CSc72010

DHCP

Outline

Basics

Comer: Chapter 22 (Chapter 23 in the the 4th edition)
Peterson: Section 4.1.6
RFC 2131

What is DHCP?

Dynamic Host Configuration Protocol: provides for configuring hosts that have been newly connected to the network with:

- IP address/subnet mask
- Default gateway address
- Domain/name server address
- tftp server address and filename for boot file
- Time server
- Time offset
- Print server
- ...

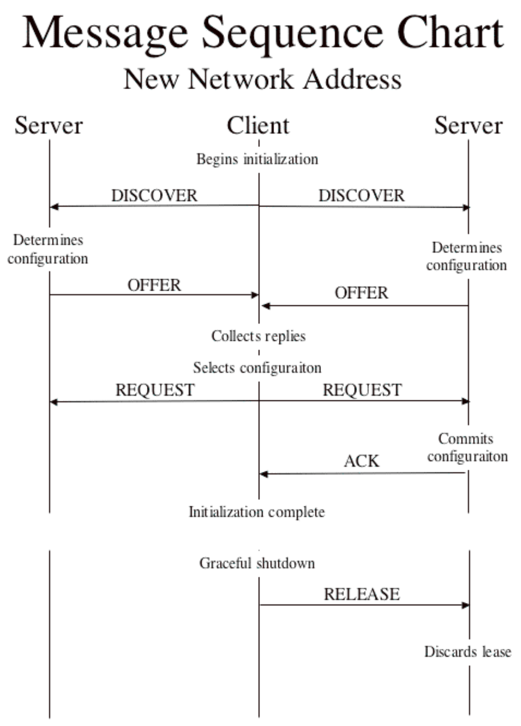
Why DHCP?

DHCP simplifies network administration and thus makes networks more cost-effective and more likely to be correct. All of the parameters for a subnet can be configured centrally.

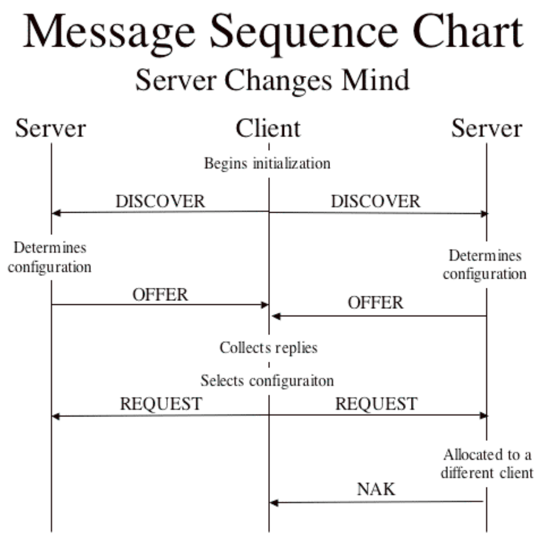
How does DHCP work?

A client looks for a DHCP server; the DHCP server “offers” configuration parameters; if there are multiple servers in range of the client, it may receive more than one offer. The client picks an offer and “requests” it from the server, then the server either acknowledges it or negative-acknowledges it. The first is a confirmation that the request has been accepted, the second is a rejection.

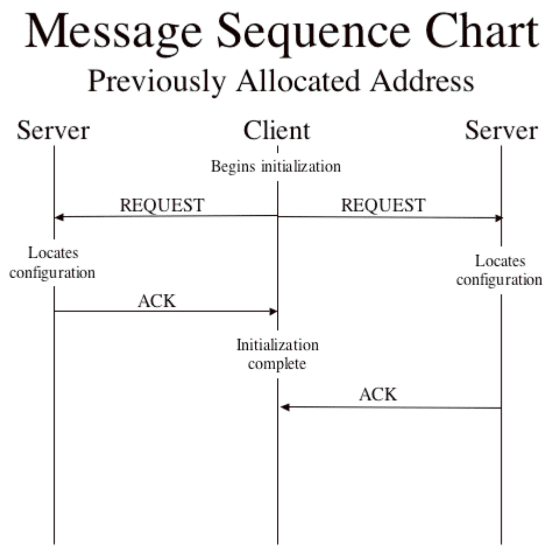
The following chart shows the normal behavior when a host is added to a network and initializes a network interface on the network. For this to happen, the Server offers an IP address that has not been allocated to any other host and it doesn’t commit it to another host during the time-span of the chart.



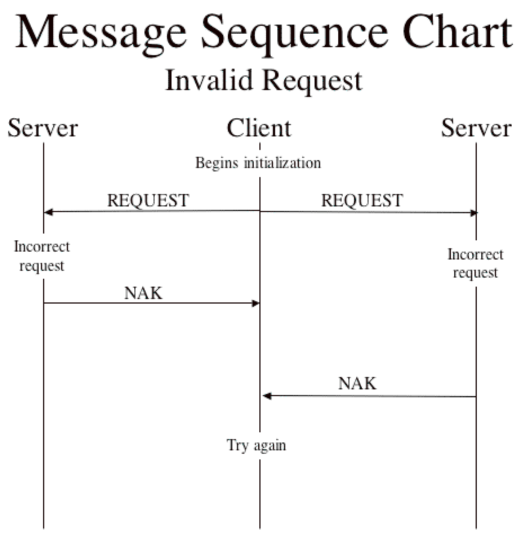
The next chart shows what happens if another client is allocated the offered IP address before it has been committed to the client shown in the figure.



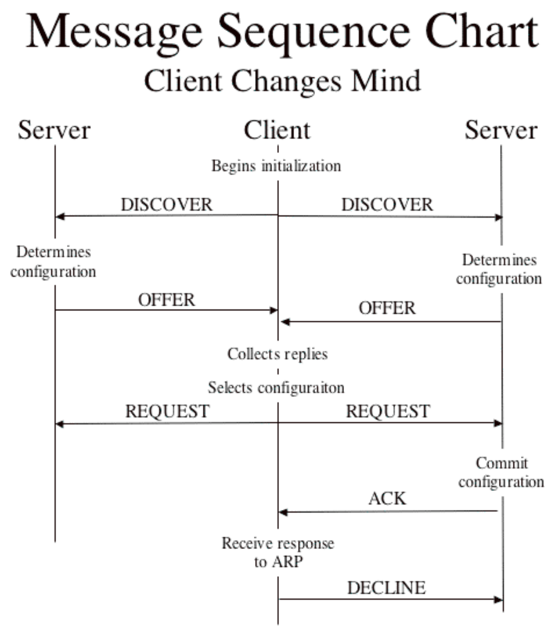
The third chart shows the normal behavior when a host re-joins a network on which it has previously been allocated an IP addres.



The fourth chart shows what happens when a host joins a network asking for an invalid network address, for whatever reason – for example, it left a long time ago and the IP address has been reallocated to a different client or it is asking for an IP address on a different subnet (because that was its last IP address).



The last slide shows what is supposed to happen when a DHCP server commits an IP address to a client incorrectly.



Important DHCP Properties

Looking at the RFC

Toward the beginning of each RFC, there is a section concerning general goals or design goals for the protocol. Here is the list for DHCP.

“The following list gives general design goals for DHCP.

1. DHCP should be a mechanism rather than a policy. DHCP must allow local system administrators control over configuration parameters where desired; e.g., local system administrators should be able to enforce local policies concerning allocation and access to local resources where desired.
2. Clients should require no manual configuration. Each client should be able to discover appropriate local configuration parameters without user intervention and incorporate those parameters into its own configuration.
3. Networks should require no manual configuration for individual clients. Under normal circumstances, the network manager should not have to enter any per-client configuration parameters.

4. DHCP should not require a server on each subnet. To allow for scale and economy, DHCP must work across routers or through the intervention of BOOTP relay agents.
5. A DHCP client must be prepared to receive multiple responses to a request for configuration parameters. Some installations may include multiple, overlapping DHCP servers to enhance reliability and increase performance.
6. DHCP must coexist with statically configured, non-participating hosts and with existing network protocol implementations.
7. DHCP must interoperate with the BOOTP relay agent behavior as described by RFC 951 and by RFC 1542 [21].
8. DHCP must provide service to existing BOOTP clients.

The following list gives design goals specific to the transmission of the network layer parameters. DHCP must:

9. Guarantee that any specific network address will not be in use by more than one DHCP client at a time,
10. Retain DHCP client configuration across DHCP client reboot. A DHCP client should, whenever possible, be assigned the same configuration parameters (e.g., network address) in response to each request,
11. Retain DHCP client configuration across server reboots, and, whenever possible, a DHCP client should be assigned the same configuration parameters despite restarts of the DHCP mechanism,
12. Allow automated assignment of configuration parameters to new clients to avoid hand configuration for new clients,

Support fixed or permanent allocation of configuration parameters to specific clients.

Let's consider how to turn some of them into trace properties.

Safety

No IP address is allocated to two different clients at the same time. [9]

> Between any pair of ACKs to two different clients, there is either a RELEASE or the lease expires.

Retain the same IP address across client reboot. [10]

> The above property holds even if there is a client reboot between the ACKs.

Retain the same IP address across server reboot. [11]

> The above property holds even if there is a server reboot between the ACKs.

Liveness

Manual configuration is not necessary; configuration is assigned to a client if possible. [12]

> Following a DISCOVER there is an OFFER if any IP address is available.
> Following a REQUEST there is an ACK if the IP address is available.

Tricks and Techniques

DHCP runs in the application layer, over UDP, using ports 67 and 68. How can a host communicate over IP before it has an IP address?

Suppose a host is assigned an IP address manually and the IP address conflicts with a dynamically-allocated IP address. Can this be detected and dealt with?

The goal is to centralize configuration, but does the network administrator have to configure a DHCP server on every subnet?

Using IP without an IP address

A client sends packets from IP address 0.0.0.0 before it is allocated an IP address. Also, since it can't know the IP address of the DHCP server, it must broadcast its DISCOVER messages (address 255.255.255.255). In fact, the client broadcasts all of its messages until it has been allocated an IP address. At that point, it knows the address of the server that allocated the IP address, and will communicate directly to it for renewals.

The server will know the client's hardware address once it receives the DISCOVER and so could send the responses to that hardware address, but, not all TCP/IP stacks will deliver IP packets, even if sent to the right hardware address, before IP has been configured.

So, if the packet can't be sent to the hardware unicast address, the server will use the local hardware broadcast address (all 1's for Ethernet) to send its responses back (o the client. The client sets a flag to tell the server it has to do this.

Avoiding IP address conflicts

There is a provision for avoiding conflicts with a manually configured IP address even if it comes from the DHCP pool.
When a client receives an ACK from a server, as a last minute check just before installing the IP address, the client should ARP for the address. It shouldn't get a response, but if it does, that means that some other host already has the address. So, the client should respond "DECLINE" to the server's ACK.
Unfortunately, DHCP clients that truly run as applications, without any special permissions, usually can't get access to the responses to ARPs in a timely manner! As a result, the ISC DHCP server actually pings each address before semdomg am ACL fpr it.

Using a single DHCP server/backup pair

If a network administrator had to run around to every subnet to configure a DHCP server on the subnet, it would hardly seem worth the bother. Instead of this, we usually have a single DHCP server pair in an organization, and the routers function as relay agents, forwarding all DHCP messages to the appropriate server, This is set up wehn the router is configured.
When the relay agent forwards the message, it inserts its own address in the message so the server knows what subnet to use for choosing an IP address and so the server can

send tis responses back.

Defining a DHCP ioa

See diagram.

Proving DHCP properties

The main properties that we can prove are no double allocation and allocation if possible.

No double allocation

The general idea is to model or implement a server that remembers IP addresses and lease durations in a database (/var/db/dhcp-leases or something like that). Then a server that is working correctly will never reallocate than IP address unitl its latest lease has expired. Suppose the server is working correctly. What can go wrong?

1) The server clock runs much faster than the client clocks.
2) The settings of the server and client clocks are way off: Note that lease times are expressed as durations in the messages, so the actual value of the clocks won't matter; but if there is a lot of skew, there is still a problem.
3) The server or client clock is reset during a lease.

   Does this relate in any way to TCP? What about determining RTT?
   An important fact of life on distributed systems: it's dangerous to assume synchronized clocks.

4) There are two DHCP servers allocating addresses from the same DHCP address pool.
5) Someone has manually assigned an IP address that is already in use.

We might be able to state a theorem: If all clocks run at exactly the same rate and if the clocks are never reset and if each IP address can be assigned by only one server and if there is no other way to assign an IP address, then there will be no double allocation.

Allocate if possible

Now, let's suppose a server always offers an IP address if there is a free one in the database when a DISCOVER arrives. Let's also make the same assumptions as above. Is starvation possible?

Here's a scenario: split the IP addresses between two servers, S1 and S2. Three clients start up: A, B, and C. All send DISCOVER. Both servers have one address left; they both offer their one address to all clients. A and B send a request to S1 for the S1 address; C asks S2 for the S2 address.

The "exactly once" requirement

This requirement is hard to implement at any time, but especially in a distributed system. Another case of it involves updates to databases, where we want to ensure that each transaction runs exactly once.

Two-phase commit.

The problem is that when there is a “partial” failure, the transaciton may have partial results or it may not be knowable whether it had any result at all. (A “partial failure” involving only one computer may seem impossible, but consider that there may be many different writes to the database required and only some of these may be completed before the computer fails.)

The solution: Before fnalizing the results of a transaction in a database, the transaction “pre-commits,” i.e., it makes sure that all results are safely stored on stable storage and can be identified as the results of this transaction. Once this is done, the transaction commits by atomically writing its ID in a commit log. Note that we have reduced multiple writes to a single write in this way. Now it is possible to process the commit log to determine which transactions have committed and to make sure that their results are in the database (by re-writing them if necessary.

Distributed commit.

Now suppose that multiple computers need to agree on a single bit (0 or 1). This is an abstraction of the database problem – multiple computers can perform the two-phase commit, but they first need to ensure that all perform it or none do. Thus they try to agree on 0 or 1.
Dale Skeen designed a three-phase commit, which may actually go on longer depending on how many processes fail but requires at least three phases. You can read about this in the text.

Other protocols

In any case, note that the startup and finalize mechanisms in TCP are attempting agreement (however, instead of guaranteeing agreement, a TCP participant can just drop out if it gets confused by sending a “RESET” message). What is guaranteed is that if the two participants exchange both SYNs and ACKs,, they will have agreed on the starting sequence numbers and window sizes, and if not, there is no session. TCP errs on the side of failure here – there is no continued attempt to establish “exactly one” session. Instead, at most one session is established.
In DHCP, the DISCOVER/OFFER/REQUEST/ACK sequence performs somewhat the same function – the client gets one IP address and the server knows which one it has,, ie, they agree on an IP address. If they don’t make it through the sequence or don’t manage to agree, then they have to start over – so it’s at most once, again.
Question – the answer may be in the Lynch book – are three messages required for agreeing on a 1 (no need to agree if the decision is 0, since the default will be 0). Or could it be four?

DHCP Failover

Suppose we need to guarantee that DHCP addresses are always allocated, even if a server goes down. How?

1) Use two servers – but if their IP address pools overlap, we know that there’s a possibility of double allocation.

2) Use disjoint address pools – but if one DHCP server goes down, half the addresses will be unavailable. Remember, we're running out of IP addresses – an organization can't hope to get twice as many as it needs.
3) Have the two servers communicate about allocated IP addresses and when one goes down, the second can renew any currently allocated addresses that belong to the other's pool.
4) What about "synchronizing events" – for example, 9 am Monday morning or when recovering from a network outage? If one of the DHCP servers is down, the number of expired leases may be larger than the remaining IP addresses on a single server.

Let's make the simplest possible assumption, that Server A knows that by time T, Server B was down. Then it will be safe to allocate IP addresses from Server B's pool if the IP address was leased at time T but is now expired. There are two cases:

1) Server B allocated the IP address before time T and told Server A about it. In this case, Server A needs to wait until it knows the lease has expired.
2) Server B allocated the IP address before time T and didn't tell Server A about it. In this case, Server A needs to wait until any possible lease would expire – ie, T+MAXLEASE, where MAXLEASE is the maximum lease that Server B can allocate.

Case 1 is fine and works as if Server B were still around, but Case 2 is not – any IP address whose lease status is unknown to A has to be untouched until T+MAXLEASE, even though most of these probably haven't been allocated.

The idea to fix this is simple: There are, in effect, two values for MAXLEASE: one very short, representing the maximum lease that the other server doesn't know about. This is called Maximum Client Lead Time, MCLT. The second is the normal, long MAXLEASE.

When an IP address is first allocated to a client, the client gets the short lease (MCLT). But the server sends a message to the other server, saying "I have allocated a lease of length .5*MCLT+MAXLEASE to client X." Once the second server has acknowledged this – hopefully, before the client comes back for renewal – the entire lease can be allocated. Note that for the short lease, no agreement is required, but for the long lease, it is.