Dynamic Host Configuration Working Group
Internet-Draft
Intended status: Informational
Expires: April 3, 2012

D. Hankins
Google

October 1, 2011

Guidelines for Creating New DHCP Options
draft-ietf-dhc-option-guidelines-07

Abstract

This document seeks to provide guidance to prospective DHCP Option authors, to help them in producing option formats that are easily adoptable.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

Most protocol developers ask themselves if a protocol will work, or work efficiently. These are important questions, but another less frequently considered question is whether the proposed protocol presents itself needless barriers to adoption by deployed software.

DHCPv4 [RFC2131] and DHCPv6 [RFC3315] software implementors are not merely faced with the task of a given option's format on the wire. The option must "fit" into every stage of the system's process, from the user interface where configuration is entered to the machine interfaces where configuration is consumed. To help understand the potential implementation challenges of any new DHCP Option, one implementation's approach to tackling DHCP Option formats (Appendix A) has been included as an Appendix.

Another more frequently overlooked aspect of rapid adoption is the question: Would the option would require operators to be intimately familiar with the option's internal format in order to make use of it? Most DHCP software provides a facility for "unknown options" at the time of publication to be configured by hand by an operator. But if doing so requires extensive reading (more than can be covered in a simple FAQ for example), it inhibits adoption.

So although a given solution would work, and might even be space, time, or aesthetically optimal, a given option is presented with a series of ever-worsening challenges to be adopted;

o If it doesn't fit neatly into existing config files.

o If it requries new source code changes to be adopted, and hence upgrades of deployed software.

o If it does not share its deployment fate in a general manner with other options, standing alone in requiring code changes or reworking configuration file syntaxes.

There are many things DHCP option authors can do to form DHCP Options to stay off this list entirely, or failing that, to make software implementors lives easier and improve its chances for widespread adoption.

2. When to Use DHCP

Principally, DHCP carries configuration parameters for its clients. Any knob, dial, slider, or checkbox on the client system, such as "my domain name servers", "my hostname", or even "my shutdown

temperature" are candidates for being configured by DHCP.

The presence of such a knob isn't enough, because DHCP also presents the extension of an administrative domain - the operator of the network to which the client is currently attached. Someone runs not only the local switching network infrastructure that the client is directly (or wirelessly) attached to, but the various methods of accessing the external Internet via local assist services that network must also provide (such as domain name servers, or routers). This means that in addition to the existence of a configuration parameter, one must also ask themselves if it is reasonable for this parameter to be set by the directly attached network's administrators.

Bear in mind that the client still reserves the right to ignore values received via DHCP (for example, due to having a value manually configured by its own operator), and that at least one main use case for DHCP is the corporate enterprise. So even if the local Net Cafe's operator is not a likely source of the candidate configuration, there may be other DHCP servers in a client's lifetime which are.

3. General Principles

The primary principle to follow in order to enhance an option's adoptability is certainly simplification. But more specifically, to create the option in such a way that it should not require any new or special case software to support. If old software currently deployed and in the field can adopt the option through supplied configuration conveniences then it's fairly well assured that new software can easily formally adopt it.

There are at least two classes of DHCP options: A bulk class of options which are provided explicitly to carry data from one side of the DHCP exchange to the other (such as nameservers, domain names, or time servers), and a protocol class of options which require special processing on the part of the DHCP software or are used during special processing (such as the FQDN options ([RFC4702], [RFC4704]), DHCPv4 message type option [RFC2132], link selection options ([RFC3011], [RFC3527]), and so forth; these options carry data that is the result of a routine in some DHCP software).

The guidelines laid out here should be understood to be relaxed for the protocol class of options. Wherever special-case-code is already required to adopt the DHCP option, it is substantially more reasonable to format the option in a less generic fashion, if there are measurable benefits to doing so.

4. Reusing Other Options

In DHCPv4, there are now nearly one hundred and thirty options, at least as IETF standards, which might be used as an example. There is also one handy document [RFC2132] containing many option definitions.

There is a tradeoff between the adoptability of previously defined option formats, and the advantages new or specialized formats can provide. In the balance, it is usually preferrable to reuse previously used option formats.

However, it isn't very practical to consider the bulk of DHCP options already allocated, and consider which of those solve a similar problem. So, the following list of common option format fragments is provided as a shorthand. Please note that it is not complete in terms of exampling every option format ever devised...it is only a list of option format fragments which are used in two or more options.

Fragment	Size	Types of Uses
ipv4-address	4	Default gateway, requested address, subnet mask [RFC2132], addresses of servers ([RFC2132], [RFC2241], [RFC2242], [RFC3495], [RFC3634], [RFC4174]), as a component in a list of routes [RFC3442].
ipv6-address	16	DHCPv6 server unicast address [RFC3315], addresses of servers ([RFC3319], [RFC3646], [RFC3898], [RFC4075], [RFC4280]).
32-bit integer	4	Signed or unsigned varieties. Used for timezone time offset [RFC2132] (deprecated by [RFC4833]). Other uses for host configuration values such as path MTU aging timeouts, ARP cache timeouts, TCP keepalive intervals [RFC2132]. Also used by the DHCPv4 protocol for relative times, and times since epoch.
16-bit integer	2	Client configuration parameters, such as MTU, maximum datagram reassembly limits, the DHCPv4 maximum message size [RFC2132], or the elapsed time option [RFC3315] in DHCPv6.

8-bit integer	1	Used for host configuration parameters, such as the default IP TTL, default TCP TTL, NetBIOS node type [RFC2132]. Also used for protocol features, such as the DHCPv4 Option Overload (as flags), DHCP Message Type (as an enumeration) or DHCPv6 Preference [RFC3315].
NVT-ASCII Text	unlim	This is the kitchen sink of common fragments. Common uses are for filenames (such as TFTP paths), host or domain names (but this should be discouraged), or protocol features such as textual messages such as verbose error indicators. Since the size of this format cannot be determined (it is not NULL terminated), it consumes any remaining space in the option.
DNS Wire Format Domain Name List [RFC1035]	unlim	Presently used for 'domain search' lists in both DHCPv4 [RFC3397] and DHCPv6 [RFC3646], but also used in DHCPv6 for any host or domain name. A field formatted this way may have a determinate length if the number of root labels is limited, but use of this format as being a determinate length should be discouraged in DHCPv4, less so in DHCPv6.
'suboption' encapsulation	unlim	The Relay Agent Information Option [RFC3046], vendor options [RFC2132], Vendor Identified Vendor SubOptions ([RFC3925], [RFC3315]). Commonly used for situations where the full format cannot be known initially, such as where there seems to be some room for later protocol work to expand the amount of information carried, or where the full extent of data carried is defined in a private specification (such as with vendor options). Encapsulations do not use 'PAD' and 'END' options in DHCPv4, and there are no such options in DHCPv6, so this format also is of indeterminate length.

Table 1: Common Option Fragments

The easiest approach to manufacturing trivially deployable DHCP Options is to assemble the option out of whatever common fragments fit - possibly allowing a group of fragments to repeat to fill the

remaining space (if present) and so provide multiple values. Place all fixed size values at the start of the option, and any variable/ indeterminate sized value at the tail end of the option.

This estimates that implementations will be able to reuse code paths designed to support the other options.

5. Avoid Conditional Formatting

Placing a octet at the start of the option which informs the software how to process the remaining octets of the option may appear simple to the casual observer. But the only conditional formatting methods that are in widespread use today are 'protocol' class options. So conditional formatting requires new code to be written, as well as introduces an implementation problem; as it requires that all speakers implement all current and future conditional formats.

Conditional formatting is absolutely not recommended, except in cases where the DHCP option has already been deployed experimentally, and all but one conditional format is deprecated.

6. Avoid Aliasing

Options are said to be aliases of each other if they provide input to the same configuration parameter. A commonly proposed example is to configure the location of some new service ("my foo server") using a binary IP address, a domain name field, and a URL. This kind of aliasing is undesirable, and is best avoided.

In this case, where three different formats are supposed, it triples the work of the software involved, requiring support for not merely one format, but support to produce and digest all three. Since clients cannot predict what values the server will provide, they must request all formats...so in the case where the server is configured with all formats, DHCP option space is wasted on option contents that are redundant.

It also becomes unclear which types of values are mandatory, and how configuring some of the options may influence the others. For example, if an operator configures the URL only, should the server synthesize a domain name and IP address?

A single configuration value on a host is probably presented to the operator (or other software on the machine) in a single field or channel. If that channel has a natural format, then any alternative formats merely make more work for intervening software in providing

conversions.

So the best advice is to choose the one method that best fulfills the requirements, be that for simplicity (such as with an IP address and port pair), late binding (such as with DNS), or completeness (such as with a URL).

On the specific subject of desiring to configure a value using a Fully Qualified Domain Name instead of a binary IP address, note that most DHCP server implementations will happily accept a Domain Name entered by the administrator, and use DNS resolution to render binary IP addresses in DHCP replies to clients. Consequently, consider the extra packet overhead incurred on the client's end to perform DNS resolution itself. The client may be operating on a battery and packet transmission is a non-trivial use of power, and the extra RTT delays the client must endure before the service is configured are at least two factors to consider in making a decision on format.

7. Considerations for Creating New Formats

If the option simply will not fit into any existing work by using fragments, the last recourse is to create a new format to fit.

When doing so, it is not enough to gauge whether or not the option format will work in the context of the option presently being considered. It is equally important to consider if the new format's fragments might reasonably have any other uses, and if so, to create the option with the foreknowledge that its parts may later become a common fragment.

One specific consideration to evaluate is whether or not options of a similar format would need to have multiple or single values encoded (whatever differs from the current option), and how that might be accomplished in a similar format.

8. The Dangers of Sub Options

Some DHCP options, such as the DHCPv4 Relay Agent Information Option [RFC3046] are defined to contain a series of DHCP options, possibly using code tags specific to that option (but not in some limited "protocol feature" cases in DHCPv6 [RFC3315]). These are commonly referred to as Encapsulated Option Spaces or more simply, Sub Options.

Sub options seem very attractive, because they allow the encoding of multiple variable length fields within the single "parent" option.

However, DHCP software will only include these options on an "all or nothing" basis, there is no well deployed mechanism for "Sub Option Parameter Request Lists" (although some defined sub-option spaces, such as for DOCSIS, do describe sub-option scoped PRL analogues), and the software will not include the entire option if there is not sufficient space.

Consequently, it is not advisable to group options that may not be requested at the same time by the same client under an encapsulated space.

Another attraction sub options present is ease of extending the configuration value for later, related configuration. This must be weighed against the cost associated with asking IANA to maintain the space's internally assigned option codes. Generally, the cost to IANA is greater, as it is unlikely that options will be later extended.

The use of sub-options is not a solution to aliasing problems. Sub-options that contain multiple configuration values that alias the same configuration element actually makes matters worse. The only solution to aliasing problems is to select a single option format, or where that is literally impossible, to use multiple DHCP options. In this way, clients may place only the options they support on their parameter request list, in the order they support them. Later protocol maintenance may incorporate a means to select a single DHCP option code out of a list of aliased options, so do not concern yourself with packet space issues arising from receiving all the aliases.

Additionally, DHCPv4 option concatenation (Section 9) has not been defined in any DHCPv4 sub-options space. Currently there is some DHCP software which does concatenate multiple DHCP options present in a sub-option space. There is also software that treats multiple DHCP option codes present in a sub-option as individual single options. So there is no reliably predictable default behaviour.

Because no sub-options space has yet been defined that includes the possibility of having more than one instance of an option of the same code, any attempt to do so is discouraged.

9. Option Size

DHCPv4 [RFC2131] options payload space is limited, as there are a number of unaddressed deployment problems with DHCPv4 packet sizes. The end result is that you should build your option to the assumption that the packet will be no larger than 576 octets. This means that

the options payload space will be 312 octets, which you will have to share with other options. This space can be extended by making use of Option Overloading [RFC2132], which allows the use of the BOOTP FILE and SNAME header fields for carrying DHCPv4 options (adding 192 octets), but these header fields will not be available for overloading if they have been configured to carry a value.

DHCPv6 [RFC3315] is much better off. First, through its use of link-local addresses, it steps aside many of the deployment problems that plague DHCPv4, and looks a great deal more like any other UDP based application; oblivious to packet sizes up to 64KB. Second, RFC 3315 explicitly refers readers to RFC 2460 Section 5, which describes an MTU of 1280 octets and a minimum fragment reassembly of 1500 octets. It's much more feasible to suggest that DHCPv6 is capable of having larger options deployed over it, and at least no common upper limit is yet known to have been encoded by its implementors. It is impossible to describe any fixed limit that cleanly divides those too big from the workable.

So in either protocol, it is advantageous to prefer option formats which contain the desired information in the smallest form factor that solves the requirements. One example is to use a 4-octet IPv4 address rather than a fully qualified domain name, because many DHCP servers will perform DNS resolution on configured FQDN's (so the DNS recursive lookup is performed anyway). There may be motivations to use the fully qualified domain name anyway, such as if the intended RRSET is not an address, or if the client must refresh the name more frequently than common lease renewal periods.

When it is not possible to compress the configuration contents either because of the simple size of the parameters, or because it is expected that very large configurations are valid, it may be preferable to use a second stage configuration. Some examples of this are to provide TFTP server and pathnames, or a URL, which the client will load and process externally to the DHCP protocol.

The DHCPv4 code and length tags are each a single octet. As the length field describes the length of the DHCP option's contents (not including the code and length octets), any option whose contents' length exceeds 255 octets can not be contained in a single option. These 'long options' will simply be fragmented into multiple options within the packet. DHCP software processing these fragments will concatenate them, in the order they appear as defined by [RFC2131], prior to evaluating their contents. This is an important distinction that is sometimes overlooked by authors - these multiple options are not individually formatted to convey one unit of information precisely as you have defined, but rather one option that has been split along any arbitrary octet boundary into multiple containers.

When documenting an example, then, try to make sure that the division point you select as an example does not lie on a clean division of your option contents - place it at an offset so as to reinforce that these values must be concatenated rather than processed individually.

DHCPv4 option fragments are a basic protocol feature, so there usually is no reason to mention this feature in new option definitions, and no requirement for every option definition to be presented as a series of fragments. It is only recommended to reinforce the existence of DHCP option fragmentation when the potential for large options is likely. In this case, try to choose a large example data value.

Note that option fragmentation is also a very common side-effect of running out of options space, and moving to overloaded FILE or SNAME fields. Although the option may be considerably shorter than 255 octets, if it does not fit in the remaining space then software may consume all remaining options space with one option fragment, and place the remainder in an overloaded field.

Primarily it is important to remember that DHCPv4 differs from DHCPv6 on this point: DHCPv4 can only convey one option of a given option code at any time - additional options (or possibly sub options, which do not have concisely defined semantics) of the same code will be concatenated together and processed at once. DHCPv6 does allow multiple instances of a given option, and they are treated as distinct values following the defined format, however this feature is generally preferred to be restricted to protocol class features (such as the IA_* series of options); it is better to define your option as an array if it is possible.

So remember that it is out of the question to define a case for multiple instances of your option in DHCPv4, and it is recommended to clarify (with normative language) if any DHCPv6 option may appear once or multiple times.

10. Clients Request their Options

The DHCPv4 Parameter Request List [RFC2132], and the DHCPv6 Option Request Option (OPTION_ORO) [RFC3315], are both options that serve two purposes - to inform the server what options the client supports and is willing to digest, and in what order of priority the client places those option contents (in the event that they will not fit in the packet, later options are to be dropped).

It doesn't make sense for some options to appear on this Parameter Request List, such as those formed by elements of the protocol's

internal workings, or are formed on either end by DHCP-level software engaged in some exchange of information. When in any form of doubt, assume that any new option must be present on the relevant option request list if the client desires it.

It is a frequent mistake of option draft authors, then, to create text that implies that a server will simply provide the new option, and clients will digest it. Generally, it's best to also specify that clients MUST place the new option code on the relevant list option, clients MAY include the new option in their packets to servers with hints as to values they desire, and servers MAY respond with the option contents (if they have been so configured).

Under only the most dire of circumstances should a new option definition require special ordering of options either in the relevant request option, or in the order of options within the packet. Although the request option does imply a priority, which might be processed in order, a server may shuffle options around in a DHCPv4 packet in order to make them fit, and server software may sort DHCPv6 options into strange orders. There is not one universal approach.

11. Security Considerations

DHCP does have an Authentication mechanism ([RFC3118], [RFC3315], [RFC4030]), where it is possible for DHCP software to discriminate between authentic endpoints and men in the middle.

However, at this date the mechanism is poorly deployed. It also does not provide end-to-end encryption.

So, while creating a new option, bear in mind that DHCP packet contents are always transmitted in the clear, and actual production use of the software will probably be vulnerable at least to man-in-the-middle attacks from within the network, even where the network itself is protected from external attacks by firewalls. In particular, some DHCP message exchanges are transmitted to broadcast or multicast addresses that are likely broadcast anyway.

If an option is of a specific fixed length, it is useful to remind the implementer of the option data's full length. This is easily done by declaring the specific value of the 'length' tag of the option. This helps to gently remind implementers to validate option length before digesting them into likewise fixed length regions of memory or stack.

If an option may be of variable size (such as having indeterminate length fields, such as domain names or text strings), it is advisable

to explicitly remind the implementor to be aware of the potential for long options. Either define a reasonable upper limit (and suggest validating it), or explicitly remind the implementor that an option may be exceptionally long (to be prepared to handle errors rather than truncate values).

For some option contents, "insane values" may be used to breach security. An IP address field might be made to carry a loopback address, or local broadcast address, and depending on the protocol this may lead to undesirable results. A domain name field may be filled with contrived contents that exceed the limitations placed upon domain name formatting...as this value is possibly delivered to "internal configuration" records of the system, it may be trusted, rather than validated.

So it behooves an option's definition to contain any validation measures as can reasonably be made.

12. IANA Considerations

This document has no actions for IANA.

13. Informative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.

[RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, March 1997.

[RFC2241] Provan, D., "DHCP Options for Novell Directory Services", RFC 2241, November 1997.

[RFC2242] Droms, R. and K. Fong, "NetWare/IP Domain Name and Information", RFC 2242, November 1997.

[RFC3011] Waters, G., "The IPv4 Subnet Selection Option for DHCP", RFC 3011, November 2000.

[RFC3046] Patrick, M., "DHCP Relay Agent Information Option", RFC 3046, January 2001.

[RFC3118] Droms, R. and W. Arbaugh, "Authentication for DHCP

Messages", RFC 3118, June 2001.

[RFC3315] Droms, R., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, July 2003.

[RFC3319] Schulzrinne, H. and B. Volz, "Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol (SIP) Servers", RFC 3319, July 2003.

[RFC3397] Aboba, B. and S. Cheshire, "Dynamic Host Configuration Protocol (DHCP) Domain Search Option", RFC 3397, November 2002.

[RFC3442] Lemon, T., Cheshire, S., and B. Volz, "The Classless Static Route Option for Dynamic Host Configuration Protocol (DHCP) version 4", RFC 3442, December 2002.

[RFC3495] Beser, B. and P. Duffy, "Dynamic Host Configuration Protocol (DHCP) Option for CableLabs Client Configuration", RFC 3495, March 2003.

[RFC3527] Kinnear, K., Stapp, M., Johnson, R., and J. Kumarasamy, "Link Selection sub-option for the Relay Agent Information Option for DHCPv4", RFC 3527, April 2003.

[RFC3634] Luehrs, K., Woundy, R., Bevilacqua, J., and N. Davoust, "Key Distribution Center (KDC) Server Address Sub-option for the Dynamic Host Configuration Protocol (DHCP) CableLabs Client Configuration (CCC) Option", RFC 3634, December 2003.

[RFC3646] Droms, R., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3646, December 2003.

[RFC3898] Kalusivalingam, V., "Network Information Service (NIS) Configuration Options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3898, October 2004.

[RFC3925] Littlefield, J., "Vendor-Identifying Vendor Options for Dynamic Host Configuration Protocol version 4 (DHCPv4)", RFC 3925, October 2004.

[RFC3942] Volz, B., "Reclassifying Dynamic Host Configuration Protocol version 4 (DHCPv4) Options", RFC 3942, November 2004.

[RFC4030] Stapp, M. and T. Lemon, "The Authentication Suboption for the Dynamic Host Configuration Protocol (DHCP) Relay Agent Option", RFC 4030, March 2005.

[RFC4075] Kalusivalingam, V., "Simple Network Time Protocol (SNTP) Configuration Option for DHCPv6", RFC 4075, May 2005.

[RFC4174] Monia, C., Tseng, J., and K. Gibbons, "The IPv4 Dynamic Host Configuration Protocol (DHCP) Option for the Internet Storage Name Service", RFC 4174, September 2005.

[RFC4280] Chowdhury, K., Yegani, P., and L. Madour, "Dynamic Host Configuration Protocol (DHCP) Options for Broadcast and Multicast Control Servers", RFC 4280, November 2005.

[RFC4702] Stapp, M., Volz, B., and Y. Rekhter, "The Dynamic Host Configuration Protocol (DHCP) Client Fully Qualified Domain Name (FQDN) Option", RFC 4702, October 2006.

[RFC4704] Volz, B., "The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Client Fully Qualified Domain Name (FQDN) Option", RFC 4704, October 2006.

[RFC4833] Lear, E. and P. Eggert, "Timezone Options for DHCP", RFC 4833, April 2007.

Appendix A. **Background on ISC DHCP**

The ISC DHCP software package was mostly written by Ted Lemon in cooperation with Nominum, Inc. Since then, it has been given to Internet Systems Consortium, Inc. ("ISC") where it has been maintained in the public interest by contributors and ISC employees.

It includes a DHCP Server, Relay, and Client implementation, with a common library of DHCP protocol handling procedures.

The DHCP Client may be found on some Linux distributions, and FreeBSD 5 and earlier. Variations ("Forks") of older versions of the client may be found on several BSDs, including FreeBSD 6 and later.

The DHCP Server implementation is known to be in wide use by many Unix-based servers, and comes pre-installed on most Linux distributions.

The ISC DHCP Software Suite has to allow:

- Administrators to configure arbitrary DHCP Option Wire Formats for options that either were not published at the time the software released, or are of the System Administrator's invention (such as 'Site-Local' [RFC3942] options), or finally were of Vendor design (Vendor Encapsulated Options [RFC2132] or similar).

- Pre-defined names and formats of options allocated by IANA and defined by the IETF Standards body.

- Applications deriving their configuration parameters from values provided by these options to receive and understand their content. Often, the binary format on the wire is not helpful or digestable by, for example, 'ifconfig' or '/etc/resolv.conf'.

So, one can imagine that this would require a number of software functions:

1. To read operator-written configuration value into memory.

2. To write the in-memory representation into protocol wire format.

3. To read the protocol wire format into memory.

4. To write the in-memory format to persistent storage (to cache across reboots for example).

5. To write the in-memory format to a format that can be consumed by applications.

If every option were formatted differently and uniquely, then we would have to write 5 functions for every option. As there is the potential for as many as 254 DHCPv4 options, or 65536 DHCPv6 options, not to mention the various encapsulated spaces ("suboptions"), this is not scalable.

One simple trick is to make the in-memory format the same as the wire format. This reduces the number of functions required from 5 to 3. This is not always workable - sometimes an intermediate format is required, but it is a good general case.

Another simple trick is to use the same (or very nearly the same) format for persistent storage as is used to convey parameters to applications. This reduces the number of functions again from 3 to 2.

This is still an intractable number of functions per each DHCP option, even without the entire DHCP option space populated. So, we need a way to reduce this to small orders.

A.1. Atomic DHCP

To accomplish these goals, a common "Format String" is used to describe, in abstract, all of the above. Each octet in this format string represents a "DHCP Atom". We chain these 'atoms' together, forming a sort of molecular structure for a particular DHCP option's defined format.

The Configuration Syntax allows the user to construct such a format string without having to understand how the Atom is encoded on the wire, and how it is configured or presented.

You can reasonably imagine that the various common formats of DHCP options described above (Table 1) each have an Atom associated with it. There are special use Atoms, such as one to repeat the previous Atoms indefinitely (for example, for options with multiple IPv4 addresses one after the other), and one which makes the previous Atom optional.

As the software encounters a format string, it processes each Atom individually to read from configuration into wire format, and also to validate and convert wire format into output format (which with some small exclusions is identical to the configuration format).

The format strings themselves are either hard coded by the software in a table of option definitions, or are compiled at runtime through configuration syntax generated by the operator.

```
        option <space>.<option> code <number> = <definition>;
```

The <space> refers to the option space, which may be the DHCPv4 option space, the DHCPv6 option space, or any suboption space such as the DHCPv4 Relay Agent Information suboptions or similar.

The <option> refers to the option's symbolic name within that space.

The code <number> refers to the binary code assigned to this option.

The <definition> is a complex statement that brings together DHCP Atoms, many of which are the aforementioned common formats, that compose this option.

Below is a sample configuration for two options, whose wire formats are defined in [RFC2132]. The Path MTU Plateau Table option, and the Static Routes option.

```
        option dhcp.path-mtu-plateau-table code 25 =
                                           array of unsigned integer 16;
        option dhcp.static-routes code 33 = array of { ip-address,
                                                       ip-address };
```

Once the options' syntax configuration is out of the way, values to be carried in the options may be configured. These acts are distinct; the previous configuration only prepares the parser system to accept the configuration below. The below configuration actually supplies a value to be transmitted on the wire, relying on the above format definition.

```
        option dhcp.path-mtu-plataeu-table 4352, 1500, 576;
        option dhcp.static-routes 10.10.10.10 10.10.10.9,
                                  10.10.10.11 10.10.10.9;
```

Author's Address

David W. Hankins
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
USA

Email: dhankins@google.com