork: how long the administrator wants client leases to last. This choice will depend on the network, the server, and the clients. The choice of lease time, like so many other networking parameters, comes down to a trade-off between *stability* and *allocation efficiency*.

The primary benefit of using long lease times is that the addresses of devices are relatively stable. A device doesn't need to worry about its IP address changing all the time, and neither does its user. This is a significant advantage in many cases, especially when it is necessary for the client to perform certain server functions, accept incoming connections, or use a DNS domain name (ignoring for the moment dynamic DNS capabilities). In those situations, having the IP address of a device moving all over the place can cause serious complications.

The main drawback of using long leases is that they substantially increase the amount of time that an IP address, once it is no longer needed, is tied up before it can be reused. In the worst-case scenario, the amount of wasted time for an allocation can be almost as long as the lease itself. If we give a device a particular address for six months and after two weeks the device is shut down and no longer used, the IP address that it was using is still unavailable for another five and a half more months.

For this reason, many administrators prefer to use short leases. This forces a client to continually renew the lease as long as it needs it. When it stops asking for permission, the address is quickly put back into the pool. This makes shorter leases a better idea in environments where the number of addresses is limited and must be conserved. The drawback is the opposite of the benefit of long leases: constantly changing IP addresses.

Administrators do not need to pick from short and long lease durations. They can compromise by choosing a number that best suits the network. The following are some examples of lease times and the reasoning behind them:

One Hour or Less Ensures maximum IP address allocation efficiency in a very dynamic environment where there are many devices connecting and disconnecting from the network, and the number of IP addresses is limited.

One Day Suitable for situations where guest machines typically stay for a day, to increase IP efficiency when many employees work part time, or otherwise to ensure that every day each client must ask again for permission to use an address.

Three Days The default used by Microsoft, which alone makes it a popular choice.

One Week A reasonable compromise between the shorter and longer times.

One Month Another compromise, closer to the longer end of the lease time range.

Three Months Provides reasonable IP address stability so that addresses don't change very often in reasonably static environments. Also a good idea if there are many IP addresses available and machines are often turned off for many days or weeks at a time. For example, this duration may be used in a university setting to ensure that IP addresses of returning students are maintained over the summer recess.

One Year An approximation of an infinite lease.

Not only is the administrator not restricted to a limited number of possible lease durations, it is not necessary for the administrator to choose a constant lease length policy for all clients. Depending on the capabilities of the DHCP server, an administrator may select different lease lengths for certain clients. For example, the administrator may decide to use long leases for desktop computers that are permanently assigned to a particular subnet and not moved, and a pool of short-leased addresses for notebook computers and visitors. In some DHCP implementations, this can be done by assigning clients to particular classes. Of course, this requires more work (and may even require multiple servers).

In selecting a lease time policy, the administrator must also bear in mind that, by default, after half the length of a lease, the client will begin attempting to renew the lease. This may make it more advisable to use a longer lease time, to increase the amount of time between when a client tries to renew the lease and when the lease expires. For example, in a network with a single DHCP server, an administrator may want to use leases no shorter than eight hours. This provides a four-hour window for maintenance on the server without leases expiring.

When a lease is very short, such as minutes or hours, it will typically expire when a client machine is turned off for a period of time, such as overnight. Longer leases will persist across reboots. The client in this case will still contact the DHCP server each time it is restarted to *reallocate* the address—confirm that it may continue using the address it was assigned.

KEY CONCEPT A key decision that a network administrator using DHCP must make is what the network's *lease length policy* will be. Longer leases allow devices to avoid changing addresses too often; shorter leases are more efficient in terms of reallocating addresses that are no longer required. An administrator can choose from a variety of different lease times and may choose longer leases for some devices than for others.

Issues with Infinite Leases

In addition to choosing a particular lease length number, it is possible to specify an infinite lease length duration for certain clients. This effectively turns dynamic allocation into automatic allocation for a particular client. As I said earlier, however, this is generally not done. The reason is that an infinite lease never expires, and as the old saw goes, "Never is a long time."

Permanently assigning an IP address from a pool is a somewhat risky move, because once assigned, if anything occurs that causes that address to be no longer used, it can never be recovered. A worst-case scenario would be a visitor to a company site who plugs a notebook computer in to the network to check email or transfer a file. If that machine is assigned an IP address using automatic allocation, the visitor will take it with him when he leaves. Obviously, this is not a great idea.

For this reason, most administrators prefer to use dynamic allocation instead, with addresses set to a very long time frame, such as a year or two years. This is considered near enough to infinity that it approximates a permanent assignment, but allows an IP address to eventually be recovered if a device stops using it. In such a policy, anything that really, truly needs a permanent assignment is given an address using *manual* assignment, which requires a conscious decision to dedicate the address to a particular device.

RELATED INFORMATION *For a little more information related to lease length selection, see the section on DHCP server implementation problems and issues in Chapter 64.*

DHCP Lease Life Cycle and Lease Timers

The use of dynamic address allocation in DHCP means a whole new way of thinking about addresses. A client no longer owns an address, but rather leases it. This means that when a client machine is set to use DHCP dynamic addressing, it can never assume that it has an address on a permanent basis. Each time it powers up, it must engage in communications with a DHCP server to begin or confirm the lease of an address. It also must perform other activities over time to manage this lease and possibly terminate it.

Calling dynamic address assignments leases is a good analogy, because a DHCP IP address lease is similar to a real-world lease in a number of respects. For example, when you rent an apartment, you sign the lease. Then you use the apartment for a

period of time. Typically, assuming you are happy with the place, you will *renew* the lease before it expires, so you can keep using it. If by the time you get near the end of the lease the owner of the apartment has not allowed you to renew it, you will probably lease a different apartment to ensure you have somewhere to live. And if you decide, say, to move out of the country, you may terminate the lease and not get another.

DHCP Lease Life Cycle Phases

DHCP leases follow a lease life cycle that generally consists of the following six phases:

Allocation A client begins with no active lease, and hence, no DHCP-assigned address. It acquires a lease through a process of *allocation*.

Reallocation If a client already has an address from an existing lease, then when it reboots or starts up after being shut down, it will contact the DHCP server that granted it the lease to confirm the lease and acquire operating parameters. This is sometimes called *reallocation*; it is similar to the full allocation process but shorter.

Normal Operation Once a lease is active, the client functions normally, using its assigned IP address and other parameters during the main part of the lease. The client is said to be *bound* to the lease and the address.

Renewal After a certain portion of the lease time has expired, the client will attempt to contact the server that initially granted the lease to *renew* the lease, so it can keep using its IP address.

Rebinding If renewal with the original leasing server fails (because, for example, the server has been taken offline), the client will try to *rebind* to any active DHCP server, in an attempt to extend its current lease with any server that will allow it to do so.

Release The client may decide at any time that it no longer wishes to use the IP address it was assigned, and may terminate the lease, *releasing* the IP address. Like the apartment renter moving out of the country, this may be done if a device is moving to a different network, for example. (Of course, unlike DHCP servers, landlords usually don't let you cancel a lease at your leisure, but hey, no analogy is perfect.)

Figure 61-1 illustrates the DHCP life cycle using an example that spans three individual leases.

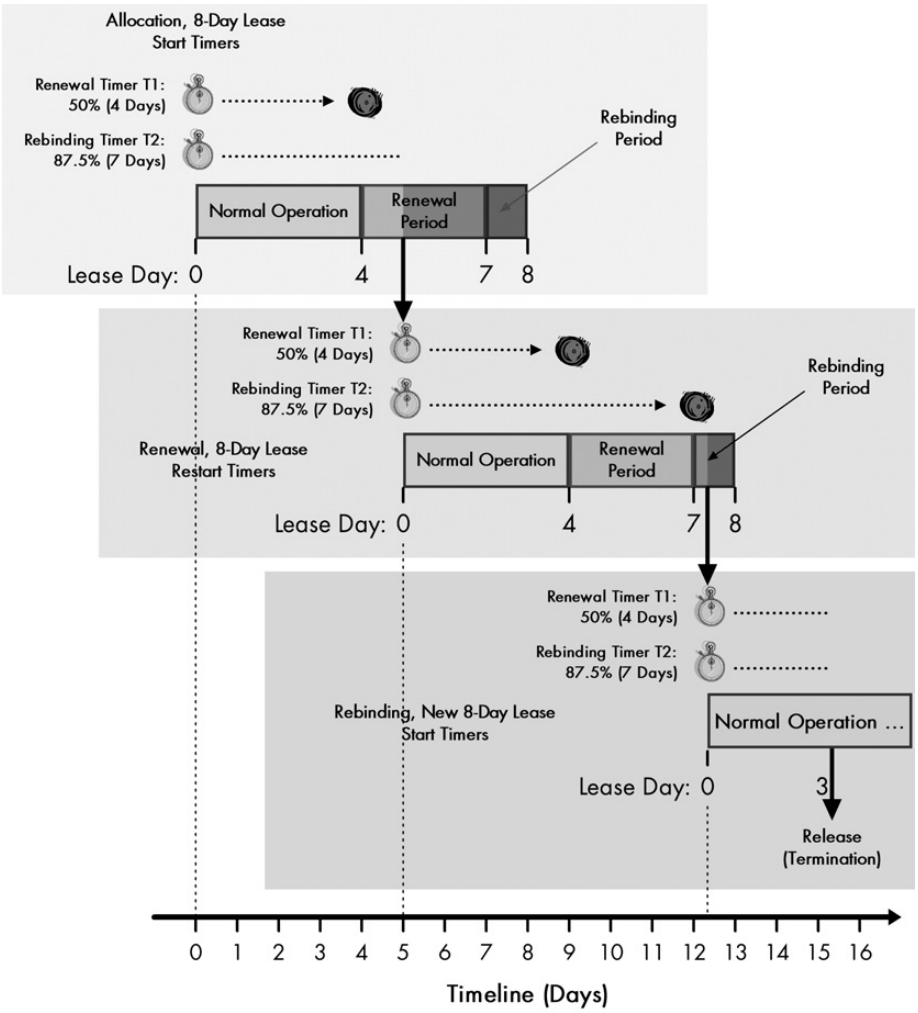


Figure 61-1: DHCP life cycle example *In this example, the initial lease has a duration of 8 days and begins at day 0. The T1 and T2 timers are set for 4 days and 7 days, respectively. When the T1 timer expires, the client enters the renewal period and successfully renews at day 5 with a new 8-day lease. When this second lease's T1 timer expires, the client is unable to renew with the original server. It enters the rebinding period when its T2 timer goes off, and it is granted a renewed 8-day lease with a different server. Three days into this lease, it is moved to a different network and no longer needs its leased address, so it voluntarily releases it.*

Renewal and Rebinding Timers

The processes of renewal and rebinding are designed to ensure that a client's lease can be extended before it is scheduled to end, so no loss of functionality or interruption occurs to the user of the client machine. Each time an address is allocated or reallocated, the client starts two timers that control the renewal and rebinding process:

Renewal Timer (T1) This timer is set by default to 50 percent of the lease period. When it expires, the client will begin the process of renewing the lease. It is simply called *T1* in the DHCP standards.

Rebinding Timer (T2) This timer is set by default to 87.5 percent of the length of the lease. When it expires, the client will try to rebind, as described in the previous section. It is given the snappy name *T2* in the DHCP standards.

Naturally, if the client successfully renews the lease when the T1 timer expires, this will result in a fresh lease, and both timers will be reset. T2 comes into play only if the renewal is not successful. It is possible to change the amount of time to which these timers are set, but obviously T1 must expire before T2, which must expire before the lease itself ends. These usually are not changed from the default, but may be modified in certain circumstances.

KEY CONCEPT DHCP leases follow a conceptual *life cycle*. The lease is first assigned to the client through a process of *allocation*; if the device later reboots, it will *reallocate* the lease. After a period of time controlled by the *renewal timer (T1)*, the device will attempt to *renew* its lease with the server that allocated it. If this fails, the *rebinding timer (T2)* will go off, and the device will attempt to *rebind* the lease with any available server. The client may also *release* its IP address if it no longer needs it.

The lease life cycle is described in the DHCP standards in the form of states that the client moves through as it acquires a lease, uses it, and then either renews or ends it. The next chapter describes these states and the specific exchanges of messages between a client and server to accomplish different lease activities.

DHCP Lease Address Pools, Ranges, and Address Management

Simpler host configuration methods such as BOOTP (or DHCP manual allocation for that matter) associate a single IP address with each client machine. DHCP dynamic addressing removes this one-to-one correspondence, in favor of flexible address mapping to clients on an as-needed basis. The clients no longer own the addresses, but lease them from the true owner, the server. Thus, a primary job of both a DHCP server and the administrator of that server is to maintain and manage these client addresses.

Address Pool Size Selection

The set of all addresses that a DHCP server has available for assignment is most often called the *address pool*. The first issue related to address management is ensuring that the address pool is large enough to serve all the clients that will be using the server. The number of addresses required depends on several factors:

Number of Clients This is an obvious factor.

Stability and Frequency of Use of Clients If most clients are left on and connected to the network all the time, you will probably need to plan on an address for each one. In contrast, if you are serving part-time employees or consultants who frequently travel, you can get away with sharing a smaller number of addresses.

Consequences of Overallocation If having certain clients be unable to get a free address is a problem, you need to more carefully manage the address pool to ensure that you don't run out of IP addresses. If having a client not get an address is *never* acceptable, make sure you have as many or more addresses as clients.

I'm sure you've probably noticed that these issues are similar to those that I raised in discussing lease lengths earlier in this chapter. In fact, the two matters are intimately related. Generally speaking, having more addresses gives the administrator the luxury of using longer leases. If you are short on addresses, you probably need to use shorter leases to reduce the chances of any unused addresses continuing to be allocated to devices not needing them.

Lease Address Ranges (Scopes)

In its simplest form, the address pool takes the form of a list of all addresses that the DHCP server has reserved for dynamic client allocation. Along with each address, the server stores certain parameters, such as a default lease length for the address and other configuration information to be sent to the client when it is assigned that address (for example, a subnet mask and the address of a default router). All of this data is stored in a special database on the server.

Of course, many clients will request addresses from this pool. Most of these clients are equals as far as the DHCP server is concerned, and it doesn't matter which address each individual client gets. This means most of the information stored with each of the addresses in a pool may be the same, except for the address number itself. Due to this similarity, it would be inefficient to need to specify each address and its parameters individually. Instead, a *range* of addresses is normally handled as a single group defined for a particular network or subnet. These are not given any particular name in the DHCP standards, but are commonly called *scopes*. This term has been popularized by Microsoft in its DHCP server implementations. Other operating systems sometimes just call these blocks of addresses *ranges*, but I prefer scope.

KEY CONCEPT Each DHCP server maintains a set of IP addresses that it uses to allocate leases to clients. These are usually contiguous blocks of addresses assigned to the server by an administrator, called DHCP *address ranges* or *scopes*.

The exact method for setting up scopes depends on the particular operating system and DHCP server software. However, each scope definition typically begins by specifying a range of addresses using a starting and an ending IP address. For example, if a company was assigned the IP address block 111.14.56.0/24, the administrator might set up a scope encompassing addresses 111.14.56.20 through 111.14.56.254, as shown in Figure 61-2. Then for that scope, the administrator can set up various parameters to be specified to each client assigned an address from the scope.

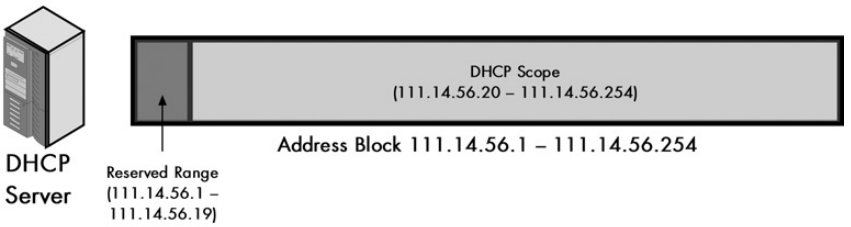


Figure 61-2: DHCP scope *A single DHCP server scope, encompassing addresses 111.14.56.1 through 111.14.56.254.*

Why not start at 111.14.56.1? Usually, we will want to set aside certain IP addresses for manual configuration of servers, routers, and other devices requiring a fixed address. One easy way to do that is to simply reserve a block of addresses that aren't used by DHCP. Alternatively, most DHCP server software will allow you to specify a range but *exclude* an address or set of addresses from the range. So, we could specify 111.14.56.1 through 111.14.56.254 and individually mark as not available addresses we manually assign. Or we could specify that 111.14.56.1 through 111.14.56.19 are reserved.

Instead of putting all of its addresses (except excluded ones) in a single scope, a server may use *multiple* scopes. One common reason for the latter approach is to support more than one subnet on a server. Multiple scopes are also commonly used when multiple DHCP servers are used to serve the same clients. There are two ways to do this: by having either *overlapping* or *non-overlapping* scopes.

Overlapping scopes allows each server to assign any address from the same pool. However, the DHCP standard doesn't specify any way for servers to communicate with each other when they assign an address, so if both servers were told they could assign addresses from the same address pool, this could result in both servers trying to assign a particular address to two different devices. As a result, if you are using two DHCP servers (as is often recommended for redundancy reasons), the administrator generally gives them different, non-overlapping scope assignments. Alternatively, the same scope is given to each server, with each server told to exclude from use the addresses the other server is assigning.

For example, suppose we have two DHCP servers: Server A (the main server) and Server B (the backup). We want to assign most of the addresses to Server A and a few as backup to Server B. We could give both Server A and Server B the scope 111.14.56.1 through 111.14.56.254. We would exclude 111.14.56.1 through 111.14.56.19 from both. Then we would exclude from Server A the range 111.14.56.200 through 111.14.56.254 and exclude from Server B the range 111.14.20 through 111.14.56.199. Figure 61-3 shows how this would work. The

main advantage of this method is that if one server goes down, the administrator can quickly remove the exclusion and let the remaining server access all addresses. Also, if one server runs out of addresses while the other has plenty, the allocations can be shifted easily.

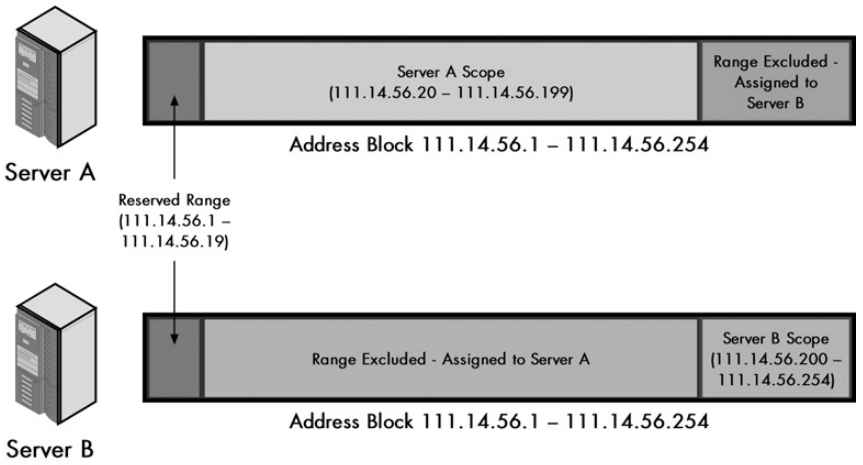


Figure 61-3: DHCP multiple-server non-overlapping scopes *DHCP Servers A and B have been assigned non-overlapping scopes to ensure that they do not conflict. This has been done by starting with the same scope definition for both. The common reserved range is excluded from each. Then Server A has Server B's address range excluded (hatched area at right in the top bar), and Server B has Server A's range excluded (hatched area in the middle at bottom).*

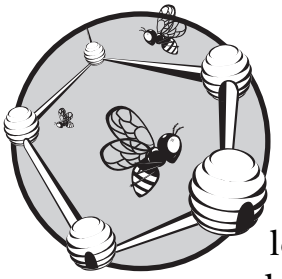
Other Issues with Address Management

There are many other issues related to address management, which start to get into the guts of DHCP server implementation. For example, as was the case with BOOTP, we may need to use relay agents when the DHCP server is responsible for addresses on a subnet different from its own. There are also special DHCP features that affect how addresses are managed. For example, the DHCP conflict detection feature can actually allow two servers to have overlapping scopes, despite what I said in the previous section. Chapter 64, which covers DHCP implementation and features, describes these issues in more detail.

KEY CONCEPT If a site has multiple DHCP servers, they can be set up with either *overlapping* or *non-overlapping* scopes. Overlapping scopes allow each server to assign from the same pool, providing flexibility, but raising the possibility of two clients being assigned the same address unless a feature such as *server conflict detection* is employed. non-overlapping scopes are safer because each server has a dedicated set of addresses for its use, but this means one server could run out of addresses while the other still has plenty, and if a server goes down, its addresses will be temporarily unallocatable.

62

DHCP CONFIGURATION AND OPERATION



The big news in DHCP is dynamic address allocation, along with the concept of address leasing. It is this new functionality that makes DHCP significantly more complex than its predecessor, the Boot Protocol (BOOTP). BOOTP is a simple request/reply protocol—a server only needs to look up a client's hardware address and send back the client's assigned IP address and other parameters. In contrast, DHCP clients and servers must do much more to carry out both parameter exchange and the many tasks needed to manage IP address leasing.

In this chapter, I delve into the nuts and bolts of how DHCP operates. I begin with two background topics. The first provides an overview of the responsibilities of clients and servers in DHCP, and shows in general terms how they relate to each other. The second discusses DHCP configuration parameters and how they are stored and communicated.

In the rest of the chapter, I illustrate the operation of DHCP in detail. I explain the DHCP client *finite state machine*, which will give you a high-level look at the entire client lease life cycle, including address allocation,

reallocation, renewal, rebinding, and optionally, lease termination. This theoretical description is then used as the basis for several topics that explain the actual processes by which DHCP client lease activities occur. These show the specific actions taken by both client and server and when and how DHCP messages are created and sent. The last part of the chapter describes the special mechanism by which a device not using DHCP for address allocation can request configuration parameters.

DHCP Overview of Client and Server Responsibilities

DHCP is the newest and most current TCP/IP host configuration protocol. However, as you saw in the previous chapter, it wasn't built from scratch—it was designed as an extension of BOOTP. In many ways, DHCP is like BOOTP with more features, and this can be seen in the basic setup of the protocol and how it works.

Both BOOTP and DHCP are designed based on the common TCP/IP model of client/server operation (see Chapter 8). In any interaction, one device plays the role of client and the other server. Each has specific responsibilities and must send and receive messages following the protocol described in the DHCP standard. The difference is that where BOOTP involves relatively little work for servers and clients and uses a simple single-message exchange for communication, DHCP requires that both servers and clients do more, and it uses several types of message exchanges.

DHCP Server Responsibilities

A *DHCP server* is a network device that has been programmed to provide DHCP services to clients. The server plays a central role in DHCP because DHCP's main function is host configuration, and the server configures hosts (clients) that communicate with it. Smaller networks may have only a single server to support many clients, while larger networks may use multiple servers. Regardless of the number of servers, each will usually service many clients.

The following are the key responsibilities of servers in making DHCP work:

Address Storage and Management DHCP servers are the owners of the addresses used by all DHCP clients. The server stores the addresses and manages their use, keeping track of which addresses have been allocated and which are still available.

Configuration Parameter Storage and Management DHCP servers also store and maintain other parameters that are intended to be sent to clients when requested. Many of these are important configuration values that specify in detail how a client is to operate.

Lease Management DHCP servers use leases to dynamically allocate addresses to clients for a limited time. The DHCP server maintains information about each of the leases it has granted to clients, as well as policy information such as lease lengths.

Response to Client Requests DHCP servers respond to different types of requests from clients to implement the DHCP communication protocol. This includes assigning addresses; conveying configuration parameters; and granting, renewing, and terminating leases.

Administration Services To support all of its other responsibilities, the DHCP server includes functionality to allow a human administrator to enter, view, change, and analyze addresses, leases, parameters, and all other information needed to run DHCP.

DHCP Client Responsibilities

A *DHCP client* is any device that sends DHCP requests to a server to obtain an IP address or other configuration information. Due to the advantages of DHCP, most host computers on TCP/IP internetworks today include DHCP client software, making them potential DHCP clients if their administrator chooses to enable the function. There are several main responsibilities of a DHCP client:

Configuration Initiation The client takes the *active* role by initiating the communication exchange that results in it being given an IP address and other parameters. The server, in contrast, is *passive* and will not really do anything for the client until the client makes contact.

Configuration Parameter Management The client maintains parameters that pertain to its configuration, some or all of which may be obtained from a DHCP server.

Lease Management Assuming its address is dynamically allocated, the client keeps track of the status of its own lease. It is responsible for renewing the lease at the appropriate time, rebinding if renewal is not possible, and terminating the lease early if the address is no longer needed.

Message Retransmission Since DHCP uses the unreliable User Datagram Protocol (UDP, see Chapter 44) for messaging, clients are responsible for detecting message loss and retransmitting requests if necessary.

DHCP Client/Server Roles

The DHCP server and client obviously play complementary roles. The server maintains configuration parameters for all clients. Each client maintains its own parameters, as discussed in the next section.

IP address assignment and lease creation, renewal, rebinding, and termination are accomplished through specific exchanges using a set of eight DHCP message types, as discussed in the "DHCP General Operation and Client Finite State Machine" and "DHCP Lease Allocation, Reallocation, and Renewal" sections later in this chapter. To accomplish this messaging, special rules are followed to generate, address, and transport messages, as explained in Chapter 63.

DHCP Relay Agents

Like BOOTP, DHCP also supports a third type of device: the *relay agent*. Relay agents are neither clients nor servers, but rather intermediaries that facilitate cross-network communication between servers and clients. They are described in more detail in Chapter 64 (where you can also find more of the implementation details of servers and clients).

KEY CONCEPT *DHCP servers* are devices programmed to provide DHCP services to clients. They manage address information and other parameters and respond to client configuration requests. *DHCP clients* are TCP/IP devices that have been set to use DHCP to determine their configuration. They send requests and read responses, and are responsible for managing their own leases, including renewing or rebinding a lease when necessary.

DHCP Configuration Parameters, Storage, and Communication

One of the more important oversights in DHCP's predecessor, BOOTP, was that it allowed a server to tell a client only three pieces of information: its IP address, the name of the server it could use to download a boot file, and the name of the boot file to use. This was a result of BOOTP's legacy as a protocol created primarily to let diskless workstations be bootstrapped.

Obviously, the IP address is a very important parameter, but in modern networks it isn't the only one that a client needs to be given for it to function properly. A typical host needs to be given other essential information to allow it to know how it should operate on its local network and interact with other devices. For example, it needs to know the address of a default local router, the subnet mask for the subnet it is on, parameters for creating outgoing IP datagrams, and much more.

Configuration Parameter Management

The inability to specify additional configuration parameters in BOOTP was resolved by using the special BOOTP Vendor-Specific Area for vendor-independent *vendor information fields*, as first defined in RFC 1048. In DHCP, this idea has been extended further, and more important, formalized, as part of the effort to make DHCP a more general-purpose configuration tool. Configuration parameter storage, maintenance, and communication are no longer optional features; they are an essential part of the host configuration process.

Just as DHCP servers are the bosses that own and manage IP addresses, they also act as the repository for other configuration parameters that belong to DHCP clients. This centralization of parameter storage provides many of the same benefits that centralizing IP addresses in DHCP does: Administrators can check and adjust parameters in a single place, rather than needing to go to each client machine.

Each DHCP server is programmed with parameters that are to be communicated to clients in addition to an IP address when an address is assigned. Alternatively, a client that has already been assigned an address using some other mechanism may

still query the DHCP server to get parameter information, using the DHCPINFORM message type. (This was actually added to the protocol in RFC 2131; it was not in the original DHCP standard.)

Parameter Storage

The exact method of storage of client parameters is to some extent implementation-dependent. Typically, there will be some parameters that apply to all clients. For example, on a small network with only one router, that router will probably be the default router for every DHCP client, regardless of address.

The DHCP server will also have certain parameters that are client-specific. The IP address itself is an obvious example, but there are other parameters that may apply to only certain clients on a network. These parameters are stored in some sort of a database and indexed using a particular *client identifier*. The default identifier consists of the client's IP subnet number and its hardware address. Thus, when a server gets a request from a particular subnet, it can use the client's hardware address in the request to look up client-specific parameters and return them. The client identifier can be changed if a different identification scheme is desired.

Clients are also responsible for storing their own parameters. Many of these will be obtained from the DHCP server, although some may be supplied in other ways. The specific implementation of the client determines which parameters it considers important and how they are discovered.

Configuration Parameter Communication

Communication of configuration parameters between DHCP clients and servers is accomplished using *DHCP options*, which replace BOOTP vendor information fields. A number of options were defined when DHCP was first created, and additional new ones have been created over the years.

Today, there are several dozen DHCP options. Obviously, the ability to have so many different parameters automatically delivered to a client provides a great deal of host configuration flexibility to administrators. DHCP options are described further in Chapter 63.

DHCP General Operation and the Client Finite State Machine

Dynamic address allocation is probably the most important new capability introduced by DHCP. In the previous chapter, I discussed the significance of the change from IP address *ownership* to IP address *leasing*. I also provided a high-level look of the activities involved in leasing, by providing an overview of the DHCP lease life cycle.

An overview of this sort is useful to get a general handle on how leases work, but to really understand the mechanics of DHCP address assignment and client/server communication, you need more details on how the devices behave and the messages they send. One tool often employed by networking engineers to describe a protocol is a theoretical model called a *finite state machine (FSM)*. Using this technique, the protocol's specific behavior is illustrated by showing the different *states* a device can be in, what possible *transitions* exist from one state to another, what

events cause transitions to occur, and what *actions* are performed in response to an event. The TCP operational overview contains more general background information on FSMs (see Chapter 47).

The DHCP standard uses an FSM to describe the lease life cycle from the perspective of a DHCP client. The client begins in an initial INIT state where it has no lease and then transitions through various states as it acquires, renews, rebinds, and/or releases its IP address. The FSM also indicates which message exchanges occurs between the server and client at various stages.

NOTE *The DHCP standard does not describe the DHCP server's behavior in the form of a FSM; only the client's is described this way.*

Some people think FSMs are a little dense and hard to understand, and I can see why. You can skip this topic, of course, but I think the FSM provides a useful way of illustrating in a comprehensive way most of the behavior of a DHCP client.

Table 62-1 describes each of the DHCP client states, and summarizes the messages sent and received by the client in each, as well as showing the state transitions that occur in response. The FSM's states, events, and transitions are easier to envision in Figure 62-1, which also incorporates a shading scheme so you can see which states are associated with each of the main DHCP processes.

Table 62-1: DHCP Client Finite State Machine

State	**State Description**	**Event and Transition**
INIT	This is the initialization state, where a client begins the process of acquiring a lease. It also returns here when a lease ends or when a lease negotiation fails.	Client sends DHCPDISCOVER. The client creates a DHCPDISCOVER message and broadcasts it to try to find a DHCP server. It transitions to the SELECTING state.
SELECTING	The client is waiting to receive DHCPOFFER messages from one or more DHCP servers, so it can choose one.	Client receives offers, selects preferred offer, and sends DHCPREQUEST. The client chooses one of the offers it has been sent, and broadcasts a DHCPREQUEST message to tell DHCP servers what its choice was. It transitions to the REQUESTING state.
REQUESTING	The client is waiting to hear back from the server to which it sent its request.	Client receives DHCPACK, successfully checks that IP address is free. The client receives a DHCPACK message from its chosen server, confirming that it can have the lease that was offered. It checks to ensure that address is not already used, and assuming it is not, records the parameters the server sent it, sets the lease timers T1 and T2, and transitions to the BOUND state.
		Client receives DHCPACK, but IP address is in use. The client receives a DHCPACK message from its chosen server, confirming that it can have the lease that was offered. However, it checks and finds the address already in use. It sends a DHCPDECLINE message back to the server and returns to the INIT state.
		Client receives DHCPNAK. The client receives a DHCPNAK message from its chosen server, which means the server has withdrawn its offer. The client returns to the INIT state.

(continued)

Table 62-1: DHCP Client Finite State Machine (continued)

State	**State Description**	**Event and Transition**
INIT-REBOOT	When a client that already has a valid lease starts up after a power down or reboot, it starts here instead of the INIT state.	Client sends DHCPREQUEST. The client sends a DHCPREQUEST message to attempt to verify its lease and reobtain its configuration parameters. It then transitions to the REBOOTING state to wait for a response.
REBOOTING	A client that has rebooted with an assigned address is waiting for a confirming reply from a server.	Client receives DHCPACK, successfully checks that IP address is free. The client receives a DHCPACK message from the server that has its lease information, confirming that the lease is still valid. To be safe, the client checks anyway to ensure that the address is not already in use by some other device. Assuming it is not, the client records the parameters the server sent it and transitions to the BOUND state.
		Client receives DHCPACK, but IP address is in use. The client receives a DHCPACK message from the server that had its lease, confirming that the lease is still valid. However, the client checks and finds that while the client was offline, some other device has grabbed its leased IP address. The client sends a DHCPDECLINE message back to the server and returns to the INIT state to obtain a new lease.
		Client receives DHCPNAK. The client receives a DHCPNAK message from a server. This tells it that its current lease is no longer valid; for example, the client may have moved to a new network where it can no longer use the address in its present lease. The client returns to the INIT state.
BOUND	A client has a valid lease and is in its normal operating state.	Renewal timer (T1) expires. The client transitions to the RENEWING state.
		Client terminates lease and sends DHCPRELEASE. The client decides to terminate the lease (due to user command, for example). It sends a DHCPRELEASE message and returns to the INIT state.
RENEWING	A client is trying to renew its lease. It regularly sends DHCPREQUEST messages with the server that gave it its current lease specified, and waits for a reply.	Client receives DHCPACK. The client receives a DHCPACK reply to its DHCPREQUEST. Its lease is renewed, it restarts the T1 and T2 timers, and it returns to the BOUND state.
		Client receives DHCPNAK. The server has refused to renew the client's lease. The client goes to the INIT state to get a new lease.
		Rebinding timer (T2) expires. While the client is attempting to renew its lease, the T2 timer expires, indicating that the renewal period has ended. The client transitions to the REBINDING state.
REBINDING	The client has failed to renew its lease with the server that originally granted it and now seeks a lease extension with any server that can hear it. It periodically sends DHCPREQUEST messages with no server specified, until it gets a reply or the lease ends.	Client receives DHCPACK. Some server on the network has renewed the client's lease. The client binds to the new server granting the lease, restarts the T1 and T2 timers, and returns to the BOUND state.
		Client receives DHCPNAK. A server on the network is specifically telling the client it needs to restart the leasing process. This may be the case if a new server is willing to grant the client a lease, but only with terms different from the client's current lease. The client goes to the INIT state.
		Lease expires. The client receives no reply prior to the expiration of the lease. It goes back to the INIT state.

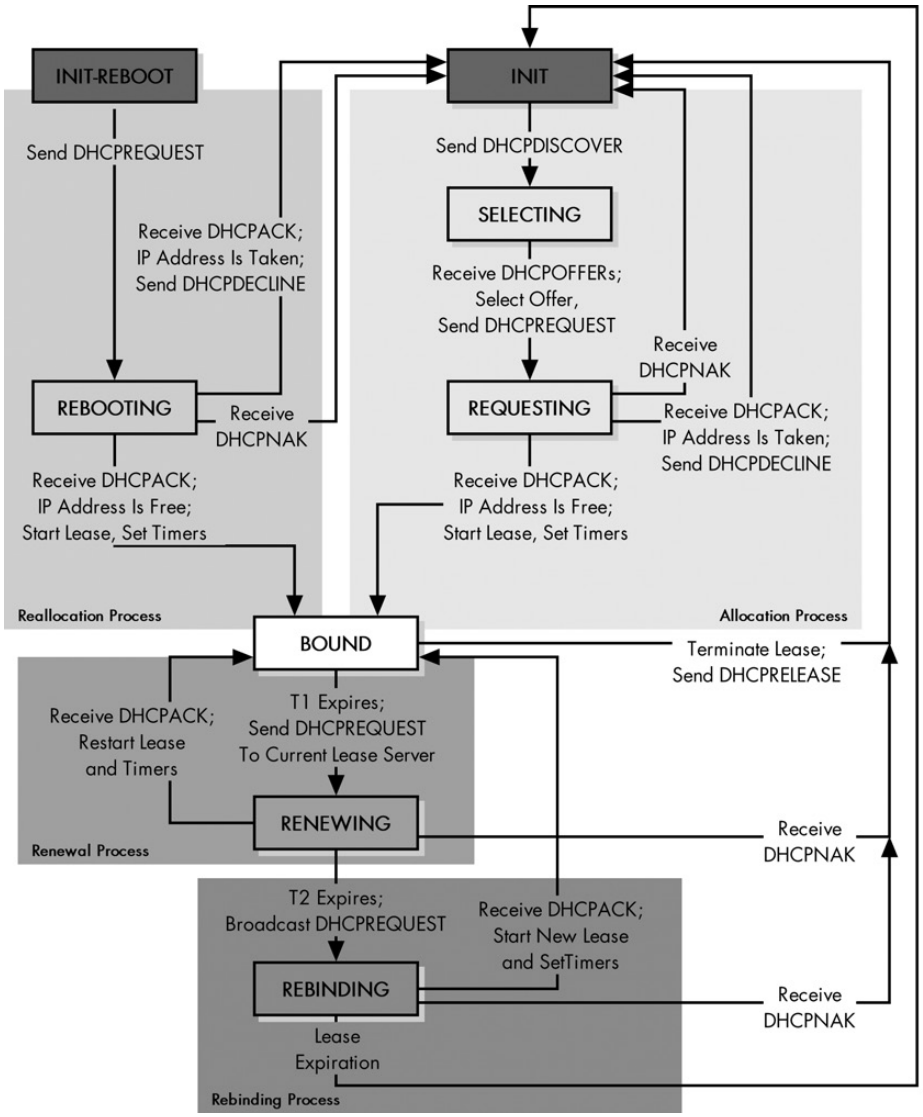


Figure 62-1: DHCP client finite state machine *This diagram shows the finite state machine (FSM) used by DHCP clients. The shaded background areas show the transitions taken by a DHCP client as it moves through the four primary DHCP processes: allocation, reallocation, renewal, and rebinding.*

This is just a summary of the FSM, and it does not show every possible event and transition, since it is complex enough already. For example, if a client that received two offers in the SELECTING state receives a DHCPNAK from its chosen server in the REQUESTING state, it may choose to send a new DHCPREQUEST to its second choice, instead of starting over from scratch. Also, the client must have logic that lets it time out if it receives no reply to sent messages in various states, such as not receiving any offers in the SELECTING state. The next sections discuss these matters in more detail.

Also note that this FSM applies to dynamically allocated clients—that is, ones with conventional leases. A device configured using automatic allocation will go through the same basic allocation process, but does not need to renew its lease. The process for manual allocation is somewhat different.

DHCP Lease Allocation, Reallocation, and Renewal

To implement DHCP, an administrator must first set up a DHCP server and provide it with configuration parameters and policy information: IP address ranges, lease length specifications, and configuration data that DHCP hosts will need to be delivered to them. Host devices can then have their DHCP client software enabled, but nothing will happen until the client initiates communication with the server. When a DHCP client starts up for the first time, or when it has no current DHCP lease, it will be in an initial state where it doesn't have an address and needs to acquire one. It will do so by initiating the process of *lease allocation*.

Before we examine the steps in the lease allocation, reallocation, and renewal processes, I need to clarify some issues related to DHCP lease communications. First, DHCP assumes that clients will normally broadcast messages, since they don't know the address of servers when they initiate contact, but that servers will send replies back unicast to the client. This can be done even before the client has an IP address, by sending the message at the link layer. Some clients don't support this and require that messages to them be broadcast instead.

DHCP uses many of the same basic fields as BOOTP, but much of the extra information the protocol requires is carried in DHCP *options*. Some of these options aren't really optional, despite the name—they are needed for the basic function of DHCP. An obvious example is the DHCP Message Type option, which is what specifies the message type itself.

The details of how messages are created and addressed, along with a full description of all DHCP fields and options, are presented in Chapter 63.

NOTE *I have assumed that no relay agents are in use here. See the discussion of DHCP/BOOTP relay agents in Chapter 60 for more on how they change the allocation process (and other processes).*

Initial Lease Allocation Process

The following are the basic steps taken by a DHCP client and server in the initial allocation of an IP address lease, focusing on the most important tasks each device performs (see Figure 62-2).

1. **Client Creates DHCPDISCOVER Message**

   The client begins in the INIT (initialization) state. It has no IP address and doesn't even know whether or where a DHCP server may be on the network. To find one, it creates a DHCPDISCOVER message, including the following information:

   - Its own hardware address in the CHAddr field of the message, to identify itself

- A random transaction identifier, put into the XID field (used to identify later messages as being part of the same transaction)

Optionally, the client may request a particular IP address using a Requested IP Address DHCP option, a particular lease length using an IP Address Lease Time option, and/or specific configuration parameters by including a Parameter Request List option in the message

2. **Client Sends DHCPDISCOVER Message**

   The client broadcasts the DHCPDISCOVER message on the local network. The client transitions to the SELECTING state, where it waits for replies to its message.

3. **Servers Receive and Process DHCPDISCOVER Message**

   Each DHCP server on the local network receives the client's DHCPDISCOVER message and examines it. The server looks up the client's hardware address in its database and determines if it is able to offer the client a lease and what the terms of the lease will be. If the client has made requests for a particular IP address, lease length, or other parameters, the server will attempt to satisfy these requests, but it is not required to do so. A server may decide not to offer a lease to a particular client if it has not been programmed to provide service for it, it has no remaining IP addresses, or for other reasons.

4. **Servers Create DHCPOFFER Messages**

   Each server that chooses to respond to the client creates a DHCPOFFER message including the following information:

   - The IP address to be assigned to the client, in the YIAddr field (if the server previously had a lease for this client, it will attempt to reuse the IP address it used last time; failing that, it will try to use the client's requested address, if present; otherwise, it will select any available address)
   - The length of the lease being offered
   - Any client-specific configuration parameters either requested by the client or programmed into the server to be returned to the client
   - Any general configuration parameters to be returned to all clients or clients in this client's class
   - The server's identifier in the DHCP Server Identifier option
   - The same transaction ID (XID) used in the DHCPDISCOVER message

5. **Servers Probe and/or Reserve Offered Address (Optional)**

   The DHCP standard specifies that before sending a DHCPOFFER to a client, the server should check to see that the IP address isn't already in use by sending an ICMP Echo message to that address. It is considered a key part of the DHCP server conflict detection feature (discussed in Chapter 64). This may be disabled by an administrator. Whether or not it probes the address offered, the server may also *reserve* the address so that if the client decides to use it, it will be available. This isn't mandatory, because the protocol handles the case where an offered lease is retracted. It is more efficient if servers do reserve addresses, but if IP addresses are in very short supply, such reservations may not be practical.

6. **Servers Send DHCPOFFER Messages**

   Each server sends its DHCPOFFER message. They may not all be sent at exactly the same time. The messages are sent either unicast or broadcast, as mentioned earlier.

7. **Client Collects and Processes DHCPOFFER Messages**

   The client waits for DHCPOFFER messages to arrive in response to its DHCPDISCOVER message. The exact behavior of the client here is implementation-dependent. The client may decide to simply take the first offer it receives, for expediency. Alternatively, it may choose to shop around by waiting for a period of time. It can then process each offer and take the one with the most favorable terms—for example, the one with the longest lease. If no DHCPOFFER messages are received, the client will enter a retransmission mode and try sending the DHCPDISCOVER again for a period of time.

8. **Client Creates DHCPREQUEST Message**

   The client creates a DHCPREQUEST message for the server offer it has selected. This message serves two purposes: It tells the server whose offer the client has accepted, "Yes, I accept your offer, assuming it is still available," and also tells the other servers, "Sorry, your offer was rejected." (Well, except for the "sorry" part; servers are pretty thick-skinned about rejection.) In this message, the client includes the following information:

   - The identifier of the chosen server in the DHCP Server Identifier option, so everyone knows who won
   - The IP address that the DHCP server assigned the client in the DHCPOFFER message, which the client puts in the Requested IP Address DHCP option as a confirmation
   - Any additional configuration parameters it wants in a Parameter Request List option in the message

9. **Client Sends DHCPREQUEST Message**

   The client sends the DHCPREQUEST message. Since it is intended for not just the selected DHCP server, but all servers, it is broadcast. After doing this, the client transitions to the REQUESTING state, where it waits for a reply from the chosen server.

10. **Servers Receive and Process DHCPREQUEST Message**

    Each of the servers receives and processes the client's request message. The servers not chosen will take the message as a rejection. However, a client may select one offer, attempt to request the lease, and have the transaction not complete successfully. The client may then come back and try its second-choice offer by sending a DHCPREQUEST containing a different Server Identifier. This means that if Server A receives a single DHCPREQUEST with a Server Identifier of Server B, that doesn't necessarily mean that Server A is finished with the transaction. For this reason, rejected servers will wait for a while before offering a previously offered lease to another client.

11. **Server Sends DHCPACK or DHCPNAK Message**

    The chosen server will see that its lease has been selected. If it did not previously reserve the IP address that was offered to the client, it must check to make sure it is still available. If it is not, the server sends back a DHCPNAK (*negative acknowledgment*) message, which essentially means, "Never mind, that lease is no longer available." Usually, however, the server will still have that lease. It will create a *binding* for that client, and send back a DHCPACK (*acknowledgment*) message that confirms the lease and contains all the pertinent configuration parameters for the client.

12. **Client Receives and Processes DHCPACK or DHCPNAK Message**

    The client receives either a positive or negative acknowledgment for its request. If the message is a DHCPNAK, the client transitions back to the INIT state and starts over—back to square one (step 1). If it is a DHCPACK, the client reads the IP address from the YIAddr field, and records the lease length and other parameters from the various message fields and DHCP options. If the client receives neither message, it may retransmit the DHCPREQUEST message one or more times. If it continues to hear nothing, it must conclude that the server flaked out and go back to step 1.

13. **Client Checks That Address Is Not in Use**

    The client device should perform a final check to ensure that the new address isn't already in use before it concludes the leasing process. This is typically done by generating an Address Resolution Protocol (ARP) request on the local network, to see if any other device thinks it already has the IP address this client was just leased. If another device responds, the client sends a DHCPDECLINE message back to the server, which basically means, "Hey server, you messed up. Someone is already using that address." The client then goes back to step 1 and starts over.

14. **Client Finalizes Lease Allocation**

    Assuming that the address is not already in use, the client finalizes the lease and transitions to the BOUND state. It also sets its two lease timers, T1 and T2. It is now ready for normal operation.

As you can see in this description, there are a number of situations that may occur that require a client to retransmit messages. This is because DHCP uses UDP, which is unreliable and can cause messages to be lost. If retransmissions don't fix a problem such as not receiving a DHCPOFFER or a DHCPACK from a server, the client may need to start the allocation process over from scratch. The client must include enough intelligence to prevent it from simply trying forever to get a lease when there may not be a point. For example, if there are no DHCP servers on the network, no number of retransmissions will help.

Thus, after a number of retries, the client will give up and the allocation process will fail. If the client is configured to use the Automatic Private IP Addressing (APIPA) feature (see Chapter 64), this is where it would be used to give the client a default address. Otherwise, the client will be, well, dead in the water.

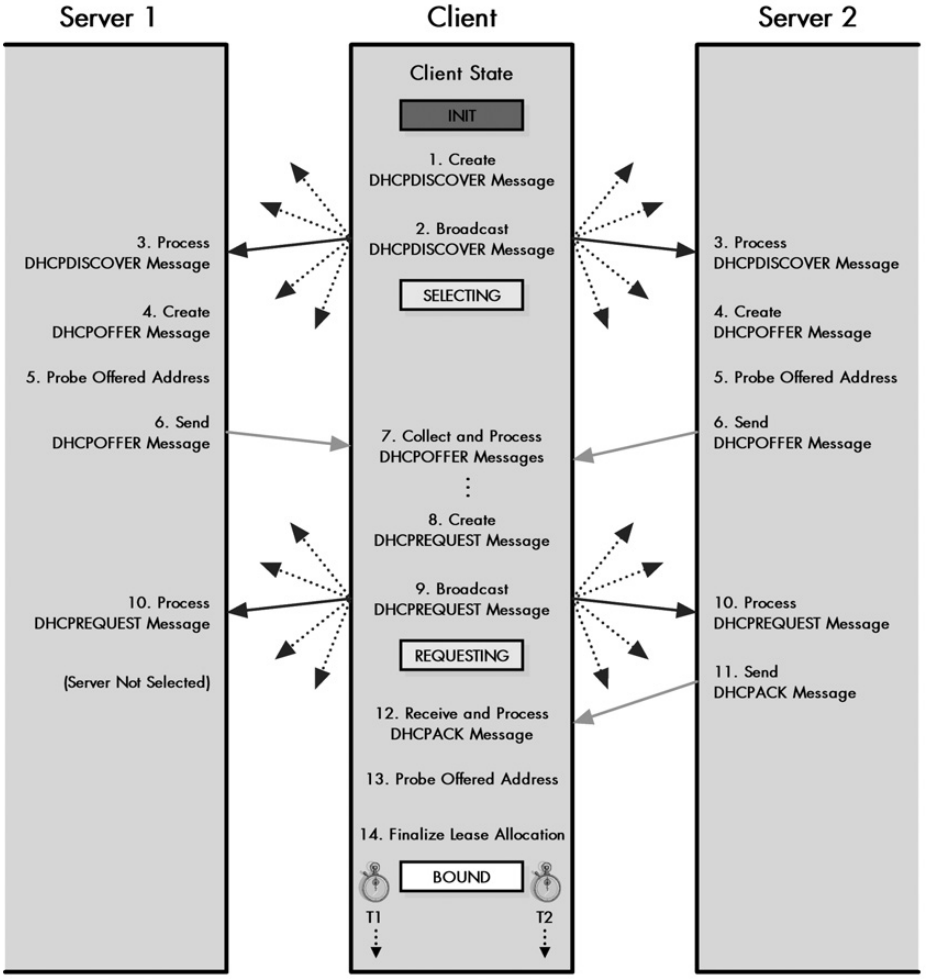


Figure 62-2: DHCP lease allocation process *This diagram shows the steps involved in DHCP client lease allocation. This diagram is a bit different from most of the other client/server exchange diagrams in this book, in that I have shown two servers instead of one. This shows how a client handles responses from multiple DHCP servers and how each server reacts differently, depending on whether its lease offer was chosen by the client.*

KEY CONCEPT The most important configuration process in DHCP is the *lease allocation process*, used by clients to acquire a lease. The client broadcasts a request to determine if any DHCP servers can hear it. Each DHCP server that is willing to grant the client a lease sends it an offer. The client selects the lease it prefers and sends a response to all servers telling them its choice. The selected server then sends the client its lease information.

DHCP Lease Reallocation Process

When a DHCP client starts up for the first time and has no lease, it begins in the INIT (initialize) state and goes through the allocation process described in the preceding section to acquire a lease. The same process is used when a lease ends, if a lease renewal fails, or if something happens to cause a client to need a new lease.

There are, however, certain situations in which a client starts up while it still has a lease already in place. In this situation, the client does not need to go through the entire process of getting an IP address allocation and a new lease setup. Instead, it simply tries to reestablish its existing lease, through a *reallocation process*.

A client performs reallocation rather than allocation when it restarts with an existing lease. The length of time that a client lease lasts can range from minutes to years; it is entirely a matter of the lease length policy set for the network and client by the administrator. Many, if not most, client machines are not connected to the network 24 hours a day. They are turned on during the day and then shut down at night, and also shut down on weekends. A client with a very short lease that is shut down and then later started again will probably find that its lease has expired, and it will need to get a new one. However, if a lease is longer than a few days, it will still probably be in effect when the client starts up again. Clients are also sometimes rebooted, to install new software or correct a problem. In this case, even when the lease length is very short, the restarting client will still have a valid lease when it starts up.

The reallocation process is essentially an abbreviated version of the allocation process described in the previous section. There is no need for the client to go through the whole "Yoohoo, any servers out there want to give me a lease?" routine. Instead, the client attempts to find the server that gave it the lease in the first place, seeking a confirmation that the lease is still valid and that it may resume using its previously allocated IP address. It also receives confirmation of the parameters it should use.

The following steps summarize the reallocation process (see Figure 62-3).

1. **Client Creates DHCPREQUEST Message**

   The client begins in the INIT-REBOOT state instead of the INIT state. It creates a DHCPREQUEST message to attempt to find a server with information about its current lease. This may or may not be the server that originally granted the lease. The server responsible for a lease could, theoretically, have changed in the time since the client obtained the lease. Thus, unlike the DHCPREQUEST message in step 8 in the allocation process, the client does not include a DHCP Server Identifier option. It does includes the following information:

   - Its own hardware address in the CHAddr field of the message, to identify itself
   - The IP address of its existing lease, in the Requested IP Address DHCP option (this address is not put into the CIAddr field)
   - A random transaction identifier, put into the XID field (used to identify later messages as being part of the same transaction)
   - Any additional configuration parameters it wants, in a Parameter Request List option in the message

2. **Client Sends DHCPREQUEST Message**

   The client broadcasts the DHCPREQUEST message. It then transitions to the REBOOTING state, where it waits for a reply from a server.

3. **Servers Receive and Process DHCPREQUEST Message and Generate Replies**

   Each server on the network receives and processes the client's request. The server looks up the client in its database, attempting to find information about the lease. Each server then decides how to reply to the client:

   - If the server has valid client lease information, it sends a DHCPACK message to confirm the lease. It will also reiterate any parameters the client should be using.
   - If the server determines the client lease is invalid, it sends a DHCPNAK message to negate the lease request. Common reasons for this happening are the client trying to confirm a lease after it has moved to a different network or after the lease has already expired.
   - If the server has no definitive information about the client lease, it does not respond. A server is also required not to respond unless its information is guaranteed to be accurate. So, for example, if a server has knowledge of an old expired lease, it cannot assume that the lease is no longer valid and send a DHCPNAK, unless it also has certain knowledge that no other server has a newer, valid lease for that client.

4. **Servers Send Replies**

   Servers that are going to respond to the client's DHCPREQUEST send their DHCPACK or DHCPNAK messages.

5. **Client Receives and Processes DHCPACK or DHCPNAK Message**

   The client waits for a period of time to get a reply to its request. Again, there are three possibilities that match the three in step 3:

   - The client receives a DHCPACK message, which confirms the validity of the lease. The client will prepare to begin using the lease again, and continue with step 6.
   - The client receives a DHCPNAK message, which tells the client that its lease is no longer valid. The client transitions back to the INIT state to get a new lease—step 1 in the allocation process.
   - If the client receives no reply at all, it may retransmit the DHCPREQUEST message. If no reply is received after a period of time, it will conclude that no server has information about its lease and will return to the INIT state to try to get a new lease.

6. **Client Checks That Address Is Not in Use**

   Before resuming use of its lease, the client device should perform a final check to ensure that the new address isn't already in use. Even though this should not be the case when a lease already exists, it's done anyway, as a safety measure. The check is the same as described in step 13 of the allocation process: an ARP request is issued on the local network, to see if any other device thinks it already has the IP address this client was just leased. If another device responds, the

client sends a DHCPDECLINE message back to the server, which tells it that the lease is no good because some other device is using the address. The client then goes back to the INIT state to get a new lease.

7. **Client Finalizes Lease Allocation**

   Assuming that the address is not already in use, the client finalizes the lease and transitions to the BOUND state. It is now ready for normal operation.

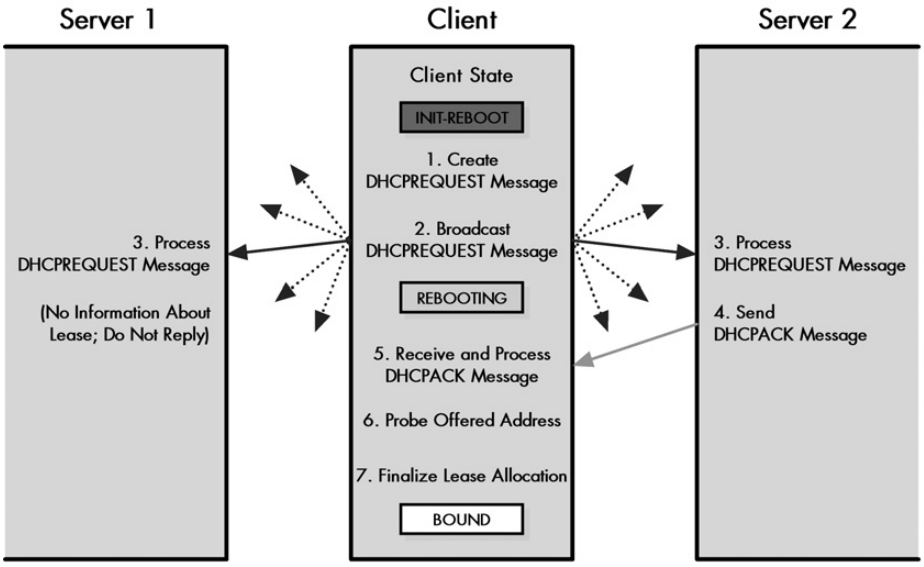


Figure 62-3: DHCP lease reallocation process *The lease reallocation process consists of seven steps that correspond approximately to steps 8 through 14 of the full lease allocation process shown in Figure 62-2. In this example, the server that originally granted the lease to the client is Server 2, so it is normally the only one that responds.*

KEY CONCEPT If a client starts up and already has a lease, it does not need to go through the full lease allocation process; instead, it can use the shorter *reallocation process*. The client broadcasts a request to find the server that has the current information on its lease. That server responds back to confirm that the client's lease is still valid.

DHCP Lease Renewal and Rebinding Processes

Once a DHCP client completes the allocation or reallocation process, it enters the BOUND state. The client is now in its regular operating mode, with a valid IP address and other configuration parameters it received from the DHCP server, and it can be used like any regular TCP/IP host.

While the client is in the BOUND state, DHCP essentially lies dormant. As long as the client stays on and functioning normally, no real DHCP activity will occur while in this state. The most common occurrence that causes DHCP to wake up and become active again is arrival of the time when the lease is to be *renewed*. Renewal ensures that a lease is perpetuated so it can be used for a prolonged period of time,

and involves its own message-exchange procedure. (The other way that a client can leave the BOUND state is when it terminates the lease early, as described in the next section.)

If DHCP's automatic allocation is used, or if dynamic allocation is used with an infinite lease period, the client's lease will never expire, so it never needs to be renewed. Short of early termination, the device will remain in the BOUND state forever, or at least until it is rebooted. However, most leases are finite in nature. A client must take action to ensure that its lease is extended and normal operation continues.

To manage the lease extension process, two timers are set at the time that a lease is allocated. The *renewal timer (T1)* goes off to tell the client it is time to try to renew the lease with the server that initially granted it. The *rebinding timer (T2)* goes off if the client is not successful in renewing with that server, and tells it to try any server to have the lease extended. If the lease is renewed or rebound, the client goes back to normal operation. If it cannot be rebound, it will expire, and the client will need to seek a new lease.

The following steps summarize the renewal/rebinding process (see Figure 62-4). Obviously, the exact sequence of operations taken by a client depends on what happens in its attempts to contact a server. For example, if it is successful with renewal, it will never need to attempt rebinding.

1. **Renewal Timer (T1) Expires**

   The renewal timer, T1, is set by default to 50 percent of the length of the lease. When the timer goes off, the client transitions from the BOUND state to the RENEWING state. Note that a client may initiate lease renewal prior to T1 timer expiration, if it desires.

2. **Client Sends DHCPREQUEST Renewal Message**

   The client creates a DHCPREQUEST message that identifies itself and its lease. It then transmits the message directly to the server that initially granted the lease, unicast. Note that this is different from the DHCPREQUEST messages used in the allocation/reallocation processes, where the DHCPREQUEST is broadcast. The client may request a particular new lease length, just as it may request a lease length in its requests during allocation, but as always, the server makes the final call on lease length.

3. **Server Receives and Processes DHCPREQUEST Message and Creates Reply**

   Assuming the server is reachable, it will receive and process the client's renewal request. There are two possible responses:

   - The server decides that the client's lease can be renewed. It prepares to send to the client a DHCPACK message to confirm the lease's renewal, indicating the new lease length and any parameters that may have changed since the lease was created or last renewed.
   - The server decides, for whatever reason, not to renew the client's lease. It will create a DHCPNAK message.

4. **Server Sends Reply**

   The server sends the DHCPACK or DHCPNAK message back to the client.

5. **Client Receives and Processes Server Reply**

   The client takes the appropriate action in response to the server's reply:

   - If the client receives a DHCPACK message, renewing the lease, it notes the new lease expiration time and any changed parameters sent by the server, resets the T1 and T2 timers, and transitions back to the BOUND state. The client does not need to do an ARP IP address check when it is renewing.
   - If the client receives a DHCPNAK message, which tells it its lease renewal request has been denied, it will immediately transition to the INIT state to get a new lease (step 1 in the allocation process).

6. **Rebinding Timer (T2) Expires**

   If the client receives no reply from the server, it will remain in the RENEWING state and will regularly retransmit the unicast DHCPREQUEST to the server. During this period, the client is still operating normally, from the perspective of its user. If no response from the server is received, eventually the rebinding timer (T2) expires. This will cause the client to transition to the REBINDING state. Recall that by default, the T2 timer is set to 87.5 percent (seven-eighths) of the length of the lease.

7. **Client Sends DHCPREQUEST Rebinding Message**

   Having received no response from the server that initially granted the lease, the client gives up on that server and tries to contact any server that may be able to extend its existing lease. It creates a DHCPREQUEST message and puts its IP address in the CIAddr field, indicating clearly that it presently owns that address. It then broadcasts the request on the local network.

8. **Servers Receive and Process DHCPREQUEST Message and Send Reply**

   Each server receives the request and responds according to the information it has for the client (a server that has no information about the lease or may have outdated information does not respond):

   - A server may agree to rebind the client's lease. This happens when the server has information about the client's lease and can extend it. It prepares for the client a DHCPACK message to confirm the lease's renewal, indicating any parameters that may have changed since the lease was created or last renewed.
   - A server may decide that the client cannot extend its current lease. This occurs when the server determines that, for whatever reason, this client's lease should not be extended. It gets ready to send back to the client a DHCPNAK message.

9. **Server Sends Reply**

   Each server that is responding to the client sends its DHCPACK or DHCPNAK message.

10. **Client Receives Server Reply**

    The client takes the appropriate action in response to the two possibilities in the preceding step:

    - The client receives a DHCPACK message, rebinding the lease. The client makes note of the server that is now in charge of this lease, the new lease expiration time, and any changed parameters sent by the server. It resets the T1 and T2 timers, and transitions back to the BOUND state. (It may also probe the new address as it does during regular lease allocation.)
    - The client receives a DHCPNAK message, which tells it that some server has determined that the lease should not be extended. The client immediately transitions to the INIT state to get a new lease (step 1 in the allocation process).

11. **Lease Expires**

    If the client receives no response to its broadcast rebinding request, it will, as in the RENEWING state, retransmit the request regularly. If no response is received by the time the lease expires, it transitions to the INIT state to get a new lease.

So, why bother with a two-step process: rebinding and renewal? The reason is that this provides the best blend of efficiency and flexibility. We first try to contact the server that granted the lease using a unicast request, to avoid taking up the time of other DHCP servers and disrupting the network as a whole with broadcast traffic. Usually this will work, because DHCP servers don't change that often and are usually left on continuously. If that fails, we then fall back on the broadcast, giving other servers a chance to take over the client's existing lease.

KEY CONCEPT Each client's lease has associated with it a *renewal timer (T1)*, normally set to 50 percent of the length of the lease, and a *rebinding timer (T2)*, usually set to 87.5 percent of the lease length. When the T1 timer goes off, the client will try to renew its lease by contacting the server that originally granted it. If the client cannot renew the lease by the time the T2 timer expires, it will broadcast a rebinding request to any available server. If the lease is not renewed or rebound by the time the lease expires, the client must start the lease allocation process over again.

DHCP Early Lease Termination (Release) Process

A TCP/IP host can't really do much without an IP address; it's a fundamental component of the Internet Protocol (IP), on which all TCP/IP protocols and applications run. When a host has either a manual IP address assignment or an infinite lease, it obviously never needs to worry about losing its IP address. When a host has a finite DHCP lease, it will use the renewal/rebinding process to try to hang on to its existing IP address as long as possible.

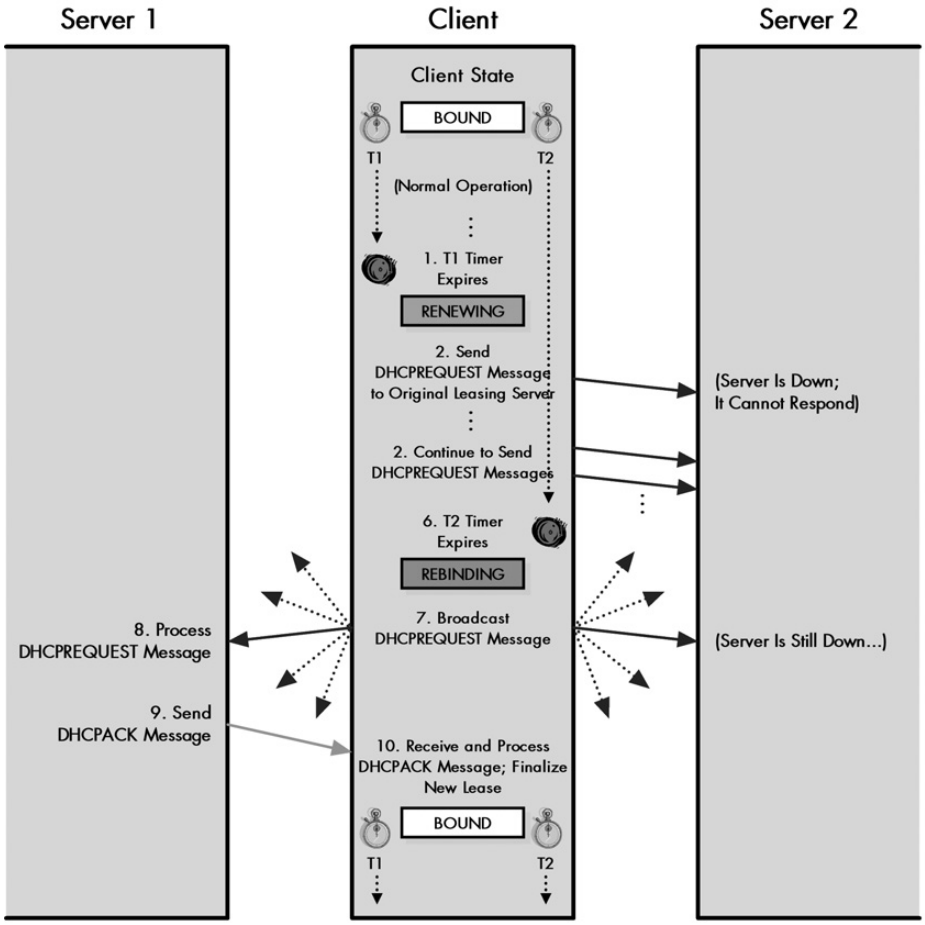


Figure 62-4: DHCP lease renewal and rebinding processes *This diagram shows the example of a client presently holding a lease with Server 2 attempting to contact it to renew the lease. However, in this case, Server 2 is down for maintenance. The server is unable to respond, and the client remains stuck at step 2 in the renewal/rebinding process. It keeps sending DHCPREQUEST messages to Server 2 until its T2 timer expires. It then enters the rebinding state and broadcasts a DHCPREQUEST message, which is heard by Server 1, which agrees to extend its current lease.*

So, under normal circumstances, a client will continue trying to extend its existing lease indefinitely. In certain cases, however, a host may decide to terminate its lease. This usually will not be something the client just decides to do spontaneously. It will occur in response to a specific request from the user to end the lease. A user may terminate a lease for a number of reasons, including the following:

- The client is being moved to a different network.
- The network is having its IP addresses renumbered.
- The user wants the host to negotiate a new lease with a different server.
- The user wants to reset the lease to fix some sort of a problem.

In any of these cases, the user can end the lease through a process called *early lease termination* or *lease release*. This is a very simple, unidirectional communication. The client sends a special DHCPRELEASE message unicast to the server that holds its current lease, to tell it that the lease is no longer required. The server then records the lease as having been ended. It does not need to reply back to the client.

The reason that the client can just assume that the lease termination has been successful is that this is not a mandatory part of the DHCP protocol. Having clients send DHCPRELEASE to end a lease is considered a courtesy, rather than a requirement. It is more efficient to have clients inform servers when they no longer need a lease, and this also allows the IP address in the terminated lease to be reused more quickly. However, DHCP servers are designed to handle the case where a client seemingly disappears without formally ending an existing lease.

DHCP Parameter Configuration Process for Clients with Non-DHCP Addresses

The majority of DHCP clients make use of the protocol to obtain both an IP address and other configuration parameters. This is the reason why so much of DHCP is oriented around address assignment and leasing. A conventional DHCP client obtains all its configuration parameters at the same time it gets an IP address, using the message exchanges and processes described in the preceding sections of this chapter.

There are cases where a device with an IP address assigned using a method other than DHCP still wants to use DHCP servers to obtain other configuration parameters. The main advantage of this is administrative convenience; it allows a device with a static IP address to still be able to automatically get other parameters the same way that regular DHCP clients do.

Ironically, one common case where this capability can be used is in configuring DHCP servers themselves! Administrators normally do not use DHCP to provide an IP address to a DHCP server, but they may want to use it to tell the server other parameters. In this case, the server requesting the parameters actually acts as a client for the purpose of the exchange with another server.

The original DHCP standard did not provide any mechanism for this sort of non-IP configuration to take place. RFC 2131 revised the protocol, adding a new message type (DHCPINFORM) that allows a device to request configuration parameters without going through the full leasing process. This message is used as part of a simple bidirectional communication that is separate from the leasing communications we have looked at so far. Since it doesn't involve IP address assignment, it is not part of the lease life cycle, nor is it part of the DHCP client FSM.

The following steps show how a device with an externally configured address uses DHCP to get other parameters (see Figure 62-5):

1. **Client Creates DHCPINFORM Message**

   The client (which may be a DHCP server acting as a client) creates a DHCPINFORM message. It fills in its own IP address in the CIAddr field, since that IP address is current and valid. It may request specific parameters using the Parameter Request List option or simply accept the defaults provided by the server.

2. **Client Sends DHCPINFORM Message**
   The client sends the DHCPINFORM message unicast, if it knows the identity and address of a DHCP server; otherwise, it broadcasts it.
3. **Server Receives and Processes DHCPINFORM Message**
   The message is received and processed by the DHCP server or servers (if there are multiple servers and the request was broadcast). Each server checks to see if it has the parameters needed by the client in its database.
4. **Server Creates DHCPACK Message**
   Each server that has the information the client needs creates a DHCPACK message, which includes the needed parameters in the appropriate DHCP option fields. (Often, this will be only a single server.)
5. **Server Sends DHCPACK Message**
   The server sends the message unicast back to the client.
6. **Client Receives and Processes DHCPACK Message**
   The client receives the DHCPACK message sent by the server, processes it, and sets its parameters accordingly.

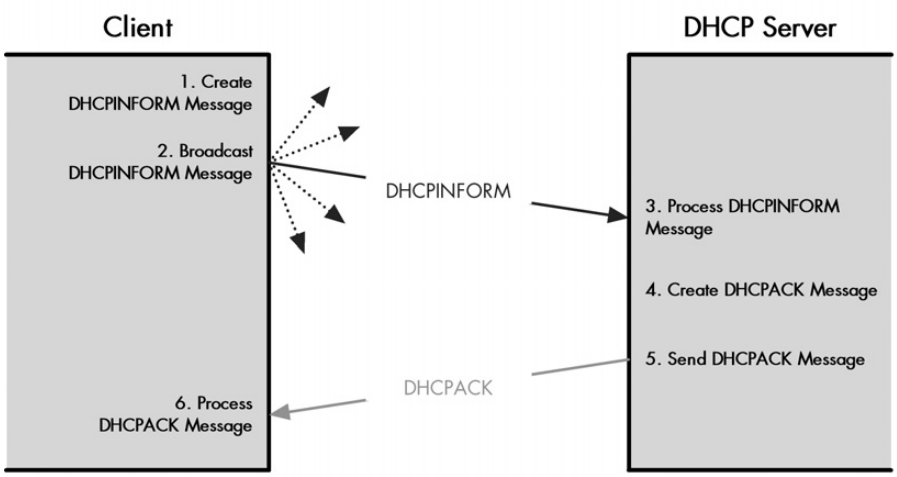


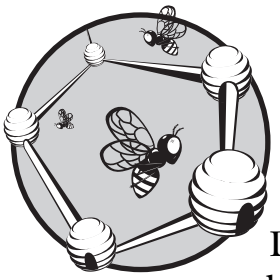
Figure 62-5: DHCP parameter configuration process *A device that already has an IP address can use the simple request/reply exchange shown in this figure to get other configuration parameters from a DHCP server. In this case, the client is broadcasting its request.*

If a client receives no reply to its DHCPINFORM message, it will retransmit it periodically. After a retry period, it will give up and use default configuration values. It will also typically generate an error report to inform an administrator or user of the problem.

KEY CONCEPT Devices that are not using DHCP to acquire IP addresses can still use its other configuration capabilities. A client can broadcast a *DHCPINFORM* message to request that any available server send it parameters for how the network is to be used. DHCP servers respond with the requested parameters and/or default parameters, carried in DHCP options of a *DHCPACK* message.

63

DHCP MESSAGING, MESSAGE TYPES, AND FORMATS



The preceding chapter on DHCP configuration and operation demonstrated how DHCP works by showing the various leasing and information-exchange processes. All of these procedures rely heavily on the exchange of information between the client and server, which is accomplished through DHCP *messages*. Like all protocols, DHCP uses a special message format and a set of rules that govern how messages are created, addressed, and transported.

In this chapter, I provide the details of how DHCP creates and sends messages, and show the formats used for DHCP messages and options. I begin with a description of how DHCP creates, addresses, and transports messages, and how it deals with message retransmission. I then outline the DHCP general message format, showing how it is similar to the BOOTP message format on which it is based and also where it differs. I describe DHCP options, the format used for them, and the special option overloading feature used for efficiency. I conclude the section with a complete list of DHCP options.

RELATED INFORMATION *DHCP is most closely related to BOOTP in the area of messaging. DHCP options are based closely on BOOTP vendor extensions (see Chapter 60), and many of the specific DHCP option types are the same as BOOTP vendor information fields. To avoid duplication, the summary table in this chapter lists the options/extensions for both protocols, indicating which ones are used by both BOOTP and DHCP, and which are used only by DHCP.*

DHCP Message Generation, Addressing, Transport, and Retransmission

As you've learned, nearly every aspect of DHCP's operation is oriented around the notion of a client device exchanging information with a server. You can also see this reflected in all of the major characteristics of DHCP messaging. This includes the format of DHCP messages, as well as the specifics of how DHCP messages are created, addressed, and transmitted, and when necessary, retransmitted.

Message Generation and General Formatting

DHCP messaging is similar in many ways to that of BOOTP, the protocol on which DHCP was based. BOOTP defined only two message types: a request and a reply. DHCP is much more complex. It uses eight different types of messages, but these are still categorized as either request or reply messages, depending on who sends them and why. DHCP uses a special DHCP Message Type option to indicate the exact DHCP message type, but still treats a message from a client seeking information as a request, and a response from a server containing information as a reply.

A client generates a message using the general DHCP message format, which is very similar to the BOOTP message format. When a server replies to a client message, it does not generate the reply as a completely new message, but rather copies the client request, changes fields as appropriate, and sends the reply back to the client. A special transaction identifier (XID) is placed in the request and maintained in the reply, which allows a client to know which reply goes with a particular request.

Message Transport

DHCP uses the User Datagram Protocol (UDP) for transport, just as BOOTP does, and for the same reasons: simplicity and support for broadcasts. It also has many of the same addressing concerns as BOOTP, as discussed in Chapter 60. Clients usually will send requests by broadcast on the local network, to allow them to contact any available DHCP server. The exception to this is when a client is trying to renew a lease with a server that it already knows. For compatibility with BOOTP, DHCP uses the same well-known (reserved) UDP port number, 67, for client requests to servers.

Some DHCP message exchanges require a server to respond back to a client that has a valid and active IP address. An example is a DHCPACK message sent in reply to a DHCPINFORM request. In this situation, the server can always send a reply unicast back to the client. Other message exchanges, however, present the same chicken-and-egg conundrum that we saw with BOOTP: If a client is using DHCP to obtain an IP address, we can't assume that IP address is available for us to use to send a reply.

In BOOTP, there were two possible solutions to this situation: The server could send back its reply using broadcast addressing as well, or the server could send back a reply directly to the host at layer 2. Due to the performance problems associated with broadcasts, DHCP tries to make the latter method the default for server replies. It assumes that a client's TCP/IP software will be capable of accepting and processing an IP datagram delivered at layer 2, even before the IP stack is initialized.

As the standard itself puts it, "DHCP requires creative use of the client's TCP/IP software and liberal interpretation of RFC 1122." RFC 1122 is a key standard describing the detailed implementation requirements of TCP/IP hosts. The DHCP standard, however, acknowledges the fact that not all devices may support this behavior. It allows a client to force servers to send back replies using broadcasts instead. This is done by the client setting the special Broadcast (B) flag to 1 in its request.

Since DHCP, like BOOTP, must use either layer 2 delivery or layer 3 broadcasts for server replies, it requires a separate well-known port number for servers to send to. Again, for compatibility with BOOTP, the same port number is used, 68. This port number is used whether a server reply is sent unicast or broadcast.

KEY CONCEPT Requests from BOOTP clients are normally sent broadcast, to reach any available DHCP server. However, there are certain exceptions, such as in lease renewal, when a request is sent directly to a known server. DHCP servers can send their replies either broadcast to the special port number reserved for DHCP clients or unicast using layer 2. The DHCP standards specify that layer 2 delivery should be used when possible to avoid unnecessary broadcast traffic.

Retransmission of Lost Messages

Using UDP provides benefits such as simplicity and efficiency to DHCP, but since UDP is unreliable, there is no guarantee that messages will get to their destination. This can lead to potential confusion on the part of a client. Consider, for example, a client sending a DHCPDISCOVER message and waiting for DHCPOFFER messages in reply. If it gets no response, does this mean that there is no DHCP server willing to offer it service or simply that its DHCPDISCOVER got lost somewhere on the network? The same applies to most other request/reply sequences, such as a client waiting for a DHCPACK or DHCPNAK message in reply to a DHCPREQUEST or DHCPINFORM message.

The fact that messages can be lost means that DHCP itself must keep track of messages sent, and if there is no response, retransmit them. Since there are so many message exchanges in DHCP, there is much that can go wrong. As in BOOTP, DHCP puts responsibility for this squarely on the shoulders of the client. This makes sense, since the client initiates contact and can most easily keep track of messages sent and retransmit them when needed. A server can't know when a client's request is lost, but a client can react when it doesn't receive a reply from the server.

In any request/reply message exchange, the client uses a retransmission timer that is set to a period of time that represents how long it is reasonable for it to wait for a response. If no reply is received by the time the timer expires, the client assumes that either its request or the response coming back was lost. The client then retransmits the request. If this request again elicits no reply, the client will continue retransmitting for a period of time.

To prevent large numbers of DHCP clients from retransmitting requests simultaneously (which would potentially clog the network), the client must use a randomized exponential backoff algorithm to determine when exactly a retransmission is made. As in BOOTP, this is similar to the technique used to recover from collisions in Ethernet. The DHCP standard specifies that the delay should be based on the speed of the underlying network between the client and the server. More specifically, it says that in a standard Ethernet network, the first retransmission should be delayed 4 seconds plus or minus a random value from 0 to 1 second; in other words, some value is chosen between 3 and 5 seconds. The delay is then doubled with each subsequent transmission (7 to 9 seconds, then 15 to 17 seconds, and so forth) up to a maximum of 64 +/− 1 second.

To prevent it from retrying endlessly, the client normally has logic that limits the number of retries. The amount of time that retransmissions go on depends on the type of request being sent; that is, what process is being undertaken. If a client is forced to give up due to too many retries, it will generally either take some sort of default action or generate an error message.

KEY CONCEPT Like BOOTP, DHCP uses UDP for transport, which does not provide any reliability features. DHCP clients must detect when requests are sent and no response is received, and retransmit requests periodically. Special logic is used to prevent clients from sending excessive numbers of requests during difficult network conditions.

DHCP Message Format

When DHCP was created, its developers had a bit of an issue related to how exactly they should structure DHCP messages. BOOTP was already widely used, and maintaining compatibility between DHCP and BOOTP was an important goal. This meant that DHCP's designers needed to continue using the existing BOOTP message format. However, DHCP has more functionality than BOOTP, and this means that it can hold more information than can easily fit in the limited BOOTP message format.

This apparent contradiction was resolved in two ways:

- The existing BOOTP message format was maintained for basic functionality, but DHCP clients and servers were programmed to use the BOOTP message fields in slightly different ways.
- The BOOTP vendor extensions were formalized and became DHCP *options*. Despite the name *options*, some of these additional fields are for basic DHCP functionality, and they are quite mandatory!

With this dual approach, DHCP devices have access to the extra information they need. Meanwhile, the basic field format is unchanged, allowing DHCP servers to communicate with older BOOTP clients, which ignore the extra DHCP information that doesn't relate to them. See the discussion of BOOTP/DHCP interoperability (in Chapter 64) for more information.

The DHCP message format is illustrated in Figure 63-1 and described fully in Tables 63-1 and 63-2. In the table, I have specifically indicated which fields are used in DHCP in a manner similar to how they are used in BOOTP, and which are significantly different.

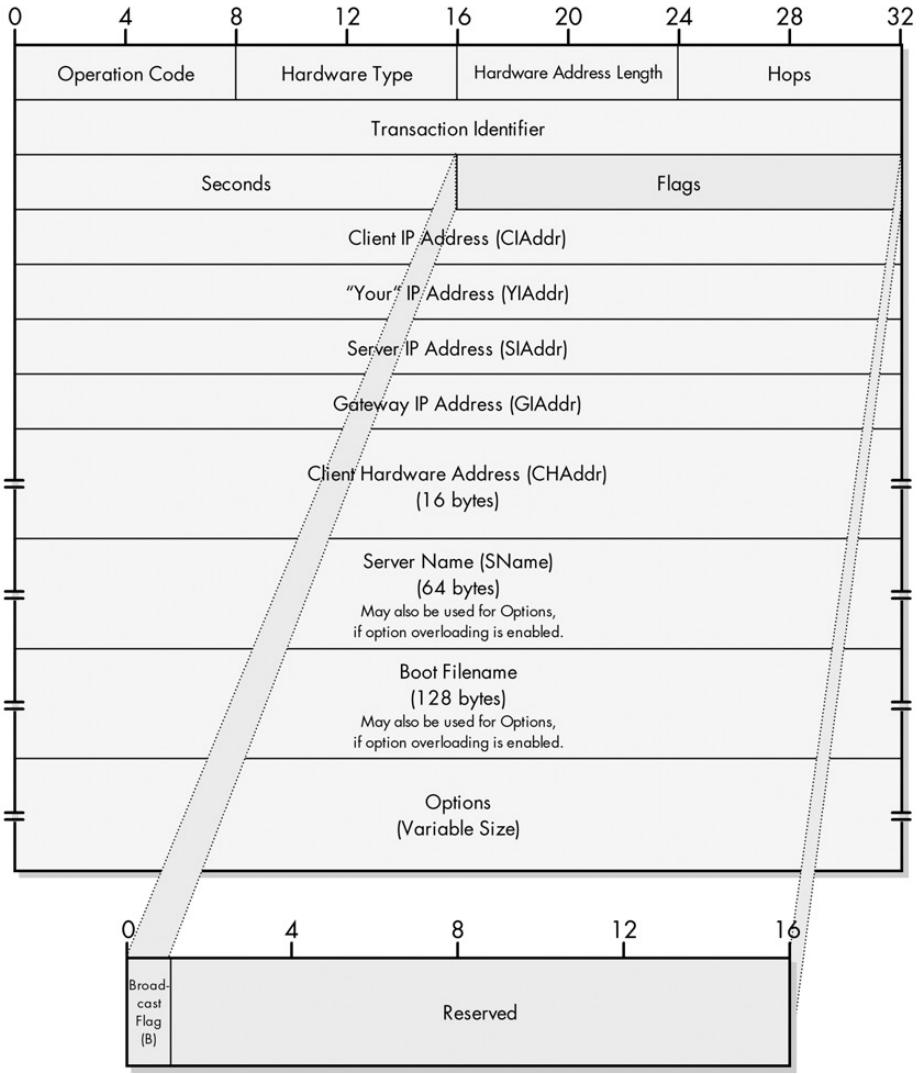


Figure 63-1: *DHCP message format*

Table 63-1: DHCP Message Format

Field Name	Size (Bytes)	Description
Op	1	Operation Code: This code represents the general category of the DHCP message. A client sending a request to a server uses an opcode of 1; a server replying uses a code of 2. So, for example, a DHCPREQUEST would be a request, and a DHCPACK or DHCPNAK is a reply. The actual specific type of DHCP message is encoded using the DHCP Message Type option.
HType	1	Hardware Type: This field specifies the type of hardware used for the local network, and it is used in exactly the same way as the equivalent field (HRD) in the Address Resolution Protocol (ARP) message format. Some of the most common values for this field are shown in Table 63-2.

(continued)

Table 63-1: DHCP Message Format (continued)

Field Name	Size (Bytes)	Description
Hlen	1	Hardware Address Length: Specifies how long hardware addresses are in this message. For Ethernet or other networks using IEEE 802 MAC addresses, the value is 6. This is also the same as a field in the ARP field format, HLN.
Hops	1	Hops: Set to 0 by a client before transmitting a request and used by relay agents to control the forwarding of BOOTP and/or DHCP messages.
XID	4	Transaction Identifier: A 32-bit identification field generated by the client, to allow it to match up the request with replies received from DHCP servers.
Secs	2	Seconds: In BOOTP, this field was vaguely defined and not always used. For DHCP, it is defined as the number of seconds elapsed since a client began an attempt to acquire or renew a lease. This may be used by a busy DHCP server to prioritize replies when multiple client requests are outstanding.
Flags	2	Flags: This corresponds to the formerly empty 2-byte field in the BOOTP message format defined by RFC 951, which was redefined as a Flags field in RFC 1542. The field presently contains just one flag subfield. This is the B (Broadcast) flag subfield, 1 bit in size, which is set to 1 if the client doesn't know its own IP address at the time it sends its request. This serves as an immediate indicator to the DHCP server or relay agent that receives the request that it should send its reply back by broadcast. The other subfield, which is 15 bits, is reserved, set to 0, and not used.
CIAddr	4	Client IP Address: The client puts its own current IP address in this field if and only if it has a valid IP address while in the *BOUND*, *RENEWING*, or *REBINDING* states; otherwise, it sets the field to 0. The client can only use this field when its address is actually valid and usable, not during the process of acquiring an address. Specifically, the client does not use this field to request a particular IP address in a lease; it uses the Requested IP Address DHCP option.
YIAddr	4	Your IP Address: The IP address that the server is assigning to the client.
SIAddr	4	Server IP Address: The meaning of this field is slightly changed in DHCP. In BOOTP, it is the IP address of the BOOTP server sending a BOOTREPLY message. In DHCP, it is the address of the server that the client should use for the next step in the bootstrap process, which may or may not be the server sending this reply. The sending server always includes its own IP address in the Server Identifier DHCP option.
GIAddr	4	Gateway IP Address: This field is used just as it is in BOOTP, to route BOOTP messages when BOOTP relay agents are involved to facilitate the communication of BOOTP requests and replies between a client and a server on different subnets or networks. See the description of DHCP relaying. As with BOOTP, this field is not used by clients and does not represent the server giving the client the address of a default router (that's done using the Router DHCP option).
CHAddr	16	Client Hardware Address: The hardware (layer 2) address of the client, which is used for identification and communication.
SName	64	Server Name: The server sending a DHCPOFFER or DHCPACK message may optionally put its name in this field. This can be a simple text nickname or a fully qualified DNS domain name (such as myserver.organization.org). This field may also be used to carry DHCP options, using the option overload feature, indicated by the value of the DHCP Option Overload option.
File	128	Boot Filename: Optionally used by a client to request a particular type of boot file in a DHCPDISCOVER message. Used by a server in a DHCPOFFER to fully specify a boot file directory path and filename. This field may also be used to carry DHCP options, using the option overload feature, indicated by the value of the DHCP Option Overload option.
Options	Variable	Options: Holds DHCP options, including several parameters required for basic DHCP operation. Note that this field was fixed at 64 bytes in length in BOOTP but is variable in length in DHCP. See the next section for more information. This field may be used by both the client and server.

Table 63-2: DHCP Message HType Values

HType Value	Hardware Type
1	Ethernet (10 Mb)
6	IEEE 802 Networks
7	ARCNet
11	LocalTalk
12	LocalNet (IBM PCNet or SYTEK LocalNET)
14	Switched Multimegabit Data Service (SMDS)
15	Frame Relay
16	Asynchronous Transfer Mode (ATM)
17	High-Level Data Link Control (HDLC)
18	Fibre Channel
19	ATM
20	Serial Line

The DHCP standard does not specify the details of how DHCP messages are encapsulated within UDP. I would assume that due to the other similarities to BOOTP, DHCP maintains BOOTP's optional use of message checksums. It also most likely assumes that messages will not be fragmented (sent with the Do Not Fragment bit set to 1 in the IP datagram). This is to allow BOOTP clients to avoid the complexity of reassembling fragmented messages.

Unlike with BOOTP, which has a fixed message size, DHCP messages are variable in length. This is the result of changing BOOTP's 64-byte Vend field into the variable-length Options field. DHCP relies on options much more than BOOTP does, and a device must be capable of accepting a message with an Options field at least 312 bytes in length. The SName and File fields may also be used to carry options, as described in the next section.

DHCP Options

When BOOTP was first developed, its message format included a 64-byte Vend field, called the Vendor-Specific Area. The idea behind this field was to provide flexibility to the protocol. The BOOTP standard did not define any specific way of using this field. Instead, the field was left open for the creators of different types of hardware to use it to customize BOOTP to meet the needs of their clients and/or servers.

Including this sort of undefined field is a good idea because it makes a protocol easily *extensible*—allowing the protocol to be easily enhanced in the future through the definition of new fields while not disturbing any existing fields. The problem with the BOOTP Vendor-Specific Area, however, is that the extensibility was vendor-specific. It was useful only for special fields that were particular to a single vendor.

What was really needed was a way to define new fields for general-purpose, vendor-independent parameter communication, but there was no field in the BOOTP message format that would let this happen. The solution came in the form of RFC 1048, which defined a technique called BOOTP *vendor information extensions*.

This method redefines the Vendor-Specific Area to allow it to carry general parameters between a client and server. This idea was so successful that it largely replaced the older vendor-specific use of the Vend field.

DHCP maintains, formalizes, and further extends the idea of using the Vend field to carry general-purpose parameters. Instead of being called vendor information extensions or vendor information fields, these fields are now called simply DHCP *options*. Similarly, the Vend field has been renamed the Options field, reflecting its new role as a way of conveying vendor-independent options between a client and server.

Options and Option Format

Keeping with the desire to maintain compatibility between BOOTP and DHCP, the DHCP Options field is, in most ways, the same as the vendor-independent interpretation of the BOOTP Vend field introduced by RFC 1048. The first four bytes of the field still carry the magic cookie value 99.130.83.99 to identify the information as vendor-independent option fields. The rest of the Option field consists of one or more subfields, each of which has a *type, length, value (TLV-encoded)* substructure, as in BOOTP.

The main differences between BOOTP vendor information fields and DHCP options are the field names and the fact that the DHCP Options field is variable in length (the BOOTP Vend field is fixed at 64 bytes). The structure of the DHCP Options field as a whole is shown in Figure 63-2, and the subfield names of each option are described in Table 63-3.

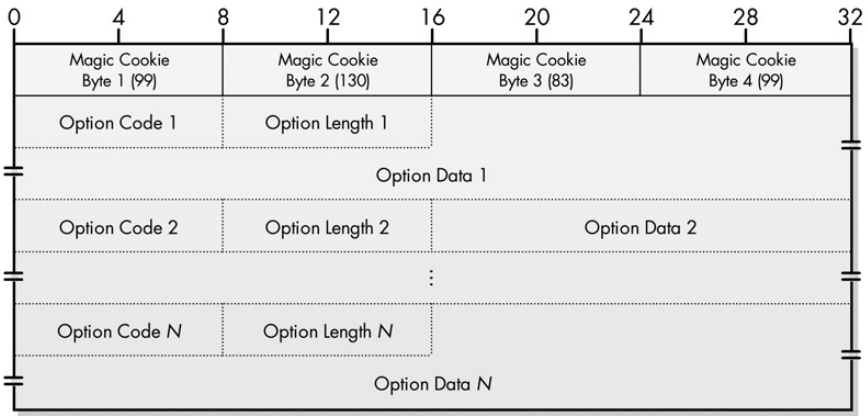


Figure 63-2: DHCP Options field format *The format of the DHCP Options field is, unsurprisingly, very similar to that of the BOOTP Vendor-Specific Area, as shown in Figure 60-4 in Chapter 60. The Options field begins with the same four-byte magic cookie and then contains a number of variable-length option fields. Each option has the format described in Table 63-3.*

All of the DHCP options follow the format shown in Table 63-3, except for two special cases, again the same as with BOOTP. A Code value of 0 is used as a *pad*, when subfields need to be aligned on word boundaries; it contains no information. The value 255 is used to mark the end of the vendor information fields. Both of these codes contain no actual data, so to save space, when either is used, just the single Code value is included and the Len and Data fields are omitted. A device

seeing a Code value of 0 just skips it as filler. A device seeing a Code value of 255 knows it has reached the end of the fields in this Options field.

Table 63-3: DHCP Option Format

Subfield Name	Size (Bytes)	Description
Code	1	Option Code: A single octet that specifies the option type.
Len	1	Option Length: The number of bytes in this particular option. This does not include the two bytes for the Code and Len subfields.
Data	Variable	Option Data: The data being sent, which has a length indicated by the Len subfield, and which is interpreted based on the Code subfield.

Option Categories

Before DHCP was invented, a series of BOOTP standards was published defining the current list of BOOTP vendor information extensions. When DHCP was developed, a single standard was created that merged both BOOTP vendor information extensions and DHCP options, since again, they are basically the same. The most recent of these is RFC 2132, entitled (ta-da!) "DHCP Options and BOOTP Vendor Extensions."

RFC 2132 lists several dozen fields that can be used either as DHCP options or BOOTP vendor information fields, grouped into several categories. In addition, there is also a set of fields that are used only in DHCP, not in BOOTP. Despite being called *options*, only some really are optional; others are necessary for the basic operation of DHCP. They are carried as option fields for only one reason: to allow DHCP to keep using the same basic message format as BOOTP for compatibility. Table 63-4 summarizes the categories used for DHCP options.

Table 63-4: DHCP Option Categories

Option Category	Description
RFC 1497 Vendor Extensions	The BOOTP vendor extensions defined in RFC 1497, the last RFC describing vendor extension fields that was BOOTP-specific (before DHCP was created). For easier reference, these were kept in a single group when DHCP options were created, even though some of the functions they represent might fit better in other categories. (See Table 63-5.)
IP Layer Parameters per Host	Parameters that control the operation of the Internet Protocol (IP) on a host, which affect the host as a whole and are not interface-specific. (See Table 63-6.)
IP Layer Parameters per Interface	Parameters that affect the operation of IP for a particular interface of a host. (Some devices have only one interface; others have more.) (See Table 63-7.)
Link Layer Parameters per Interface	Parameters that affect the data link layer operation of a host, on a per-interface basis. (See Table 63-8.)
TCP Parameters	Parameters that impact the operation of the TCP layer; specified on a per-interface basis. (See Table 63-9.)
Application and Service Parameters	Parameters used to configure or control the operation of various miscellaneous applications or services. (See Table 63-10.)
DHCP Extensions	Parameters that are DHCP-specific and used to control the operation of the DHCP protocol itself. (See Table 63-12.)

The tables at the end of this chapter provide a complete list of the DHCP options defined in RFC 2132.

Due to the popularity of DHCP, several other options have been defined since that standard was published. Each time a new option is created, documenting it would have required a new successor to RFC 2132, which would be confusing and time-consuming. Instead, the maintenance of these options and extensions has been moved from the RFC process to a set of files maintained by the Internet Assigned Numbers Authority (IANA), just like so many other parameters. There is also a process by which a developer can request additional standard extensions to be added to DHCP. This is described in section 10 of RFC 2132.

KEY CONCEPT DHCP takes BOOTP's vendor information extensions and formalizes them into an official feature called DHCP *options*. The BOOTP Vendor Specific Area field becomes the DHCP Options field, and it can contain an arbitrary number of parameters to be sent from the server to the client. Some of these include pieces of data that are actually mandatory for the successful operation of DHCP. There are several dozen DHCP options, which are divided into functional categories.

Option Overloading

Since DHCP relies so much more on the use of options than BOOTP did, the size of the Options field could theoretically grow quite large. However, since DHCP is using UDP for transport, the overall size of a message is limited. This theoretically could have led to a situation where a message might run out of room and be unable to carry all its options. Meanwhile, there are two more spacious fields in the message format: SName and File, which are 64 bytes and 128 bytes, respectively. These fields might not even be needed in some cases, because many devices use DHCP for getting a lease and parameters, not to download a boot image. Even if they are needed, they might be carrying much less information than their large fixed size allows.

To make better use of the total space in the message format, DHCP includes a special feature called *option overloading*, which allows the SName and File fields to be used to carry more option fields instead of their conventional information. Use of this option is itself indicated through the use of a DHCP option, Option Overload, which tells a device receiving a message how to interpret the two fields. If option overloading is used, the SName and/or File fields are read and interpreted in the same way as the Options field, after all of the options in the Options field are parsed. If the message actually does need to carry a server name or boot file, these are included as separate options (number 66 and number 67, respectively), which are variable in length and can therefore be made exactly the length needed.

Incidentally, the creators of DHCP did recognize that even though vendor-independent options are important, a vendor might want to be able to send vendor-specific information just as the original BOOTP defined. To this end, they created a DHCP option called Vendor Specific Information. This option allows a vendor to encapsulate a set of vendor-specific option fields within the normal DHCP option structure. In essence, you can think of this as a way of nesting a conventional BOOTP Vend field (of variable length) within a single DHCP option. Other DHCP options can be carried simultaneously, subject to overall message-length limits. Note that this supplements an already existing BOOTP option that allows reference to be made to a file containing vendor-specific information.

KEY CONCEPT Since DHCP messages can contain so many options, a special feature called *option overloading* was created. When enabled, overloading allows options to make use of the large SName and File fields in the DHCP message format for options.

Summary of DHCP Options/BOOTP Vendor Information Fields

BOOTP *vendor information fields* are used to carry additional vendor-independent configuration parameters. These were used as the basis for DHCP *options*, which extend the concept to include parameters used to manage the operation of DHCP as a whole, as described in the previous section. Since BOOTP vendor information fields and DHCP options are essentially the same (except for the DHCP-specific fields), they are described in the same TCP/IP standard, and hence, in this single part of the book.

The following tables list each of the DHCP options/BOOTP vendor information fields. The tables show each option's Code value, the length of the Data subfield for the option, the formal name of the option, and a brief description of how it is used. For simplicity in the tables, where I say *option*, please read it as *option/vendor information field*, since they are the same (except, for the DHCP-specific options).

NOTE *There are a lot of options in these tables, and some of them define parameters that are used by somewhat obscure protocols that I do not cover in this book. The brief descriptions may not be enough for you to completely understand how each and every option is used. Note in particular that many of the original BOOTP vendor information fields that are used to communicate the addresses of certain types of servers are now archaic and may no longer be used.*

RFC 1497 Vendor Extensions

Table 63-5 shows the DHCP/BOOTP options that were originally defined in RFC 1497.

Table 63-5: DHCP/BOOTP Options: RFC 1497 Vendor Extensions

Code Value	Data Length (Bytes)	Name and Description
0	0	Pad: A single byte used as filler to align a subsequent field on a word (2-byte) boundary. It contains no information. One of two options that is a single byte in length, having no Data subfield (the other is the End option).
1	4	Subnet Mask: A 32-bit subnet mask being supplied for the client to use on the current network. It must appear in the option list before the Router option if both are present.
2	4	Time Offset: Specifies the time offset of the client's subnet in seconds from *Coordinated Universal Time (UTC, formerly Greenwich Mean Time or GMT)*. Positive values represent areas east of the prime meridian (in the United Kingdom); negative values represent areas west of the prime meridian. Essentially, this is used to indicate the time zone of the subnet.
3	Variable (multiple of 4)	Router: Specifies a list of 32-bit router addresses for the client to use on the local network. Routers are listed in the order of preference for the client to use.

(continued)

Table 63-5: DHCP/BOOTP Options: RFC 1497 Vendor Extensions (continued)

Code Value	Data Length (Bytes)	Name and Description
4	Variable (multiple of 4)	Time Server: Specifies a list of time server addresses (per RFC 868, see Chapter 88) for the client to use on the local network. Servers are listed in the order of preference for the client to use.
5	Variable (multiple of 4)	IEN-116 Name Server: Specifies a list of IEN-116 name server addresses for the client to use on the local network. Servers are listed in the order of preference for the client to use. Note that this option is not used for DNS name servers.
6	Variable (multiple of 4)	DNS Name Server: Specifies a list of DNS (see Chapter 52) name server addresses for the client to use on the local network. Servers are listed in the order of preference for the client to use.
7	Variable (multiple of 4)	Log Server: Specifies a list of MIT-LCS UDP log server addresses for the client to use on the local network. Servers are listed in the order of preference for the client to use.
8	Variable (multiple of 4)	Cookie Server: Specifies a list of RFC 865 cookie server addresses for the client to use on the local network. Servers are listed in the order of preference for the client to use.
9	Variable (multiple of 4)	LPR Server: Specifies a list of RFC 1179 line printer server addresses for the client to use on the local network. Servers are listed in the order of preference for the client to use.
10	Variable (multiple of 4)	Impress Server: Specifies a list of Imagen Impress server addresses for the client to use on the local network. Servers are listed in the order of preference for the client to use.
11	Variable (multiple of 4)	Resource Location Server: Specifies a list of RFC 887 resource location server addresses for the client to use on the local network. Servers are listed in the order of preference for the client to use.
12	Variable	Host Name: Specifies a host name for the client. This may or may not be a DNS host name; see option15.
13	2	Boot File Size: Specifies the size of the default boot image file for the client, expressed in units of 512 bytes.
14	Variable	Merit Dump File: Specifies the path and filename of the file to which the client should dump its core image in the event that it crashes.
15	Variable	Domain Name: Specifies the DNS domain name for the client. Compare this with option 12.
16	4	Swap Server: Specifies the address of the client's swap server.
17	Variable	Root Path: Specifies the path name of the client's root disk. This allows the client to access files it may need, using a protocol such as the Network File System (NFS; see Chapter 58).
18	Variable	Extensions Path: Specifies the name of a file that contains vendor-specific fields that the client can interpret in the same way as the Options or Vend field in a DHCP/BOOTP message. This was defined to allow a client and server to still exchange vendor-specific information even though the Option/Vend field is now used for the general-purpose fields described in this chapter. Also see option 43.
255	0	End: Placed after all other options to mark the end of the option list. One of two options that is a single byte in length, having no Data subfield (the other is the Pad option).

IP Layer Parameters per Host

Table 63-6 shows the parameters that control the operation of IP on a host as a whole. They are not interface-specific.

Table 63-6: DHCP/BOOTP Options: IP Layer Parameters per Host

Code Value	Data Length (Bytes)	Name and Description
19	1	IP Forwarding Enable/Disable: A value of 1 turns on IP forwarding (that is, routing) on a client that is capable of that function; a value of 0 turns it off.
20	1	Non-Local Source Routing Enable/Disable Option: A value of 1 tells a client capable of routing to allow forwarding of IP datagrams with nonlocal source routes. A value of 0 tells the client not to allow this. See the source routing IP datagram option (see Chapter 21) for a bit more information on this and option 21.
21	Variable (multiple of 8)	Policy Filter: A set of address/mask pairs used to filter nonlocal source-routed datagrams.
22	2	Maximum Datagram Reassembly Size: Tells the client the size of the largest datagram that the client should be prepared to reassemble. The minimum value is 576 bytes.
23	1	Default IP Time to Live: Specifies the default value that the client should use for the Time to Live field in creating IP datagrams.
24	4	Path MTU Aging Timeout: Specifies the number of seconds the client should use in aging path maximum transmission unit (MTU) values determined using path MTU discovery.
25	Variable (multiple of 2)	Path MTU Plateau Table: Specifies a table of values to be used in performing path MTU discovery.

IP Layer Parameters per Interface

Table 63-7 shows the parameters that are specific to a particular host interface at the IP level.

Table 63-7: DHCP/BOOTP Options: IP Layer Parameters per Interface

Code Value	Data Length (Bytes)	Name and Description
26	2	Interface MTU: Specifies the MTU to be used for IP datagrams on this interface. The minimum value is 68.
27	1	All Subnets Are Local: When set to 1, tells the client that it may assume that all subnets of the IP network it is on have the same MTU as its own subnet. When 0, the client must assume that some subnets may have smaller MTUs than the client's subnet.
28	4	Broadcast Address: Tells the client what address it should use for broadcasts on this interface.
29	1	Perform Mask Discovery: A value of 1 tells the client that it should use Internet Control Message Protocol (ICMP; see Chapter 31) to discover a subnet mask on the local subnet. A value of 0 tells the client not to perform this discovery.
30	1	Mask Supplier: Set to 1 to tell the client that it should respond to ICMP subnet mask requests on this interface.
31	1	Perform Router Discovery: A value of 1 tells the client to use the ICMP router discovery process to solicit a local router. A value of 0 tells the client to not do so. Note that DHCP itself can be used to specify one or more local routers using option 3.

(continued)

Table 63-7: DHCP/BOOTP Options: IP Layer Parameters per Interface (continued)

Code Value	Data Length (Bytes)	Name and Description
32	4	Router Solicitation Address: Tells the client the address to use as the destination for router solicitations.
33	Variable (multiple of 8)	Static Route: Provides the client with a list of static routes it can put into its routing cache. The list consists of a set of IP address pairs; each pair defines a destination and a router to be used to reach the destination.

Link Layer Parameters per Interface

Table 63-8 lists the DHCP/BOOTP options that are specific to a particular link layer (layer 2) interface.

Table 63-8: DHCP/BOOTP Options: Link Layer Parameters per Interface

Code Value	Data Length (Bytes)	Name and Description
34	1	Trailer Encapsulation: When set to 1, tells the client to negotiate the use of trailers, as defined in RFC 893. A value of 0 tells the client not to use this feature.
35	4	ARP Cache Timeout: Specifies how long, in seconds, the client should hold entries in its ARP cache (see Chapter 13).
36	1	Ethernet Encapsulation: Tells the client what type of encapsulation to use when transmitting over Ethernet at layer 2. If the option value is 0, it specifies that Ethernet II encapsulation should be used, per RFC 894; when the value is 1, it tells the client to use IEEE 802.3 encapsulation, per RFC 1042.

TCP Parameters

The options impacting the operation of TCP are shown in Table 63-9.

Table 63-9: DHCP/BOOTP Options: TCP Parameters

Code Value	Data Length (Bytes)	Name and Description
37	1	Default TTL: Specifies the default TTL the client should use when sending TCP segments.
38	4	TCP Keepalive Interval: Specifies how long (in seconds) the client should wait on an idle TCP connection before sending a keepalive message. A value of 0 instructs the client not to send such messages unless specifically instructed to do so by an application.
39	1	TCP Keepalive Garbage: When set to 1, tells a client it should send TCP keepalive messages that include an octet of "garbage" for compatibility with implementations that require this.

Application and Service Parameters

Table 63-10 shows the miscellaneous options that control the operation of various applications and services.

Table 63-10: DHCP/BOOTP Options: Application and Service Parameters

Code Value	Data Length (Bytes)	Name and Description
40	Variable	Network Information Service Domain: Specifies the client's Network Information Service (NIS) domain. Contrast this with option 64.
41	Variable (multiple of 4)	Network Information Servers: Specifies a list of IP addresses of NIS servers the client may use. Servers are listed in the order of preference for the client to use. Contrast this with option 65.
42	Variable (multiple of 4)	Network Time Protocol Servers: Specifies a list of IP addresses of Network Time Protocol (NTP) servers the client may use. Servers are listed in the order of preference for the client to use.
43	Variable	Vendor Specific Information: Allows an arbitrary set of vendor-specific information items to be included as a single option within a DHCP or BOOTP message. This information is structured using the same format as the Options or Vend field itself, except that it does not start with a magic cookie. See the "DHCP Options" section earlier in this chapter for more details.
44	Variable (multiple of 4)	NetBIOS over TCP/IP Name Servers: Specifies a list of IP addresses of NetBIOS name servers (per RFC 1001/1002) that the client may use. Servers are listed in the order of preference for the client to use.
45	Variable (multiple of 4)	NetBIOS over TCP/IP Datagram Distribution Servers: Specifies a list of IP addresses of NetBIOS datagram distribution servers (per RFC 1001/1002) that the client may use. Servers are listed in the order of preference for the client to use.
46	1	NetBIOS over TCP/IP Node Type: Tells the client what sort of NetBIOS node type it should use. Four different bit values are used to define the possible node type combinations, as listed in Table 63-11.
47	Variable	NetBIOS over TCP/IP Scope: Specifies the NetBIOS over TCP/IP scope parameter for the client.
48	Variable (multiple of 4)	X Window System Font Servers: Specifies a list of IP addresses of X Window System Font servers that the client may use. Servers are listed in the order of preference for the client to use.
49	Variable (multiple of 4)	X Window System Display Manager: Specifies a list of IP addresses of systems running the X Window System Display Manager that the client may use. Addresses are listed in the order of preference for the client to use.
64	Variable	Network Information Service+ Domain: Specifies the client's NIS+ domain. Contrast this with option 40.
65	Variable (multiple of 4)	Network Information Service+ Servers: Specifies a list of IP addresses of NIS+ servers the client may use. Servers are listed in the order of preference for the client to use. Contrast this with option 41.
68	Variable (multiple of 4)	Mobile IP Home Agent: Specifies a list of IP addresses of home agents that the client can use in Mobile IP (see Chapter 30). Agents are listed in the order of preference for the client to use; normally a single agent is specified.
69	Variable (multiple of 4)	Simple Mail Transport Protocol (SMTP) Servers: Specifies a list of IP addresses of SMTP servers the client may use. Servers are listed in the order of preference for the client to use. See Chapter 77 for more on SMTP.
70	Variable (multiple of 4)	Post Office Protocol (POP3) Servers: Specifies a list of IP addresses of POP3 servers the client may use. Servers are listed in the order of preference for the client to use. See Chapter 78.
71	Variable (multiple of 4)	Network News Transfer Protocol (NNTP) Servers: Specifies a list of IP addresses of NNTP servers the client may use. Servers are listed in the order of preference for the client to use. See Chapter 85.
72	Variable (multiple of 4)	Default World Wide Web (WWW) Servers: Specifies a list of IP addresses of World Wide Web (HTTP) servers the client may use. Servers are listed in the order of preference for the client to use. See Chapter 79.

(continued)

Table 63-10: DHCP/BOOTP Options: Application and Service Parameters (continued)

Code Value	Data Length (Bytes)	Name and Description
73	Variable (multiple of 4)	Default Finger Servers: Specifies a list of IP addresses of Finger servers the client may use. Servers are listed in the order of preference for the client to use.
74	Variable (multiple of 4)	Default Internet Relay Chat (IRC) Servers: Specifies a list of IP addresses of Internet Relay Chat (IRC) servers the client may use. Servers are listed in the order of preference for the client to use.
75	Variable (multiple of 4)	StreetTalk Servers: Specifies a list of IP addresses of StreetTalk servers the client may use. Servers are listed in the order of preference for the client to use.
76	Variable (multiple of 4)	StreetTalk Directory Assistance (STDA) Servers: Specifies a list of IP addresses of STDA servers the client may use. Servers are listed in the order of preference for the client to use.

Table 63-11: NetBIOS Over TCP/IP Node Type (Option 46) Values

Option 46 Subfield Name	Size (Bits)	Description
Reserved	4	Reserved: Not used.
H-Node	1	H-Node: Set to 1 to tell the client to act as a NetBIOS H-node.
M-Node	1	M-Node: Set to 1 to tell the client to act as a NetBIOS M-node.
P-Node	1	P-Node: Set to 1 to tell the client to act as a NetBIOS P-node.
B-Node	1	B-Node: Set to 1 to tell the client to act as a NetBIOS B-node.

DHCP Extensions

Last, but certainly not least, Table 63-12 describes the DHCP-only options that control the operation of the DHCP protocol.

Table 63-12: DHCP Options: DHCP Extensions

Code Value	Data Length (Bytes)	Name and Description
50	4	Requested IP Address: Used in a client's DHCPDISCOVER message to request a particular IP address assignment.
51	4	IP Address Lease Time: Used in a client request to ask a server for a particular DHCP lease duration, or in a server reply to tell the client the offered lease time. It is specified in units of seconds.
52	1	Option Overload: Used to tell the recipient of a DHCP message that the message's SName and/or File fields are being used to carry options, instead having their normal meanings. This option implements the option overload feature. There are three possible values for this single-byte option: 1 means the File field is carrying the option data, 2 means the SName field has the option data, and 3 means both fields have the option data.
53	1	DHCP Message Type: Indicates the specific type of DHCP message, as listed in Table 6-13.

(continued)

Table 63-12: DHCP Options: DHCP Extensions (continued)

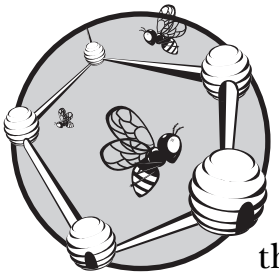
Code Value	Data Length (Bytes)	Name and Description
54	4	Server Identifier: The IP address of a particular DHCP server. This option is included in messages sent by DHCP servers to identify themselves as the source of the message. It is also used by a client in a DHCPREQUEST message to specify which server's lease it is accepting.
55	Variable	Parameter Request List: Used by a DHCP client to request a list of particular configuration parameter values from a DHCP server.
56	Variable	Message: Used by a server or client to indicate an error or other message.
57	2	Maximum DHCP Message Size: Used by a DHCP client or server to specify the maximum size of DHCP message it is willing to accept. The minimum legal value is 576 bytes.
58	4	Renewal (T1) Time Value: Tells the client the value to use for its renewal timer.
59	4	Rebinding (T2) Time Value: Tells the client the value to use for its rebinding timer.
60	Variable	Vendor Class Identifier: Included in a message sent by a DHCP client to specify its vendor and configuration. This may be used to prompt a server to send the correct vendor-specific information using option 43.
61	Variable	Client Identifier: Used optionally by a client to specify a unique client identification for itself that differs from the DHCP default. This identifier is expected by servers to be unique among all DHCP clients and is used to index the DHCP server's configuration parameter database.
66	Variable	TFTP Server Name: When the DHCP message's SName field has been used for options using the option overload feature, this option may be included to specify the Trivial File Transfer Protocol (TFTP) server name that would normally appear in the SName field.
67	Variable	Bootfile Name: When the DHCP message's File field has been used for options using the option overload feature, this option may be included to specify the boot filename that would normally appear in the File field.

Table 63-13: DHCP Message Type (Option 53) Values

Option 53 Value	DHCP Message Type
1	DHCPDISCOVER
2	DHCPOFFER
3	DHCPREQUEST
4	DHCPDECLINE
5	DHCPACK
6	DHCPNAK
7	DHCPRELEASE
8	DHCPINFORM

64

DHCP CLIENT/SERVER IMPLEMENTATION, FEATURES, AND IPV6 SUPPORT



The preceding chapters in this part describe the fundamentals of the operation of DHCP: the address leasing system, configuration processes, and messaging. With this foundation in place, we can now proceed to look into some of the more interesting details of how DHCP is implemented. We can also delve into some of the extra capabilities and special features that change the basic DHCP mechanisms we have already studied.

In this chapter, I discuss DHCP client/server implementation issues, special features that enhance the protocol, and some of the problems and issues related to making DHCP work. I begin with a discussion of DHCP server and client implementation and management issues. I discuss DHCP message relaying and how it is related to the relaying feature used for the Boot Protocol (BOOTP). I describe the DHCP feature for providing automatic default addressing when a client cannot contact a server, and the conflict detection feature for multiple servers. I then cover some of the issues related

to interoperability of DHCP and BOOTP, and provide an outline of some of the more important problems and issues related to DHCP security. I conclude with an overview of DHCP for IP version 6 (DHCPv6).

DHCP Server and Client Implementation and Management Issues

DHCP is a client/server protocol, relying on both the server and client to fulfill certain responsibilities. Of the two device roles, the DHCP server is arguably the more important, because it is in the server that most of the functionality of DHCP is actually implemented.

DHCP Server Implementations

The server maintains the configuration database, keeps track of address ranges, and manages leases. For this reason, DHCP servers are typically much more complex than DHCP clients. In essence, without a DHCP server, there really is no DHCP. Thus, deciding how to implement DHCP servers is a large part of implementing the protocol.

A classic DHCP server consists of DHCP server software running on a server hardware platform of one sort or another. A DHCP server usually will not be a dedicated computer, except on very large networks. It is more common for a hardware server to provide DHCP services along with performing other functions, such as acting as an application server, serving as a general database server, providing DNS services, and so forth. So, a DHCP server does not need to be a special computer; any device that can run a DHCP server implementation can act as a server.

In fact, the DHCP server may not even need to be a host computer at all. Today, many routers include DHCP functionality. Programming a router to act as a DHCP server allows clients that connect to the router to be automatically assigned IP addresses. This provides numerous potential advantages in an environment where a limited number of public IP addresses is shared among multiple clients, or where IP Network Address Translation (NAT; see Chapter 28) is used to dynamically share a small number of addresses. Since DHCP requires a database, a router that acts as a DHCP server requires some form of permanent storage. This is often implemented using flash memory on routers; "true" servers use hard disk storage.

Virtually all modern operating systems include support for DHCP, including most variants of UNIX, Linux, newer versions of Microsoft Windows, Novell NetWare, and others. In some cases, you may need to run the server version of the operating system to have a host act as a DHCP server. For example, while Microsoft Windows XP supports DHCP, I don't believe that a DHCP server comes in the Windows XP Home Edition, though you could install one yourself.

DHCP Server Software Features

In most networks, you will choose the operating system based on a large number of factors. The choice of operating system will then dictate what options you have for selecting DHCP server software. Most common operating systems have a number of

options available for software. While all will implement the core DHCP protocol, they will differ in terms of the usual software attributes: cost, performance, ease of use, and so on. They may also differ in terms of their features, such as the following:

- How they allow address ranges (scopes) to be defined
- How clients can be grouped and managed
- The level of control an administrator has over parameters returned to a client
- The level of control an administrator has over general operation of the protocol, such as specification of the T1 and T2 timers and other variables, and how leases are allocated and renewals handled
- Security features
- Ability to interact with DNS to support dynamic device naming
- Optional features such as BOOTP support, conflict detection, and Automatic Private IP Addressing (all discussed later in this chapter)

Choosing the Number of Servers

In setting up DHCP for a network, there are a number of important factors to consider and decisions to be made. One of the most critical is the number of servers you want to have. In theory, each network requires only one DHCP server; in practice, this is often not a great idea. Servers sometimes experience hardware or software failures, or they must be taken down for maintenance. If there is only one server and clients can't reach it, no DHCP clients will be able to get addresses. For this reason, two or more servers are often used.

If you do use more than one server, you need to carefully plan how you will configure each one. One of the first decisions you will need to make is which servers will be responsible for which addresses and clients. You need to determine whether you want the servers to have distinct or overlapping address pools, as discussed in the explanation of DHCP address ranges in Chapter 61. Distinct pools ensure that addresses remain unique, but result in unallocatable addresses if a server fails. Overlapping addresses are more flexible, but risk address conflicts unless a feature like conflict detection (described later in this chapter) is used.

Server Placement, Setup, and Maintenance

Once you know how many servers you want, you need to determine on which part of the network you want to place them. If you have many physical networks, you may also need to use DHCP relaying to allow all clients to reach a server. Since the structure of the network may affect the number of servers you use, many of these decisions are interrelated.

You must make policy decisions related to all the DHCP operating parameters discussed in the previous chapters. The two big decisions are the size and structure of the address pool, and making lease policy decisions such as the lease length and the settings for the T1 and T2 timers. You also must decide which clients will be dynamically allocated addresses and how manually configured clients will be handled.

Finally, it's essential for the administrator to remember that an organization's DHCP server is a database server and must be treated accordingly. Like any database server, it must be maintained and managed carefully. Administrative policies must be put into place to ensure the security and efficient operation of the server. Also, unlike certain other types of database systems, the DHCP database is not automatically replicated; the server database should therefore be routinely backed up, and using RAID storage is also a good idea.

DHCP Client Implementations

Just as a DHCP server consists of server software running on a server platform or hardware acting as a server, a DHCP client is simply DHCP client software running on a client device. Most often, a client device is a host computer connected to a TCP/IP internetwork. DHCP is so widely accepted today that virtually all hosts include DHCP client software. The DHCP client is usually integrated into graphical operating systems like Windows, or is implemented using a specific client daemon like *dhclient* or *dhcpd* on UNIX/Linux.

Since the entire idea behind DHCP is to put the server in charge of parameter storage, configuration, and address management, DHCP clients are relatively simple. The client implements the messaging protocol and communicates parameters received from the DHCP server to the rest of the TCP/IP software components as needed. It doesn't do a whole lot else.

In fact, there's not really much for an administrator to do to set up a client to use DHCP. In some operating systems, it's as simple as "throwing a switch," by enabling DHCP support within the client itself. This prompts the client to then stop using any manually configured parameters and start searching for a DHCP server instead. The server then becomes responsible for the client's configuration and address assignment.

Since the client doesn't do a great deal in DHCP other than communicate with the server, not much is required in the way of user software for a DHCP client. In most cases, control over the DHCP client software is accomplished using a TCP/IP configuration utility, as described in Chapter 88. Windows clients use the programs *ipconfig* or *winipcfg* to display the status of their current DHCP leases. These programs also allow the client to manually release the current lease or renew it.

Releasing the lease means early lease termination using the DHCPRELEASE message. This is usually the only way that a lease is terminated. Renewing the lease is a manual version of the automated renewal process. Releasing and renewing the lease may be done in sequence to reset a client that is in a confused state or is having some other type of DHCP or connectivity problem.

DHCP Message Relaying and BOOTP Relay Agents

DHCP is the third-generation host configuration protocol for TCP/IP. We've already discussed extensively how it was based directly on BOOTP, which was, in turn, an enhancement of the earlier Reverse Address Resolution Protocol (RARP). Even though each new protocol has made significant improvements over its predecessor, each iteration has retained certain limitations that are actually common to all host configuration protocols.

One of the most important limitations with host configuration protocols is the reliance on broadcasts for communication. Whenever we are dealing with a situation where a client needs to communicate but doesn't know its IP address and doesn't know the address of a server that will provide it, the client needs to use broadcast addressing. However, for performance reasons, broadcasts are normally propagated only on the local network. This means that the client and server would always need to be on the same physical network for host configuration to occur. Of course, we don't want this to be the case. It would require that a large internetwork have a different server on every network, greatly reducing the benefits of centralized configuration information and creating numerous administrative hassles.

RARP didn't have any solution to this problem, which is one reason why it was so limited in usefulness. BOOTP's solution is to allow a client and server to be on different networks through the use of BOOTP *relay agents*.

BOOTP Relay Agents for DHCP

A *relay agent* is a device that is not a BOOTP server, but which runs a special software module that allows it to act in the place of a server. A relay agent can be placed on networks where there are BOOTP clients but no BOOTP servers. The relay agent intercepts requests from clients and relays them to the server. The server then responds back to the agent, which forwards the response to the client. A full rationale and description of operation of BOOTP relay agents can be found in Chapter 60.

The designers of DHCP were satisfied with the basic concepts and operation behind BOOTP relay agents, which had already been in use for many years. For this reason, they made the specific decision to continue using BOOTP relay agent functionality in DHCP. In fact, this is one of the reasons why the decision was made to retain the BOOTP message format in DHCP, and also the basic two-message request/reply communication protocol. This allows BOOTP relay agents to handle DHCP messages as if they were BOOTP messages. This is also why the mention of BOOTP in the title of this topic is not a typo—DHCP uses BOOTP relay agents. Even the DHCP standard says that a "BOOTP relay agent is an Internet host or router that passes DHCP messages between DHCP clients and DHCP servers."

In practice, the agents are indeed sometimes called *DHCP relay agents*. You may also see the terms *BOOTP/DHCP relay agent* and *DHCP/BOOTP relay agent*.

DHCP Relaying Process

Since DHCP was designed specifically to support BOOTP relay agents, the agents behave in DHCP much as they do in BOOTP. Although DHCP has much more complex message exchanges, they are all still designed around the notion of a client request and server response. There are just more requests and responses.

The BOOTP agent looks for broadcasts sent by the client and then forwards them to the server (as described in the BOOTP relay agent behavior discussion in Chapter 60), and then returns replies from the server. The additional information in the DHCP protocol is implemented using additions to the BOOTP message format in the form of DHCP options, which the relay agent doesn't look at. It just treats them as it does BOOTP requests and replies.

KEY CONCEPT To permit DHCP clients and DHCP servers to reside on different physical networks, an intermediary device is required to facilitate message exchange between networks. DHCP uses the same mechanism for this as BOOTP: the deployment of *BOOTP relay agents*. The relay agent captures client requests, forwards them to the server, and then returns the server's responses back to the client.

In summary, when a relay agent is used, here's what the various client requests and server replies in the DHCP operation section become:

Client Request When a client broadcasts a request, the relay agent intercepts it on UDP port 67. It checks the Hops field and discards the request if the value is greater than 16; otherwise, it increments the field. The agent puts its own address into the GIAddr field unless another relay agent has already put its address in the field. It then forwards the client request to a DHCP server, either unicast or broadcast on another network.

Server Reply The server sees a nonzero value in the GIAddr field and sends the reply to the relay agent whose IP address is in that field. The relay agent then sends the reply back to the client, using either unicast or broadcast (as explained in the discussion of DHCP addressing in Chapter 61).

One difference between BOOTP and DHCP is that certain communications from the client to the server are unicast. The most noticeable instance of this is when a client tries to renew its lease with a specific DHCP server. Since it sends this request unicast, it can go to a DHCP server on a different network using conventional IP routing, and the relay agent does not need to be involved.

DHCP Autoconfiguration/Automatic Private IP Addressing (APIPA)

The IP address of a TCP/IP host is, in many ways, its identity. Every TCP/IP network requires that all hosts have unique addresses to facilitate communication. When a network is manually configured with a distinct IP address for each host, the hosts permanently know who they are. When hosts are made DHCP clients, they no longer have a permanent identity; they rely on a DHCP server to tell them who they are.

This dependency is not a problem as long as DHCP is functioning normally and a host can get a lease, and, in fact, has many benefits that we have explored. Unfortunately, a number of circumstances can arise that result in a client failing to get a lease. The client may not be able to obtain a lease, reacquire one after reboot, or renew an existing lease. There are several possible reasons why this might happen:

- The DHCP server may have experienced a failure or may be taken down for maintenance.
- The relay agent on the client's local network may have failed.

- Another hardware malfunction or power failure may make communication impossible.
- The network may have run out of allocatable addresses.

Without a lease, the host has no IP address, and without an address, the host is effectively dead in the water. The base DHCP specification doesn't really specify any recourse for the host in the event that it cannot successfully obtain a lease. It is left up to the implementor to decide what to do, and when DHCP was first created, many host implementations would simply display an error message and leave the host unusable until an administrator or user took action.

Clearly, this is far from an ideal situation. It would be better if we could just have a DHCP client that is unable to reach a server automatically configure itself. In fact, the Internet Engineering Task Force (IETF) reserved a special IP address block for this purpose (see Chapter 17). This block, 169.254.0.1 through 169.254.255.254 (or 169.254.0.0/16 in classless notation) is reserved for autoconfiguration, as mentioned in RFC 3330, "Hosts obtain these addresses by auto-configuration, such as when a DHCP server may not be found."

Strangely, however, no TCP/IP standard was defined to specify how such autoconfiguration works. To fill the void, Microsoft created an implementation that it calls *Automatic Private IP Addressing (APIPA)*. Due to Microsoft's market power, APIPA has been deployed on millions of machines, and has thus become a de facto standard in the industry. Many years later, the IETF did define a formal standard for this functionality, in RFC 3927, "Dynamic Configuration of IPv4 Link-Local Addresses."

APIPA Operation

APIPA is really so simple that it's surprising it took so long for someone to come up with the idea. It takes over at the point where any DHCP lease process fails. Instead of just halting with an error message, APIPA randomly chooses an address within the aforementioned private addressing block. It then performs a test very similar to the one in step 13 in the DHCP allocation process (see Chapter 62): It uses ARP to generate a request on the local network to see if any other client responds using the address it has chosen. If there is a reply, APIPA tries another random address and repeats the test. When the APIPA software finds an address that is not in use, it is given to the client as a default address. The client will then use default values for other configuration parameters that it would normally receive from the DHCP server. This process is illustrated in Figure 64-1.

A client using an autoconfigured address will continue to try to contact a DHCP server periodically. By default, this check is performed every five minutes. If and when it finds one, it will obtain a lease and replace the autoconfigured address with the proper leased address.

APIPA is ideally suited to small networks, where all devices are on a single physical link. Conceivably, with 20 APIPA-enabled DHCP clients on a network with a single DHCP server, you could take the server down for maintenance and still have all the clients work properly, using 169.254.*x.x* addresses.

Bear in mind, however, that APIPA is not a proper replacement for full DHCP.

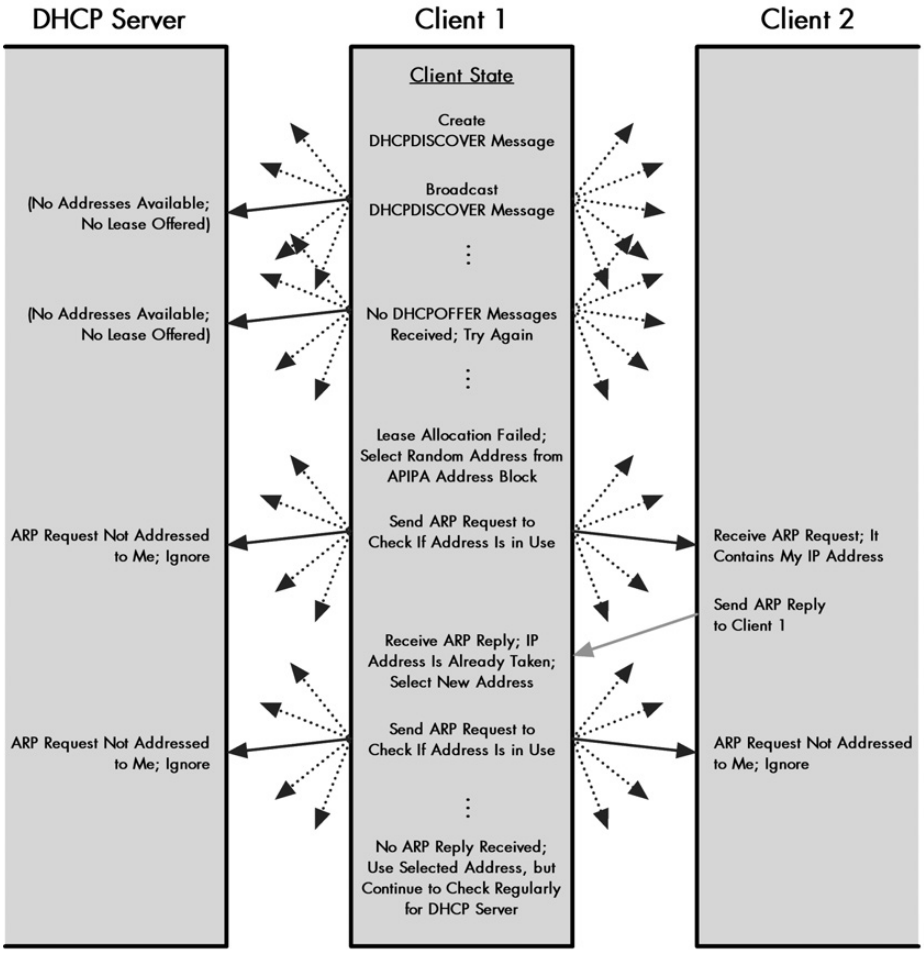


Figure 64-1: DHCP Automatic Private IP Addressing (APIPA) *In this example, Client 1 is trying to get an IP address from its DHCP server, but the server is out of addresses, so it does not respond to the client's requests. The client is configured to use APIPA, so it randomly selects an address from the APIPA address block. It sends an ARP request on the local network to see if any other device is using that address. Usually, there will be no conflict, but here Client 2 is using the address, so it responds. Client 1 chooses a different address, and this time gets no reply. It begins using that address, while continuing to check regularly for a DHCP server to come online.*

APIPA Limitations

The 169.254.0.0/16 block is a private IP range and comes with all the limitations of private IP addresses, including inability to use these addresses on the Internet. Also, APIPA cannot provide the other configuration parameters that a client may need to get from a DHCP server. Finally, APIPA will not work properly in conjunction with proxy ARP, because the proxy will respond for any of the private addresses, so they will all appear to be used.

Since it uses ARP to check for address conflicts, APIPA is not well suited for large internetworks. To use it on an internetwork with multiple subnets would require software that allows each subnet to use a different portion of the full 169.254.0.0/16 blocks, to avoid conflicts.

In practice, APIPA is a solution for small networks. Large internetworks deal with the problem of not being able to contact a DHCP server by taking steps to ensure that a client can always contact a DHCP server.

KEY CONCEPT An optional DHCP feature called *Automatic Private IP Addressing (APIPA)* was developed to allow clients to still be able to communicate in the event that they are unable to obtain an IP address from a DHCP server. When enabled, the client chooses a random address from a special reserved block of private IP addresses and checks to make sure the address is not already in use by another device. It continues to check for a DHCP server periodically until it is able to find one.

DHCP Server Conflict Detection

One of the primary decisions any TCP/IP administrator using DHCP must make is how many DHCP servers to deploy. A single server has the advantage of simplicity, but provides no redundancy in the event of failure. It also means that whenever the DHCP server is down, clients can't get addresses. For these reasons, most larger networks use two or more servers.

When you have two servers or more—let's say two for sake of this discussion—you then have another decision to make: How do you divide the address pool between the servers? As I explored in detail in the discussion of DHCP address pools in Chapter 61, there are two options: give the servers overlapping addresses or making them non-overlapping. Unfortunately, in classic DHCP, neither is really a great solution. Overlapping ranges mean both servers might try to assign the same address, since DHCP includes no provision for communication between servers. Non-overlapping ranges avoid this problem, but make only some of the addresses available to each server.

It's strange that the DHCP standard didn't provide better support for cross-server coordination, even though there clearly was a need for it. However, certain DHCP implementations include an optional feature to allow two servers to have overlapping scopes without address clashes occurring. This is a feature commonly found on Microsoft DHCP servers and may also be present in other implementations. It is called *DHCP server conflict detection*.

The idea behind conflict detection is very simple. Suppose a DHCP server receives a DHCPDISCOVER message from a client and decides to offer it a lease. Before sending the DHCPOFFER message, the server conducts a *probe* by sending ICMP Echo (ping) messages (see Chapter 33) out to the address it plans to offer. It then waits a short period of time to hear if it receives any ICMP Echo Reply messages back. If it does, it knows the IP address is in use and chooses a different one.

If all DHCP servers are configured to do this before offering an address, then it is possible to give all of them the same, overlapping addresses for assignment. They won't have any way of coordinating with each other, but as long as they ask first by doing an ICMP check, there won't be any problems. This provides an administrator

with the advantages of overlapping address ranges—simplicity and access to all addresses by all servers—without risk of address conflicts. The only small drawback is a little extra network traffic to perform the check, and possibly a few milliseconds of server CPU time if a new address needs to be chosen.

If you were paying attention when you read about the DHCP allocation process in Chapter 62, you may have noticed that what I am describing here sounds familiar. In fact, it's true that this feature isn't anything new. The use of ICMP to check an address before offering it is actually mentioned in RFC 2131 as part of the standard DHCP allocation process, and you can find it mentioned as step 5 in the allocation process description.

So why was conflict detection required to be an extra feature? The reason is that the use of ICMP wasn't mandatory because the standard says servers *should* do it, not that they *must* do it. This choice was made to provide flexibility in implementing DHCP, but that flexibility comes at a cost. So, if you want to use this feature, you need to look for support for it in your server software.

KEY CONCEPT Some DHCP implementations include a feature called *server conflict detection*. When this feature is activated, it causes each server to always check to make sure an address is not in use before granting it to a client. When conflict detection is used by all DHCP servers on a network, the servers can be given overlapping scopes, so each can assign any of the organization's IP addresses, while at the same time not needing to be concerned about two clients being assigned the same address by different servers.

DHCP and BOOTP Interoperability

I've talked extensively about how DHCP was designed based on BOOTP and how they use the same basic communication method and message format. This was done for several reasons, one of the most important of which was ensuring interoperability of the two protocols. Given this, you might expect that we could simply say that BOOTP and DHCP are compatible with each other, and that's that.

It is true that DHCP was intended to be compatible with BOOTP. RFC 2131 lists the following as one of DHCP's design goals: "DHCP must provide service to existing BOOTP clients." This seems pretty clear. The reuse of the BOOTP message format is one of the keys to DHCP and BOOTP compatibility. DHCP functionality is implemented not through new fields, but rather through DHCP-specific options, such as the DHCP Message Type option that specifies the all-important type of DHCP messages. DHCP devices can look for this extra information, while BOOTP devices can ignore it.

However, while DHCP and BOOTP are similar, they are not the same, and so there are some interoperability concerns that crop up when they are used together. The DHCP message format is structurally the same as the BOOTP format, but the interpretation of certain fields is slightly different. BOOTP clients don't understand DHCP, so when BOOTP and DHCP are used together, the DHCP client or server must sometimes behave slightly differently to compensate. Further complicating matters are the facts that not all implementations of DHCP and BOOTP are necessarily exactly the same and that certain specifications in the DHCP standard are not mandatory.

For these reasons, we cannot just assume that DHCP and BOOTP will work together. To address some of these issues, the IETF published RFC 1534, "Interoperation Between DHCP and BOOTP," at the same time that DHCP was originally created. This document looks at how the protocols work together, focusing on the two distinct client/server interoperating combinations: a BOOTP client connecting to a DHCP server, and a DHCP client connecting to a BOOTP server. Let's consider each case.

BOOTP Clients Connecting to a DHCP Server

As indicated by the preceding quote from RFC 2131, DHCP was specifically intended to allow a DHCP server to handle requests from BOOTP clients. The protocol itself is set up to enable this, but it does require that the DHCP server be given certain intelligence to know how to deal with BOOTP clients.

One of the most important issues is that BOOTP clients will follow the BOOTP configuration process and not the DHCP leasing processes. The DHCP server must use BOOTP messages with the BOOTP meanings for fields when dealing with BOOTP clients. A server determines that a client is using BOOTP instead of DHCP by looking for the presence of the DHCP Message Type option, which must be present in all DHCP messages but is not used for BOOTP.

If a DHCP server detects that it is dealing with a BOOTP client, it can respond with configuration information for the client. The server can use either manual or automatic allocation for the client. Automatic allocation means the server chooses an address from its pool of unused addresses, but assigns it permanently. BOOTP clients are not capable of dynamic allocation, since BOOTP is static in nature.

A DHCP server may include BOOTP vendor information fields in its response to a BOOTP client, including ones defined since BOOTP was created. However, it obviously must not send any DHCP-specific options.

DHCP Clients Connecting to a BOOTP Server

A DHCP client can obtain configuration information from a BOOTP server, because the server will respond to the client's initial DHCPDISCOVER message as if it were a BOOTP BOOTREQUEST message. The DHCP client can tell that a BOOTP reply has been received because there will be no DHCP Message Type option.

A response from a BOOTP server should be treated as an infinite lease, since again, that's all that BOOTP supports. Note that if a DHCP client receives a response from both a BOOTP server and a DHCP server, it should use the DHCP response and not the BOOTP response (even if this means it gets a shorter lease).

DHCP Security Issues

DHCP was designed in the early 1990s, when the number of organizations on the Internet was relatively small. Furthermore, it was based on BOOTP, which was created in the 1980s, when the Internet as we know it today barely even existed. In those days, Internet security wasn't a big issue, because it was mostly a small group

of research and educational organizations using TCP/IP on the Internet. As a result, DHCP, like many protocols of that era, doesn't do much to address security concerns.

Actually, this is a bit understated. Not only does DHCP run over the Internet Protocol (IP) and the User Datagram Protocol (UDP), which are inherently insecure, but the DHCP protocol itself has no security provisions whatsoever. This is a fairly serious issue in modern networks, because of the sheer power of DHCP, which deals with critical configuration information.

DHCP Security Concerns

There are two different classes of potential security problems related to DHCP:

Unauthorized DHCP Servers If a malicious person plants a rogue DHCP server, it is possible that this device could respond to client requests and supply them with spurious configuration information. This could be used to make clients unusable on the network, or worse, set them up for further abuse later on. For example, a hacker could exploit a bogus DHCP server to direct a DHCP client to use a router under the hacker's control, rather than the one the client is supposed to use.

Unauthorized DHCP Clients A client could be set up that masquerades as a legitimate DHCP client and thereby obtain configuration information intended for that client. This information could then be used to compromise the network later on. Alternatively, a malicious person could use software to generate a lot of bogus DHCP client requests to use up all the IP addresses in a DHCP server's pool. More simply, this could be used by a thief to steal an IP address from an organization for his own use.

These are obviously serious concerns. The normal recommended solutions to these risks generally involve providing security at lower layers. For example, one of the most important techniques for preventing unauthorized servers and clients is careful control over physical access to the network: layer 1 security. Security techniques implemented at layer 2 may also be of use—for example, in the case of wireless LANs. Since DHCP runs over UDP and IP, one could use IPSec at layer 3 to provide authentication.

DHCP Authentication

To try to address some of the more specific security concerns within DHCP itself, in June 2001, the IETF published RFC 3118, "Authentication for DHCP Messages." This standard describes an enhancement that replaces the normal DHCP messages with authenticated ones. Clients and servers check the authentication information and reject messages that come from invalid sources. The technology involves the use of a new DHCP option type, the Authentication option, and operating changes to several of the leasing processes to use this option.

Unfortunately, 2001 was pretty late in the DHCP game, and there are millions of DHCP clients and servers around that don't support this new standard. Both the client and server must be programmed to use authentication for this method to

have value. A DHCP server that supports authentication could use it for clients that support the feature and skip it for those that do not. However, the fact that this option is not universal means that it is not widely deployed, and most networks must rely on more conventional security measures.

DHCP for IP Version 6 (DHCPv6)

DHCP is currently the standard host configuration protocol for the TCP/IP protocol suite. TCP/IP is built on version 4 of IP (IPv4). However, development work has been under way since the early 1990s on a successor to IPv4: version 6 of the Internet Protocol (IPv6; see Part II-4 for more information). This new IP standard will be the future of TCP/IP.

While most of the changes that IPv6 brings impact technologies at the lower layers of the TCP/IP architectural model, the significance of the modifications means that many other TCP/IP protocols are also affected. This is particularly true of protocols that work with addresses or configuration information, including DHCP. For this reason, a new version of DHCP is required for IPv6. Development has been under way for quite some time on *DHCP for IPv6*, also sometimes called *DHCPv6*. At the time of writing, DHCPv6 has not yet been formally published—it is still an Internet draft under discussion.

NOTE *In discussions purely oriented around IPv6, DHCPv6 is sometimes just called* DHCP*, and the original DHCP is called* DHCPv4*.*

Two Methods for Autoconfiguration in IPv6

One of the many enhancements introduced in IPv6 is an overall strategy for easier administration of IP devices, including host configuration. There are two basic methods defined for autoconfiguration of IPv6 hosts:

Stateless Autoconfiguration A method defined to allow a host to configure itself without help from any other device.

Stateful Autoconfiguration A technique where configuration information is provided to a host by a server.

Which of these methods is used depends on the characteristics of the network. Stateless autoconfiguration is described in RFC 2462 and discussed in Chapter 24. Stateful autoconfiguration for IPv6 is provided by DHCPv6. As with regular DHCP, DHCPv6 may be used to obtain an IP address and other configuration parameters, or just to get configuration parameters when the client already has an IP address.

DHCPv6 Operation Overview

The operation of DHCPv6 is similar to that of DHCPv4, but the protocol itself has been completely rewritten. It is not based on the older DHCP or on BOOTP, except in conceptual terms. It still uses UDP, but it uses new port numbers, a new

message format, and restructured options. All of this means that the new protocol is not strictly compatible with DHCPv4 or BOOTP, though I believe work is under way on a method to allow DHCPv6 servers to work with IPv4 devices.

KEY CONCEPT Since DHCP works with IP addresses and other configuration parameters, the change from IPv4 to IPv6 requires a new version of DHCP commonly called *DHCPv6*. This new DHCP represents a significant change from the original DHCP and is still under development. DHCPv6 is used for IPv6 *stateful autoconfiguration*. The alternative is *stateless autoconfiguration*, a feature of IPv6 that allows a client to determine its IP address without need for a server.

DHCPv6 is also oriented around IPv6 methods of addressing, especially the use of link-local scoped multicast addresses (see Chapter 25). This allows efficient communication even before a client has been assigned an IP address. Once a client has an address and knows the identity of a server, it may communicate with the server directly using unicast addressing.

DHCPv6 Message Exchanges

There are two basic client/server message exchanges that are used in DHCPv6: the *four-message exchange* and the *two-message exchange*. The former is used when a client needs to obtain an IPv6 address and other parameters. This process is similar to the regular DHCP address allocation process. Highly simplified, it involves these steps:

1. The client sends a multicast Solicit message to find a DHCPv6 server and ask for a lease.
2. Any server that can fulfill the client's request responds to it with an Advertise message.
3. The client chooses one of the servers and sends a Request message to it, asking to confirm the offered address and other parameters.
4. The server responds with a Reply message to finalize the process.

There is also a shorter variation of the four-message process above, where a client sends a Solicit message and indicates that a server should respond back immediately with a Reply message.

If the client already has an IP address, either assigned manually or obtained in some other way, a simpler process can be undertaken, similar to how in regular DHCP the DHCPINFORM message is used:

1. The client multicasts an Information-Request message.
2. A server with configuration information for the client sends back a Reply message.
3. As in regular DHCP, a DHCPv6 client renews its lease after a period of time by sending a Renew message. DHCPv6 also supports relay agent functionality, as in DHCPv4.