

Chatter on the Wire:

A look at DHCP traffic

by Eric Kollmann

aka xnih

v.1.0

September 2007

Disclaimer

All information provided in here is my take on the RFC's and on data I came across in my testing. Some of it could be incorrect!

This paper will be updated from time to time if new data is available that needs added. This is by no means a completely finished project or a guaranteed 100% accurate document.

Acknowledgments

Thanks go out to

- The crew at OpenOffice.org that made a great product which I wrote most of this in.
- WeOnlyDo.com for the code and sample project for a DHCP Server. I tried some of the other free ones out there and couldn't get any to work the way I wanted.
- winpcap.org which without Satori would have never happened.
- wireshark.org who without all of the packet capture screenshots in here wouldn't have been possible.
- A couple of security researchers who have helped out on the project either directly with code into Satori or with information or posts about the program, because without interest in Satori most of this paper wouldn't have been written because it would have fallen off my radar.
- packetfence.org project and one of the people over there who pushed the idea of doing more on this whole DHCP idea.
- NetworkSorcery.net, whose breakdown of different packets has come in very helpful throughout both papers and multiple programming projects over the years!

Suggested reading material

- For those that want to know more about DHCP in general a good book to read is 'The DHCP Handbook' 2nd Edition by Ralph Droms and Ted Lemon. Droms has been a name on the RFC's for DHCP for years!
- Though this will be mentioned later check out 'ICMP Usage in Scanning' by Ofir Arkin, you can find it at: http://www.sys-security.com/archive/papers/ICMP_Scanning_v3.0.pdf Many of the initial ideas on how else to identify systems came from ideas I first read about in some of Ofir's earlier papers.
- From a passive identification process I'd also recommend taking a look at 'Silence on the wire: A field guide to passive reconnaissance and indirect attacks'. This is an all encompassing book on passive identification by Michal Zalewski and I know my papers have the same general name, Michal gave me a bad time about it once.

Contents

Disclaimer.....	i
Acknowledgments.....	ii
Thanks go out to.....	ii
Suggested reading material.....	ii
Contents.....	iii
Brief recap.....	1
RFC History for DHCP.....	3
How DHCP works.....	5
DHCP DISCOVER Packet.....	7
DHCP OFFER Packet.....	7
DHCP REQUEST.....	9
DHCPACK & DHCPNAK.....	9
Renewing a DHCP Lease.....	9
DHCP RELEASE – Giving the IP back.....	10
DHCP INFORM packets – Getting more information.....	10
Format of a DHCP Packet.....	12
The Hard Way to Fingerprint DHCP.....	15
Windows 95.....	18
Windows 98.....	23
Windows 98 SE.....	24
Window ME.....	25
Windows NT.....	26
Windows 2000.....	27
Windows Vista.....	28
Arudis.....	29
Gentoo 2005.0.....	29
Gentoo 2006.1.....	30
CentOS 4.....	30
Fedora Core 3.....	31
Fedora Core 4.....	31
Fedora Core 5.....	31
Knoppix 5.1.....	32
Other Linux Builds.....	32
IP TTL on DHCP Packets.....	33
DHCP Options – the easy way.....	34
Using all Options.....	34
Option 55.....	35
Windows 95.....	35
Windows 98.....	36
Microsoft as a whole.....	37
Fedora Core and Cent OS.....	38
Backtrack, Gentoo and Slax using Option 60.....	38
Other Linux Distributions using Option 51 and 57.....	39

Beyond Option 55 – how we can track a few other things..... 41

 Option 61..... 41

 Option 77..... 42

PXE Boot and what we can Learn..... 43

 Option 93 – Client System Architecture..... 43

 Option 94 – Client Network Device Interface..... 44

Utilizing Lease Information..... 45

 What happens when a lease expires..... 45

Appendix A..... 47

 DHCP Options..... 47

 DHCP Message Type 53 Values - per [RFC2132]..... 50

Brief recap

Back in August 2005 I did a paper titled: 'Chatter on the Wire: A look at excessive network traffic and what it can mean to network security'. I don't know how widespread it was or how many people read it, but it kept me busy during my time in Iraq. It was about active and passive identification and more importantly about all the different ways that passive OS identification can be used.

I briefly touched on DHCP fingerprinting in that paper. Originally I thought I had a brand new idea, but a search on google popped up this article: <http://www.nts.ku.edu/about/projects/dhcp/NGDHCP.pdf> from sysadminmag.com from February 2005, a few months earlier than my first discovery of this great idea. Dave Hull one of the authors of the article later put me in contact with the people at the packetfence.org project and now over 2 years after that article was written, DHCP fingerprinting still has not really taken off at all! I hope to shed some more light on it and maybe someone with more programming skills than I will expand and run with it.

There are only a few projects that I know of that utilize this technique for OS identification, below are a few that I know are currently still receiving updates and are actively supported:

- Satori, which uses it and other packets it sees on the wire for OS identification, written by yours truly, located at: <http://myweb.cableone.net/xnih>
- PacketFence which uses the techniques/identification to determine if it should grant you a DHCP address, which is maintained at <http://packetfence.org>
- RogueScanner which is a free program for active fingerprinting, which also has a DHCP listener portion. One issue with it is it is a closed database and it actually ships off its fingerprints back to the company's site. You can find more out on it at <http://www.networkchemistry.com/products/roguescanner.php>

There are a few other projects that were started in regards to the original project:

- dhcprint, last version I see was 0.2 updated in October 2005, located at: <http://erwin.wpi.edu/~fs/dhcprint/>
- DHCPListener, last version I see was 1.4, which was the proof of concept one discussed in the original article. It can be located at: <http://www.nts.ku.edu/downloads/>

Other products/projects that appear to be using DHCP Fingerprinting

- DAIR, from mobisys, it appears to be geared towards wireless security and detecting rogue APs

I believe University of Kansas also utilizes this information in their RINGS project based on the original writeup in sysadminmag.com.

So to wrap this all up, no DHCP Fingerprinting isn't something specific I came up with, but 2 years later little has been done with it, that I've seen, to improve/expand it beyond the original idea presented by Dave Hull and George F. Willard III of using the Option 55 and Option 60 data.

Note: In September 2007 I ran across an article dated February 2003 entitled 'New scheme for passive OS fingerprinting using DHCP message' from the Journal 'Joho Shori Gakkai Kenkyu Hokoku'. Actually all I ran across was the abstract that discusses using DHCP messages. I have not been able to find a copy of the actual article. So, this approach to OS fingerprinting is even older than initially thought!

I've been working on this approach to OS fingerprinting since March 2005 or so, so not sure how I've never come across this before?

RFC History for DHCP

Before we go any farther we really need a little RFC history lesson. I know, boring to some of you, at the mere mention of 'RFC' your eye's have rolled back into your head, you've started drooling and gasping for breath, but it needs to be done since as things have changed with the RFC's so have what the vendors do with their DHCP stack.

RFC 1541 – Written in October 1993, so for the purposes of this paper it will be the first one we deal a lot with since the earliest OS I'm writing on is Windows 95. RFC 1541 can be found at:

<http://www.faqs.org/rfcs/rfc1541.html>

RFC 2131 – Replaced RFC 1541 and was written in March 1997. It is the main one we'll be using. Some things may have changed in later RFC's. It can be found at: <http://www.faqs.org/rfcs/rfc2131.html>

According to RFC 2131 here is what has changed from the previous RFC:

1.1 Changes to [RFC 1541](#)

This document updates the DHCP protocol specification that appears in [RFC1541](#). A new DHCP message type, DHCPINFORM, has been added; see section 3.4, 4.3 and 4.4 for details. The classing mechanism for identifying DHCP Clients to DHCP Servers has been extended to include "vendor" classes as defined in sections 4.2 and 4.3. The minimum lease time restriction has been removed. Finally, many editorial changes have been made to clarify the text as a result of experience gained in DHCP interoperability tests.

RFC 2132 – Released in March 1997. It gives us information on DHCP Options and BOOTP Vendor Extensions which is one of the main ways we are using to ID systems.

RFC 4361 – Released in February 2006. It provides a few updates to RFC 2131 and 2132, in respect to the Client Identification, Option 61, which we use a little in identification also. This one may provide us some useful information, but could be a waste of time also, only time will tell.

RFC 4388 – This RFC provides some new DHCP Message Types, primarily DHCPLeaseQuery and the different available responses. It was released in February 2006. It can be found here:

<http://www.faqs.org/rfcs/rfc4388.html>

Note: It would be fun to get a DHCP Server that supports DHCP Lease Query and send that to all these OS's and see how they respond. This would be a good test and paper at a later time.

RFC 4578 – Provides some info on PXEboot. It was written recently in November 2006. It can be found here: <http://www.faqs.org/rfcs/rfc4578.html> Interesting things to look at here will be section 2.1 which will help provide information about the underlying hardware platform.

Ok, this is by no means an exhaustive list of the RFC's that talk about DHCP, but it is a start. There are a ton that deal with IPv6, but since I don't have an IPv6 network to play with, we won't be touching on that in this paper.

Notice a lot of these RFC's have been released in the past year (as of the initial writing of this paper). Unless vendors were planning ahead or not waiting for the RFC, odds are most of the OS's we'll be looking at won't support some of these new things (such as DHCPLeaseQuery). But we'll see.

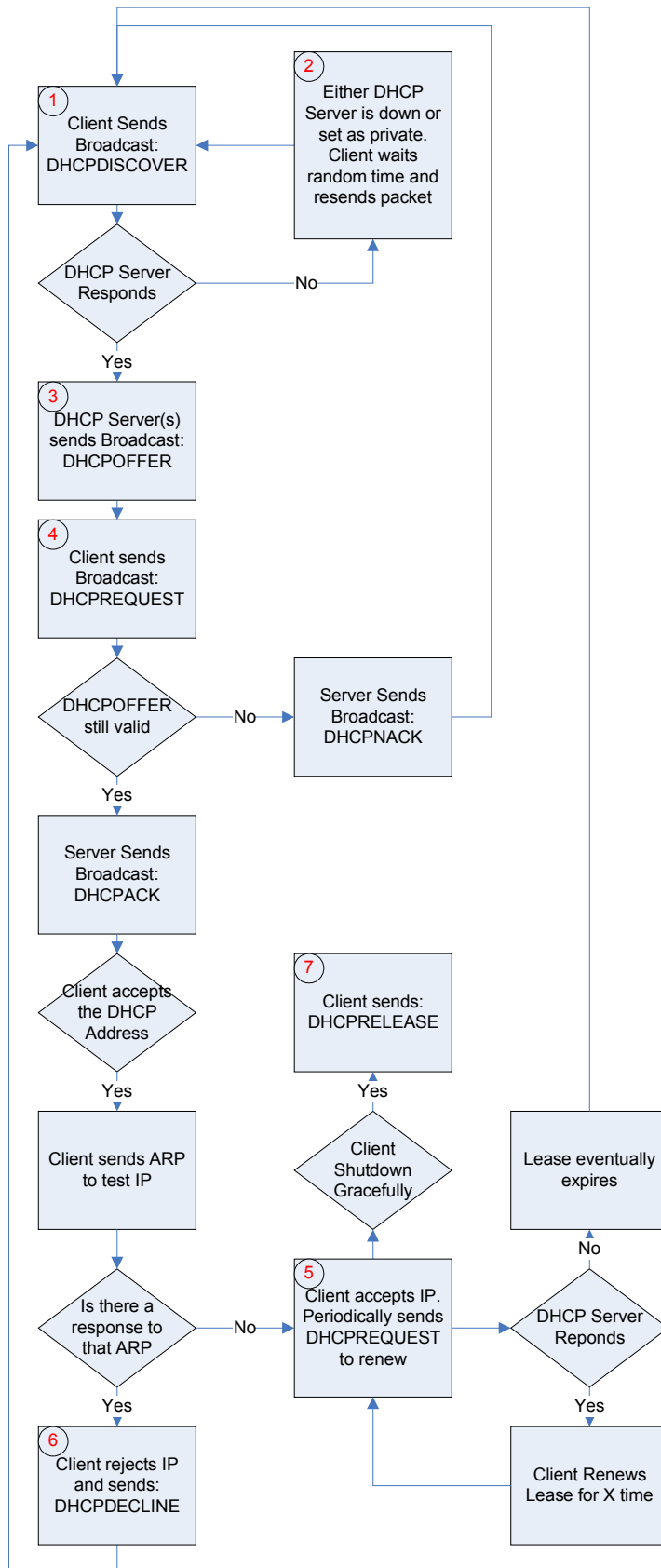
How DHCP works

Ok, now that we have the RFC's out of the way and an idea of what has come before, in this field of study, we can start looking a bit more at it. DHCP has been with us for a long time now. It was an extension or maybe better put a replacement to BOOTP, but so as to utilize the existing infrastructure already built up in the BOOTP world it utilized the same format/structure. Some of this we will see may come back to bite us a bit, or more precisely we are left with information 20+ years later in a DHCP packet that does not appear to be used anymore. (This can always be a problem when we continue to worry about backward compatibility).

Before we get much further we need to understand how DHCP actually works. As you've seen there are numerous RFC's out there, and I know the mere mention of that makes some of your eyes glaze over. For those of you who don't want to look at RFC's at all, sorry, you're going to have to do a lot of scanning of this document, at least in some areas, since I quote some of them, quite heavily. If it wasn't for these RFC's and each vendors interpretation of them, or very lack adherence to them it would make the first part of this paper very hard to do. Granted the first part is about how to do DHCP identification the hard way, so maybe it really doesn't matter there!

Part of the reason DHCP Fingerprinting works so well is because DHCP is a broadcast packet. Even with all of those nice expensive, high end switches, replacing all those cheap hubs, when that DHCP Client decides to come up and request an IP address, they broadcast out for everyone to hear (at least on their local segment) that "Here I am; Here's my MAC; Here's what IP I had before; Here's what I want". From a passive OS identification perspective how can I complain about this lovely "feature"?

Before we go any further I guess we should look at how DHCP actually works, or at least my interpretation and the drawing that I like. I've noted in this flowchart where we can potentially fingerprint the machines.



At point 1, we can fingerprint the Client with the Options and information in each of those Options since most OSs have a unique set of Options, in a unique order, that they ask for in a DHCP Discover packet. This will at least give us a base OS or general idea of the system we may be dealing with.

At point 2, we can also fingerprint the Client. We can use the information from Point 1, but better yet, we can use the time between DHCPDISCOVER packets. This is going to be the first main way we look at identifying the remote client.

At point 3, we can fingerprint the Server, this is only useful with SOHO type devices that do not allow you to change things. If you can change the Options that your Server advertises, any fingerprinting done here is near useless.

At point 4, we will use the same idea as point 1, except DHCP Requests now.

At point 5, we can use the same idea as point 2, assuming that the DHCP Server does not Respond. This DHCPREQUEST packet may also be sent as the machine boots up if the lease is still valid.

At point 6, we may find it to be useful, don't know yet. The work required to track this is beyond what I'm willing to put in at this time!

At point 7, we see something very few OSs behave nicely and do. That is sending a DHCP RELEASE packet. This is an advantage since you never know when you might spot a system. Also, add to that fact that most OS's don't do this, it brings down the list of systems we'll identify this way to just a few!

Figure 1.1 – Flowchart of How DHCP packets flow

DHCP DISCOVER Packet

Let's first take a look at the initial bootup process of the machine. This will give us a general idea of what these packets look like that the DHCP Clients and Servers are sending around to each other, look at how to read them, what the different sections are, and why we care about them.

Initially, upon bootup most clients will send a DHCPDISCOVER packet. Lets look at a typical one:

```

Bootstrap Protocol
  Message type: Boot Request (1)
  Hardware type: Ethernet
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0xc2ead4f3
  Seconds elapsed: 0
  Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0 (0.0.0.0)
  Your (client) IP address: 0.0.0.0 (0.0.0.0)
  Next server IP address: 0.0.0.0 (0.0.0.0)
  Relay agent IP address: 0.0.0.0 (0.0.0.0)
  Client MAC address: 3com_9b:64:4d (00:60:97:9b:64:4d)
  Server host name not given
  Boot file name not given
  Magic cookie: (OK)
  Option: (t=53,l=1) DHCP Message Type = DHCP Discover
  Option: (t=61,l=7) Client identifier
  Option: (t=50,l=4) Requested IP Address = .152.217
  Option: (t=12,l=9) Host Name = " 33591"
  Option: (t=55,l=8) Parameter Request List
  End option
  Padding

```

Figure 1.2 - Typical DHCP Discover Packet

The Client machine initially sends out a broadcast packet to 255.255.255.255, with a source of 0.0.0.0. Again, with this being a broadcast packet we are able to see it anywhere on the local segment. On a /24 segment this may not be as big a deal as say a /16, but even on a small network every little piece of information may come in useful for you, or for an attacker.

DHCP OFFER Packet

Once a DHCP Server receives a DHCPDISCOVER it will send a DHCP OFFER packet. Any DHCP Server that hears the packet will send this OFFER back to the client. (This is under the assumption that there are leases still available and that the server is not setup as a private DHCP server, only handing out addresses to pre-approved or pre-registered MACs.)

```

Bootstrap Protocol
Message type: Boot Reply (2)
Hardware type: Ethernet
Hardware address length: 6
Hops: 0
Transaction ID: 0x004a4241
Seconds elapsed: 0
+ Bootp flags: 0x8000 (Broadcast)
Client IP address: 0.0.0.0 (0.0.0.0)
Your (client) IP address:      .62.103 (      .62.103)
Next server IP address: 0.0.0.0 (0.0.0.0)
Relay agent IP address:      .58.2 (      .58.2)
Client MAC address: Cisco_a4:67:40 (00:01:96:a4:67:40)
Server host name not given
Boot file name not given
Magic cookie: (OK)
+ Option: (t=53,l=1) DHCP Message Type = DHCP Offer
+ Option: (t=54,l=4) Server Identifier =      .2.52
+ Option: (t=51,l=4) IP Address Lease Time = 7 days
+ Option: (t=1,l=4) Subnet Mask = 255.255.252.0
+ Option: (t=6,l=12) Domain Name Server
+ Option: (t=3,l=4) Router =      .60.1
+ Option: (t=150,l=8) Private
+ Option: (t=12,l=13) Host Name = "      14762"
End option

```

Figure 1.3 - Typical DHCP Offer sent from a DHCP Server

Each DHCP Server is going to do its own thing in regards to what Options it tells the Client about, but the main things are going to be:

IP Address, Subnet Mask, and Lease Time.

More Options may be sent to the Client, but none of those are actually needed for the Client to work inside of its current subnet. Granted if you want it to participate on the Internet you may want to provide a default gateway/router and a DNS Server or two.

The Client may receive more than one DHCP Offer. It just depends on your network. If this were to happen the Client must make a decision on which DHCP Offer to accept. It may do this by which one answered first, or it may make that decision based on the Options it received from the DHCP Server. This is going to be very client specific.

This would be an interesting place of study in the future, but typically, at least in my experience there are not multiple DHCP Servers per segment, except in a failover scenario. Who wants DHCP Servers fighting over clients, possibly giving out bad info to it or giving out conflicting information! On the other hand, there are always those people that plug in a little SOHO router when they shouldn't and end up handing out private/non-routable addresses and cause problems! Determining how different Clients figure out which DHCP Server to use would be interesting.

DHCP REQUEST

At this point, the Client will send a DHCPREQUEST, requesting an IP address from the DHCP Server. In some cases the machine will request different Options than it requested in the DHCPDISCOVER packet, but not in all cases. Again, this is going to be Client specific.

DHCPACK & DHCPNAK

At this point the DHCP Server acknowledges that that IP is ok by sending a DHCPACK to the client if the offer is still valid. If the offer has expired, or another DHCP Server has handed out that IP Address, the DHCP Server will send out a DHCPNAK.

If a DHCPNAK, or negative acknowledgment is sent then the Client will send a DHCPDECLINE packet and then it will start over sending DHCPDISCOVER packets, at least by RFC. (Is this always the case or will it just go back and send out DHCPREQUESTS, yet one more place to spend some time researching).

If a DHCPACK, or acknowledgment is sent it is then up to the client to verify that the IP is good for it to use. Some clients will send out a Gratuitous ARP packet, but not all, asking who owns that IP address. If it gets an answer it means that someone else is using it, and it then starts over at the DHCPDISCOVER step. If no ARP Reply is seen, it assumes the IP is free and assigns it to its IP stack.

Again, when time permits we should look into what OS's actually do this, sending out the Gratuitous ARP, to verify that the IP is free. How many OS's out there, just believe the DHCP Server, accept the IP, and try to start using it? At least at this time, outside of the scope of this paper like many other things mentioned so far.

Renewing a DHCP Lease

First, we need to look at the RFC here a bit. We need to understand what the Client is supposed to do according to the RFC. For those of you that want to look this up, here is the RFC that this is from:

<http://www.faqs.org/rfcs/rfc2131.html>

Section 4.4.5

```
The client maintains two times, T1 and T2, that specify the times at
which the client tries to extend its lease on its network address.
T1 is the time at which the client enters the RENEWING state and
attempts to contact the server that originally issued the client's
network address. T2 is the time at which the client enters the
REBINDING state and attempts to contact any server. T1 MUST be
earlier than T2, which, in turn, MUST be earlier than the time at
which the client's lease will expire.
```

[cut]...

```
Times T1 and T2 are configurable by the server through options. T1
defaults to (0.5 * duration_of_lease). T2 defaults to (0.875 *
duration_of_lease). Times T1 and T2 SHOULD be chosen with some
random "fuzz" around a fixed value, to avoid synchronization of
client reacquisition.
```

[cut]...

In both RENEWING and REBINDING states, if the client receives no response to its DHCPREQUEST message, the client SHOULD wait one-half of the remaining time until T2 (in RENEWING state) and one-half of the remaining lease time (in REBINDING state), down to a minimum of 60 seconds, before retransmitting the DHCPREQUEST message.

Ok, so in a nutshell, assuming a lease time of 10 mins (600 secs): 5 mins (300 secs) in the client will attempt to renew that IP address, sending a DHCPREQUEST, if for some reason it doesn't get a response it waits half the time between 300 and 525 secs or 112.5 seconds, so at approximately 413 seconds from getting its IP it would do yet another DHCPREQUEST. Eventually it would get to that 525 second mark (10 mins = 600 seconds, 0.875 of $600 = 525$). At this point it would be in REBINDING state.

When it runs into REBINDING State it stops sending out DHCPREQUEST packets in unicast and starts sending them in broadcast mode.

If the lease expires, it is supposed to stop sending data out on its leased IP and start the whole process over like it just booted up, doing a DHCPDISCOVER, DHCPREQUEST, etc. If it gets the original IP back then it is not required to do the Gratuitous ARP, but if it gets a new IP, then it needs to go through all of the steps.

Now the question, from a fingerprinting perspective is:

1. How big is the "fuzz" factor around the .50 and .875 marks in each OS? Are we talking 0-1 second, 1-5 secs, etc?
2. Do machines do the 50% of time remaining between T1 and T2 as they are supposed to?
3. When a lease expires, does the OS stop processing packets to that address, or does it continue using it?

The amount of tracking required to keep track of this is more than I'm willing to put in. Other issues that arise here are how long the lease time is, how long machines stay up and running, etc. To get accurate results here you'd have to have:

1. Short lease times
2. Client devices that stay up and running constantly

So it is something that may give out useful results, but it is not the easiest way to determine an OS.

DHCP RELEASE – Giving the IP back

Section 4.4.6 talks briefly about DHCPRELEASE, saying it is not a required feature for DHCP to work, but that if a client no longer needs the IP address (ie it is being shut down) it should do a release.

As we'll see later, very few OSs implement this feature.

DHCP INFORM packets – Getting more information

DHCPINFORM gives you the ability to request information that may not have been needed upon bootup. During testing this has been seen mostly in the case of Novell Netware client machines requesting information on NDS tree, context information or SLP location. This is typically information that is not sent out in the original DHCP OFFER, but needed by the client.

From a fingerprinting aspect this can give us information about software on the client machine, but may not get us much closer in determining what OS is on the client.

After further testing, with newer OS's, we've seen that Windows Vista also now utilizes DHCPINFORM packets. The difference here is that Vista sends the same type of information in the INFORM packets as it did in its others, so no additional information has really been seen from this OS's perspective with the INFORM packets.

Format of a DHCP Packet

Before we get any deeper into this lets look, again, at a typical DHCP packet. This one happens to be from a Windows 95 box.

```

Bootstrap Protocol
  Message type: Boot Request (1)
  Hardware type: Ethernet
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x16011601
  Seconds elapsed: 0
  Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0 (0.0.0.0)
  Your (client) IP address: 0.0.0.0 (0.0.0.0)
  Next server IP address: 0.0.0.0 (0.0.0.0)
  Relay agent IP address: 0.0.0.0 (0.0.0.0)
  Client MAC address: vmware_b7:97:b0 (00:0c:29:b7:97:b0)
  Server host name not given
  Boot file name not given
  Magic cookie: (OK)
  Option 53: DHCP Message Type = DHCP Discover
  Option 61: Client identifier
  Option 50: Requested IP Address = 192.168.0.3
  Option 12: Host Name = "w95"
  End Option
  Padding

```

A quick run down on each part of the above message:

- Message Type - Also known as 'op' if you look at the RFC. It has a length of 1. Possible values for this field are: 1 – DHCP Request; 2 – DHCP Reply
 - Don't confuse 'Message Type' with Option 53 'DHCP Message Type'. They may look a lot alike, but they will hold slightly different information.
- Hardware Type – Also known as 'htype'. It has a length of 1. Possible values for this field, that we'll see are: 1 – Ethernet; There are others, but we won't be seeing them, but for those interested in a list see: <http://www.networksorcery.com/enp/protocol/dhcp.htm> It looks like there are currently about 33.
- Hardware Address – Also known as 'hlen'. It has a length of 1. For our purposes the value we'll be seeing is 6, the length of a MAC.
- Hops – Length of 1. This value is typically 0, the only time it should be anything different is if the packet is forwarded on by a DHCP Relay agent.
- Transaction ID – Also known as 'xid'. Length of 4. Should be a random number generated by the client which is used for the Client and Server to keep track of the current transaction.
- Seconds elapsed – Also known as 'secs'. Length of 2. Set by the client and is supposed to tell the Server how much time has passed since it started trying to get an IP or has been working on the renewal process. As we'll see, this is one of those places where each vendor does their own thing.
- Bootp flags – Or just simply 'flags' in the RFC. Length of 2. Should be 1 of 2 values, either Broadcast or Unicast. The Server is supposed to adhere to what the client requests here. If the client sends it in Unicast, the Server should respond with Unicast. The only bit, at this point in time, that is to be set is the first one. All the rest are to be set to 0. So far I haven't seen any that set the rest of the bits to

anything but 0, but it may be interesting for future research from identifying the DHCP Server is to set that value and see if the Server sends it back with the same flags or if it resets them all to 0! According to the RFC the Relay Agents and Servers should ignore anything besides 0, but we know how well vendors adhere to RFCs!

- Client IP Address – Or simply 'ciaddr'. Length of 4 for an Ipv4 address of course. Should only hold a value of anything besides 0.0.0.0 if the client currently has a valid IP address and has it bound to its IP stack. If the clients DHCP lease has expired, this should be 0.0.0.0
- Your (client) IP address – Or simply 'yiaddr'. Length of 4. Used in the DHCP Offer and ACK packets returned from the Server. Simply put it is the Server telling you what IP address you should use. An interesting test would be for the DHCP Client to send something besides 0.0.0.0 in its packet to the Server. What will the Server do?
- Next Server IP address – Or 'siaddr' in the RFC. Length of 4. Again a field filled in by the Server in Offer and ACK packets. I've never seen anything but 0.0.0.0 in it, probably depends on the DHCP Server software and setup. Again, what happens if the client sends the Server something in this field?
- Relay agent IP address – 'giaddr'. Length of 4. This will only be something besides 0.0.0.0 if you are utilizing a DHCP Relay Agent in your setup.
- Client MAC address – 'chaddr' in the RFC. Length of 16. Just what it states, it is the MAC of the client sending the DHCP packet. But for those of you paying attention you're saying wait, a MAC isn't 16 in length, it is only 6! Well think back a few bullets here, we had this nice option called 'Hardware Length'. Maybe just maybe these 2 tie together in some way. For our purposes, expect a value of length 6 with a lot of 00 padding.
- Server host name not given – Or from the RFC 'sname'. It has a length of 64 and can hold Server host name information. Typically a lot more of those 00's from my experience.
- Boot file name not given – Or from the RFC 'file' which has a length of 128. As I recall it is for those systems that will be doing a tftp of their boot file. I believe it was mostly for those old diskless workstations, but may also be used by cable modems. Haven't looked at this field in a long time.
- Magic Cookie, Options, End Option and Padding – Or all just 'options' via the RFC. This is a variable length field and each piece breaks down a little bit more. The Options field can be up to 312 in length, which would make the total packet 576. One of the Options that can be set in this section lets the client inform the Server of what size packet it is willing to accept. We typically see this with a Mac OS X machine and some flavors of Linux (and I'm sure Unix as well, but I didn't have any test subjects there).

Note: A little side trip down memory lane here... The Magic Cookie feature was first introduced back in RFC 951 in September 1985, here is what they had to say:

If the 'vend' field is used, it is recommended that a 4 byte 'magic number' be the first item within 'vend'. This lets a server determine what kind of information it is seeing in this field. Numbers can be assigned by the usual 'magic number' process --you pick one and it's magic. A different magic number could be used for bootreply's than bootrequest's to allow the client to take special action with the reply information.

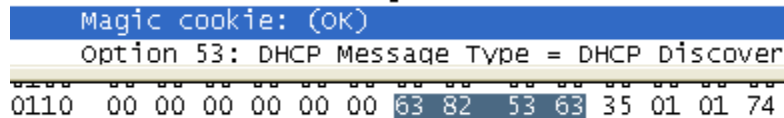
Then in RFC 1497, from August 1993 we get:

Vendor Information "Magic Cookie"

As suggested in [[RFC-951](#)], the first four bytes of this field have been assigned to the magic cookie, which identifies the mode in

which the succeeding data is to be interpreted. The value of the magic cookie is the 4 octet dotted decimal 99.130.83.99 (or hexadecimal number 63.82.53.63) in network byte order.

And guess what, taking a look at Windows Vista Beta 2 we see:



The screenshot shows a network packet capture window. The top bar is blue and contains the text "Magic cookie: (OK)". Below this, a yellow bar contains the text "Option 53: DHCP Message Type = DHCP Discover". The bottom part of the window shows a hex dump of the packet data. The first line of the hex dump is "0110 00 00 00 00 00 00 63 82 53 63 35 01 01 74". The bytes "63 82 53 63" are highlighted in a blue box, corresponding to the magic cookie value 99.130.83.99 in hexadecimal.

So something originally planned for BOOTP back in 1985 to be used to allow the client to “take special action with the reply information” and which never appears to have been implemented is still sitting out there! I know we have to worry about backward compatibility and whatnot, but a 20+ year old RFC feature that as far as I can determine was never actually implemented..... just sad. An interesting test here would be to set it to different values, see what different DHCP Servers would do with it. Or in writing your own DHCP Server, fill it in with other values and see if any clients actually notice. My guess is most don't even read the value, but just skip it, but who knows. Outside of testing I'll be doing at this time.

Ok, now that we have a run down on what a typical DHCP packet has in it we'll start utilizing that information in this section. We'll start by treating this like those high school math classes I'm sure we all loved so much. First we'll do it the hard way, then, after we understand how to do it that way we'll sit down and get to know how to do it the easy way. In case high school has changed in the 15 years or so since I was there, or the math classes at least, it used to be you always got to prove the formula before you could use it, or would learn how to do it the hard way first and then after that get shown the easy way. So here goes....

The Hard Way to Fingerprint DHCP

We can use multiple little differences in the way an OS implements its DHCP stack to identify the OS. Some of these ideas are nothing new to this paper, they are borrowed from ideas presented elsewhere and used for other OS identification products on different parts of the IP stack.

First we'll look at timing of how often a DHCP Discover packet is sent out when a DHCP Server is not present. We'll look at Microsoft OS's first since they have the same tendencies here as they do in their ICMP stack. A great paper on ICMP OS Fingerprinting is by Ofir Arkin and can be found at: http://www.sys-security.com/archive/papers/ICMP_Scanning_v3.0.pdf

To setup this test, each OS was started up on a segment without a DHCP Server on it to answer them. Clients were allowed up to 20 minutes to see what they'd do and what packets they would send out. All of the Microsoft based OS's would do their initial barrage of packets, go to "sleep" for about 5 minutes and then start over. Some of the Linux based systems would do something similar, where others would do their initial few attempts, and when they weren't answered, never appeared to attempt to contact the DHCP Server again.

Note: Microsoft Definitions and History

Before we get much farther here I'd like to throw out a definition or two:

Gold, as in Windows 95 Gold is the original release of an Microsoft OS, so in the case of Windows 95 we had the following releases:

*Windows 95 Gold
Windows 95 SP1 (Win 95A)
Windows 95 OSR2 (Win 95B)
Windows 95 OSR2.1 (Win 95B)
Windows 95 OSR2.5 (Win 95C)*

Below is a list of Windows based OS's and when they were released, this list is based off of wikipedia.org's list here: http://en.wikipedia.org/wiki/List_of_Microsoft_Windows_versions

DOS-Based

*1985 November - Windows 1.0
1987 December - Windows 2.0
1990 May - Windows 3.0
1992 August - Windows 3.1
1992 October - Windows for Workgroups 3.1
1993 November - Windows for Workgroups 3.11
1995 August - Windows 95 (4.00.950)
1995 December - Windows 95 SP1 (4.00.950A)
1996 August - Windows 95 OSR2 (4.00.1111 and 4.0.950B)
1997 August - Windows 95 OSR2.1 (4.00.1212 and 4.0.950B)
1997 August - Windows 95 OSR2.5 (4.00.1214 and 4.0.950C)*

1998 June - Windows 98 (4.10.1998 and 4.10.1998A)
1999 May - Windows 98 SE (4.10.2222, 4.10.2222A and 4.10.2222C)
2000 June - Windows Me (4.90.3000 and 4.90.3000A)

NT Kernel-Based

1993 August - Windows NT 3.1
1994 September - Windows NT 3.5
1995 June - Windows NT 3.51 (3.5.1057)
1996 July - Windows NT 4.0 (4.0.1381)
2000 February - Windows 2000 (5.0.2195)
2001 October - Windows XP
2003- Windows 2003 Server
2007 - Windows Vista

CE-based

1996 November - Windows CE 1.0
1997 November - Windows CE 2.0
1998 July - Windows CE 2.1
1998 October - Windows CE 2.11 for the Handheld PC
1999 August - Windows CE 2.12
2000 July - Windows CE 3.0

Note: Linux History and Definitions

Below is a list of Windows based OS's and when they were released, this list is based off of wikipedia.org's list here: http://en.wikipedia.org/wiki/Linux_kernel

Kernel

1991 September – Linux 0.1
1991 December – Linux 0.11
1992 March – Linux 0.95
1994 March – Linux 1.0.0
1995 March – Linux 1.2
1996 June – Linux 2.0
1999 January – Linux 2.2.0
2001 January – Linux 2.4.0
2003 December – Linux 2.6.0

Distributions

Redhat Linux

1995 May – RHL 1.0
1995 October – RHL 2.1
1996 March – RHL 3.0.3
1996 October – RHL 4.0
1997 December – RHL 5.0
1999 April – RHL 6.0
2000 September – RHL 7.0

2002 September – RHL 8.0
2003 April – RHL 9.0
Redhat Advanced Server/Enterprise Server
2002 March – 2.1
2003 October – 3.0
2005 February – 4.0
Fedora Core
2003 November – FC 1
2004 May – FC 2
2004 November – FC 3
2005 June – FC 4
2006 March – FC 5
2006 October – FC 6

Debian
1996 June – 1.1
1998 July – 2.0
2002 July – 3.0
2007 April – 4.0

We could do this all day on Linux distributions, if you need specific ones for where they fall in for RFC information check out wikipedia and you should be able to find it!

A large chunk of the identification process, in this paper, will be Windows based, of course not all of them, but a lot of them. There are a number of reasons for this, not least of which is I work as a Microsoft Engineer and am much more comfortable playing with it than with some of the other OS's. One of the main reasons though is that it, at least currently, is a "Microsoft world". I know, I just offended half of the purists out there. Sure other OS's are making in-roads into this, but the average user out there is still buying quite a few more Microsoft systems than anything else. Therefore, most of what I see on my work network is Microsoft machines. Since most of what I see there is Microsoft if just follows it is easier for me to verify those systems than all the others.

We will delve into quite a few Linux distributions, but not nearly as deeply. Most of the Linux distributions will be of the 'Live CD' flavor because they were easy to download and setup in a VM. Since most were live distributions we are also able to see some distinct similarities between them, but more on all of that later.

Again, as a reminder each of these tests was run for 15-20 minutes. Collecting packets from the time the system booted up, until it was shut down. The system was setup on a network without a DHCP Server on it to answer its queries. We are able to see the actual time packets showed up in the capture, the difference between each of those DHCP packets, the type of packet it was, the value that was stored in the Seconds Elapsed field and the TransactionID on that set of packets.

Unless otherwise noted all packets are DHCP DISCOVER packets since the Client has no knowledge of an existing DHCP Server out there or a currently valid IP address. We will see that this does not stop some clients from sending DHCP REQUEST packets from the get-go instead of sending initial DHCP DISCOVER packets.

Now that we have a general understanding of what is going on lets take a look at a Win95 machine when no DHCP Server is on the network to answer it.

Windows 95

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Windows 95 Gold					
	18.548022	0.000000	Unicast	0	0x15011501
	20.557468	2.009446	Unicast	512	0x15011501
	22.572565	2.015097	Unicast	1024	0x15011501
	24.576440	2.003875	Unicast	1536	0x15011501
	326.547115	301.970675	Unicast	0	0x7f977f97
	328.549272	2.002157	Unicast	512	0x7f977f97
	330.553576	2.004304	Unicast	1024	0x7f977f97
	332.559363	2.005787	Unicast	1536	0x7f977f97
	634.527055	301.967692	Unicast	0	0xe72de82d
	636.530027	2.002972	Unicast	512	0xe72de82d
	638.535967	2.005940	Unicast	1024	0xe72de82d
	640.540041	2.004074	Unicast	1536	0xe72de82d
	943.166368	302.626327	Unicast	0	0x4ec44fc4
	945.179732	2.013364	Unicast	512	0x4ec44fc4
	947.183176	2.003444	Unicast	1024	0x4ec44fc4
	949.189149	2.005973	Unicast	1536	0x4ec44fc4

Seen in both Windows 95 Gold and Windows 95B Testing

Note: I'm providing as much information in a lot of these so that someone else may be able to utilize the information provided here and find other things I may have missed. I know some of these tables go on longer than needed and that a few more trees will probably be killed in the printing of this, but I'm willing to live with that.

A coworker of mine used to joke, as we watched Windows copy files, that we only had “5 more minutes in Microsoft time” for the file to complete its copy. As most of you know, that little feature to tell how much time is left in a file copy is by no means accurate most of the time, or at least never has been for me. Why bring this up?

Look at the 'Seconds Elapsed' column. The Windows 95 box says that 512 seconds have elapsed since the previous DHCP Discover packet was sent. Now look back at the 'Time Difference' column. In actuality only 2 seconds have passed. Each 1 second in real life is equal to 256 in Microsoft time. If that is the case here, I'm not surprised it has such a hard time figuring out how long anything is going to take!

Ok, enough making fun of Microsoft's bad time management feature and back to something more useful. In Windows 95, when no DHCP Server is present, it will send 4 DHCP Discover packets, all spaced 2 seconds apart, sent in Unicast. It will then wait 5 minutes and 2 seconds (302 seconds) and start the process over again.

Now that we've talked a little about the 'Seconds Field' it is probably time to go back and look at what one of the RFC's has to say about this.

RFC 1532, which dealt with BOOTP, not DHCP had this to say about it:

3.2 Definition of the 'secs' field

The 'secs' field of a BOOTREQUEST message SHOULD represent the elapsed time, in seconds, since the client sent its first BOOTREQUEST message. Note that this implies that the 'secs' field of the first BOOTREQUEST message SHOULD be set to zero.

Clients SHOULD NOT set the 'secs' field to a value which is constant for all BOOTREQUEST messages.

DISCUSSION:

The original definition of the 'secs' field was vague. It was not clear whether it represented the time since the first BOOTREQUEST message was sent or some other time period such as the time since the client machine was powered-up. This has limited its usefulness as a policy control mechanism for BOOTP servers and relay agents. Furthermore, certain client implementations have been known to simply set this field to a constant value or use incorrect byte-ordering. Incorrect byte-ordering usually makes it appear as if a client has been waiting much longer than it really has, so a relay agent will relay the BOOTREQUEST sooner than desired (usually immediately). These implementation errors have further undermined the usefulness of the 'secs' field. These incorrect implementations SHOULD be corrected.

This RFC was released in October 1993 and was labeled 'Clarifications and Extensions for the Bootstrap Protocol'.

Remember this little bit here:

Clients SHOULD NOT set the 'secs' field to a value which is constant for all BOOTREQUEST messages.

This will come back up in later Microsoft OS's and in some of the Linux builds we look at.

Now that we've seen what Windows 95 did lets go back and look at what RFC 1541. (For those tracking RFC's you're probably asking "Hey I thought we were using RFC 2131 as the main one?" We are, but Windows 95 was released in 1995, RFC 2131 was released in 1997, so to give Microsoft a break we'll be nice and look at the RFC in place during this time frame).

RFC 1541

4.4.1 Initialization and allocation of network address

The client begins in INIT state and forms a DHCPDISCOVER message. The client should wait a random time between one and ten seconds to desynchronize the use of DHCP at startup. The client sets 'ciaddr' to 0x00000000. The client MAY request specific parameters by including the 'parameter request list' option. The client MAY suggest a network address and/or lease time by including the 'requested IP address' and 'IP address lease time' options. The

client MUST include its hardware address in the 'chaddr' field for use in delivery of DHCP reply messages. The client MAY include a different unique identifier in the 'client identifier' option. If the client does not include the 'client identifier' option, the server will use the contents of the 'chaddr' field to identify the client's lease.

The client generates and records a random transaction identifier and inserts that identifier into the 'xid' field. The client records its own local time for later use in computing the lease expiration. The client then broadcasts the DHCPDISCOVER on the local hardware broadcast address to 0xffffffff IP broadcast address and 'DHCP server' UDP port.

If the 'xid' of an arriving DHCP OFFER message does not match the 'xid' of the most recent DHCPDISCOVER message, the DHCP OFFER message must be silently discarded. Any arriving DHCPACK messages must be silently discarded.

The client collects DHCP OFFER messages over a period of time, selects one DHCP OFFER message from the (possibly many) incoming DHCP OFFER messages (e.g., the first DHCP OFFER message or the DHCP OFFER message from the previously used server) and extracts the server address from the 'server identifier' option in the DHCP OFFER message. The time over which the client collects messages and the mechanism used to select one DHCP OFFER are implementation dependent. The client may perform a check on the suggested address to ensure that the address is not already in use. For example, if the client is on a network that supports ARP, the client may issue an ARP request for the suggested request. When broadcasting an ARP request for the suggested address, the client must fill in its own hardware address as the sender's hardware address, and 0 as the sender's IP address, to avoid confusing ARP caches in other hosts on the same subnet. If the network address appears to be in use, the client sends a DHCPDECLINE message to the server and waits for another DHCP OFFER. As the client does not have a valid network address, the client must broadcast the DHCPDECLINE message.

Ok, so the client starts in the INIT state, or DHCPDISCOVER state with the client waiting 1-10 seconds to “desynchronize” during startup. Well this is going to be hard to determine since we really don't know when the OS's IP stack is online, so we'll assume it does this correctly.

To look at some of the other stuff it will be easier to look at a packet now:

```

Bootstrap Protocol
  Message type: Boot Request (1)
  Hardware type: Ethernet
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x16011601
  Seconds elapsed: 0
  Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0 (0.0.0.0)
  Your (client) IP address: 0.0.0.0 (0.0.0.0)
  Next server IP address: 0.0.0.0 (0.0.0.0)
  Relay agent IP address: 0.0.0.0 (0.0.0.0)
  Client MAC address: vmware_b7:97:b0 (00:0c:29:b7:97:b0)
  Server host name not given
  Boot file name not given
  Magic cookie: (OK)
  Option 53: DHCP Message Type = DHCP Discover
  Option 61: Client identifier
    Hardware type: Ethernet
    Client MAC address: vmware_b7:97:b0 (00:0c:29:b7:97:b0)
  Option 50: Requested IP Address = 192.168.0.3
  Option 12: Host Name = "w95"
  End Option
  Padding

```

Next the client is supposed to set its 'ciaddr' to 0x00000000, it is set to 0.0.0.0 so, so far so good. The client "MAY" request an IP address and lease time. As we see in the Options, Option 50 it requests the IP it has had in the past, but does not request a specific lease time.

The client "MUST" provide its hardware address, which would be found in the Client MAC address field.

Another "MAY" pops up now, it may provide a different unique ID for the Client Identifier Field (Option 61). As we see here Windows 95 provided the same information in Option 61 as its Client MAC address. This isn't always the case, but it is the typical thing you will see.

The Client is to create a random transaction ID. As we see above Windows 95 chose 0x16011601. At first I thought it kept picking something and just repeating it. 12341234 type idea. But it only does this on the first 2 sets of packets it sends in the first table above (0x15011501 and 0x7f977f97), then in the 3rd set of packets this pattern falls apart. We see this in some of the other OS's also, so there has to be something there, but I haven't dug around enough to figure out the pattern.

So from what we've seen above, Microsoft appears to be adhering to the RFC (except for the seconds elapsed field so far). But that can't be true can it, doesn't Microsoft always do its own thing?

Lets back up a little in the RFC

4.1 Constructing and sending DHCP messages

DHCP Clients and servers both construct DHCP messages by filling in fields in the fixed format section of the message and appending tagged data items in the variable length option area. The options area includes first a four-octet 'magic cookie' (which was described

in section 3), followed by the options. The last option must always be the 'end' option.

[cut]

DHCP Clients are responsible for all message retransmission. The client MUST adopt a retransmission strategy that incorporates a randomized exponential backoff algorithm to determine the delay between retransmissions. The delay before the first retransmission MUST be 4 seconds randomized by the value of a uniform random number chosen from the range -1 to +1. Clients with clocks that provide resolution granularity of less than one second may choose a non-integer randomization value. The delay before the next retransmission MUST be 8 seconds randomized by the value of a uniform number chosen from the range -1 to +1. The retransmission delay MUST be doubled with subsequent retransmissions up to a maximum of 64 seconds. The client MAY provide an indication of retransmission attempts to the user as an indication of the progress of the configuration process. The protocol specification in the remainder of this section will describe, for each DHCP message, when it is appropriate for the client to retransmit that message forever, and when it is appropriate for a client to abandon that message and attempt to use a different DHCP message.

Normally, DHCP Servers and BOOTP relay agents attempt to deliver DHCP OFFER, DHCP ACK and DHCP NAK messages directly to the client using unicast delivery. The IP destination address (in the IP header) is set to the DHCP 'yiaddr' address and the link-layer destination address is set to the DHCP 'chaddr' address. Unfortunately, some client implementations are unable to receive such unicast IP datagrams until the implementation has been configured with a valid IP address (leading to a deadlock in which the client's IP address cannot be delivered until the client has been configured with an IP address).

A client that cannot receive unicast IP datagrams until its protocol software has been configured with an IP address SHOULD set the BROADCAST bit in the 'flags' field to 1 in any DHCP DISCOVER or DHCP REQUEST messages that client sends. The BROADCAST bit will provide a hint to the DHCP Server and BOOTP relay agent to broadcast any messages to the client on the client's subnet. A client that can receive unicast IP datagrams before its protocol software has been configured SHOULD clear the BROADCAST bit to 0. The BOOTP clarifications document discusses the ramifications of the use of the BROADCAST bit [21].

A server or relay agent sending or relaying a DHCP message directly to a DHCP Client (i.e., not to a relay agent specified in the 'giaddr' field) SHOULD examine the BROADCAST bit in the 'flags' field. If this bit is set to 1, the DHCP message SHOULD be sent as an IP broadcast using an IP broadcast address (preferably 255.255.255.255) as the IP destination address and the link-layer broadcast address as the link-layer destination address. If the BROADCAST bit is cleared to 0, the message SHOULD be sent as an IP unicast to the IP address specified in the 'yiaddr' field and the link-layer address specified in the 'chaddr' field. If unicasting is not possible, the message MAY be sent as an IP broadcast using an IP

broadcast address (preferably 255.255.255.255) as the IP destination address and the link-layer broadcast address as the link-layer destination address.

Ok, I cut some of the 4.1 section out above, but left a chunk at the end that is useful information for other purposes. The whole flag section there about Unicast and Broadcast will come in eventually. But back to the problem at hand and the idea that Microsoft may have been following the RFC.

There is this little chunk about,

The client MUST adopt a retransmission strategy that incorporates a randomized exponential backoff algorithm to determine the delay between retransmissions. The delay before the first retransmission MUST be 4 seconds randomized by the value of a uniform random number chosen from the range -1 to +1.

It goes on to say that the next one should double this (so 4, 8, 16, 32 ±1) continuing to double to to a max of 64. Well lets look back at what the actual Time Difference was between Windows 95 DHCP Discover packets when the DHCP Server wasn't answering.... 2, 2, 2, 302

Ok, so there are a few things that Windows 95 seems to miss in the RFC:

- Time frame between retransmissions
- Whole point of the seconds elapsed field

Each Microsoft OS is a little different on this and I'll touch on each one below.

Windows 98

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Windows 98 Gold					
	0.000000	0.000000	Unicast	0	0xe800e800
	5.993846	5.993846	Unicast	1536	0xe800e800
	11.993841	5.999995	Unicast	3072	0xe800e800
	17.992411	5.998570	Unicast	4608	0xe800e800
	326.326851	308.334440	Unicast	0	0xaa9eaa9e
	332.335218	6.008367	Unicast	1536	0xaa9eaa9e
	338.344430	6.009212	Unicast	3072	0xaa9eaa9e
	344.356020	6.011590	Unicast	4608	0xaa9eaa9e
	649.329187	304.973167	Unicast	0	0x653c663c
	655.400866	6.071679	Unicast	1536	0x653c663c
	661.472528	6.071662	Unicast	3072	0x653c663c
	667.486649	6.014121	Unicast	4608	0x653c663c
	972.162776	304.676127	Unicast	0	0x2eda2fda
	978.470199	6.307423	Unicast	1536	0x2eda2fda
	984.489159	6.018960	Unicast	3072	0x2eda2fda
	990.555391	6.066232	Unicast	4608	0x2eda2fda

Similar to Windows 95 we see Seconds elapsed actually being a factor of 256, so again for every one second elapsed in real life, the OS is incrementing this field in the DHCP packet by 256. Also like Windows 95 we see 4 DHCP packets sent out when no DHCP Server is there to respond, but something a little different now, instead of every 2 seconds, we now get packets ever 6 seconds, then a break of 5 minutes and 6 seconds and the

process starts over again.

So there are a few things that Windows 98 Gold seems to miss in the RFC:

- Time frame between retransmissions
- Whole point of the seconds elapsed field

Windows 98 SE

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Windows 98 SE					
	0.000000	0.000000	Unicast	0	0x4264c24f
	4.016893	4.016893	Unicast	53345	0x4264c24f
	11.993695	7.976802	Unicast	53345	0x4264c24f
	354.526033	342.532338	Unicast	0	0x451ac06f
	358.534833	4.008800	Unicast	40874	0x451ac06f
	366.538299	8.003466	Unicast	40874	0x451ac06f
	381.493376	14.955077	Unicast	40874	0x451ac06f
	691.511730	310.018354	Unicast	0	0x5913326d
	696.513772	5.002042	Unicast	20137	0x5913326d
	705.525158	9.011386	Unicast	20137	0x5913326d
	722.485332	16.960174	Unicast	20137	0x5913326d
	1035.460828	312.975496	Unicast	0	0xce351b1d
	1039.470282	4.009454	Unicast	63143	0xce351b1d
	1046.475661	7.005379	Unicast	63143	0xce351b1d
	1061.483806	15.008145	Unicast	63143	0xce351b1d

With Windows 98 SE we get a little change. First it appears someone got tired of that whole 256 thing and decided to change it up a bit for this release (who said 98SE was just a re-release of the same OS?). Instead of incrementing it they decided to lock it to a specific value for all packets sent in that round. No idea what this value will be each time, it appears to be random.

Here is where RFC 1532 shows up again, remember that little bit about:

```
Clients SHOULD NOT set the 'secs' field to a value which is constant
for all BOOTREQUEST messages.
```

Well they didn't exactly set it to a constant value for all of the packets, just all of the packets after the initial one.

As you can also see in the table above, the first volley of DHCP Discover packets, unlike in the previous OS releases we've seen, we only get 3 DHCP Discover packets, then a little over 5 minute break and then back to the 4 packet volley we've come to expect from a Microsoft OS. Also a little difference here is that each packet is spaced a little differently. In previous OS's each packet was evenly spaced. Now we see the 2nd packet being sent 4 seconds (± 1) after the first, then the next packet being sent 8 seconds (± 1) after that, then 15 seconds (± 1) after that.

With Windows 98 SE they sorta figured out:

- Time frame between retransmission in the RFC

But then they missed this even more than they had before:

- Whole point of the seconds elapsed field

My only conclusion at this point is that Microsoft hired a new group every time to write the DHCP stuff and they either didn't like what the previous group had done or figured they could do it better and did things completely different. Got me, all I know is based on what we've seen up to here it sure looks that way!

Window ME

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Windows ME					
	18.510131	0.000000	Unicast	0	0x4947fe04
	34.077567	15.567436	Unicast	13155	0x4947fe04
	41.869045	7.791478	Unicast	13155	0x4947fe04
	56.266649	14.397604	Broadcast	0	0x1d738766
	60.266023	3.999374	Broadcast	2659	0x1d738766
	68.274701	8.008678	Broadcast	2659	0x1d738766
	84.279593	16.004892	Broadcast	2659	0x1d738766
	407.452938	323.173345	Broadcast	0	0x2e64e045
	411.452450	3.999512	Broadcast	45377	0x2e64e045
	420.464766	9.012316	Broadcast	45377	0x2e64e045
	436.481583	16.016817	Broadcast	45377	0x2e64e045
	797.448305	360.966722	Broadcast	0	0x8626547d
	801.420618	3.972313	Broadcast	10048	0x8626547d
	808.432701	7.012083	Broadcast	10048	0x8626547d
	823.463090	15.030389	Broadcast	10048	0x8626547d

Windows ME made a few changes to the previous consumer version of Windows, but kept a few also (maybe the comment about a different group every time was a bit premature here). With Windows ME we now get an initial volley of 3 DHCP Discover packets sent in Unicast mode, when there is no answer here it waits about 14 seconds and tries again. This time it sets the DHCP flag as Broadcast instead of Unicast. Waits times are 4, 8, and 16 seconds, each ± 1 and then a 5 minutes and some odd seconds break, to start over again. Like Windows 98 SE it decides that the whole Seconds Elapsed field is just something it needs to fill and shoves some junk in it.

This wraps up the consumer version of Windows for back in the day. Next we'll take a look at the business version of Windows NT and 2000. Eventually looking at some of their latest stuff of Vista.

But before that, a quick wrap up on the Win9x OS's for this section (unfortunately for us all, they'll show up again somewhere in this paper). For the most part we've seen the DHCP Discover packet in Unicast mode, the seconds elapsed being tied to 256 and the time between packets being set to a fairly straight forward algorithm with a 5 minute delay between DHCP Discover volleys.

Windows NT

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Windows NT 4 SP1					
	3.849133	0.000000	Unicast	0	0x6b154665
	13.360092	9.510959	Unicast	2560	0x6b154665
	26.562041	13.201949	Unicast	5888	0x6b154665
	44.719590	18.157549	Unicast	10496	0x6b154665
	365.917669	321.198079	Unicast	0	0x7f0f2410
	375.602385	9.684716	Unicast	2560	0x7f0f2410
	389.665813	14.063428	Unicast	6144	0x7f0f2410
	407.550019	17.884206	Unicast	10752	0x7f0f2410
	729.690455	322.140436	Unicast	0	0x5f12c41d
	739.490730	9.800275	Unicast	2304	0x5f12c41d
	753.685147	14.194417	Unicast	6144	0x5f12c41d
	772.512828	18.827681	Unicast	10752	0x5f12c41d
	1094.304309	321.791481	Unicast	0	0x7a126f6e
	1104.051941	9.747632	Unicast	2560	0x7a126f6e
	1118.582369	14.530428	Unicast	6400	0x7a126f6e
	1137.338007	18.755638	Unicast	11008	0x7a126f6e

I thought I had a copy of NT Gold around, but haven't found it, so we'll look at SP1 here first. Again we see the 256 factor, packets sent in Unicast, but a bit different time difference in between. It appears to be 9, 14, and 18, all ± 1 .

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Windows NT 4 SP5					
	3.907075	0.000000	Unicast	0	0x2272886f
	7.871992	3.964917	Unicast	1024	0x2272886f
	15.912602	8.040610	Unicast	3072	0x2272886f
	31.884816	15.972214	Unicast	7168	0x2272886f
	364.976805	333.091989	Unicast	0	0xb53c0626
	368.943210	3.966405	Unicast	1024	0xb53c0626
	377.932114	8.988904	Unicast	3328	0xb53c0626
	393.960682	16.028568	Unicast	7424	0xb53c0626
	748.980794	355.020112	Unicast	0	0xca00bb73
	753.949018	4.968224	Unicast	1280	0xca00bb73
	761.990595	8.041577	Unicast	3328	0xca00bb73
	778.976421	16.985826	Unicast	7680	0xca00bb73
	1114.010100	335.033679	Unicast	0	0x7720cd50
	1117.975139	3.965039	Unicast	1024	0x7720cd50
	1124.962010	6.986871	Unicast	2816	0x7720cd50
	1140.996640	16.034630	Unicast	6912	0x7720cd50

SP5 seems to have kept most of the same stuff, but again it has changed the pause between packets. Now using 4, 8, and 16, ± 1 . I'm not sure when this changed, but I assume either in SP3 or SP5 since that seems to be when most changes were made to this OS from what I recall (sorry was never a big NT 4 fan).

Even though they seemed to fix the spacing issue, the seconds elapsed seems to be uniquely Microsoft even in

the business version of Windows.

Windows 2000

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Windows 2000 Server SP4					
	0.000000	0.000000	Unicast	0	0x1e2c1abc
	4.006332	4.006332	Unicast	1024	0x1e2c1abc
	11.991062	7.984730	Unicast	3072	0x1e2c1abc
	28.022814	16.031752	Unicast	7168	0x1e2c1abc
	63.983527	35.960713	Broadcast	0	0x662ead22
	68.960069	4.976542	Broadcast	1280	0x662ead22
	77.012748	8.052679	Broadcast	3328	0x662ead22
	93.004581	15.991833	Broadcast	7424	0x662ead22
	410.031348	317.026767	Broadcast	0	0x0cff5ecd
	414.004389	3.973041	Broadcast	1024	0x0cff5ecd
	421.999163	7.994774	Broadcast	3072	0x0cff5ecd
	439.022150	17.022987	Broadcast	7424	0x0cff5ecd
	791.009081	351.986931	Broadcast	0	0x4f706246
	795.035815	4.026734	Broadcast	1024	0x4f706246
	803.022808	7.986993	Broadcast	3072	0x4f706246
	819.149566	16.126758	Broadcast	7168	0x4f706246

Windows 2000, released right around the same time as Windows ME, they kicked over to this Unicast/Broadcast type feature. Starts out with 4 Unicast packets, when it gets no response it changes to Broadcast. Like SP5 of NT we have a 4, 8, and 16 ± 1 'Time Difference' and they are still clinging to their 256 seconds in Microsoft time being equal to 1 second in real life feature.

Before we move on to their current OS's of Vista (was only a beta as of the initial writing of this) we'll do a quick rehash here. Like the previous set of OS's we get this lovely 256 seconds = 1 second rule, we have a set of rules to determine how many seconds the OS will wait between DHCP Discover packets and then its roughly 5 minute wait between volleys.

Windows Vista

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Windows Vista					
	36.643228	0.000000	Broadcast	0	0xa9e19fdc
	36.774136	0.130908	Broadcast	0	0x47c9f24d
	40.861436	4.087300	Broadcast	1024	0x47c9f24d
	48.107649	7.246213	Broadcast	3072	0x47c9f24d
	63.972979	15.865330	Broadcast	6912	0x47c9f24d
	403.203627	339.230648	Broadcast	0	0x5837e59f
	407.173866	3.970239	Broadcast	1024	0x5837e59f
	416.166606	8.992740	Broadcast	3328	0x5837e59f
	431.169484	15.002878	Broadcast	7168	0x5837e59f
	740.306192	309.136708	Broadcast	0	0x7feedd14
	745.314252	5.008060	Broadcast	1280	0x7feedd14
	753.309754	7.995502	Broadcast	3328	0x7feedd14
	770.342450	17.032696	Broadcast	7680	0x7feedd14
	1073.395371	303.052921	Broadcast	0	0xdd85322a
	1078.349264	4.953893	Broadcast	1280	0xdd85322a
	1086.366535	8.017271	Broadcast	3328	0xdd85322a
	1102.362298	15.995763	Broadcast	7424	0xdd85322a

Windows Vista has changed the Way Microsoft does DHCP a little bit, but not much really. They've kicked over to Broadcast for the DHCP Flag field completely now and they also appear to send an extra initial packet that is not tied into the rest of them (see how the Transaction ID changes between packet 1 and packet 2). But other than that, they are in the same basic boat they've been in for quite some time. They are still doing something close to 4, 8, and 16 second intervals, with a 5 minute and some odd second interval between volleys. According to the RFC they really should have had a 32 and 64 second packet in there. They also have never got around to changing their whole 1 second in real life = 256 seconds in Microsoft time feature.

Supposedly this Broadcast Flag can be turned off in Vista, but it is on by default.

Now that we've had a lot of fun, can we really call it that, fun, anyway, now that we've had all this fun at Microsoft's expense, lets look at some of the Linux boxes I came up with for testing.

Arudis

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Arudis					
	0.000000	0.000000	Unicast	10	0x4591d25f
	1.651587	1.651587	Unicast	10	0x957746a
	3.044003	1.392416	Unicast	10	0x4591d25f
	5.672182	2.628179	Unicast	10	0x957746a
	11.069977	5.397795	Unicast	10	0x4591d25f
	13.776210	2.706233	Unicast	10	0x957746a
	27.173683	13.397473	Unicast	10	0x4591d25f
	29.893524	2.719841	Unicast	10	0x957746a

This box had a packet capture run for only about 11 minutes. You can see all the data collected. For the first 30 seconds it sent out DHCP DISCOVER packets, after getting no response it stopped trying.

One of the things we see completely different here than we did in the Microsoft tests is that it has set a value in the Seconds Elapsed field and kept it consistently throughout its DHCP startup sequence. One thing to note is that it had 2 different DHCP DISCOVER processes going at the same time. It is supposed to keep the same Transaction ID throughout this process. In this case we see 2 distinct ones. We can break that down to this:

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Arudis					
	0.000000	0.000000	Unicast	10	0x4591d25f
	3.044003	3.044003	Unicast	10	0x4591d25f
	11.069977	8.025974	Unicast	10	0x4591d25f
	27.173683	16.103706	Unicast	10	0x4591d25f
OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Arudis					
	1.651587	0.000000	Unicast	10	0x957746a
	5.672182	4.020595	Unicast	10	0x957746a
	13.776210	8.104028	Unicast	10	0x957746a
	29.893524	16.117314	Unicast	10	0x957746a

Based on this we get it sending out packets at 4, 8, and 16 second intervals ± 1 . Why it was doing two at the same time I'm not sure, but something to look into later if time permits.

Gentoo 2005.0

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Gentoo 2005.0					
	0.000000	0.000000	Unicast	10	0x8e2b2c51
	3.550021	3.550021	Unicast	10	0x8e2b2c51
	14.915227	11.365206	Unicast	10	0x8e2b2c51
	36.471822	21.556595	Unicast	10	0x8e2b2c51

We see the same basic things here. A set Seconds Elapsed field and a failure to try to do DHCP Discovers again after the initial attempt fails.

Gentoo 2006.1

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Gentoo 2006.1					
(DHCP Request)	0.000000	0.000000	Unicast	10	0xeceeee772
(DHCP Request)	4.194787	4.194787	Unicast	10	0xeceeee772
(DHCP Request)	12.401223	8.206436	Unicast	10	0xeceeee772
(DHCP Request)	28.860002	16.458779	Unicast	10	0xeceeee772
(DHCP Discover)	60.106819	31.246817	Unicast	10	0xa5a7371a
(DHCP Discover)	64.175171	4.068352	Unicast	10	0xa5a7371a
(DHCP Discover)	72.249659	8.074488	Unicast	10	0xa5a7371a
(DHCP Discover)	88.294355	16.044696	Unicast	10	0xa5a7371a

This build seems to have something a bit different going on than some of the rest. It started off doing a DHCP REQUEST, this is typically done when you have an existing IP address or are doing a renewel/rebind, not something you do when the machine hasn't been turned on before. Was this a fluke, does it have to do with when they made the original LiveCD that this was booted off of, not sure, sometime we need to rerun it for validity.

CentOS 4

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
CentOS 4					
	0.000000	0.000000	Unicast	0	0x75f6c938
	3.222040	3.222040	Unicast	0	0x65782858
	6.403152	3.181112	Unicast	6	0x75f6c938
	8.383582	1.980430	Unicast	7	0x65782858
	33.829241	25.445659	Unicast	0	0x89863f03
	37.761298	3.932057	Unicast	4	0x89863f03
	46.568221	8.806923	Unicast	13	0x89863f03
	61.236646	14.668425	Unicast	28	0x89863f03
	80.741610	19.504964	Unicast	48	0x89863f03
	90.459483	9.717873	Unicast	58	0x89863f03

Again, we see 2 separate processes going on at the same time here. One with a Transaction ID of 0x75f6c938 and another of 0x65782858. Each approximately 6 ± 1 seconds apart from each other. It then takes a little break and starts over sending out 6 packets and then quitting when it doesn't get a response.

One change though is that the Seconds Elapsed Field is actually correct.

Later when we look at the easy way to ID systems we'll see that CentOS and Fedora can't be differentiated by the simple means, but as we will see here in a second maybe in the case of no DHCP Server being present we can utilize it to tell the difference?

Fedora Core 3

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Fedora Core 3					
	91.557369	0.000000	Unicast		00x9708cf05
	96.513013	4.955644	Unicast		50x9708cf05
	110.525387	14.012374	Unicast		190x9708cf05
	123.491597	12.966210	Unicast		320x9708cf05
	139.500417	16.008820	Unicast		480x9708cf05

Here, unlike in CentOS 4, we don't get the 2 sets of DHCP DISCOVER packets. More tests need to be done to see if this always happens or if one of these was a fluke due to some unknown reason. Again on this one, like the previous ones, after the initial flurry of DHCP DISCOVER packets, this OS never seemed interested in trying again. I gave it 10 or so minutes and never saw any other packets.

Fedora Core 4

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Fedora Core 4					
	2.780580	0.000000	Unicast		00xd1cd455f
	7.785394	5.004814	Unicast		50xd1cd455f
	18.762807	10.977413	Unicast		160xd1cd455f
	29.743068	10.980261	Unicast		270xd1cd455f
	36.779749	7.036681	Unicast		340xd1cd455f
	49.768023	12.988274	Unicast		470xd1cd455f

It looks a lot like the Fedora Core 3 one, except we picked up one more packet in there. Instead of only 5 packets, we got 6. Again this could have to do with the algorithm that they use to determine when to send out packets. This was a one time test and hasn't been repeated yet.

Fedora Core 5

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Fedora Core 5					
	92.332620	0.000000	Unicast		00x5aff2e17
	96.956827	4.624207	Unicast		50x5aff2e17
	105.951240	8.994413	Unicast		140x5aff2e17
	119.962996	14.011756	Unicast		280x5aff2e17
	127.968500	8.005504	Unicast		360x5aff2e17
	136.983728	9.015228	Unicast		450x5aff2e17
	148.983644	11.999916	Unicast		570x5aff2e17

Like the previous test, we've increased the number of packets. Instead of 6 packets like in Fedora Core 4, we got 7 packets here. Other than that everything looks basically the same.

Knoppix 5.1

OS	Actual Time	Time Diff	Broad/Unicast	Sec elapsed	TransID
Knoppix 5.0.1					
	0.000000	0.000000	Unicast		00x60d7d6c3
	4.005759	4.005759	Unicast	1024	0x60d7d6c3
	11.037472	7.031713	Unicast	2816	0x60d7d6c3
	20.833675	9.796203	Unicast	5120	0x60d7d6c3

For a Linux build, here is something a little odd, they seem to be taking pointers from Microsoft and sticking to a factor of 256 for that Seconds Elapsed field. I guess we can't just divide the number in their by 256 and see if it is a factor of 256 and assume it is a Microsoft box anymore. Oh well.

Other Linux Builds

All of the Linux builds so far have decided that if the DHCP Server wasn't up initially, then why try again, at least in the short term (again, only gave them 10 minutes to try again). Other builds, such as Lindows (Linspire), OpenBSD and a few others did not seem to have this same short coming, but instead continued to send out DHCP DISCOVER packets every few seconds, waiting for that DHCP Server to come back on line and provide it a way onto the network.

There was a lot more data collected on these other builds, but due to time constraints they will not be added at this time. I hope to release version 1.5 of this paper down the road at some time and to add some of the other OS's that were skipped, both Linux and Windows machines, along with other OS's such as BeOS.

IP TTL on DHCP Packets

OS	TTL
Anonym.os 1.0	16
Auditor 20060502	16
Cent OS 4	16
Fedora Core 3	16
Fedora Core 4	16
Fedora Core 5	16
Fedora Core 6	16
Helix 1.8	16
Insert 1.3.8	16
Kanotix 2005-01	16
Kanotix 2005-02	16
Kanotix 2005-03	16
Kanotix 2005-04	16
Kanotix 2006-01 RC4	16
Knoppix 3.6	16
Knoppix 3.7	16
Knoppix 3.8	16
Knoppix 3.9	16
Knoppix 4.0	16
Knoppix 5.0	16
Knoppix 5.1	16
Knoppix STD 0.1	16
Open BSD 3.8	16
Operator 3.3	16
PC BSD 1.2	16
PC BSD 1.3	16
PC Linux	16
PHLAK 0.1	16
PHLAK 0.2	16
PHLAK 0.3	16
SLED 10	16

OS	TTL
Window s 95	32

OS	TTL
Backtrack 1.0	64
Backtrack 2.0	64
Freespire 1.0	64
Freespire 2.0	64
Gentoo 2005.0	64
Gentoo 2006.0	64
Gentoo 2006.1	64
Linspire 4.5	64
Linspire 5.0	64
Red Hat 6.2	64
Slax 5.1.8	64
SUSE Linux 10.1	64
Ubuntu - Gnome 2.10	64
Ubuntu 5.10	64
Ubuntu 6.10	64

OS	TTL
Window s 98	128
Window s 98 SE	128
Window s ME	128
Window s NT 4	128
Window s 2000	128
Window s XP	128
Window s Vista	128

OS	TTL
OSX	255

If you don't find your favorite OS or Distribution in there, look at one that is based off the same family branch. Odds are when they built it, they didn't tweak it down at the lower layer, just the pretty front end that they put on it!

As noted in the title of this, this is the TTL on DHCP packets. Windows seems to pretty much stick to a TTL of 128 across all IP packets. Linux seems to take into account that your DHCP Server really shouldn't be up to 127 hops away from you and drops this number down a bit more. OS X on the other hand seems to want to make sure your DHCP server can be contacted no matter how far away it is, so it gives it the most it can at 255!

DHCP Options – the easy way

Using all Options

Now that we've looked at the hard way to do identification lets take a look at an easier way. Each machine, as it boots up, requests certain DHCP Options and Parameters on those Options. We can see that in the following packet capture:

```

Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
  Version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  Total Length: 328
  Identification: 0x0000 (0)
  Flags: 0x00
  Fragment offset: 0
  Time to live: 32
  Protocol: UDP (0x11)
  Header checksum: 0x99a6 [correct]
  Source: 0.0.0.0 (0.0.0.0)
  Destination: 255.255.255.255 (255.255.255.255)
User Datagram Protocol, Src Port: bootpc (68), Dst Port: bootps (67)
Bootstrap Protocol
  Message type: Boot Request (1)
  Hardware type: Ethernet
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0xa000a000
  Seconds elapsed: 0
  Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0 (0.0.0.0)
  Your (client) IP address: 0.0.0.0 (0.0.0.0)
  Next server IP address: 0.0.0.0 (0.0.0.0)
  Relay agent IP address: 0.0.0.0 (0.0.0.0)
  Client MAC address: vmware_8b:69:93 (00:0c:29:8b:69:93)
  Server host name not given
  Boot file name not given
  Magic cookie: (OK)
  Option: (t=53,l=1) DHCP Message Type = DHCP Discover
  Option: (t=61,l=7) Client identifier
  Option: (t=50,l=4) Requested IP Address = 192.168.0.27
  Option: (t=12,l=5) Host Name = "w95b"
  End Option
  Padding

```

The first thing we have boxed/marked up there is 'Differentiated Services Field'. I'm not going to touch on this in this paper, but I have noticed that some OS's have a value of 0x00, where others seem to always be 0x10. This could come into some use in differentiating between OS's at a later time. This has more to do with the IP

stack as a whole than the DHCP Stack, but it appears to be tied into DHCP, in a very loose way for identification, and that is the only reason I mention it in this paper.

'Time To Live' happens to be next. We've already seen the typical time to live size for a lot of the OS's out there in the IP stack in my other paper, but the TTL on IP packets that happen to be DHCP packets seem to be different than the default OS IP TTL, at least on non-Windows machines.

'Second's Elapsed' we've discussed in the first section and how this is one of those fields that each OS seems to do its own thing with.

'Bootp Flags', here we basically have two options, either it is Unicast or it is Broadcast, that simple.

'Client IP Address', here we'll typically see the 0.0.0.0, but if the lease expires and the machine is still online the Client will request its existing IP address again (typically at least, some OS's seem to figure if they have the lease it is good until they shut down and they don't do a request for that address again until they reboot.

But now onto the main feature, the Options that a machine requests upon bootup. In this case we have a Windows 95B machine asking for Options 53, 61, 50, 12. At this point it actually is just doing a DHCP Discover (see Option 53) and then providing the DHCP Server with some other information.

Some of that other information is its current hostname, and since most machines like to keep the same IP address they had in the past, it tells the DHCP Server what IP it would like to have if it is available.

Option 55

By large Option 55 will give us the best information in regards to what OS is being run. In some cases it will be the same across multiple OS's, or normally over multiple flavors of the same underlying kernel in the Linux world. Utilizing the parameters requested in Option 55, and the order in which they are requested will/can typically identify the OS by requesting the IP address.

We'll see more about this throughout the next few sections!

Windows 95

During a normal startup process we'll see these 2 packets from a Windows 95 machine:

OS	Type	TTL	Options	Option 55
Window s 95	Discover	32	53,61,50,12	
Window s 95	Request	32	53,61,50,54,12,55,43	1,3,15,6,44,46,47

A TTL of 32 is unique to Windows 95 from my testing on DHCP. With Windows in general the DHCP TTL and the TTL it sends on almost all other IP packets is the same. So 32 for Windows 95 machines, 128 for all the rest.

If we were only to see the DHCP Discover packet, we could utilize the DHCP Options to identify this machine. Typically though, most networks will have a DHCP Server on the network, so after the machine sends out the Discover packet it will get an Offer from the Server, once that happens it will then send a DHCP Request. At this time we can utilize the information provided in parameter list of DHCP Option 55 to uniquely identify the

machine.

In most cases we never need to rely on the DHCP Options as a whole, but can utilize the DHCP Option 55 parameter list to identify machines. There are some cases though where Option 55 data is the same on multiple OS's. In that case if we fall back to utilizing all of the Options we may be able to get a better guess as to the OS.

With that being the case, you may ask why we don't utilize the DHCP Options alone. The problem is they are, typically, too generic. Options alone may give you the family the OS belongs to, but they typically will not give you the actual build in that family. That being said, new builds of OS's are coming out all the time, so to dismiss the Options field completely would be a bad idea. Some times the only thing we have to go on is the Options it gives us.

Windows 98

Next we'll look at Windows 98 SE because it adds another piece to the puzzle:

```
⊕ Option: (t=53,l=1) DHCP Message Type = DHCP Discover
⊕ Option: (t=61,l=7) Client identifier
⊕ Option: (t=50,l=4) Requested IP Address = 192.168.0.23
⊕ Option: (t=12,l=6) Host Name = "w98se"
⊕ Option: (t=60,l=7) vendor class identifier = "MSFT 98"
⊕ Option: (t=55,l=9) Parameter Request List
```

We now have what is known as the DHCP Vendor Class Identifier, or as I typically refer to it the Vendor Code. Windows 98SE and Window ME both started adding MSFT 98.

The Vendor Code was introduced so that vendors could provide specific information from the client. This allows the DHCP Server to send specific information back to the client for the type of hardware it is. This of course assumes that the Vendor Class Identifier is accurate. If Windows 98SE and Windows ME both say they are MSFT 98, that may be fine in most cases, since you'd be sending both 98 and ME the same general information, but what about those cases where one flavor of the OS is no longer supported and the other is. Say you wanted to get all Windows 98SE boxes off the network, but ME was still ok, well this information alone isn't enough to do that.

Microsoft as a whole

Here we see all of the main information from the Microsoft Systems out there:

OS	Type	TTL	Options	Option 55	Vendor Code
Windows 95	Discover	32	53,61,50,12		
Windows 95	Request	32	53,61,50,54,12,55,43	1,3,15,6,44,46,47	
Windows 98	Discover	128	53,61,50,12,55	1,3,6,15,44,46,47,57	
Windows 98	Request	128	53,61,50,54,12,55	1,3,6,15,44,46,47,57	
Windows 98 SE	Discover	128	53,61,50,12,60,55	1,15,3,6,44,46,47,43,77	MSFT 98
Windows 98 SE	Request	128	53,61,50,54,12,81,60,55	1,15,3,6,44,46,47,43,77	MSFT 98
Windows 98 SE (Renewel)	Request	128	53,61,50,12,81,60,55	1,15,3,6,44,46,47,43,77	MSFT 98
Windows ME	Discover	128	53,251,61,50,12,60,55	1,15,3,6,44,46,47,31,33,43,77	MSFT 98
Windows ME	Request	128	53,61,50,54,12,60,55	1,15,3,6,44,46,47,31,33,43,77	MSFT 98
Windows ME (Renewel)	Request	128	53,61,12,60,55	1,15,3,6,44,46,47,31,33,43,77	MSFT 98
Windows NT 4	Discover	128	53,61,50,12,55	1,15,3,44,46,47,6	
Windows NT 4	Request	128	53,61,50,54,12,55	1,15,3,44,46,47,6	
Windows NT 4 (Renewel)	Request	128	53,61,12,55	1,15,3,44,46,47,6	
Windows 2000	Discover	128	53,251,61,50,12,60,55	1,15,3,6,44,46,47,31,33,43	MSFT 5.0
Windows 2000	Request	128	53,61,50,54,12,81,60,55	1,15,3,6,44,46,47,31,33,43	MSFT 5.0
Windows 2000 (Renewel)	Request	128	53,61,12,81,60,55	1,15,3,6,44,46,47,31,33,43	MSFT 5.0
Windows XP	Discover	128	53,116,61,50,12,60,55	1,15,3,6,44,46,47,31,33,249,43	MSFT 5.0
Windows XP	Request	128	53,61,50,54,12,81,60,55	1,15,3,6,44,46,47,31,33,249,43	MSFT 5.0
Windows XP (Renewel)	Request	128	53,61,12,81,60,55	1,15,3,6,44,46,47,31,33,249,43	MSFT 5.0
Windows Vista	Discover	128	53,116,61,50,12,60,55,43	1,15,3,6,44,46,47,31,33,121,249,43	MSFT 5.0
Windows Vista	Request	128	53,61,50,54,12,81,60,55,43	1,15,3,6,44,46,47,31,33,121,249,43	MSFT 5.0
Windows Vista (Renewel)	Request	128	53,61,12,81,60,55,43	1,15,3,6,44,46,47,31,33,121,249,43	MSFT 5.0
Windows Vista	Inform	128	53,61,12,60,55,43	1,15,3,6,44,46,47,31,33,121,249,43,252	MSFT 5.0

The problem is the OS, for whatever reason, from time to time will add something onto Option 55. Typically it is just a '252' as we see in the Vista Inform Packet, but that isn't always the case.

Below are the 3 I've typically seen from a Windows XP machine, but I have reports of others. Most of the others are variations where the '1' is dropped off of them:

```
1,15,3,6,44,46,47,31,33,249,43
1,15,3,6,44,46,47,31,33,249,43,252
1,15,3,6,44,46,47,31,33,249,43,252,12
```

So it looks like an addition of '252' or '252,12' to the end of a known Windows Option 55 parameter list will probably be a feasible way to ID machines also. But this hasn't been extensively tested yet!

Next we'll start looking at some Linux machines. Since a lot of Linux builds all come from the same parent, we can see this in their DHCP bootup process. One other thing that has been interesting to see is that some builds have not changed their DHCP bootup process throughout quite a few iterations.

Fedora Core and Cent OS

Fedora Core 3	Request	16	53,54,50,55	1,28,2,3,15,6,12,40,41,42
Fedora Core 4 (Renewel)	Request	16	53,55	1,28,2,3,15,6,12,40,41,42
Fedora Core 5	Discover	16	53,50,55	1,28,2,3,15,6,12,40,41,42
Fedora Core 3	Discover	16	53,50,55	1,28,2,3,15,6,12,40,41,42
Fedora Core 3 (Renewel)	Request	16	53,55	1,28,2,3,15,6,12,40,41,42
Fedora Core 4	Discover	16	53,50,55	1,28,2,3,15,6,12,40,41,42
Fedora Core 4	Request	16	53,54,50,55	1,28,2,3,15,6,12,40,41,42
Fedora Core 5	Request	16	53,54,50,55	1,28,2,3,15,6,12,40,41,42
Fedora Core 6 (Renewel)	Request	16	53,55	1,28,2,3,15,6,12,40,41,42
Cent OS 4	Discover	16	53,50,55	1,28,2,3,15,6,12,40,41,42
Cent OS 4	Request	16	53,54,50,55	1,28,2,3,15,6,12,40,41,42
Fedora Core 5 (Renewel)	Request	16	53,55	1,28,2,3,15,6,12,40,41,42
Fedora Core 6	Discover	16	53,50,55	1,28,2,3,15,6,12,40,41,42
Fedora Core 6	Request	16	53,54,50,55	1,28,2,3,15,6,12,40,41,42
CentOS 4 (Renewel)	Request	16	53,55	1,28,2,3,15,6,12,40,41,42

So as you can see above Fedora Core 3-6 and Cent OS 4 (and I believe the other versions of Cent OS also) all use the same underlying DHCP stack. So here is one place that we can't differentiate between OS versions or different OS builds.

Backtrack, Gentoo and Slax using Option 60

We see this on other Linux OS's also, but in some cases we can catch a break by looking at yet another field that some of them, like Microsoft OS's, have in them. Back to that lovely Vendor Code. In this case though we get better differentiation because of it.

Backtrack 1.0	Request	64	1,3,6,12,15,17,23,28,29,31,33,40,41,42	Linux 2.6.15.6 i686
Gentoo 2005.0	Request	64	1,3,6,12,15,17,23,28,29,31,33,40,41,42	Linux 2.6.11-gentoo-r3 i686

The nice builders of Gentoo wanted to put a little special touch on their build to set it apart from others. So they tagged the name right in there. Not just telling us that it was Gentoo Release 3, but that it was running the Linux 2.6.11 Kernel.

The people at Backtrack didn't give us as much added information to inform us that it was Backtrack 1.0, but they did provide us with the info that it was running the Linux 2.6.15.6 Kernel.

We can see that they have both continued this practice in later versions:

Slax 5.1.8	Request	64	1,3,6,12,15,17,23,28,29,31,33,40,41,42,119	Linux 2.6.16 i686
Gentoo 2006.0	Request	64	1,3,6,12,15,17,23,28,29,31,33,40,41,42,119	Linux 2.6.15-gentoo-r5 i686
Gentoo 2006.1	Request	64	1,3,6,12,15,17,23,28,29,31,33,40,41,42,119	Linux 2.6.17-gentoo-r8 i686
Backtrack 2.0	Request	64	1,3,6,12,15,17,23,28,29,31,33,40,41,42,119	Linux 2.6.18-rc5 i686

So here again we see that Gentoo has told us that Gentoo R5 is running Linux 2.6.15 Kernel. Gentoo R8 happened to be 2006.1, not sure what happened to R6 and R7.

Backtrack upgraded from the 2.6.11 kernel to the 2.6.18 RC5 one.

Slax 5.1.8 got in here also with the 2.6.16 kernel.

Now could there be overlaps here? Sure, Slax 5.1.9 or some other such build may also be running the Linux 2.6.18-rc5 kernel, so utilizing DHCP alone will not be 100% on your OS identification, but it will get you into the ball park. And in the case of Gentoo builds may give you a very good idea!

Other Linux Distributions using Option 51 and 57

Here we add a little new twist. We've seen how the Option 55 data will be the same across multiple distributions of Linux boxes that are all based off the same underlying build, well some of the other Options that they may provide are Option 51, the Lease time that the Client would like to request, and Option 57, telling the DHCP Server the size of DHCP packets that it can accept.

PHLAK 0.1	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
PHLAK 0.2	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
PHLAK 0.3	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
Knoppix 3.6	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
Knoppix 3.7	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
Knoppix 3.8	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
Knoppix 3.9	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
Knoppix 4.0	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
Knoppix STD 0.1	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
Operator 3.3	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
Auditor 20060502	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
Kanotix 2005-01	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
Kanotix 2005-02	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
Kanotix 2005-03	16	1,3,6,15,28,12,7,9,42,48,49	43200	548
Red Hat 6.2	64	1,3,6,15,28,12,7,9,42,48,49	43200	548

As you can see here PHLAK, Knoppix 3.6 – 4.0, Knoppix STD 0.1, Operator 3.3, Auditor 20060502, Kanotix 2005-01 through 03 are all using the same underlying DHCP stack. At first glance at the Option 55 parameter list we may assume that Red Hat 6.2 is also, and it may be, but it also has a little twist to it. Instead of a TTL of 16, it has a TTL of 64 on its DHCP packets.

All of these OS's are requesting a DHCP Lease Time of 43200 seconds (720 mins or 12 hours). They are also telling the DHCP Server that they can accept DHCP messages up to a size of 548, instead of the default of 576. Not sure how this is supposed to help them since according to the RFC it should NEVER be less than 576. There has to be something here, but, unknown at this time.

As you can see below with a newer version, of the above OS's, the Option 55 data changed:

Knoppix 5.0	16	1,3,6,15,28,12,7,9,42,48,49,26	43200	548
Knoppix 5.1	16	1,3,6,15,28,12,7,9,42,48,49,26	43200	548
Kanotix 2005-04	16	1,3,6,15,28,12,7,9,42,48,49,26	43200	548
Kanotix 2006-01 RC4	16	1,3,6,15,28,12,7,9,42,48,49,26	43200	548
Helix 1.8	16	1,3,6,15,28,12,7,9,42,48,49,26	43200	548
Insert 1.3.8	16	1,3,6,15,28,12,7,9,42,48,49,26	43200	548

Late in 2005 (guessing based on Kanotix) the underlying Knoppix stuff must have been updated. Helix and Insert distributions seem to be based on Knoppix based on the Option 55 parameter list. Not sure if that is accurate without more digging into them, but it seems feasible from what we've seen so far.

Anyway, they started requesting Option 26, MTU for the NIC interface.

Beyond Option 55 – how we can track a few other things

Option 61

Option 61 Client Identifier is used as a unique identifier to link a Client to its lease. Typically this will be the Clients MAC, but in cases of RRAS Servers it is a bit of other “stuff” and then its MAC address. This makes identify MS RRAS Servers fairly trivial from their DHCP packets.

```

Option: (t=61,l=7) Client identifier
  Option: (61) Client identifier
  Length: 7
  Value: 010060979B644D
  Hardware type: Ethernet
  Client MAC address: 3com_9b:64:4d (00:60:97:9b:64:4d)

```

This one is a non RRAS machine, a typical Client you’d see on the network. Value of its data is: 01 + MAC

```

Option: (t=61,l=17) Client identifier
  Option: (61) Client identifier
  Length: 17
  Value: 015241532000080DBE58DA0000000000000

```

.10	00 00 00 00 00 00 63 82	53 63 35 01 03 4d 0e 52C. Sc5..M.R
.20	52 41 53 2e 4d 69 63 72	6f 73 6f 66 74 3d 11 01	RAS.Micr osoft=..
.30	52 41 53 20 00 08 0d be	58 da 00 00 00 00 00 00	RAS X.....

This one is a MS RRAS Server. Value of its data is: 01 + RAS + ‘ ‘ + MAC + 000000000000

```

Option: (t=61,l=27) Client identifier
  Option: (61) Client identifier
  Length: 27
  Value: 00636973636F2D303030312E393661342E363734302D564C...
Option: (t=55,l=7) Parameter Request List
Option: (t=60,l=9) vendor class identifier = "docsis1.0"
Option: (t=52,l=1) option overload = Boot file and server host names hold options
End option
Padding

```

.20	01 39 02 04 80 3d 1b 00	63 69 73 63 6f 2d 30 30	.9...=. cisco-00
.30	30 31 2e 39 36 61 34 2e	36 37 34 30 2d 56 4c 35	01.96a4. 6740-VL5
.40	38 30 37 07 01 42 06 03	43 0c 96 3c 09 64 6f 63	807..B.. C..<.doc
.50	73 69 73 31 2e 30 34 01	03 ff 00 00 00 00 00 00	sis1.04.

This last one is of a Cisco 2900 Catalyst XL device. We also see its MAC and the VLAN that it is sitting on.

Option 77

Option 77 is User Class Information. This field can be used in quite a few ways, but a default will give you this on a RRAS Server from Microsoft. It is typically used to help identify a type of computer or category of a user. This is a user tweakable setting on the Client machine, so while semi useful for Fingerprinting, the fact that it is changeable from the clients end has to be taken into account.

```
Option: (t=77,l=14) User Class Information
Option: (77) User Class Information
Length: 14
Value: 525241532E4D6963726F736F6674
00 00 00 00 00 00 63 82 53 63 35 01 03 4d 0e 52 .....C. SC5..M.R
52 41 53 2e 4d 69 63 72 6f 73 6f 66 74 3d 11 01 RAS.Micr osoft=..
```


PXE Boot and what we can Learn

Ok, you may be thinking there is nothing specific to learn from PXE boots beside the fact that there is a machine out there trying to do a network boot. We'll that is what I thought originally too until I stumbled across 2 different PXE boots.

Back in the introduction to all of the RFC's I mentioned:

RFC 4578 – Provides some info on PXEboot. It was written recently in November 2006. It can be found here: <http://www.faqs.org/rfcs/rfc4578.html> Interesting things to look at here will be section 2.1 which will help provide information about the underlying hardware platform.

Well lets look at what this actually provides us with:

```

+ Internet Protocol, Src: 0.0.0.0 (0.0.0.0), Dst: 255.255.255.255 (255.255.255.255)
+ User Datagram Protocol, Src Port: bootpc (68), Dst Port: bootps (67)
- Bootstrap Protocol
  Message type: Boot Request (1)
  Hardware type: Ethernet
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x2ac64089
  Seconds elapsed: 4
+ Bootp flags: 0x8000 (Broadcast)
  Client IP address: 0.0.0.0 (0.0.0.0)
  Your (client) IP address: 0.0.0.0 (0.0.0.0)
  Next server IP address: 0.0.0.0 (0.0.0.0)
  Relay agent IP address: 0.0.0.0 (0.0.0.0)
  Client MAC address: vmware_c6:40:89 (00:0c:29:c6:40:89)
  Server host name not given
  Boot file name not given
  Magic cookie: (OK)
+ Option: (t=53,l=1) DHCP Message Type = DHCP Discover
+ Option: (t=55,l=24) Parameter Request List
+ Option: (t=57,l=2) Maximum DHCP Message Size = 1260
+ Option: (t=97,l=17) UUID/GUID-based Client Identifier
+ Option: (t=93,l=2) Client System Architecture = IA x86 PC
+ Option: (t=94,l=3) Client Network Device Interface
+ Option: (t=60,l=32) vendor class identifier = "PXEClient:Arch:00000:UNDI:002001"

```

Option 93 – Client System Architecture

- 0 Intel x86PC
- 1 NEC/PC98
- 2 EFI Itanium
- 3 DEC Alpha
- 4 Arc x86
- 5 Intel Lean Client
- 6 EFI IA32

- 7 EFI BC
- 8 EFI Xscale
- 9 EFI x86-64

So by looking at Option 93 we can determine what the underlying Hardware is. May be useful, may not, time will tell.

Option 94 – Client Network Device Interface

- 2.00 LANDesk service agent boot ROMs. No PXE APIs.
- 2.00 First generation PXE boot ROMs. (PXENV+)
- 2.01 Second generation PXE boot ROMs. (!PXE)
- 3.00 32/64-bit UNDI specification. (Alpha)
- 3.10 32/64-bit UNDI specification. (Beta) First generation EFI runtime driver support.
- 3.20 32/64-bit UNDI specification. (Release) Second generation EFI runtime driver support.

With Option 94 we can now see what PXE compliance level the NIC is. VMWare uses 2.01.

Utilizing Lease Information

What happens when a lease expires

One thing that needs more looking into is how many OS's actually renew their IP addresses when the lease expires and which ones just wait until the next time they reboot.

From my testing it did not look like the majority of Linux builds renewed their leases if they were currently up and running, only requesting new ones the next time they rebooted. To utilize this information for identification purposes would be hard. You would need to track individual machines upon bootup, knowing how long the lease is for and then watching them once 50% of the lease time was up and see if they try to do a DHCP Request to renew their lease. This may be feasible on a network with a short lease time, but not as practical on networks with long leases. Granted the use of any passive OS identification program or utility is typically a long term project and not a 20 minute deal anyway!

Here is a list of OS's that, appear to just continue to use the IP Address, this may have been because of the short lease times I gave them (10 minutes), or it may have been for other reasons:

- Auditor 20060502
- Backtrack 1.0
- Backtrack 2.0
- Helix 1.8
- Insert 1.3.8
- Kanotix 2005-01
- Kanotix 2005-02
- Kanotix 2005-03
- Kanotix 2005-04
- Kanotix 2006-01 RC4
- Knoppix 3.6
- Knoppix 3.7
- Knoppix 3.8
- Knoppix 3.9
- Knoppix 4.0
- Knoppix 5.0
- Knoppix 5.1
- Knoppix STD 0.1
- PHLAK 0.1
- PHLAK 0.2
- PHLAK 0.3
- Red Hat 6.2
- Ubuntu - Gnome 2.10
- Windows 95
- Windows 98

Here are a list known to do DHCP Renewal Requests (DHCP Request) while they currently have an existing IP address:

- Cent OS 4
- Fedora Core 3
- Fedora Core 4
- Fedora Core 5
- Fedora Core 6
- Freespire 1.0
- Freespire 2.0
- Gentoo 2005.0
- Gentoo 2006.0
- Linspire 4.5
- Linspire 5.0
- PC BSD 1.2
- PC BSD 1.3
- PC Linux
- Slax 5.1.8
- Ubuntu 5.10
- Ubuntu 6.10
- Windows 2000
- Windows 98 SE
- Windows ME
- Windows NT 4
- Windows Vista
- Windows XP

Here is a list of OS's I'm unsure of due to the fact they didn't like my cobbled together DHCP Server (or I just didn't pay enough attention to see if they every did):

- Anonym.os 1.0
- Gentoo 2006.1
- Open BSD 3.8
- SLED 10
- SUSE Linux 10.1

Appendix A

DHCP Options

Straight copy of: <http://www.iana.org/assignments/bootp-dhcp-parameters>, but it is always nice to have in a piece of reference material so you don't have to always go out and look it up.

Tag	Name	Length	Meaning	Reference
----	----	-----	-----	-----
0	Pad	0	None	[RFC2132]
1	Subnet Mask	4	Subnet Mask Value	[RFC2132]
2	Time Offset	4	Time Offset in Seconds from UTC	[RFC2132]
3	Router	N	N/4 Router addresses	[RFC2132]
4	Time Server	N	N/4 Timeserver addresses	[RFC2132]
5	Name Server	N	N/4 IEN-116 Server addresses	[RFC2132]
6	Domain Server	N	N/4 DNS Server addresses	[RFC2132]
7	Log Server	N	N/4 Logging Server addresses	[RFC2132]
8	Quotes Server	N	N/4 Quotes Server addresses	[RFC2132]
9	LPR Server	N	N/4 Printer Server addresses	[RFC2132]
10	Impress Server	N	N/4 Impress Server addresses	[RFC2132]
11	RLP Server	N	N/4 RLP Server addresses	[RFC2132]
12	Hostname	N	Hostname string	[RFC2132]
13	Boot File Size	2	Size of boot file in 512 byte chunks	[RFC2132]
14	Merit Dump File	N	Client to dump and name the file to dump it to	[RFC2132]
15	Domain Name	N	The DNS domain name of the client	[RFC2132]
16	Swap Server	N	Swap Server addeess	[RFC2132]
17	Root Path	N	Path name for root disk	[RFC2132]
18	Extension File	N	Path name for more BOOTP info	[RFC2132]
19	Forward On/Off	1	Enable/Disable IP Forwarding	[RFC2132]
20	SrcRte On/Off	1	Enable/Disable Source Routing	[RFC2132]
21	Policy Filter	N	Routing Policy Filters	[RFC2132]
22	Max DG Assembly	2	Max Datagram Reassembly Size	[RFC2132]
23	Default IP TTL	1	Default IP Time to Live	[RFC2132]
24	MTU Timeout	4	Path MTU Aging Timeout	[RFC2132]
25	MTU Plateau	N	Path MTU Plateau Table	[RFC2132]
26	MTU Interface	2	Interface MTU Size	[RFC2132]
27	MTU Subnet	1	All Subnets are Local	[RFC2132]
28	Broadcast Address	4	Broadcast Address	[RFC2132]
29	Mask Discovery	1	Perform Mask Discovery	[RFC2132]
30	Mask Supplier	1	Provide Mask to Others	[RFC2132]
31	Router Discovery	1	Perform Router Discovery	[RFC2132]
32	Router Request	4	Router Solicitation Address	[RFC2132]
33	Static Route	N	Static Routing Table	[RFC2132]
34	Trailers	1	Trailer Encapsulation	[RFC2132]
35	ARP Timeout	4	ARP Cache Timeout	[RFC2132]
36	Ethernet	1	Ethernet Encapsulation	[RFC2132]
37	Default TCP TTL	1	Default TCP Time to Live	[RFC2132]
38	Keepalive Time	4	TCP Keepalive Interval	[RFC2132]
39	Keepalive Data	1	TCP Keepalive Garbage	[RFC2132]
40	NIS Domain	N	NIS Domain Name	[RFC2132]

41	NIS Servers	N	NIS Server Addresses	[RFC2132]
42	NTP Servers	N	NTP Server Addresses	[RFC2132]
43	Vendor Specific	N	Vendor Specific Information	[RFC2132]
44	NETBIOS Name Srv	N	NETBIOS Name Servers	[RFC2132]
45	NETBIOS Dist Srv	N	NETBIOS Datagram Distribution	[RFC2132]
46	NETBIOS Node Type	1	NETBIOS Node Type	[RFC2132]
47	NETBIOS Scope	N	NETBIOS Scope	[RFC2132]
48	X Window Font	N	X Window Font Server	[RFC2132]
49	X Window Manager	N	X Window Display Manager	[RFC2132]
50	Address Request	4	Requested IP Address	[RFC2132]
51	Address Time	4	IP Address Lease Time	[RFC2132]
52	Overload	1	Overload "sname" or "file"	[RFC2132]
53	DHCP Msg Type	1	DHCP Message Type	[RFC2132]
54	DHCP Server Id	4	DHCP Server Identification	[RFC2132]
55	Parameter List	N	Parameter Request List	[RFC2132]
56	DHCP Message	N	DHCP Error Message	[RFC2132]
57	DHCP Max Msg Size	2	DHCP Maximum Message Size	[RFC2132]
58	Renewal Time	4	DHCP Renewal (T1) Time	[RFC2132]
59	Rebinding Time	4	DHCP Rebinding (T2) Time	[RFC2132]
60	Class Id	N	Class Identifier	[RFC2132]
61	Client Id	N	Client Identifier	[RFC2132]
62	Netware/IP Domain	N	Netware/IP Domain Name	[RFC2242]
63	Netware/IP Option	N	Netware/IP sub Options	[RFC2242]
64	NIS-Domain-Name	N	NIS+ v3 Client Domain Name	[RFC2132]
65	NIS-Server-Addr	N	NIS+ v3 Server Addresses	[RFC2132]
66	Server-Name	N	TFTP Server Name	[RFC2132]
67	Bootfile-Name	N	Boot File Name	[RFC2132]
68	Home-Agent-Addrs	N	Home Agent Addresses	[RFC2132]
69	SMTP-Server	N	Simple Mail Server Addresses	[RFC2132]
70	POP3-Server	N	Post Office Server Addresses	[RFC2132]
71	NNTP-Server	N	Network News Server Addresses	[RFC2132]
72	WWW-Server	N	WWW Server Addresses	[RFC2132]
73	Finger-Server	N	Finger Server Addresses	[RFC2132]
74	IRC-Server	N	Chat Server Addresses	[RFC2132]
75	StreetTalk-Server	N	StreetTalk Server Addresses	[RFC2132]
76	STDA-Server	N	ST Directory Assist. Addresses	[RFC2132]
77	User-Class	N	User Class Information	[RFC3004]
78	Directory Agent	N	directory agent information	[RFC2610]
79	Service Scope	N	service location agent scope	[RFC2610]
80	Rapid Commit	0	Rapid Commit	[RFC4039]
81	Client FQDN	N	Fully Qualified Domain Name	[RFC4702]
82	Relay Agent Information	N	Relay Agent Information	[RFC3046]
83	iSNS	N	Internet Storage Name Service	[RFC4174]
84	REMOVED/Unassigned			[RFC3679]
85	NDS Servers	N	Novell Directory Services	[RFC2241]
86	NDS Tree Name	N	Novell Directory Services	[RFC2241]
87	NDS Context	N	Novell Directory Services	[RFC2241]
88	BCMCS Controller Domain Name list			[RFC4280]
89	BCMCS Controller IPv4 address option			[RFC4280]
90	Authentication	N	Authentication	[RFC3118]
91	client-last-transaction-time option			[RFC4388]
92	associated-ip option			[RFC4388]
93	Client System	N	Client System Architecture	[RFC4578]
94	Client NDI	N	Client Network Device Interface	[RFC4578]
95	LDAP	N	Lightweight Directory Access Protocol	[RFC3679]
96	REMOVED/Unassigned			[RFC3679]
97	UUID/GUID	N	UUID/GUID-based Client Identifier	[RFC4578]
98	User-Auth	N	Open Group's User Authentication	[RFC2485]

99	GEOCONF_CIVIC			[RFC4776]
100	REMOVED/Unassigned			[RFC3679]
101	REMOVED/Unassigned			[RFC3679]
102-107	REMOVED/Unassigned			[RFC3679]
108	REMOVED/Unassigned			[RFC3679]
109	Unassigned			[RFC3679]
110	REMOVED/Unassigned			[RFC3679]
111	Unassigned			[RFC3679]
112	Netinfo Address	N	NetInfo Parent Server Address	[RFC3679]
113	Netinfo Tag	N	NetInfo Parent Server Tag	[RFC3679]
114	URL	N	URL	[RFC3679]
115	REMOVED/Unassigned			[RFC3679]
116	Auto-Config	N	DHCP Auto-Configuration	[RFC2563]
117	Name Service Search	N	Name Service Search	[RFC2937]
118	Subnet Selection Option	4	Subnet Selection Option	[RFC3011]
119	Domain Search	N	DNS domain search list	[RFC3397]
120	SIP Servers DHCP Option	N	SIP Servers DHCP Option	[RFC3361]
121	Classless Static Route Option	N	Classless Static Route Option	[RFC3442]
122	CCC	N	CableLabs Client Configuration	[RFC3495]
123	GeoConf Option	16	GeoConf Option	[RFC3825]
124	V-I Vendor Class		Vendor-Identifying Vendor Class	[RFC3925]
125	V-I Vendor-Specific Information		Vendor-Identifying Vendor-Specific Information	[RFC3925]
126	Removed/Unassigned			[RFC3679]
127	Removed/Unassigned			[RFC3679]
128	PXE - undefined (vendor specific)			[RFC4578]
128	Etherboot signature. 6 bytes: E4:45:74:68:00:00			
128	DOCSIS "full security" server IP address			
128	TFTP Server IP address (for IP Phone software load)			
129	PXE - undefined (vendor specific)			[RFC4578]
129	Kernel options. Variable length string			
129	Call Server IP address			
130	PXE - undefined (vendor specific)			[RFC4578]
130	Ethernet interface. Variable length string.			
130	Discrimination string (to identify vendor)			
131	PXE - undefined (vendor specific)			[RFC4578]
131	Remote statistics server IP address			
132	PXE - undefined (vendor specific)			[RFC4578]
132	IEEE 802.1Q VLAN ID			
133	PXE - undefined (vendor specific)			[RFC4578]
133	IEEE 802.1D/p Layer 2 Priority			
134	PXE - undefined (vendor specific)			[RFC4578]
134	Diffserv Code Point (DSCP) for VoIP signalling and media streams			
135	PXE - undefined (vendor specific)			[RFC4578]
135	HTTP Proxy for phone-specific applications			
136-149	Unassigned			[RFC3942]
150	TFTP server address (Tentatively Assigned - 23 Jun 2005)			
150	Etherboot			
150	GRUB configuration path name			
151-174	Unassigned			[RFC3942]
175	Etherboot (Tentatively Assigned - 23 Jun 2005)			
176	IP Telephone (Tentatively Assigned - 23 Jun 2005)			
177	Etherboot (Tentatively Assigned - 23 Jun 2005)			
177	PacketCable and CableHome (replaced by 122)			
178-207	Unassigned			[RFC3942]
208	pxelinux.magic (string) = F1:00:74:7E (241.0.116.126) (Tentatively Assigned - 23 Jun 2005)			

209	pxelinux.configfile (text) (Tentatively Assigned - 23 Jun 2005)		
210	pxelinux.pathprefix (text) (Tentatively Assigned - 23 Jun 2005)		
211	pxelinux.reboottime (unsigned integer 32 bits) (Tentatively Assigned - 23 Jun 2005)		
212-219	Unassigned		
220	Subnet Allocation Option (Tentatively Assigned - 23 Jun 2005)		
221	Virtual Subnet Selection Option (Tentatively Assigned - 23 Jun 2005)		
222-223	Unassigned		[RFC3942]
224-254	Private Use		
255	End	0	None [RFC2132]

DHCP Message Type 53 Values - per [RFC2132]

Registration Procedures:

Value	Message Type	Reference
-----	-----	-----
1	DHCPDISCOVER	[RFC2132]
2	DHCPOFFER	[RFC2132]
3	DHCPREQUEST	[RFC2132]
4	DHCPDECLINE	[RFC2132]
5	DHCPACK	[RFC2132]
6	DHCPNAK	[RFC2132]
7	DHCPRELEASE	[RFC2132]
8	DHCPINFORM	[RFC2132]
9	DHCPFORCERENEW	[RFC3203]
10	DHCPLEASEQUERY	[RFC4388]
11	DHCPLEASEUNASSIGNED	[RFC4388]
12	DHCPLEASEUNKNOWN	[RFC4388]
13	DHCPLEASEACTIVE	[RFC4388]

The only ones I've captured in the wild are Values 1-8. The rest have fairly recently been added and I assume are going to be directed at the Client directly via IP, not a broadcast message for everyone to see.