

Dynamic Host Configuration Working
Group
Internet-Draft
Updates: 3315 (if approved)
Intended status: BCP
Expires: March 22, 2014

D. Hankins
Google
T. Mrugalski
M. Siodelski
ISC
S. Jiang
Huawei Technologies Co., Ltd
S. Krishnan
Ericsson
September 18, 2013

Guidelines for Creating New DHCPv6 Options
draft-ietf-dhc-option-guidelines-14

Abstract

This document provides guidance to prospective DHCPv6 Option developers to help them creating option formats that are easily adoptable by existing DHCPv6 software. This document updates RFC3315.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 22, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Requirements Language	4
2. Introduction	4
3. When to Use DHCPv6	5
4. General Principles	5
5. Reusing Other Options Formats	6
5.1. Option with IPv6 addresses	7
5.2. Option with a single flag (boolean)	8
5.3. Option with IPv6 prefix	8
5.4. Option with 32-bit integer value	9
5.5. Option with 16-bit integer value	10
5.6. Option with 8-bit integer value	10
5.7. Option with URI	10
5.8. Option with Text String	11
5.9. Option with variable length data	13
5.10. Option with DNS Wire Format Domain Name List	13
6. Avoid Conditional Formatting	14
7. Avoid Aliasing	14
8. Choosing between FQDN and address	15
9. Encapsulated options in DHCPv6	16
10. Additional States Considered Harmful	17
11. Configuration changes occur at fixed times	18
12. Multiple provisioning domains	19
13. Chartering Requirements and Advice for Responsible ADs	19
14. Considerations for Creating New Formats	20
15. Option Size	21
16. Singleton options	21
17. Option Order	22
18. Relay Options	22
19. Clients Request their Options	23
20. Transition Technologies	24
21. Recommended sections in the new document	24
21.1. DHCPv6 Client Behavior Text	25
21.2. DHCPv6 Server Behavior Text	26
21.3. DHCPv6 Relay Agent Behavior Text	26
22. Should the new document update existing RFCs?	26
23. Security Considerations	27
24. IANA Considerations	28
25. Acknowledgements	28
26. References	28
26.1. Normative References	28
26.2. Informative References	28
Authors' Addresses	30

1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. 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.

DHCPv6 [RFC3315] software implementors are not merely faced with the task of handling a given option's format on the wire. The option must fit into every stage of the system's process, starting with the user interface used to enter the configuration up to the machine interfaces where configuration is ultimately consumed.

Another frequently overlooked aspect of rapid adoption is whether the option requires operators to be intimately familiar with the option's internal format in order to use it? Most DHCPv6 software provides a facility for handling unknown options at the time of publication. The handling of such options usually needs to be manually configured by the 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 requires 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.
- o If the option would work well in the particular deployment environment the proponents currently envision, but has equally valid uses in some other environment where the proposed option format would fail or would produce inconsistent results.

There are many things DHCPv6 option creators can do to avoid the

pitfalls in this list entirely, or failing that, to make software implementors lives easier and improve its chances for widespread adoption.

3. When to Use DHCPv6

Principally, DHCPv6 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 DHCPv6.

The presence of such a knob isn't enough, because DHCPv6 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 the network must also provide (such as domain name servers, or routers). This means that, even if a configuration parameter can potentially be delivered by DHCPv6, it is necessary to evaluate whether it is reasonable for this parameter to be under the control of the administrator of whatever network a client is attached to at any given time.

Note that the client is not required to configure any of these values received via DHCPv6 (e.g., due to having these values locally configured by its own administrator). But it needs to be noted that overriding DHCPv6-provided values may cause the client to be denied certain services in the network to which it has attached. The possibility of having higher level of control over client node configuration is one of the reasons that DHCPv6 is preferred in enterprise networks.

4. General Principles

The primary guiding principle to follow in order to enhance an option's adoptability is reuse. The option should be created in such a way that does 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 facilities then it's fairly certain that new software can easily formally adopt it.

There are at least two classes of DHCPv6 options: simple options which are provided explicitly to carry data from one side of the DHCPv6 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 DHCPv6 software or are used during special processing (such as the Fully Qualified Domain Name (FQDN) option [RFC4704]), and so forth; these options carry data that is the result of a routine in some DHCPv6 software.

The guidelines laid out here should be applied in a relaxed manner for the protocol class of options. Wherever special case code is already required to adopt the DHCPv6 option, it is substantially more reasonable to format the option in a less generic fashion, if there are measurable benefits to doing so.

5. Reusing Other Options Formats

The easiest approach to manufacturing trivially deployable DHCPv6 Options is to assemble the option out of whatever common fragments fit - possibly allowing a group of data elements 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 means that implementations will likely be able to reuse code paths designed to support the other options.

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

However, it isn't very practical to consider the bulk of DHCPv6 options already allocated, and consider which of those solve a similar problem. So, the following list of common option format data elements is provided as a shorthand. Please note that it is not complete in terms of exemplifying every option format ever devised.

If more complex options are needed, those basic formats mentioned here may be considered as primitives (or 'fragment types') that can be used to build more complex formats. It should be noted that it is often easier to implement two options with trivial formats than one option with more complex format. That is not unconditional requirement though. In some cases splitting one complex option into two or more simple options introduces inter-option dependencies that should be avoided. In such a case, it is usually better to keep one complex option.

5.1. Option with IPv6 addresses

This option format is used to carry one or many IPv6 addresses. In some cases the number of allowed address is limited (e.g. to one):

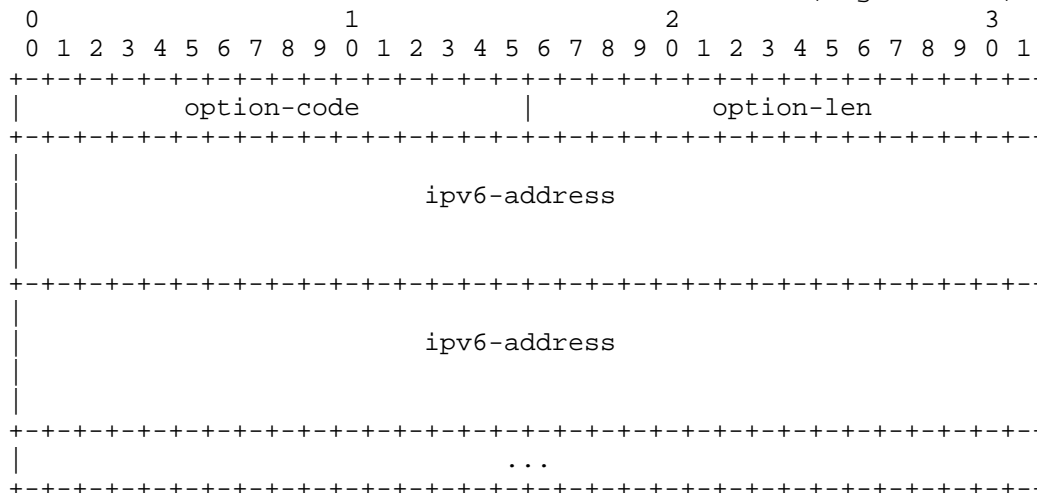


Figure 1: Option with IPv6 address

Examples of use:

- o DHCPv6 server unicast address (a single address only) [RFC3315]
- o SIP Servers IPv6 Address List [RFC3319]
- o DNS Recursive Name Server [RFC3646]
- o NIS Servers [RFC3898]
- o SNTP Servers [RFC4075]
- o Broadcast and Multicast Service Controller IPv6 Address Option for DHCPv6 [RFC4280]
- o MIPv6 Home Agent Address [RFC6610] (a single address only)
- o NTP server [RFC5908] (a single address only)
- o NTP Multicast address [RFC5908] (a single address only)

5.2. Option with a single flag (boolean)

Sometimes it is useful to convey a single flag that can either take on or off values. Instead of specifying an option with one bit of usable data and 7 bits of padding, it is better to define an option without any content. It is the presence or absence of the option that conveys the value. This approach has the additional benefit of absent option designating the default, i.e. administrator has to take explicit actions to deploy the opposite of the default value.

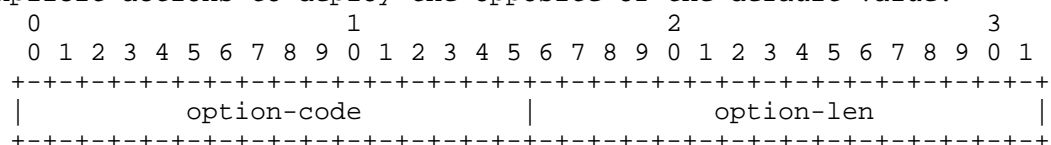


Figure 2: Option for conveying boolean

Examples of use:

- o DHCPv6 rapid-commit [RFC3315]

5.3. Option with IPv6 prefix

Sometimes there is a need to convey an IPv6 prefix. The information to be carried by such an option includes the 128-bit IPv6 prefix together with a length of this prefix taking values from 0 to 128. Using the simplest approach, the option could convey this data in two fixed length fields: one carrying prefix length, another carrying the prefix. However, in many cases /64 or shorter prefixes are used. This implies that the large part of the prefix data carried by the option would have its bits set to zero and would be unused. In order to avoid carrying unused data, it is recommended to store prefix in the variable length data field. The appropriate option format is defined as follows:

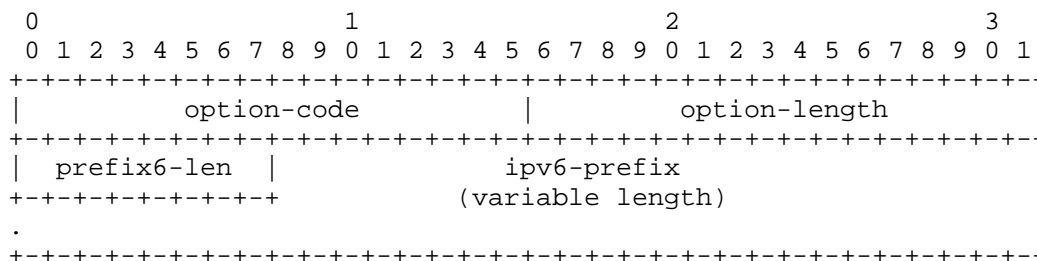


Figure 3: Option with IPv6 Prefix

option-length is set to 1 + length of the IPv6 prefix.

prefix6-len is one octet long and specifies the length in bits of the IPv6 prefix. Typically allowed values are 0 to 128.

ipv6-prefix field is a variable length field that specifies the IPv6 prefix. The length is $(\text{prefix6-len} + 7) / 8$. This field is padded with zero bits up to the nearest octet boundary when prefix6-len is not divisible by 8.

Examples of use:

- o Default Mapping Rule [I-D.ietf-softwire-map-dhcp]

For example, the prefix 2001:db8::/60 would be encoded with an option-length of 9, prefix6-len would be set to 60, the ipv6-prefix would be 8 octets and would contain octets 20 01 0d b8 00 00 00 00.

It should be noted that the IAPREFIX option defined by [RFC3633] uses a full length 16-octet prefix field. The concern about option length was not well understood at the time of its publication.

5.4. Option with 32-bit integer value

This option format can be used to carry 32 bit-signed or unsigned integer value:

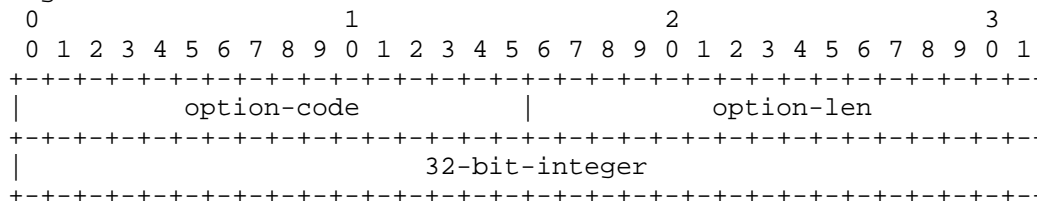


Figure 4: Option with 32-bit-integer value

Examples of use:

- o Information Refresh Time [RFC4242]

5.5. Option with 16-bit integer value

This option format can be used to carry 16-bit signed or unsigned integer values:

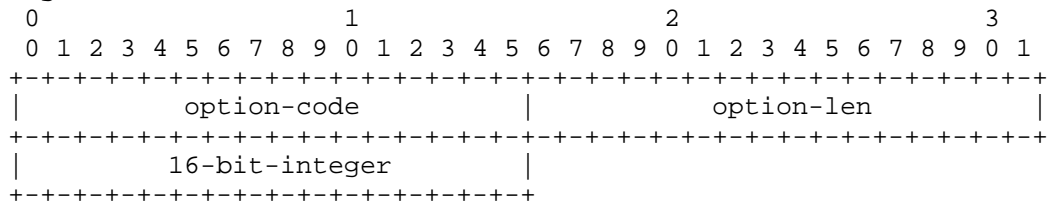


Figure 5: Option with 16-bit integer value

Examples of use:

- o Elapsed Time [RFC3315]

5.6. Option with 8-bit integer value

This option format can be used to carry 8-bit integer values:

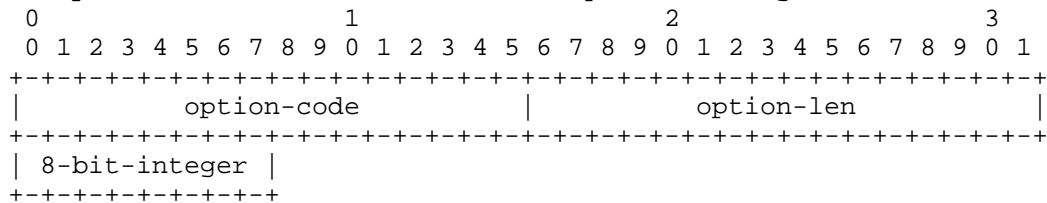


Figure 6: Option with 8-bit integer value

Examples of use:

- o DHCPv6 Preference [RFC3315]

5.7. Option with URI

A Uniform Resource Identifier (URI) [RFC3986] is a compact sequence of characters that identifies an abstract or physical resource. The term "Uniform Resource Locator" (URL) refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism (e.g., its network "location"). This option format can be used to carry a single URI:

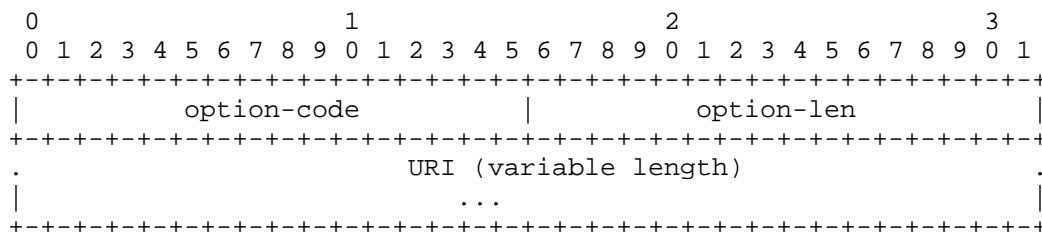


Figure 7: Option with URI

Examples of use:

- o Boot File URL [RFC5970]

An alternate encoding to support multiple URIs is available. An option must be defined to use either the single URI format above or the multiple URI format below depending on whether a single is always sufficient or if multiple URIs are possible.

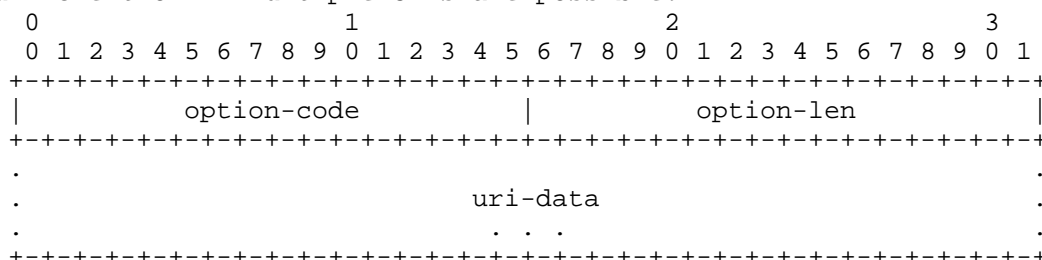
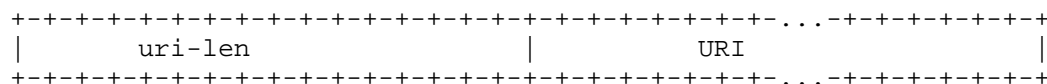


Figure 8: Option with multiple URIs

Each instance of the uri-data is formatted as follows:



The uri-len is two octets long and specifies the length of the uri data.

5.8. Option with Text String

A text string is a sequence of characters that have no semantics. The encoding of the text string **MUST** be specified. Unless otherwise specified, all text strings in newly defined options are expected to be Unicode strings that are encoded using UTF-8 [RFC3629] in Net-

Unicode form [RFC5198]. Please note that all strings containing only 7 bit ASCII characters are also valid UTF-8 Net-Unicode strings.

If a data format has semantics other than just being text, it is not a string. E.g., a FQDN is not a string, and a URI is also not a string, because they have different semantics. A string must not include any terminator (such as a null byte). This option format can be used to carry a text string:

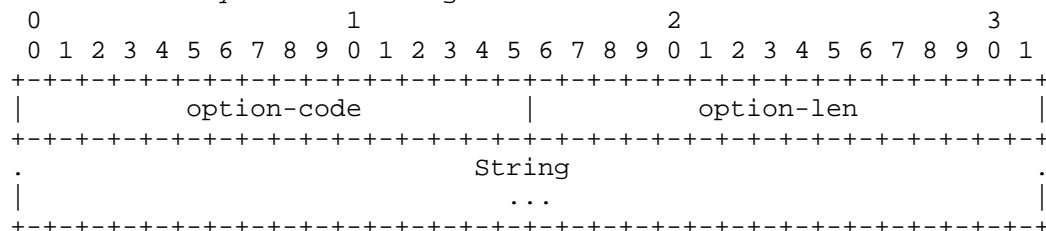


Figure 9: Option with text string

Examples of use:

- o Timezone Options for DHCPv6 [RFC4833]

An alternate encoding to support multiple text strings is available. An option must be defined to use either the single text string format above or the multiple text string format below depending on whether a single is always sufficient or if multiple text strings are possible.

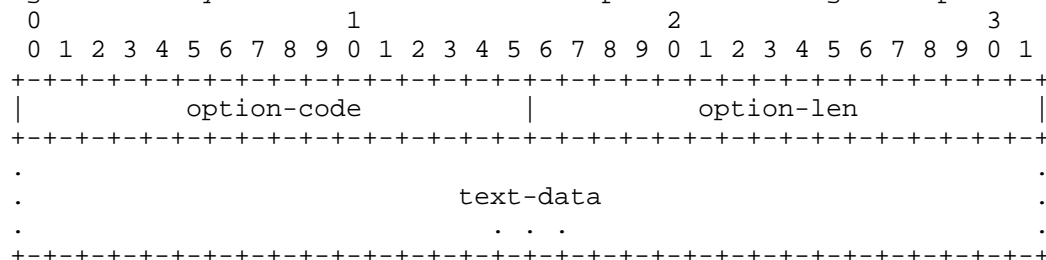
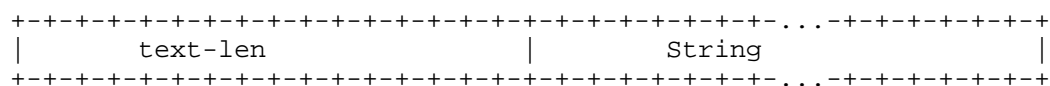


Figure 10: Option with multiple text strings

Each instance of the text-data is formatted as follows:



The text-len is two octets long and specifies the length of the string.

5.9. Option with variable length data

This option can be used to carry variable length data of any kind. Internal representation of carried data is option specific. Whenever this format is used by the new option being defined, the data encoding should be documented.

This option format provides a lot of flexibility to pass data of almost any kind. Though, whenever possible it is highly recommended to use more specialized options, with field types better matching carried data types.

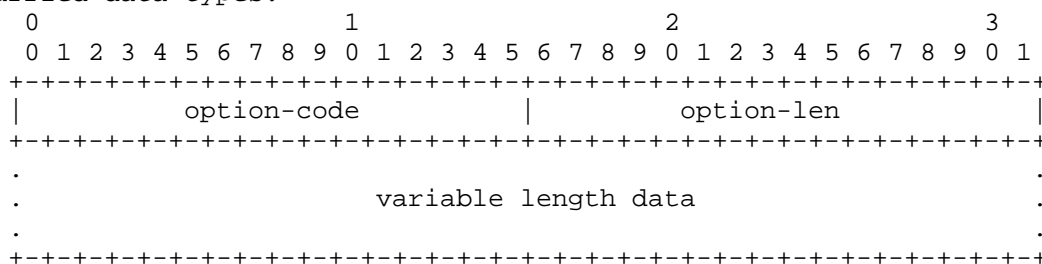


Figure 11: Option with variable length data

Examples of use:

- o Client Identifier [RFC3315]
- o Server Identifier [RFC3315]

5.10. Option with DNS Wire Format Domain Name List

This option is used to carry 'domain search' lists or any host or domain name. It uses the same format as described in Section 5.9, but with the special data encoding, described in section 8 of [RFC3315]. This data encoding supports carrying multiple instances of hosts or domain names in a single option, by terminating each instance with the byte value of 0.

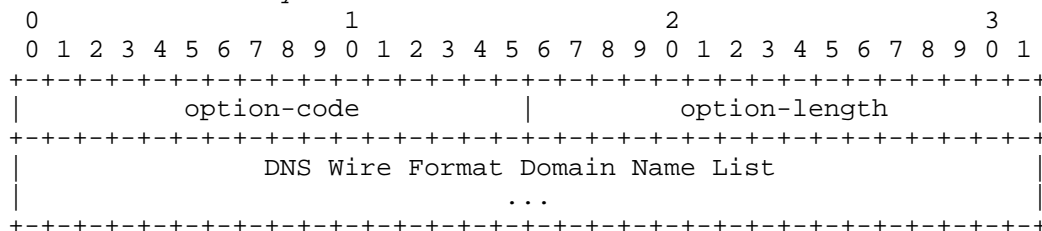


Figure 12: Option with DNS Wire Format Domain Name List

Examples of use:

- o SIP Servers Domain Name List [RFC3319] (many domains)
- o NIS Domain Name (many domains) [RFC3898] (many domains)
- o DS-Lite AFTR location [RFC6334] (a single FQDN)
- o Home Network Identifier [RFC6610] (a single FQDN)
- o Home Agent FQDN [RFC6610] (a single FQDN)

6. Avoid Conditional Formatting

Placing an 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. Therefore conditional formatting requires new code to be written and complicates future interoperability should new conditional formats be added; and existing code has to ignore conditional format that it does not support.

7. 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 an URL. This kind of aliasing is undesirable, and is not recommended.

In this case, where three different formats are supposed, it more than triples the work of the software involved, requiring support for not merely one format, but support to produce and digest all three. Furthermore, code development and testing must cover all possible combinations of defined formats. 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, DHCPv6 message bandwidth is wasted on option contents that are redundant. Also, the DHCPv6 option number space is wasted, as three new option codes are required, rather than one.

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).

8. Choosing between FQDN and address

Some parameters may be specified as FQDN or an address. It is not allowed to define both option types at the same time (see section Section 7), so one of them must be chosen. This section is intended to help make an informed decision in that regard.

On the specific subject of desiring to configure a value using a FQDN instead of a binary IP address, note that most DHCPv6 server implementations will happily accept a Domain Name entered by the administrator, and use DNS resolution to render binary IP addresses in DHCPv6 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.

Unless there are specific reasons to do otherwise, address should be used. It is simpler to use, its validation is trivial (length of 16 constitutes a valid option), is explicit and does not allow any ambiguity. It is faster (does not require extra resolution efforts), so it is more efficient, which can be especially important for energy restricted devices.

FQDN options are discouraged for options intended to configure hosts, because hosts may have multiple provisioning domains (see Section 12), and may get a different answer from the DNS depending on the provisioning domain. This is particularly a problem when the normal expected use of the option makes sense with private DNS zone(s), as might be the case with a corporate VPN.

FQDN does require a resolution into an actual address. This implies the question when the FQDN resolution should be taken. There are a couple of possible answers: a) by the server, when it is started, b) by the server, when it is about to send an option, c) by the client,

immediately after receiving an option, d) by the client, when the content of the option is actually consumed. For a), b) and possibly c), the option should really convey an address, not FQDN. The only real incentive to use FQDN is case d). It is the only case that allows possible changes in the DNS to be picked up by clients.

FQDN imposes a number of additional failure modes and issues that should be dealt with:

1. The client must have a knowledge about available DNS servers. That typically means that option DNS_SERVERS is mandatory. This should be mentioned in the draft that defines new option. It is possible that the server will return FQDN option, but not the DNS Servers option. There should be a brief discussion about it;
 2. The DNS may not be reachable;
 3. DNS may be available, but may not have appropriate information (e.g. no AAAA records for specified FQDN);
 4. Address family must be specified (A, AAAA or any);
 5. What should the client do if there are multiple records available (use only the first one, use all, use one and switch to the second if the first fails for whatever reason, etc.);
 6. Multi-homed devices may be connected to different administrative domains with each domain providing different information in DNS (e.g. an enterprise network exposing private domains). Client may send DNS queries to a different DNS server;
 7. It should be mentioned if Internationalized Domain Names are allowed. If they are, what kind of DNS option encoding should be specified.
9. Encapsulated options in DHCPv6

Most options are conveyed in a DHCPv6 message directly. Although there is no codified normative language for such options, they are often referred to as top-level options. Many options may include other options. Such inner options are often referred to as encapsulated or nested options. Those options are sometimes called sub-options, but this term actually means something else, and therefore should never be used to describe encapsulated options. It is recommended to use term "encapsulated" as this terminology is used in [RFC3315]. The difference between encapsulated and sub-options are that the former uses normal DHCPv6 option numbers, while the

latter uses option number space specific to a given parent option. It should be noted that, contrary to DHCPv4, there is no shortage of option numbers. Therefore almost all options share a common option space. For example option type 1 meant different things in DHCPv4, depending if it was located in top-level or inside of Relay Agent Information option. There is no such ambiguity in DHCPv6 (with the unfortunate exception of [RFC5908], which was published without following the advice provided during the DHC working group review, and contains many errors. [RFC5908] SHOULD NOT under any circumstances be used as a template for future DHCP option definitions.

From the implementation perspective, it is easier to implement encapsulated options rather than sub-options, as the implementers do not have to deal with separate option spaces and can use the same buffer parser in several places throughout the code.

Such encapsulation is not limited to one level. There is at least one defined option that is encapsulated twice: Identity Association for Prefix Delegation (IA_PD, defined in [RFC3633], section 9) conveys IA Prefix (IAPREFIX, defined in [RFC3633], section 10). Such delegated prefix may contain an excluded prefix range that is represented by PD_EXCLUDE option that is conveyed as encapsulated inside IAPREFIX (PD_EXCLUDE, defined in [RFC6603]). It seems awkward to refer to such options as sub-sub-option or doubly encapsulated option, therefore "encapsulated option" term is typically used, regardless of the nesting level.

When defining a DHCP-based configuration mechanism for a protocol that requires something more complex than a single option, it may be tempting to group configuration values using sub-options. That should preferably be avoided, as it increases complexity of the parser. It is much easier, faster and less error prone to parse a large number of options on a single (top-level) scope, than parse options on several scopes. The use of sub-options should be avoided as much as possible, but it is better to use sub-options rather than conditional formatting.

It should be noted that currently there is no clear way defined for requesting sub-options. Most known implementations are simply using top-level ORO for requesting both top-level options and encapsulated options.

10. Additional States Considered Harmful

DHCP is a protocol designed for provisioning clients. Less experienced protocol designers often assume that it is easy to define

an option that will convey a different parameter for each client in a network. Such problems arose during designs of MAP [I-D.ietf-softwire-map-dhcp] and 4rd [I-D.ietf-softwire-4rd]. While it would be easier for provisioned clients to get ready to use per-client option values, such requirement puts exceedingly large loads on the server side. The new extensions may introduce new implementation complexity and additional database state on the server. Alternatives should be considered, if possible. As an example, [I-D.ietf-softwire-map-dhcp] was designed in a way that all clients are provisioned with the same set of MAP options and each provisioned client uses its unique address and delegated prefix to generate client-specific information. Such a solution does not introduce any additional state for the server and therefore scales better.

It also should be noted that contrary to DHCPv4, DHCPv6 keeps several timers for renewals. Each IA_NA (addresses) and IA_PD (prefixes) contains T1 and T2 timers that designate time after which client will initiate renewal. Those timers apply only to its own IA containers. Refreshing other parameters should be initiated after a time specified in the Information Refresh Time Option (defined in [RFC4242]), carried in the Reply message and returned in response to Information-Request message. Introducing additional timers make deployment unnecessarily complex and SHOULD be avoided.

11. Configuration changes occur at fixed times

In general, DHCPv6 clients only refresh configuration data from the DHCP server when the T1 timer expires. Although there is a RECONFIGURE mechanism that allows a DHCP server to request that clients initiate reconfiguration, support for this mechanism is optional and cannot be relied upon.

Even when DHCP clients refresh their configuration information, not all consumers of DHCP-sourced configuration data notice these changes. For instance, if a server is started using parameters received in an early DHCP transaction, but does not check for updates from DHCP, it may well continue to use the same parameter indefinitely. There are a few operating systems that take care of reconfiguring services when the client moves to a new network (e.g. based on mechanisms like [RFC4436], [RFC4957] or [RFC6059]), but it's worth bearing in mind that a renew may not always result in the client taking up new configuration information that it receives.

In light of the above, when designing an option you should take into consideration the fact that your option may hold stale data that will only be updated at an arbitrary time in the future.

12. Multiple provisioning domains

In some cases there could be more than one DHCPv6 server on a link, with each providing a different set of parameters. One notable example of such a case is a home network with a connection to two independent ISPs.

The DHCPv6 protocol specification does not provide clear advice on how to handle multiple provisioning sources. Although [RFC3315] states that a client that receives more than one ADVERTISE message, may respond to one or more of them, such capability has not been observed in existing implementations. Existing clients will pick one server and will continue configuration process with that server, ignoring all other servers.

In addition, a node that acts as a DHCPv6 client may be connected to more than one physical network. In this case, it will in most cases operate a separate DHCP client state machine on each interface, acquiring different, possibly conflicting information through each. This information will not be acquired in any synchronized way.

Existing nodes cannot be assumed to systematically segregate configuration information on the basis of its source; as a result, it is quite possible that a node may receive an FQDN on one network interface, but do the DNS resolution on a different network interface, using different DNS servers. As a consequence, DNS resolution done by the DHCP server is more likely to behave predictably than DNS resolution done on a multi-interface or multi-homed client.

This is a generic DHCP protocol issue and should not be dealt within each option separately. This issue is better dealt with using a protocol-level solution and fixing this problem should not be attempted on a per option basis. Work is ongoing in the IETF to provide a systematic solution to this problem.

13. Chartering Requirements and Advice for Responsible ADs

Adding a simple DHCP option is straightforward, and generally something that any working group can do, perhaps with some help from designated DHCP experts. However, when new fragment types need to be devised, this requires the attention of DHCP experts, and should not be done in a working group that doesn't have a quorum of such experts. This is true whether the new fragment type has the same structure as an existing fragment type, but has different semantics. It is equally true when the new format has a new structure.

Responsible Area Directors for working groups that wish to add a work item to a working group charter to define a new DHCP option should get clarity from the working group as to whether the new option is a simple DHCP option with no new fragment type or new fragment semantics, or whether it in fact will require new fragment types. A working group charter item should explicitly state which of these two types is required; if it is not known at the time of chartering, the charter should state that the working group will study the question and recharter or seek help elsewhere if a new fragment type is to be defined.

If a working group needs a new fragment type, it is preferable to seek out a working group whose members already have sufficient expertise to evaluate the new work and try to come up with a new format that generalizes well and can be reused, rather than a single-use fragment type. If such a working group is available, the work should be chartered in that working group as a separate draft that documents the new fragment type. The working group that needs the new fragment type can then define their new option referencing the new fragment type document. This work can generally be done in parallel so as not to delay the process significantly.

In the event that there is no working group with DHCP expertise that can define the new fragment type, the responsible AD should seek out help from known DHCP experts within the IETF to provide advice and frequent early review as the working group defines the new fragment type. The new fragment type should still be done in a separate document, even if it's done in the same working group, so as to foster reuse of the new fragment type. The responsible AD should work with the working group chairs and designated DHCP experts to ensure that new fragment type document has in fact been carefully reviewed by the experts and appears satisfactory.

Responsible area directors for working groups that are considering defining options that actually update the DHCP protocol, as opposed to simple options, should go through a process similar to that described above when trying to determine where to do the work. Under no circumstances should a working group be given a charter deliverable to define a new DHCP option, and then on the basis of that charter item actually make updates to the DHCP protocol.

14. Considerations for Creating New Formats

When defining new options, 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.

When defining a new option, it is best to synthesize the option format using fragment types already in use. However, in some cases there may be no fragment type that accomplishes the intended purpose.

The matter of size considerations and option order are further discussed in Section 15 and Section 17.

15. Option Size

DHCPv6 [RFC3315] allows for packet sizes up to 64KB. First, through its use of link-local addresses, it avoids many of the deployment problems that plague DHCPv4, and is actually an UDP over IPv6 based protocol (compared to DHCPv4, which is mostly UDP over IPv4 protocol, but with layer 2 hacks). 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 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 not really possible to describe a fixed limit that cleanly divides workable option sizes from those that are too big.

It is advantageous to prefer option formats which contain the desired information in the smallest form factor that satisfies the requirements. Common sense still applies here. It is better to split distinct values into separate octets rather than propose overly complex bit shifting operations to save several bits (or even an octet or two) that would be padded to the next octet boundary anyway.

DHCPv6 does allow for 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). In such cases, it is better to define an option as an array if it is possible. It is recommended to clarify (with normative language) whether a given DHCPv6 option may appear once or multiple times. The default assumption is only once.

In general, if a lot of data needs to be configured (i.e. large option lengths), DHCPv6 may not be the best choice to deliver such configuration information and SHOULD simply be used to deliver an URI that specifies how to obtain the actual configuration information.

16. Singleton options

Although [RFC3315] states that each option type MAY appear more than

once, the original idea was that multiple instances are reserved for stateful options, like IA_NA or IA_PD. For most other options it is usually expected that they will appear at most once. Such options are called singleton options. Sadly, RFCs have often failed to clearly specify whether a given option can appear more than once or not. Documents that define new options SHOULD state whether these options are singletons or not. Unless otherwise specified, newly defined options are considered to be singletons.

When deciding whether a single or multiple option instances are allowed in a message, take into consideration how the content of the option will be used. Depending on the service being configured it may or may not make sense to have multiple values configured. If multiple values make sense, it is better to explicitly allow that by using option format that allows multiple values within one option instance.

Allowing multiple option instances often leads to confusion. Consider the following example. Basic DS-Lite architecture assumes that the B4 element (DHCPv6 client) will receive AFTR option and establish a single tunnel to configured tunnel termination point (AFTR). During standardization process of [RFC6334] there was a discussion whether multiple instances of DS-Lite tunnel option should be allowed. This created an unfounded expectation that the clients receiving multiple instances of the option will somehow know when one tunnel endpoint goes off-line and do some sort of failover between other values provided in other instances of the AFTR option. Others assumed that if there are multiple options, the client will somehow do a load balancing between provided tunnel endpoints. Neither failover nor load balancing was defined for DS-Lite architecture, so it caused confusion. It was eventually decided to allow only one instance of the AFTR option.

17. Option Order

Option order, either the order among many DHCPv6 options or the order of multiple instances of the same option, SHOULD NOT be significant and MUST NOT be assumed.

As there is no explicit order for multiple instance of the same option, an option definition SHOULD instead restrict ordering by using a single option that contains ordered fields.

18. Relay Options

In DHCPv4, all relay options are organized as sub-options within DHCP

Relay Agent Information Option[RFC3046]. And an independent number space called "DHCP Relay Agent Sub-options" is maintained by IANA. Different from DHCPv4, in DHCPv6, Relay options are defined in the same way as client/server options, and they too use the same number space as client/server options. Future DHCPv6 Relay options MUST be allocated from this single DHCPv6 Option number space.

E.g. the Relay-Supplied Options Option [RFC6422] may also contain some DHCPv6 options as permitted, such as the EAP Re-authentication Protocol (ERP) Local Domain Name DHCPv6 Option [RFC6440].

19. Clients Request their Options

The DHCPv6 Option Request Option (OPTION_ORO) [RFC3315], is an option that serves two purposes - to inform the server what options the client supports and to inform what options the client is willing to consume.

For some options, such as the options required for the functioning of the DHCPv6 protocol itself, it doesn't make sense to require that they be explicitly requested using the Option Request Option. In all other cases, it is prudent to assume that any new option must be present on the relevant option request list if the client desires to receive it.

It is tempting to add text that requires the client to include a new option in Option Request Option list, similar to this text: "Clients MUST place the foo option code on the Option Request Option list, clients MAY include option foo in their packets as hints for the server as values the desire, and servers MUST include option foo when the client requested it (and the server has been so configured)". Such text is discouraged as there are several issues with it. First, it assumes that client implementation that supports a given option will always want to use it. This is not true. The second and more important reason is that such text essentially duplicates mechanism already defined in [RFC3315]. It is better to simply refer to the existing mechanism rather than define it again. See Section 21 for proposed examples on how to do that.

Creators of DHCPv6 options cannot not assume special ordering of options either as they appear in the option request option, or as they appear within the packet. Although it is reasonable to expect that options will be processed in the order they appear in ORO, server software is not required to sort DHCPv6 options into the same order in reply messages.

It should also be noted that options values are never aligned within

the DHCP packet, even the option code and option length may appear on odd byte boundaries.

20. Transition Technologies

Transition from IPv4 to IPv6 is progressing. Many transition technologies are proposed to speed it up. As a natural consequence there are also DHCP options proposed to provision those proposals. The inevitable question is whether the required parameters should be delivered over DHCPv4 or DHCPv6. Authors often don't give much thought about it and simply pick DHCPv6 without realizing the consequences. IPv6 is expected to stay with us for many decades, and so is DHCPv6. There is no mechanism available to deprecate an option in DHCPv6, so any options defined will stay with us as long as DHCPv6 protocol itself. It seems likely that such options defined to transition from IPv4 will outlive IPv4 by many decades. From that perspective it is better to implement provisioning of the transition technologies in DHCPv4, which will be obsoleted together with IPv4.

When the network infrastructure becomes IPv6-only, the support for IPv4-only nodes may still be needed. In such a scenario, a mechanism for providing IPv4 configuration information over IPv6-only networks such as [I-D.ietf-dhc-v4configuration] may be needed.

21. Recommended sections in the new document

There are three major entities in DHCPv6 protocol: server, relay agent, and client. It is very helpful for implementers to include separate sections that describe operation for those three major entities. Even when a given entity does not participate, it is useful to have a very short section stating that it must not send a given option and must ignore it when received.

There is also a separate entity called requestor, which is a special client-like type that participates in leasequery protocol [RFC5007] and [RFC5460]. A similar section for the requestor is not required, unless the new option has anything to do with requestor (or it is likely that the reader may think that is has). It should be noted that while in the majority of deployments, requestor is co-located with relay agent, those are two separate entities from the protocol perspective and they may be used separately. There are stand-alone requestor implementations available.

The following sections include proposed text for such sections. That text is not required to appear, but it is appropriate in most cases. Additional or modified text specific to a given option is often

required.

Although requestor is somewhat uncommon functionality, its existence should be noted, especially when allowing or disallowing options to appear in certain message or being sent by certain entities. Additional message types may appear in the future, besides types defined in [RFC3315]. Therefore authors are encouraged to familiarize themselves with a list of currently defined DHCPv6 messages available on IANA website [iana].

Typically new options are requested by clients and assigned by the server, so there is no specific relay behavior. Nevertheless it is good to include a section for relay agent behavior and simply state that there are no additional requirements for relays. The same applies for client behavior if the options are to be exchanged between relay and server.

Sections that contain option definitions MUST include formal verification procedure. Often it is very simple, e.g. option that conveys IPv6 address must be exactly 16 bytes long, but sometimes the rules are more complex. It is recommended to refer to existing documents (e.g. section 8 of RFC3315 for domain name encoding) rather than trying to repeat such rules.

21.1. DHCPv6 Client Behavior Text

Clients MAY request option foo, as defined in [RFC3315], sections 17.1.1, 18.1.1, 18.1.3, 18.1.4, 18.1.5 and 22.7. As a convenience to the reader, we mention here that the client includes requested option codes in Option Request Option.

Optional text (if client's hints make sense): Client also MAY include option foo in its SOLICIT, REQUEST, RENEW, REBIND and INFORMATION-REQUEST messages as a hint for the server regarding preferred option values.

Optional text (if the option contains FQDN): If the client requests an option that conveys an FQDN, it is expected that the contents of that option will be resolved using DNS. Hence the following text may be useful: Clients that request option foo SHOULD also request option OPTION_DNS_SERVERS specified in [RFC3646].

Clients MUST discard option foo if it is invalid (i.e. did not pass validation steps defined in Section X.Y).

Optional text (if option foo is expected to be exchanged between relays and servers): Option foo is exchanged between relays and servers only. Clients are not aware of the usage of option foo.

Clients MUST ignore received option foo.

21.2. DHCPv6 Server Behavior Text

Sections 17.2.2 and 18.2 of [RFC3315] govern server operation in regards to option assignment. As a convenience to the reader, we mention here that the server will send option foo only if configured with specific values for foo and the client requested it.

Optional text: Option foo is a singleton. Servers MUST NOT send more than one instance of foo option.

Optional text (if server is never supposed to receive option foo): Servers MUST ignore incoming foo option.

21.3. DHCPv6 Relay Agent Behavior Text

It's never appropriate for a relay agent to add options to a message heading toward the client, and relay agents don't actually construct Relay-Reply messages anyway.

Optional text (if foo option is exchanged between clients and server or between requestors and servers): There are no additional requirements for relays.

Optional text (if relays are expected to insert or consume option foo): Relay agents MAY include option foo in a Relay-Forw when forwarding packets from clients to the servers.

22. Should the new document update existing RFCs?

Authors often ask themselves a question whether their proposal updates exist RFCs, especially 3315. In April 2013 there were about 80 options defined. Had all documents that defined them also updated RFC3315, comprehension of such a document set would be extremely difficult. It should be noted that "extends" and "updates" are two very different verbs. If a new draft defines a new option that clients request and servers provide, it merely extends current standards, so "updates 3315" is not required in the new document header. On the other hand, if a new document replaces or modifies existing behavior, it should be noted that it updates the other document. For example, [RFC6644] clearly updates [RFC3315] as it replaces existing with new text.

23. Security Considerations

DHCPv6 does have an Authentication mechanism ([RFC3315]) that makes it possible for DHCPv6 software to discriminate between authentic endpoints and man-in-the-middle. Other authentication mechanisms may optionally be deployed.

So, while creating a new option, it is prudent to assume that the DHCPv6 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 DHCPv6 message exchanges are transmitted to 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, out of bound values may be used to breach security. An IP address field might be made to carry a loopback address, or local multicast 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 implicitly trusted without being validated.

Authors of drafts defining new DHCP options are therefore strongly advised to explicitly define validation measures that recipients of such options are required to do before processing such options. However, validation measures already defined by RFC3315 or other specifications referenced by the new option document are redundant, and can introduce errors, so authors are equally strongly advised to refer to the base specification for any such validation language rather than copying it into the new specification.

24. IANA Considerations

This document has no actions for IANA.

25. Acknowledgements

Authors would like to thank Simon Perreault, Bernie Volz, Ted Lemon, Bud Millwood and Ralph Droms for their comments.

26. References

26.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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.

26.2. Informative References

- [I-D.ietf-dhc-v4configuration]
Rajtar, B. and I. Farrer, "Provisioning IPv4 Configuration Over IPv6 Only Networks",
draft-ietf-dhc-v4configuration-01 (work in progress),
May 2013.
- [I-D.ietf-software-4rd]
Despres, R., Jiang, S., Penno, R., Lee, Y., Chen, G., and M. Chen, "IPv4 Residual Deployment via IPv6 - a Stateless Solution (4rd)", draft-ietf-software-4rd-06 (work in progress), July 2013.
- [I-D.ietf-software-map-dhcp]
Mrugalski, T., Deng, X., Troan, O., Bao, C., Dec, W., and l. leaf.yeh.sdo@gmail.com, "DHCPv6 Options for configuration of Software Address and Port Mapped Clients", draft-ietf-software-map-dhcp-04 (work in progress), July 2013.
- [RFC3046] Patrick, M., "DHCP Relay Agent Information Option", RFC 3046, January 2001.
- [RFC3319] Schulzrinne, H. and B. Volz, "Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol

(SIP) Servers", RFC 3319, July 2003.

- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.
- [RFC3633] Troan, O. and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC 3633, 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.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4075] Kalusivalingam, V., "Simple Network Time Protocol (SNTP) Configuration Option for DHCPv6", RFC 4075, May 2005.
- [RFC4242] Venaas, S., Chown, T., and B. Volz, "Information Refresh Time Option for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 4242, November 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.
- [RFC4436] Aboba, B., Carlson, J., and S. Cheshire, "Detecting Network Attachment in IPv4 (DnAv4)", RFC 4436, March 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.
- [RFC4957] Krishnan, S., Montavont, N., Njedjou, E., Veerepalli, S., and A. Yegin, "Link-Layer Event Notifications for Detecting Network Attachments", RFC 4957, August 2007.
- [RFC5007] Brzozowski, J., Kinnear, K., Volz, B., and S. Zeng, "DHCPv6 Leasequery", RFC 5007, September 2007.

- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.
- [RFC5460] Stapp, M., "DHCPv6 Bulk Leasequery", RFC 5460, February 2009.
- [RFC5908] Gayraud, R. and B. Lourdelet, "Network Time Protocol (NTP) Server Option for DHCPv6", RFC 5908, June 2010.
- [RFC5970] Huth, T., Freimann, J., Zimmer, V., and D. Thaler, "DHCPv6 Options for Network Boot", RFC 5970, September 2010.
- [RFC6059] Krishnan, S. and G. Daley, "Simple Procedures for Detecting Network Attachment in IPv6", RFC 6059, November 2010.
- [RFC6334] Hankins, D. and T. Mrugalski, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Option for Dual-Stack Lite", RFC 6334, August 2011.
- [RFC6422] Lemon, T. and Q. Wu, "Relay-Supplied DHCP Options", RFC 6422, December 2011.
- [RFC6440] Zorn, G., Wu, Q., and Y. Wang, "The EAP Re-authentication Protocol (ERP) Local Domain Name DHCPv6 Option", RFC 6440, December 2011.
- [RFC6603] Korhonen, J., Savolainen, T., Krishnan, S., and O. Troan, "Prefix Exclude Option for DHCPv6-based Prefix Delegation", RFC 6603, May 2012.
- [RFC6610] Jang, H., Yegin, A., Chowdhury, K., Choi, J., and T. Lemon, "DHCP Options for Home Information Discovery in Mobile IPv6 (MIPv6)", RFC 6610, May 2012.
- [RFC6644] Evans, D., Droms, R., and S. Jiang, "Rebind Capability in DHCPv6 Reconfigure Messages", RFC 6644, July 2012.
- [iana] IANA, "DHCPv6 parameters (IANA webpage)", November 2003, <<http://www.iana.org/assignments/dhcpv6-parameters/>>.

Authors' Addresses

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

Email: dhankins@google.com

Tomek Mrugalski
Internet Systems Consortium, Inc.
950 Charter Street
Redwood City, CA 94063
USA

Phone: +1 650 423 1345
Email: tomasz.mrugalski@gmail.com

Marcin Siodelski
950 Charter Street
Redwood City, CA 94063
USA

Phone: +1 650 423 1431
Email: msiodelski@gmail.com

Sheng Jiang
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: jiangsheng@huawei.com

Suresh Krishnan
Ericsson
8400 Blvd Decarie
Town of Mount Royal, Quebec
Canada

Email: suresh.krishnan@ericsson.com

IntArea
Internet-Draft
Intended status: Informational
Expires: May 05, 2014

B. Carpenter
Univ. of Auckland
S. Jiang
Huawei Technologies Co., Ltd
W. Tarreau
HAProxy, Inc.
November 01, 2013

Using the IPv6 Flow Label for Load Balancing in Server Farms
draft-ietf-intarea-flow-label-balancing-03

Abstract

This document describes how the IPv6 flow label as currently specified can be used to enhance layer 3/4 load distribution and balancing for large server farms.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 05, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Summary of Flow Label Specification	3
3. Summary of Server Farm Load Balancing Techniques	4
4. Applying the Flow Label to L3/L4 Load Balancing	7
5. Security Considerations	10
6. IANA Considerations	11
7. Acknowledgements	11
8. Change log [RFC Editor: Please remove]	12
9. References	12
9.1. Normative References	12
9.2. Informative References	13
Authors' Addresses	13

1. Introduction

The IPv6 flow label has been redefined [RFC6437] and is now a recommended IPv6 node requirement [RFC6434]. Its use for load sharing in multipath routing has been specified [RFC6438]. Another scenario in which the flow label could be used is in load distribution for large server farms. Load distribution is a slightly more general term than load balancing, but the latter is more commonly used. In the context of a server farm, both terms refer to mechanisms that distribute the workload of a server farm among different servers in order to optimize performance. Server load balancing commonly applies to HTTP traffic, but most of the techniques described would apply to other upper layer applications as well. This document starts with brief introductions to the flow label and to server load balancing techniques, and then describes how the flow label can be used to enhance load balancers operating on IP packets and TCP sessions, commonly known as layer 3/4 load balancers.

The motivation for this approach is to improve the performance of most types of layer 3/4 load balancers, especially for traffic including multiple IPv6 extension headers and in particular for fragmented packets. Fragmented packets, often the result of customers reaching the load balancer via a VPN with a limited MTU, are a common performance problem.

2. Summary of Flow Label Specification

The IPv6 flow label [RFC6437] is a 20 bit field included in every IPv6 header [RFC2460]. It is recommended to be supported in all IPv6 nodes by [RFC6434]. There is additional background material in [RFC6436] and [RFC6294]. According to its definition, the flow label should be set to a constant value for a given traffic flow (such as an HTTP connection), and that value will belong to a uniform statistical distribution, making it potentially valuable for load balancing purposes.

Any device that has access to the IPv6 header has access to the flow label, and it is at a fixed position in every IPv6 packet. In contrast, transport layer information, such as the port numbers, is not always in a fixed position, since it follows any IPv6 extension headers that may be present. In fact, the logic of finding the transport header is always more complex for IPv6 than for IPv4, due to the absence of an Internet Header Length field in IPv6. Additionally, if packets are fragmented, the flow label will be present in all fragments, but the transport header will only be in one packet. Therefore, within the lifetime of a given transport layer connection, the flow label can be a more convenient "handle" than the port number for identifying that particular connection.

According to RFC 6437, source hosts should set the flow label, but, if they do not (i.e., its value is zero), forwarding nodes (such as the first-hop router) may set it instead. In both cases, the flow label value must be constant for a given transport session, normally identified by the IPv6 and Transport header 5-tuple. By default, the flow label value should be calculated by a stateless algorithm. The resulting value should form part of a statistically uniform distribution, regardless of which node sets it.

It is recognised that at the time of writing, very few traffic flows include a non-zero flow label value. The mechanism described below is one that can be added to existing load balancing mechanisms, so that it will become effective as more and more flows contain a non-zero label. Even if the flow label is chosen from an imperfectly uniform distribution, it will nevertheless increase the information entropy of the IPv6 header as a whole. This allows for progressive introduction of load balancing based on the flow label.

If the recommendations in Section 3 of RFC 6437 are followed for traffic from a given source accessing a well-known TCP port at a given destination, the flow label can act as a substitute for the port numbers as far as a load balancer is concerned, and it can be found at a fixed position in the layer 3 header even if any extension headers are present.

The flow label is defined as an end-to-end component of the IPv6 header, but there are three qualifications to this:

1. Until the RFC 6437 standard is widely implemented as recommended by RFC 6434, the flow label will often be set to the default value of zero.
2. Because of the recommendation to use a stateless algorithm to calculate the label, there is a low (but non-zero) probability that two simultaneous flows from the same source to the same destination have the same flow label value despite having different transport protocol port numbers.
3. The flow label field is in an unprotected part of the IPv6 header, which means that intentional or unintentional changes to its value cannot be easily detected by a receiver.

The first two points are addressed below in Section 4 and the third in Section 5.

3. Summary of Server Farm Load Balancing Techniques

Load balancing for server farms is achieved by a variety of methods, often used in combination [Tarreau]. This section gives a general overview of common methods, although the flow label is not relevant to all of them. The actual load balancing algorithm (the choice of which server to use for a new client session) is irrelevant to this discussion. We give examples for HTTP, but analogous techniques may be used for other application protocols.

- o The simplest method is simply using the DNS to return different server addresses for a single name such as `www.example.com` to different users. This is typically done by rotating the order in which different addresses within the server site are listed by the relevant authoritative DNS server, on the assumption that the client will pick the first one. Routing may be configured such that the different addresses are handled by different ingress routers. Several variants of this load balancing mechanism exist, such as expecting some clients to use all the advertised addresses when multiple connections are involved, or directing the traffic to multiple sites, also known as global load balancing. None of

these mechanisms are in the scope of this document, and what this document proposes does not affect their usability nor aim to replace them, so they will not be discussed further.

- o Another method, for HTTP servers, is to operate a layer 7 reverse proxy in front of the server farm. The reverse proxy will present a single IP address to the world, communicated to clients by a single AAAA record. For each new client session (an incoming TCP connection and HTTP request), it will pick a particular server and proxy the session to it. The act of proxying should be more efficient and less resource-intensive than the act of serving the required content. The proxy must retain TCP state and proxy state for the duration of the session. This TCP state could, potentially, include the incoming flow label value.
- o A component of some load balancing systems is an SSL reverse proxy farm. The individual SSL proxies handle all cryptographic aspects and exchange unencrypted HTTP with the actual servers. Thus, from the load balancing point of view, this really looks just like a server farm, except that it's specialised for HTTPS. Each proxy will retain SSL and TCP and maybe HTTP state for the duration of the session, and the TCP state could potentially include the flow label.
- o Finally the "front end" of many load balancing systems is a layer 3/4 load balancer. While it can be a dedicated device, it is also a standard function of some network switches or routers (e.g. using Equal Cost Multipath Routing (ECMP) [RFC2991]). In this case, it is the layer 3/4 load balancer whose IP address is published as the primary AAAA record for the service. All client sessions will pass through this device. Depending on the specific scenario, the balancer will assign new sessions among the actual application servers, across an SSL proxy farm, or among a set of layer 7 proxies. In all cases, the layer 3/4 load balancer has to classify incoming packets very quickly and choose the target server or proxy so as to ensure persistence. 'Persistence' is defined as the guarantee that a given client session will run to completion on a single server. The layer 3/4 load balancer therefore needs to inspect each incoming packet to classify it. There are two common types of layer 3/4 load balancers, the totally stateless ones which only act on single packets, generally involving a per-packet hashing of easy-to-find information such as the source address and/or port into a server number, and the stateful ones which take the routing decision on the very first packets of a session and maintain the same direction for all packets belonging to the same session. Clearly, both types of layer 3/4 balancers could inspect and make use of the flow label value.

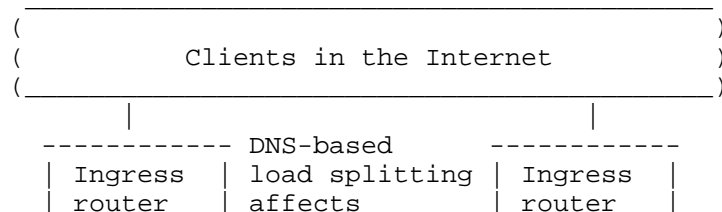
Our focus is on how the balancer identifies a particular flow. For clarity, note that two aspects of layer 3/4 load balancers are not affected by use of the flow label to identify sessions:

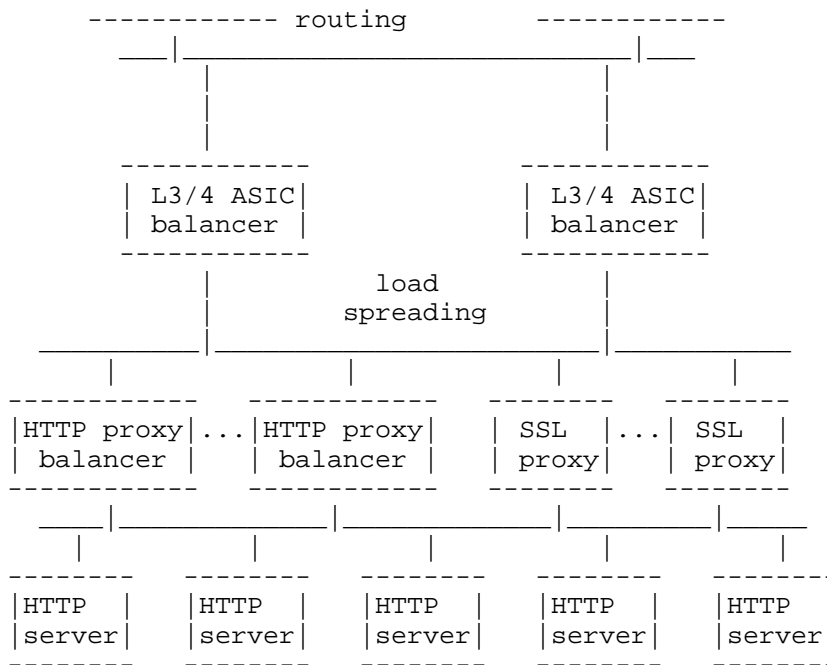
1. Balancers use various techniques to redirect traffic to a specific target server.
 - All servers are configured with the same IP address, they are all on the same LAN, and the load balancer sends directly to their individual MAC addresses. In this case, return packets from the server to the client are sent back without passing through the balancer, a technique known as direct server return, but we are not concerned here with the return packets.
 - All servers are configured with the same IP address, treated locally as an anycast address by layer 3 ECMP routing.
 - Each server has its own IP address, and the balancer uses an IP-in-IP tunnel to reach it.
 - Each server has its own IP address, and the balancer performs NAT (network address and port translation) to deliver the client's packets to that address.

The choice between these methods is not affected by use of the flow label.

2. A layer 3/4 balancer must correctly handle Path MTU Discovery by forwarding relevant ICMPv6 packets in both directions. This too is not directly affected by use of the flow label. It should be noted that there may be difficulty correlating an ICMPv6 "Packet too big" response with the session it refers to, but that is out of the scope of the present document.

The following diagram, inspired by [Tarreau], shows a layout with various methods in use together.





From the previous paragraphs, we can identify several points in this diagram where the flow label might be relevant:

1. Layer 3/4 load balancers.
2. SSL proxies.
3. HTTP proxies.

However, usage by the proxies seems unlikely to affect performance, because they must in any case process the application layer header, so in this document we focus only on layer 3/4 balancers.

4. Applying the Flow Label to L3/L4 Load Balancing

The suggested model for using the flow label to enhance a L3/L4 load balancing mechanism is as follows:

- o We are only concerned with IPv6 traffic in which the flow label value has been set according to [RFC6437]. If the flow label of an incoming packet is zero, load balancers will continue to use the transport header in the traditional way. As the use of the flow label becomes more prevalent according to RFC 6434, load

balancers, and therefore users, will reap a growing performance benefit.

- o If the flow label of an incoming packet is non-zero, layer 3/4 load balancers can use the 2-tuple {source address, flow label} as the session key for whatever load distribution algorithm they support. Alternatively, they might use the 3-tuple {dest address, source address, flow label}, especially if the server farm supports multiple server IP addresses, but this does not affect the argument. If any IPv6 extension headers, including fragment headers, are present, this will be significantly quicker than searching for the transport port numbers later in the packet. Moreover, the transport layer information such as the source port is not repeated in fragments, which generally prevents stateless load balancers from supporting fragmented traffic since they generally cannot reassemble fragments.

A stateless layer 3/4 load balancer would simply apply a hash algorithm to the 2-tuple or 3-tuple on all packets, in order to select the same target server consistently for a given flow. Needless to say, the hash algorithm has to be well chosen for its purpose, but this problem is common to several forms of stateless load balancing. The discussion in [RFC6438] applies.

A stateful layer 3/4 load balancer would apply its usual load distribution algorithm to the first packet of a session, and store the {tuple, server} association in a table so that subsequent packets belonging to the same session are forwarded to the same server. Thus, for all subsequent packets of the session, it can ignore all IPv6 extension headers, which should lead to a performance benefit. Whether this benefit is valuable will depend on engineering details of the specific load balancer.

Note that such a balancer will not identify new transport sessions from the same source that use the same flow label; they will be delivered to the same server. This is like the behavior of existing hash-based layer 4 balancers that always send similarly hashed packets to the same destination. However, a global state table in a flow label balancer cannot be shared between multiple services if these services rely on transport layer information, since the goal of using the flow label is to avoid looking up that information.

A related issue is that the balancer will not detect FIN/ACK sequences at the end of sessions. Therefore, it will rely on inactivity timers to delete session state. However, all existing balancers must maintain such timers to deal with hung sessions, and the practical impact on memory utilisation is unlikely to be significant.

- o Layer 3/4 balancers that redirect the incoming packets by NAT are not expected to obtain any saving of time by using the flow label, because they have no choice but to follow the extension header chain, in order to locate and modify the port number and transport checksum. The same would apply to balancers that perform TCP state tracking for any reason.
- o Note that correct handling of ICMPv6 for Path MTU Discovery requires the layer 3/4 balancer to keep state for the client source address, independently of either the port numbers or the flow label.
- o SSL and HTTP proxies, if present, should forward the flow label value towards the server. This usually has no performance benefit, but is consistent with the general RFC 6437 model for the flow label.

It should be noted that the performance benefit, if any, depends entirely on engineering trade-offs in the design of the L3/L4 balancer. An extra test is needed (is the label non-zero?), but if there is a non-zero label, all logic for handling extension headers can be skipped except for the first packet of a new flow. Since the identifying state to be stored is only the tuple and the server identifier, storage requirements will be reduced. Additionally, the method will work for fragmented traffic and for flows where the transport information is missing (unknown transport protocol) or obfuscated (e.g., IPsec). Traffic reaching the load balancer via a VPN is particularly prone to the fragmentation issue, due to MTU size issues. For some load balancer designs, these are very significant advantages.

In the unlikely event of two simultaneous flows from the same source address having the same flow label value, the two flows would end up assigned to the same server, where they would be distinguished as normal by their port numbers. There are approximately one million possible flow label values, and if the rules for flow label generation [RFC6437] are followed, this would be a statistically rare event, and would not damage the overall load balancing effect. Moreover, with a million possible label values, it is very likely that there will be many more flow label values than servers at most sites, so it is already expected that multiple flow label values will end up on the same server for a given client IP address.

In the case that many thousands of clients are hidden behind the same large-scale NAT (network address and port translator) with a single shared IP address, the assumption of low probability of conflicts might become incorrect, unless flow label values are random enough to avoid following similar sequences for all clients. This is not expected to be a factor for IPv6 anyway, since there is no need to implement large-scale NAT with address sharing [RFC4864]. The probability of conflicts is low for sites that implement network prefix translation [RFC6296], since this technique provides a different address for each client.

5. Security Considerations

Security aspects of the flow label are discussed in [RFC6437]. As noted there, a malicious source or man-in-the-middle could disturb load balancing by manipulating flow labels. This risk already exists today where the source address and port are used as hashing key in layer 3/4 load balancers, as well as where a persistence cookie is used in HTTP to designate a server. It even exists on layer 3 components which only rely on the source address to select a destination, making them more DDoS-prone. Nevertheless, all these methods are currently used because the benefits for load balancing and persistence hugely outweigh the risks. The flow label does not significantly alter this situation.

Specifically, the standard [RFC6437] states that "stateless classifiers should not use the flow label alone to control load distribution, and stateful classifiers should include explicit methods to detect and ignore suspect flow label values." The former point is answered by also using the source address. The latter point is more complex. If the risk is considered serious, the site ingress router or the layer 3/4 balancer should use a suitable heuristic to verify incoming flows with non-zero flow label values. If a flow from a given source address and port number does not have a constant flow label value, it is suspect and should be dropped. This would deal with both intentional and accidental changes to the flow label.

A malicious source or man-in-the-middle could generate a flow in which the flow label is constant but the transport port numbers in some packets are invalid. Such packets, if load-balanced only on the basis of the flow label, could reach the target server and create a single-source DOS attack on its TCP engine.

RFC 6437 notes in its Security Considerations that if the covert channel risk is considered significant, a firewall might rewrite non-zero flow labels. As long as this is done as described in RFC 6437, it will not invalidate the mechanisms described above.

The flow label may be of use in protecting against distributed denial of service (DDOS) attacks against servers. As noted in RFC 6437, a source should generate flow label values that are hard to predict, most likely by including a secret nonce in the hash used to generate each label. The attacker does not know the nonce and therefore has no way to invent flow labels which will all target the same server, even with knowledge of both the hash algorithm and the load balancing algorithm. Still, it is important to understand that it is always trivial to force a load balancer to stick to the same server during an attack, so the security of the whole solution must not rely on the unpredictability of the flow label values alone, but should include defensive measures like most load balancers already have against abnormal use of source address or session cookies.

New flows are assigned to a server according to any of the usual algorithms available on the load balancer (e.g., least connections, round robin, etc.). The association between the source address/flow label value and the server is stored in a table (often called stick table) so that future traffic from the same source using the same flow label can be sent to the same server. This method is more robust against a loss of server and also makes it harder for an attacker to target a specific server, because the association between a flow label value and a server is not known externally.

In the case that a stateless hash function is used to assign client packets to specific servers, it may be advisable to use a cryptographic hash function of some kind, to ensure that an attacker cannot predict the behaviour of the load balancer.

6. IANA Considerations

This document requests no action by IANA.

7. Acknowledgements

Valuable comments and contributions were made by Fred Baker, Olivier Bonaventure, Ben Campbell, Lorenzo Colitti, Linda Dunbar, Donald

Eastlake, Joel Jaeggli, Gurudeep Kamat, Warren Kumari, Julia Renouard, Julius Volz, and others.

This document was produced using the xml2rfc tool [RFC2629].

8. Change log [RFC Editor: Please remove]

draft-ietf-intarea-flow-label-balancing-03: IESG comments, 2013-11-01.

draft-ietf-intarea-flow-label-balancing-02: Last Call comments, 2013-10-07.

draft-ietf-intarea-flow-label-balancing-01: clarifications based on WG comments, 2013-05-25.

draft-ietf-intarea-flow-label-balancing-00: WG adoption, minor WG comments, 2013-01-15.

draft-carpenter-flow-label-balancing-02: updates based on external review, 2012-12-05.

draft-carpenter-flow-label-balancing-01: update following comments, 2012-06-12.

draft-carpenter-flow-label-balancing-00: restructured after IETF83, 2012-05-08.

draft-carpenter-v6ops-label-balance-02: clarified after WG discussions, 2012-03-06.

draft-carpenter-v6ops-label-balance-01: updated with community comments, additional author, 2012-01-17.

draft-carpenter-v6ops-label-balance-00: original version, 2011-10-13.

9. References

9.1. Normative References

- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
- [RFC6434] Jankiewicz, E., Loughney, J., and T. Narten, "IPv6 Node Requirements", RFC 6434, December 2011.
- [RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, November 2011.

9.2. Informative References

- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [RFC2991] Thaler, D. and C. Hopps, "Multipath Issues in Unicast and Multicast Next-Hop Selection", RFC 2991, November 2000.
- [RFC4864] Van de Velde, G., Hain, T., Droms, R., Carpenter, B., and E. Klein, "Local Network Protection for IPv6", RFC 4864, May 2007.
- [RFC6294] Hu, Q. and B. Carpenter, "Survey of Proposed Use Cases for the IPv6 Flow Label", RFC 6294, June 2011.
- [RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, June 2011.
- [RFC6436] Amante, S., Carpenter, B., and S. Jiang, "Rationale for Update to the IPv6 Flow Label Specification", RFC 6436, November 2011.
- [RFC6438] Carpenter, B. and S. Amante, "Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels", RFC 6438, November 2011.
- [Tarreau] Tarreau, W., "Making applications scalable with load balancing", 2006, <http://lwt.eu/articles/2006_lb/>.

Authors' Addresses

Brian Carpenter
Department of Computer Science
University of Auckland
PB 92019
Auckland 1142
New Zealand

Email: brian.e.carpenter@gmail.com

Sheng Jiang
Huawei Technologies Co., Ltd
Q14, Huawei Campus
No.156 Beiqing Road
Hai-Dian District, Beijing 100095
P.R. China

Email: jiangsheng@huawei.com

Willy Tarreau
HAProxy, Inc.
R&D Network Products
3 rue du petit Robinson
78350 Jouy-en-Josas
France

Email: w@lwt.eu

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 11, 2014

S. Jiang
Y. Yin
Huawei Technologies Co., Ltd
B. Carpenter
Univ. of Auckland
October 08, 2013

Network Configuration Negotiation Problem Statement and Requirements
draft-jiang-config-negotiation-ps-01

Abstract

This document describes a problem statement and general requirements for distributed autonomous configuration of multiple aspects of networks, in particular carrier networks. The basic model is that network elements need to negotiate configuration settings with each other to meet overall goals. The document describes a generic negotiation behavior model. The document also reviews whether existing management and configuration protocols may be suitable for autonomic networks.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 11, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Requirements and Application Scenarios for Network Devices Negotiation	3
2.1. Negotiation between downstream and upstream network devices	4
2.2. Negotiation between peer network devices	5
2.3. Negotiation between networks	5
2.4. Information and status query among devices	5
2.5. Unavoidable configuration	6
3. Existing protocols	6
4. A Behavior Model of a Generic Negotiation Protocol	7
5. Security Considerations	10
6. IANA Considerations	10
7. Acknowledgements	11
8. Change Log [RFC Editor please remove]	11
9. Informative References	11
Authors' Addresses	11

1. Introduction

The success of IP and the Internet has made the network model very complicated, and networks have become larger and larger. The network of a large ISP typically contains more than a hundred thousand network devices which play many roles. The initial setup configuration, dynamic management and maintenance, troubleshooting and recovery of these devices have become a huge outlay for network operators. Particularly, these devices are managed by many different staff requiring very detailed training and skills. The coordination of these staff is also difficult and often inefficient. There are therefore increased requirements for autonomy in the networks. [I-D.boucadair-network-automation-requirements] is one of the attempts to describe such requirements. It listed a "requirement for a protocol to convey configuration information towards the managed entities". However, the present document is going further by requiring a configuration negotiation protocol rather than unidirectional provisioning.

Autonomic operation means network devices could decide configurations by themselves. There are already many existing internal implementations or algorithms for a network device to decide or

compute its configuration according to its own status, often referred to as device intelligence. In one particular area, routing protocols, distributed autonomous configuration is a well established mechanism. The question is how to extend autonomy to cover all kinds of configuration, not just routing tables.

However, in order to make right or good decisions, the network devices need to know more information than just routes from the relevant or neighbor devices. There are dependencies between such information and configurations. Currently, most of these configurations require manual coordination and operation.

Today, there is no generic negotiation protocol that can be used to control decision processes among distributed devices or between networks. Proprietary network management systems are widely used but tend to be hierarchical systems ultimately relying on a console operator and a central database. An autonomous system needs to be less hierarchical and with less dependence on an operator. This requires network elements to negotiate directly with each other, with an absolute minimum or zero configuration data at the installation stage.

This document analyzes the requirements for a generic negotiation protocol and the application scenarios, then gives considerations for detailed technical requirements for designing such a protocol. Some existing protocols are also reviewed as part of the analysis. A protocol behavior model, which may be used to define such a negotiation protocol, is also described.

2. Requirements and Application Scenarios for Network Devices Negotiation

Routing protocols are a typical autonomic model based on distributed devices. But routing is mainly one-way information announcement (in both directions), rather than bi-directional negotiation. Its only focus is reachability. The future networks need to be able to manage many more dimensions of the network, such as power saving, load balancing, etc. The current routing protocols only show simple link status, as up or down. More information, such as latency, congestion, capacity, and particularly available throughput, is very helpful to get better path selection and utilization rate.

A negotiation model with no human intervention is needed when the coordination of multiple devices can provide better overall network performance.

A negotiation model provides a possibility for forecasting. A "dry run" becomes possible before the concrete configuration takes place.

Another area is tunnel management, with automatic setup, maintenance, and removal. A related area is ad hoc routes, without encapsulation, to handle specific traffic flows (which might be regarded as a form of software defined networking).

When a new user or device comes online, it might be necessary to set up resources on multiple relevant devices, coordinated and matched to each other so that there is no wasted resource. Security settings might also be needed to allow for the new user/device.

Status information and traffic metrics need to be shared between nodes for dynamic adjustment of resources.

Troubleshooting should be as autonomous as possible. Although it is far from trivial, there is a need to detect the "real" breakdown amongst many alerts, and then take action to reconfigure the relevant devices. Again, routing protocols have done this for many years, but in an autonomous network it is not just routing that needs to reconfigure itself after a failure.

2.1. Negotiation between downstream and upstream network devices

The typical scenario is that there is a new access gateway, which could be a wireless base station, WiFi hot spot, Data Center switch, VPN site switch, enterprise CE, home gateway, etc. When it is plugged into the network, bi-direction configuration/control is needed. The upstream network needs to configure the device, its delegated prefix(es), DNS server, etc. For this direction, DHCP might be suitable and sufficient. However, there is another direction: the connection of downstream devices also needs to trigger the upstream devices, for example the provider edge, to create a corresponding configuration, by setting up a new tunnel, service, authentication, etc.

Furthermore, after the communication between gateway and provider has been established, the devices would like to optimize their configurations interactively according to dynamic link status or performance measurements, power consumption, etc. For dynamical management and maintenance, there are many other network events that downstream network devices may need to report to upstream network devices and initiate some configuration change on these upstream networks. Currently, these kinds of synchronizing operations require the involvement of human operators.

Similar requirements can also appear between other types of downstream and upstream network devices.

2.2. Negotiation between peer network devices

Within a large network, in many segments, there are network devices that are of equal importance. They have a peer rather than hierarchical relationship. There are many horizontal traffic flows or tunnels between them. In order to make their connection efficient, their configurations have to match each other. Any change of their configuration may request synchronization with their peer network devices.

However, in many cases, the peer network devices may not be able to make the exact changes as requested. Instead, another slightly different change may be the best choice for optimal performance. In order to decide on this best choice, multiple rounds of information exchange between peers may be necessary. This should be done without requiring the involvement of human operators. To provide this ability, a mechanism for network devices to be able to negotiate with each other is needed.

2.3. Negotiation between networks

A network may announce some information about its internal capabilities to connected peer networks, so that the peer networks can react accordingly. BGP routing information is a simple example.

Beyond reachability, more information may enable better coordination among networks. Examples include traffic engineering among multiple connections between two networks, particularly when these connections are geographically distributed; dynamic bandwidth adjustment to match changing traffic from a peer network; dynamic establishment and adjustment of differentiated service classes to support Service Level Agreements; and so on.

2.4. Information and status query among devices

In distributed routers, many data such as status indicators or traffic measurements are dynamically changing. These may be the triggers for follow-up negotiation. For example, consider two routers sharing traffic load. Router A may request the traffic situation of router B, then start negotiation, such as requesting router B to handle all traffic, so that router A can enter power-saving mode. Another example is that a device may request its neighbor to send a forecast or dry-run result based on a given potential configuration change. Then, the initiating router can evaluate whether the potential configuration change could meet its original target.

2.5. Unavoidable configuration

Even with autonomous negotiation, some initial configuration data cannot be avoided in some devices. A design goal is to reduce this to an absolute minimum. This information may have to be pre-configured on the device before it has been deployed physically, and is typically static. A preliminary list of unavoidable configuration data is:

- o Authentic identity for each device. This may be a public key or a signed certification. This is necessary to protect the infrastructure against unauthorized replacement of equipment.
- o The role/function and capability of the device. The role/function may depend on the network planning. The capability is typically decided by the hardware.
- o On the network edge, the routers may need to be configured with the identity of each peer provider, and their entitlements to service.

Ideally, everything else (topology, link capacity, address prefixes, shared resources, customer authentication and authority, etc.) will be discovered or negotiated autonomously according to general policy for various negotiated objectives.

3. Existing protocols

Routing protocols are mainly one-way information announcements. The receiver makes decisions independently, based on the received information, and there is not much feedback information to the announcing peer. This remains true even though the protocol is used in both directions between peer routers; there is no negotiation, and each peer runs its route calculations independently.

There are many existing protocols that have some minor negotiation abilities, such as Dynamic Host Configuration Protocol for IPv6 (DHCPv6) [RFC3315], Neighbor Discovery (ND) [RFC4861], Port Control Protocol (PCP) [RFC6887], etc. Numerous other configuration or management protocols could be listed. However, they are either simple request/response models or can only negotiate on very limited aspects. Negotiation is a feature of certain connectivity protocols, for example Point to Point Protocol (PPP) [RFC1661] and Transport Layer Security (TLS) [RFC5246] but again, the negotiable aspects are strictly limited.

At present, there appears to be no generic protocol that meets the requirements discussed above.

4. A Behavior Model of a Generic Negotiation Protocol

This section describes a behavior model and some considerations for designing a generic negotiation protocol, which would act as a platform for different negotiation objectives.

- o A generic platform

The design of the network device protocol is desired to be a generic platform, which is independent from the negotiation contents. It should only take care of the general intercommunication between negotiation counterparts. The negotiation contents will vary according to the various negotiation objectives and the different pairs of negotiating counterparts.

- o Security infrastructure and trust relationship

Because this negotiation protocol may directly cause changes to device configurations and bring significant impacts to a running network, this protocol must be based on a restrictive security infrastructure. It should be carefully managed and monitored so that every device in this negotiation system behaves well and remains well protected.

On the other hand, a limited negotiation model might be deployed based on a limited trust relationship. For example, between two administrative domains, devices may also exchange limited information and negotiate some particular configurations based on a limited conventional or contractual trust relationship.

- o The uniform pattern for negotiation contents

The negotiation contents should be defined according to a uniform pattern. They could be carried either in TLV (Type, Length and Value) format or in payloads described by a flexible language, like XML. A protocol design should choose one of these two. The format must be extensible for unknown future requirements.

- o A simple initiator/responder model

Multiple-party negotiations are too complicated to be modeled and there may be too many dependencies among the parties to converge efficiently. A simple initiator/responder model is more feasible and could actually complete multiple-party negotiations by indirect steps. Naturally this process must be guaranteed to terminate and must contain tie-breaking rules.

- o Organizing of negotiation content

Naturally, the negotiation content should be organized according to the relevant function or service. The content from different functions or services should be kept independent from each other. They should not be combined into a single option or single session because these contents may be negotiated with different counterparts or may be different in response time.

- o Topology neighbor device discovery

Every network device that supports the negotiation protocol is a responder and always listens to a well-known (UDP?) port. A well-known link-local multicast address should be defined for discovery purposes. Upon receiving a discovery or request message, the recipient device should return a message in which it either indicates itself as a proper negotiation counterpart or diverts the initiator towards another more proper device.

- o Self aware network devices

Every network device should be pre-configured with its role and functions and be aware of its own capabilities. The roles may be only distinguished because of network behaviors, which may include forwarding behaviors, aggregation properties, topology location, bandwidth, tunnel or translation properties, etc. The role and function may depend on the network planning. The capability is typically decided by the hardware or firmware. It is the foundation of the negotiation behavior of a specific device.

- o Requests and responses in negotiation procedures

The initiator should be able to negotiate with its relevant negotiation counterpart devices, which may be different according to the negotiation objective. It may request relevant information from the negotiation counterpart so that it can decide its local configuration to give the most coordinated performance. It may request the negotiation counterpart to make a matching configuration in order to set up a successful communication with it. It may request certain simulation or forecast results by sending some dry run conditions.

Beyond the traditional yes/no answer, the responder should be able to reply with a suggested alternative if its answer is 'no'. This is going to start a bi-direction negotiation towards reaching a compromise between the two devices.

- o Convergence of negotiation procedures

The negotiation procedure should move towards convergent results. It means that when a responder makes a suggestion of a changed condition in a negative reply, it should be as close as possible to the original request or previous suggestion. The suggested value of the third or later negotiation steps should be chosen between the suggested values from the last two negotiation steps. In any case there must be a mechanism to guarantee rapid convergence in a small number of steps.

- o Dependencies of negotiation

In order to decide a configuration on a device, the router may need information from neighbor routers. This can be reached through the above negotiation procedure. However, a certain information on a neighbor router may depend on other information from its neighbors, which may need another negotiation procedure to obtain or decide. Therefore, there are dependencies among negotiation procedures. There need to be clear edge/convergence for these negotiation dependencies. Also some mechanisms are needed to avoid loop dependencies.

- o End of negotiation

A single negotiation procedure also needs ending conditions if it does not converge. A limit round number, for example three, should be set on the devices. It may be an implementation choice or a configurable parameter. However, the protocol design needs to clearly specify this, so that the negotiation can be terminated properly. In some cases, a timeout might be needed to break off a dependency negotiation.

- o Failed negotiation

There must be a well-defined procedure for concluding that a negotiation cannot succeed, and if so deciding what happens next (deadlock resolution, tie-breaking, or revert to best effort service).

- o Policy constraints

There must be provision for general policy rules to be applied by all devices in the network (e.g., security rules, prefix length, resource sharing rules). However, policy distribution might not use the negotiation protocol itself.

- o Management monitoring, alerts and intervention

Devices should be able to report to a monitoring system. Some events must be able to generate operator alerts and some provision for emergency intervention must be possible (e.g. to freeze negotiation in a mis-behaving device). These features may not use the negotiation protocol itself.

5. Security Considerations

This document does not include a detailed threat analysis for autonomous configuration, but it is obvious that a successful attack on autonomic nodes would be extremely harmful, as such nodes might end up with a completely undesirable configuration. A concrete protocol proposal will therefore require a threat analysis, and some form of strong authentication and, if possible, built-in protection against denial of service attacks.

Separation of network devices and user devices may become very helpful in this kind of scenario.

Also, security configuration itself should become autonomic whenever possible. However, in the security area at least, operator override of autonomic configuration must be possible for emergency use.

As noted earlier, a cryptographically authenticated identity for each device is needed in an autonomic network. It is not safe to assume that a large network is physically secured against interference or that all personnel are trustworthy. Each autonomous device should be capable of proving its identity and authenticating its messages. One approach would be to use a private/public key pair and sufficiently strong cryptography. Each device would generate its own private key, which is never exported from the device. The device identity and public key would be recorded in a network-wide database. The alternative of using symmetric keys (shared secrets) is less attractive, since it creates a risk of key leakage as well as a key management problem when devices are installed or removed.

Generally speaking, no personal information is expected to be involved in the negotiation protocol, so there should be no direct impact on personal privacy. Nevertheless, traffic flow paths, VPNs, etc. may be negotiated, which could be of interest for traffic analysis. Also, carriers generally want to conceal details of their network topology and traffic density from outsiders. Therefore, since insider attacks cannot be prevented in a large carrier network, the security mechanism for the negotiation protocol needs to provide message confidentiality.

6. IANA Considerations

This draft does not request any IANA action.

7. Acknowledgements

The authors want to thank Zhenbin Li, Bing Liu for valuable comments.

This document was produced using the xml2rfc tool [RFC2629].

8. Change Log [RFC Editor please remove]

draft-jiang-negotiation-config-ps-01, add more requirements, and add more considerations for behavior model, 2013-10-08.

draft-jiang-negotiation-config-ps-00, original version, 2013-06-29.

9. Informative References

- [I-D.boucadair-network-automation-requirements]
Boucadair, M. and C. Jacquenet, "Requirements for Automated (Configuration) Management", draft-boucadair-network-automation-requirements-01 (work in progress), June 2013.
- [RFC1661] Simpson, W., "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, July 1994.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [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.
- [RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC6887] Wing, D., Cheshire, S., Boucadair, M., Penno, R., and P. Selkirk, "Port Control Protocol (PCP)", RFC 6887, April 2013.

Authors' Addresses

Sheng Jiang
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: jiangsheng@huawei.com

Yuanbin Yin
Huawei Technologies Co., Ltd
Q15, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: yinyuanbin@huawei.com

Brian Carpenter
Department of Computer Science
University of Auckland
PB 92019
Auckland 1142
New Zealand

Email: brian.e.carpenter@gmail.com

DNS Extensions
INTERNET-DRAFT
Updates RFC 2845 (if approved)
Intended Status: Standards Track
Expires: March 27, 2014

H. Rafiee
M. v. Loewis
C. Meinel
Hasso Plattner Institute
September 27, 2013

Transaction SIGNature (TSIG) using CGA Algorithm in IPv6
<draft-rafi-ee-intarea-cga-tsig-06.txt>

Abstract

This document describes a new mechanism that can be used to reduce the need for human intervention during DNS authentication and secure DNS authentication in various scenarios such as the DNS authentication of resolvers to stub resolvers, authentication during zone transfers, authentication of root DNS servers to recursive DNS servers, and authentication during the FQDN (RFC 4703) update.

Especially in the last scenario, i.e., FQDN, if the node uses the Neighbor Discovery Protocol (NDP) (RFC 4861, RFC 4862), unlike the Dynamic Host Configuration Protocol (DHCP) (RFC 3315), the node has no way of updating his FQDN records on the DNS and has no means for a secure authentication with the DNS server. While this is a major problem in NDP-enabled networks, this is a minor problem in DHCPv6. This is because the DHCP server updates the FQDN records on behalf of the nodes on the network.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 27, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved. This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Conventions used in this document	3
3. Terminology	3
4. Problem Statement	5
5. CGA-TSIG Applications	5
5.1. IP Spoofing	6
5.2. DNS Dynamic Update Spoofing	6
5.3. Resolver Configuration Attack	7
5.4. Exposing Shared Secret	7
5.5. Replay attack	7
6. Algorithm Overview	7
6.1. The CGA-TSIG DATA structure	7
6.2. Generation of CGA-TSIG DATA	9
7. Authentication During Zone Transfer	11
7.1. Verification process	12
8. Authentication During the FQDN or PTR Update	13
8.1. Verification Process	14
9. Authentication During Query Resolving (stub to recursive)	14
9.1. Verification process	15
10. Authentication During Query Resolving (Auth. to recursive)	16
11. No cache parameters available or SeND is not supported	16
12. Security Considerations	16
13. IANA Considerations	17
14. Appendix	18
15. Acknowledgements	19
16. References	19
16.1. Normative	19
16.2. Informative	20
Authors' Addresses	22

1. Introduction

Transaction SIGNature (TSIG) [RFC2845] is a protocol that provides endpoint authentication and data integrity through the use of one-way hashing and shared secret keys in order to establish a trust relationship between two/group of hosts, which can be either a client and a server, or two servers. The TSIG keys, which are manually exchanged between a group of hosts, need to be maintained in a secure manner. This protocol is today mostly used to secure a Dynamic Update, or to give assurance to the slave name server that the zone transfer is from the original master name server and that it has not been spoofed by hackers. It does this by verifying the signature using a cryptographic key that is shared with the receiver.

It is possible to extend the TSIG protocol through the use of newly defined algorithms. This document proposes the use of Cryptographically Generated Addresses (CGA) [RFC3972] as a new algorithm in the TSIG Resource Record (RR). CGA is an important option available in Secure Neighbor Discovery (SeND) [RFC3971], which provides nodes with the necessary proof of IP address ownership by providing a cryptographic binding between a host and its IP address without the need for the introduction of a new infrastructure. CGA is a one-way hashing algorithm used to generate Interface IDs for IPv6 addresses in a secure manner. An interface ID consists of the rightmost 64 bits of the 128 bit IPv6 address. CGA verifies the ownership of the sender's IP address by finding a relationship between the sender's IP address and his public key [1,2].



Figure 1 IPv6 addresses

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC 2119 significance.

3. Terminology

The terms used in this document have the following standard meaning:

- Name server: A server that supports DNS service.
- Resolver/recursive DNS server: A resolver/recursive name server responds to queries where the query does not contain an entry for the node in its database. It first checks its own records and cache for the answer to the query and then, if it cannot find an answer there, it may recursively query name servers higher up in the hierarchy and then pass the response back to the originator of the query. This is known as a recursive query or recursive lookup.
- Stub resolver: A specific kind of DNS resolver that is unable to resolve the queries recursively. So, it relies on a recursive DNS resolver to resolve the queries.
- Authoritative: An authoritative name server provides the answers to DNS queries. For example, it would respond to a query about a mail server IP address or website IP address. It provides original, first-hand, definitive answers (authoritative answers) to DNS queries. It does not provide 'just cached' answers that were obtained from another name server. Therefore it only returns answers to queries about domain names that are installed in its system configuration.

There are two types of Authoritative Name Servers:

1. Master server (primary name server): A master server stores the original master copies of all zone records. A host master is only allowed to change the master server's zone records. Each slave server gets updated via a special automatic updating mechanism within the DNS protocol. All slave servers maintain identical copies of the master records.

2. Slave server (secondary name server): A slave server is an exact replica of the master server. It is used to share the DNS server's load and to improve DNS zone availability in cases where the master server fails. It is recommended that there be at least 2 slave servers and one master server for each domain name.

- Root DNS server: An authoritative DNS server for a specific root domain. For example, .com
- Client: a client can be any computer (server, laptop, etc) that only supports stub DNS servers and not other DNS services. It can be a mail server, web server or a laptop computer.
- Node: a node can be anything such as a client, a DNS server (resolver, authoritative) or a router.
- Host: all nodes except routers

4. Problem Statement

The authentication during any DNS query process is solely based on the source IP address when no secure mechanism is in use either during the DNS update (zone transfer, FQDN update) or during the DNS query resolving process. This makes the DNS query process vulnerable to several types of spoofing attacks -- man in the middle, reflector, source IP spoofing, etc. One example is the problem that exists between a client and a DNS resolver. When a client sends a DNS query to a resolver, an attacker can send a response to this client containing the spoofed source IP address for this resolver. The client checks the resolver's source IP address for authentication. If the attacker spoofed the resolver's IP address, and if the attacker responds faster than the legitimate resolver, then the client's cache will be updated with the attacker's response. The client does not have any way to authenticate the resolver.

If DNSSEC (RFC 6840) or TSIG, as a security mechanism is in use, then the problem would be the manual step required for the configuration. For instance when a DNSSEC needs to sign the zone offline. TSIG secures this process by providing the transaction level authentication necessary by the use of a shared secret. But, the current problem with using TSIG is that manual processing is required in order to generate and exchange the shared secrets. This is because, in TSIG, the shared secret exchange is done offline. Currently there is little deployment of TSIG for resolver authentication with clients. One reason is that resolvers respond to anonymous queries and can be located in any part of the network. A second reason is that the manual TSIG process makes it difficult to configure each new client with the shared secret of the resolver. Another catastrophic problem with TSIG would be when this shared secret, that is shared between a group of hosts, leaks and makes it necessary to repeat this manual step. The reason is, that for each group of hosts there needs to be one shared secret and the administrator will need to manually add it to the DNS configuration file for each of these hosts. This manual process will need to be invoked in the case where one of these hosts is compromised and the shared secret is well known to the attacker. It will also have to be invoked in the case where any of these hosts needs to change their IP addresses, because of different reasons such as privacy issues, as explained in RFC 4941 [RFC4941], or when moving to another subnet within the same network, etc. Therefore, the problem that exists today with the authentication processes used in different scenarios is what this document addresses. The various scenarios include authentication during zone transfer, authentication of the nodes during DNS query resolving and authentication during updating PTR and FQDN (RFC 4703).

5. CGA-TSIG Applications

The purpose of CGA-TSIG [7] is to minimize the amount of human intervention required to accomplish shared secret or key exchange and, as a byproduct, to reduce the process's vulnerability to attacks introduced by human errors (during changing the DNS configuration) when Secure Neighbor Discovery (SeND) is used for addressing purposes or when SeND is not available for use.

As explained in a prior section, CGA-TSIG can be used in different scenarios. For the FQDN update scenario CGA-TSIG is useful in dynamic networks where the nodes want to change their IP addresses frequently in order to maintain privacy. If the Dynamic Host Configuration Protocol (DHCP) is in use, then the DHCP server can do this update on behalf of the nodes in this network on a DNS server but in Neighbor Discovery Protocol (NDP), there is no feature available that allows the host security update process for its own FQDN. CGA-TSIG can be a solution.

For the resolver scenario, usually the the resolver can add the TSIG Resource Record (RR) to the DNS query response and use the CGA-TSIG algorithm in order to permit a useful authentication of the result. CGA-TSIG assures the client that the query response comes from the true originator and not from an attacker. It also ensures the integrity of the data by signing the data.

There are several types of attack that CGA-TSIG can prevent. Here we will evaluate some of them. The use of CGA-TSIG will also reduce the number of messages needed in exchange between a client and a server in order to establish a secure channel. To exchange the shared secret between a DNS resolver and a client, when TSIG is used, a minimum of four messages are required for the establishment of a secure channel. Modifying RFC 2845 to use CGA-TSIG will decrease the number of messages needed in this exchange. The messages used in RFC 2930 (TKEY RR) are not needed when CGA-TSIG is used.

5.1. IP Spoofing

During the DNS Update process or the query resolving process it is important that both communicating parties know that the one that they are communicating with is the actual owner of that IP address and that the messages are not being sent from a spoofed IP address. This can be accomplished by the use of the CGA algorithm which utilizes the node for IP address verification of other nodes.

5.2. DNS Dynamic Update Spoofing

Dynamic Update Spoofing is eliminated because the signature contains both the CGA parameters and the DNS update message. This will offer proof of the sender's IP address ownership (CGA parameters) and the

validity of the update message.

5.3. Resolver Configuration Attack

When using CGA-TSIG, the DNS server, or the client, would not need further configuration. This would reduce the possibility of human errors being introduced into the DNS configuration file. Since this type of attack is predicated on human error, the chances of it occurring, when this extension is used, are minimized.

5.4. Exposing Shared Secret

Using CGA-TSIG will decrease the number of manual steps required in generating the new shared secret and in exchanging it among the hosts where the old shared secret was shared between them for updating purposes. This manual step is required after a leakage has occurred of the shared secret to an attacker via any of these hosts.

5.5. Replay attack

Using the Time Signed value in the signature modifies the content of the signature each time the node generates and sends it to the DNS server. If the attacker tries to spoof this value with another timestamp, to show that the update message is current, the DNS server checks this message by verifying the signature. In this case, the verification process will fail thus also preventing the replay attack.

6. Algorithm Overview

The following sections explain the use of CGA or any other future algorithm in place of CGA for securing the DNS process by adding a CGA-TSIG data structure as an option to the TSIG Resource Record (RR).

6.1. The CGA-TSIG DATA structure

The CGA-TSIG data structure SHOULD be added to the Other DATA section of the RDATA field in the TSIG Resource Record (RR) (see figures 2 and 3). The DNS RRTYPE must be set to TSIG [RFC2845]. The RDATA Algorithm Name MUST be set to CGA-TSIG. A detailed explanation of the standard RDATA fields can be found in section 2.3 RFC 2845. This document focuses only on the new structure added to the Other DATA

section. These new fields are CGA-TSIG Len and CGA-TSIG DATA. The TSIG RR is added to an additional section of the DNS message. If another algorithm is used in place of CGA for SeND, such as SSAS [4 , 5], then the CGA-TSIG Len will be the length for the parameters of this algorithm and CGA-TSIG DATA will consist of the parameters required for verification of that algorithm, like signature, public key, etc.

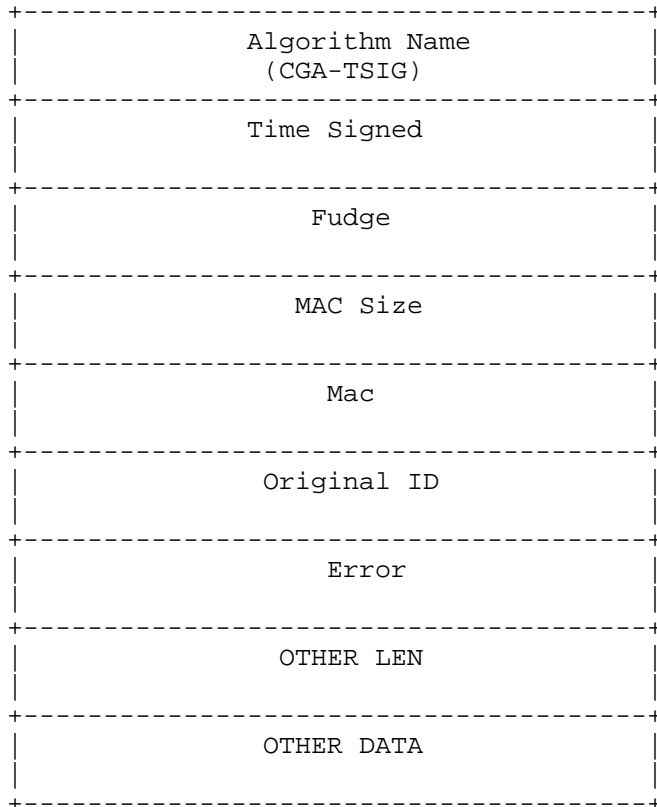
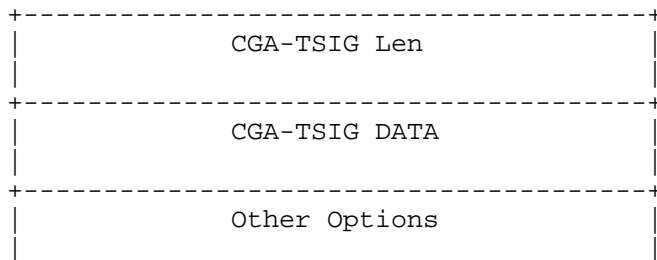


Figure 2 Modified TSIG RDATA

The CGA-TSIG DATA Field and the CGA-TSIG Len will occupy the first two slots of Other DATA. Figure 3 shows the layout.



+-----+

Figure 3 Other DATA section of RDATA field

CGA-TSIG DATA Field Name	Data Type	Notes
Algorithm type	u_int16_t	Name of the algorithm [RFC3972] RSA (by default) CGA
type	u_int16_t	Name of the algorithm used in SeND
IP tag	16 octet	the tag used to identify the IP address
Parameters Len	Octet	the length of CGA parameters
Parameters	variable	CGA parameters Section 3 RFC 3972
Signature Len	Octet	the length of CGA signature
Signature	variable	Section 3.2.1 This document
old pubkey Len	variable	the length of old public key field
old pubkey	variable	Old public key
old Signature Len	variable	the length of old signature field
old Signature	variable	Old signature generated by old public key.

Type indicates the Interface ID generation algorithm that was used in SeND. This field allows for the use of future, optional algorithms in SeND. The default value for CGA is 1. The IP tag is a node's old IP address. A client's public key can be associated with several IP addresses on a server. The DNS server, or the DNS message verifier node, SHOULD store the IP addresses and the public keys so as to indicate their association to each other. If a client wants to add RRs to the server by using a new IP address, then the IP tag field will be set to binary zeros. The server will then store the new IP address that was passed to it in storage. If the client wants to replace an existing IP address in a DNS server with a new one, then the IP tag field will be populated with the IP address which is to be replaced. The DNS server will then look for the IP address referenced by the IP tag stored in its storage and replace that IP address with the new one. This enables the client to update his own RRs using multiple IP addresses while, at the same time, giving him the ability to change IP addresses. If a node changes its public key in order to maintain privacy, then it MUST add the old public key to the old pubkey field. It MUST also retrieve the current time from Time Signed field, sign it using the old private key, and then add the digest (signature) to the old signature field. This enables the verifier node to authenticate a host with a new public key. The detailed verification steps are explained in sections 5.1, 6.1 and 7.1.

6.2. Generation of CGA-TSIG DATA

In order to use CGA-TSIG as an authentication approach, some of the parameters need to be cached during IP address generation. If no parameters are available in cache, please see section 8. If the Type (section 4.1) is CGA, then the parameters that SHOULD be cached are the modifier, algorithm type, location of the public/private keys and the IP addresses of this host generated by the use of CGA.

1. Obtain required parameters from cache.

The CGA-TSIG algorithm obtains the old IP address, modifier, subnet prefix, and public key from cache. It concatenates the old IP address with the CGA parameters, i.e., modifier, subnet prefix, public key and collision count (the order of CGA parameters are shown in section 3 RFC 3972). If the old IP address is not available, then CGA-TSIG must set the old IP address (IP tag) to zero.

Note: If the node is a DNS server (resolver or authoritative DNS server) and it does not support SeND, but the goal is to use this algorithm, then it is possible to use a script to generate the CGA parameters, which are needed to manually configure this server's IP address. Then this server can make use of these parameters for authentication purposes.

2. Generate signature

For signature generation, all CGA parameters (modifier, public key, collision count and subnet prefix), that are concatenated with the DNS update message, the IP tag and the Time Signed field, are signed by using a RSA algorithm, the default, or any future algorithm used in place of RSA, and the private key which was obtained from cache in the first step. This signature must be added to the signature field of the CGA-TSIG DATA. Time Signed is the same timestamp as is used in RDATA. This value is the number of seconds since 1 January 1970 in UTC obtained from the signature generator. This approach will prevent replay attacks by changing the content of the signature each time a node wants to send a DNS message. The format of DNS messages is explained in section 4.1.2 RFC 1035 [RFC1035].

Algorithm Name
Type
IP tag (16 bytes)
Parameter Len (1 byte)
Parameters

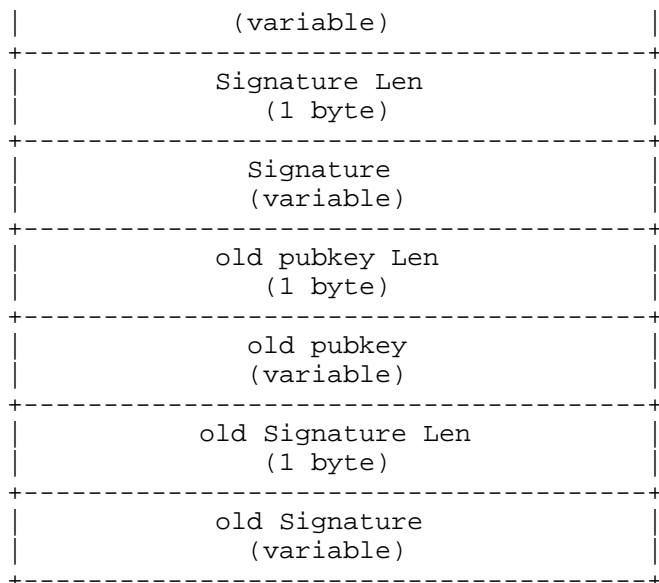


Figure 4 CGA-TSIG DATA Field

3. Generate old signature

If the nodes generated new key pairs, then they need to add the old public key and message, signed by the old private key, to CGA-TSIG DATA. A node will retrieve the timestamp from Time Signed, will use the old private key to sign it, and then will add the content of this signature to the old signature field of CGA-TSIG DATA. This step **MUST** be skipped when the node did not generate new key pairs.

7. Authentication During Zone Transfer

This section discusses the use of CGA-TSIG for the authentication of two DNS servers (a master and a slave). In the case of processing a DNS update for multiple DNS servers (authentication of two DNS servers), there are two possible scenarios with regard to the authentication process, which differs from that of the authentication of a node (client) with one DNS server. This is because of the need for human intervention.

a. Add the DNS servers' IP address to a slave configuration file

A DNS server administrator should only manually add the IP address of the master DNS server to the configuration file of the slave DNS server. When the DNS update message is processed, the slave DNS server can authenticate the master DNS server based on the source IP address and then, prove the ownership of this address by use of the CGA-TSIG option from the TSIG RR. This scenario will be valid until the IP address in any of these DNS servers changes.

To automate this step's process, the DNS Update message sender's public key must be saved on the other DNS server, after the source IP address has been successfully verified for the first time. In this case, when the sender generates a new IP address by executing the CGA algorithm using the same public key, the other DNS server can still verify it and add its new IP address to the DNS configuration file automatically.

b. Retrieve public/private keys from a third party Trusted Authority (TA)

The message exchange option of SeND [RFC3971] may be used for the retrieval of the third party certificate. This may be done automatically from the TA by using the Certificate Path Solicitation and the Certificate Path Advertisement messages. Like in scenario b, the certificate should be saved on the DNS server for later use for the generation of its address or for the DNS update process. In this case, whenever any of these servers want to generate a new IP address, then the DNS update process can be accomplished automatically without the need for human intervention.

7.1. Verification process

Sender authentication is necessary in order to prevent attackers from making unauthorized modifications to DNS servers through the use of spoofed DNS messages. The verification process executes the following steps:

1. Execute the CGA verification

These steps are found in section 5 RFC 3972. If the sender of the DNS message uses another algorithm, instead of CGA, then this step becomes the verification step for that algorithm. If the verification process is successful, then step 2 will be executed. Otherwise the message will be discarded without further action.

2. Check the Time Signed

The Time Signed value is obtained from TSIG RDATA and is called t1. The current system time is then obtained and converted to UTC time and is called t2. If t1 is in the range of t2 and t2 minus x minutes (see formula 1, x minutes may vary according to transmission lag time) then step 3 will be executed. Otherwise, the message will be considered a spoofed message and the message should be discarded without further action. The range is used in consideration of the delays that can occur during its transmission over TCP or UDP. Both times must use UTC time in order to avoid differences in time based on different geographical locations.

$$t2-x \leq t1 \leq t2 \quad (1)$$

3. Verify the signature

The signature contained in CGA-TSIG DATA should be verified. This can be done by retrieving the public key and signature from CGA-TSIG DATA and using this public key to verify the signature. If the verification process is successful, then step 4 will be executed. If the verification fails, then the message should be discarded without further action.

4. Verify the source IP address

The source IP address of the Update requester MUST be checked against the one contained in the DNS configuration file. If it is the same, then the Update Message should be processed, otherwise, step 5 will be executed.

5. Verify the public key

The DNS server checks whether or not the public key retrieved from CGA-TSIG DATA is the same as what was available in the storage where the public keys and IP addresses were saved. If no entry is found in storage for this public key, then the update will be rejected without further action. Otherwise, when the old public key length is not zero go to step 6.

6. Verify the old public key

If the old public key length is zero, then skip this step and discard the DNS update message without further action. If the old public key length is not zero, then the DNS server will retrieve the old public key from CGA-TSIG DATA and will check to see whether or not it is the same as what was saved in the DNS server's storage where the public keys and IP addresses are stored. If it is the same, then step 6 will be executed, otherwise the message should be discarded without further action.

7. Verify the old signature

The old signature contained in CGA-TSIG DATA should be verified. This can be done by retrieving the old public key and the old signature from CGA-TSIG DATA and then using this old public key to verify the old signature. If the verification is successful, then the Update Message should be processed and the new public key should be replaced with the old public key in the DNS server. If the verification process fails, then the message should be discarded without further action.

8. Authentication During the FQDN or PTR Update

Normally the DHCPv6 server will update the client's RRs on their

behalf In the scenario where SeND is used as a secure NDP, the nodes will need to do this process themselves unless there is stateless DHCPv6 server available. CGA-TSIG can be used To give nodes the ability of doing this process themselves. In this case the clients need to include the CGA-TSIG option in order to allow the DNS server to verify them. The verification process is the same as that explained in section except for step 4.

8.1. Verification Process

The verification steps are the same as those is explained in section 5.1, but removing step 4 and modifying step 5.

- 1- Execute the CGA verification
- 2- Check the Time Signed
- 3- Verify the signature
4. Verify the public key

The DNS server checks whether or not the public key retrieved from CGA-TSIG DATA is the same as what was available in the storage where the public keys and IP addresses were saved. If no entry is found in storage for this public key, and the FQDN or PTR is also not available in the DNS server, then the DNS server will store the public key of this node in his database and add this node's PTR and FQDN. Otherwise if any PTR is available, and the node IP tag is empty, or there is currently another public key associated with the node's FQDN, then the update will be rejected without further action. Otherwise go to step 5 when the old public key length is not zero.

- 5- Verify the public key
- 6- Verify the old public key
- 7- Verify the old signature

9. Authentication During Query Resolving (stub to recursive)

A DNS query request sent by a host, such as a client or a mail server, does not need to include CGA-TSIG DATA because the resolver responds to anonymous queries. But the resolver's response SHOULD contain the CGA-TSIG DATA field in order to enable this client to verify him.

In generation of the CGA-TSIG for a resolver, there is no need to include the IP tag. This is because resolvers don't usually have several IP addresses so the client does not need to keep several IP

addresses for the same resolver.

9.1. Verification process

When a resolver responds to the host's query request for the first time, the client saves its public key in a file. This allows the client to verify this resolver when it changes its IP address due to privacy or security concerns. The first 2 steps of the verification process are the same as those steps explained in section 5.1 These steps are as follows:

1. Execute the CGA verification
2. Check the Time Signed
3. Verify the Source IP address

If the resolver's source IP address is the same as that which is known for the host, then step 4 will be executed. Otherwise the message SHOULD be discarded without further action.

4. Verify the signature

The signature contained in CGA-TSIG DATA should be verified. This can be done by retrieving the public key and signature from CGA-TSIG DATA and using this public key to verify the signature. If the verification process is successful, then step 5 will be executed. If the verification fails, then the message should be discarded without further action.

5. Verify the public key

The host checks whether or not the public key retrieved from CGA-TSIG DATA matches any public key that was previously saved in the storage where the public keys and IP addresses of resolvers are saved. If there is a match, then the message is processed. If not, then step 5 will be executed.

5. Verify the old public key

If the old public key length is zero, then skip this step and discard the DNS query response without further action. If the old public key length is not zero, then the host will retrieve the old public key from CGA-TSIG DATA and will check whether or not it is the same as what was saved in the host's storage where the public keys and IP addresses are stored. If it is the same, then step 6 will be executed, otherwise the message should be discarded without further action.

6. Verify the old signature

The old signature contained in CGA-TSIG DATA should be verified. This can be done by retrieving the old public key and old signature from CGA-TSIG DATA and then using this old public key to verify the old signature. If the verification is successful, then the DNS Message should be processed and the new public key should be replaced with the old public key of the resolver in the host. If the verification process fails, then the message should be discarded without further action.

10. Authentication During Query Resolving (Auth. to recursive)

This verification step in the authentication of authoritative to recursive DNS server is the same as that explained in section 7.1. In this case the recursive DNS server does not need to include CGA-TSIG, but the root DNS server does need to include it in order to enable the recursive DNS server to verify it.

11. No cache parameters available or SeND is not supported

In the case where there are no cache parameters available during the IP address generation, there are then two scenarios that come into play here. In the first scenario there is the case where the sender of a DNS message needs to generate a key pair and generate the CGA-TSIG data structure as explained in section 4. The node SHOULD skip the first section of the verification processes explained in section 5.1 , section 6.1 and section 7.1.

In the second scenario, as explained in section 4.2 (step 1), it is not necessary for the server to support the SeND or CGA algorithm. The DNS administrator can make a one time use of a CGA script to generate the CGA parameters and then manually configure the IP address of this DNS server. Then later, this DNS server can use those values as a means for authenticating other nodes. The verifier nodes also do not necessarily need to support SeND. They only need to support CGA-TSIG.

12. Security Considerations

The approach explained in this draft, CGA-TSIG, is a solution for securing DNS messages from spoofing type attacks like those explained in section 3.

A problem that may arise here concerns attacks against the CGA algorithm. In this section we will explain the possibility of such attacks against CGA [5] and explain the available solutions that we considered in this draft.

a) Discover an Alternative Key Pair Hashing of the Victim's Node Address

In this case an attacker would have to find an alternate key pair hashing of the victim's address. The probability for success of this type of attack will rely on the security properties of the underlying hash function, i.e., an attacker will need to break the second pre-image resistance of that hash function. The attacker will perform a second pre-image attack on a specific address in order to match other CGA parameters using Hash1 and Hash2. The cost of doing this is $(2^{59}+1) * 2^{(16*1)}$. If the user uses a sufficient security level, it will be not feasible for an attacker to carry out this type of attack due to the cost involved. Changing the IP address frequently will also decrease the chance for this type of attack succeeding.

b) DoS to Kill a CGA Node

Sending a valid or invalid CGA signed message with high frequency across the network can keep the destination node(s) busy with the verification process. This type of DoS attack is not specific to CGA, but it can be applied to any request-response protocol. One possible solution, to mitigate this attack, is to add a controller to the verifier side of the process to determine how many messages a node has received over a certain period of time from a specific node. If a determined threshold rate is exceeded, then the node will stop further receipt of incoming messages from that node.

c) CGA Privacy Implication

Due to the high computational complexity necessary for the creation of a CGA, it is likely that once a node generates an acceptable CGA it will continue its use at that subnet. The result is that nodes using CGAs are still susceptible to privacy related attacks. One solution to these types of attacks is setting a lifetime for the address as explained in RFC 4941.

13. IANA Considerations

The IANA has allowed for choosing new algorithm(s) for use in the TSIG Algorithm name. Algorithm name refers to the algorithm described in this document. The requirement to have this name registered with

IANA is specified.

In section 4.1, Type should allow for the use of future optional algorithms with regard to SeND. The default value for CGA might be 1. Other algorithms would be assigned a new number sequentially. For example, a new algorithm called SSAS [4,5] could be assigned a value of 2.

14. Appendix

- A sample key storage for CGA-TSIG

```
create table cgatsigkeys (  
  id          INT auto_increment,  
  pubkey      VARCHAR(300),  
  primary key(id)  
);  
  
create table cgatsigips (  
  id          INT auto_increment,  
  idkey       INT,  
  IP          VARCHAR(20),  
  FOREIGN KEY (idkey) REFERENCES cgatsigkeys(id)  
  primary key(id)  
);
```

CGA-TSIG tables on mysql backend database

- a sample format of stored parameters in the node

For example, the modifier is stored as bytes and each byte might be separated by a comma (for example : 284,25,14,...). Algorithmtype is the algorithm used in signing the message. Zero is the default algorithm for RSA. Secval is the CGA Sec value that is, by default, one. GIP is the global IP address of this node (for example: 2001:abc:def:1234:567:89a). oGIP is the old IP address of this node, before the generation of the new IP address. Keys contains the path

where the CGA-TSIG algorithm can find the PEM format used for the public/private keys (for example: /home/myuser/keys.pem).

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<Details>
```

```
<CGATSIG>
```

```
<modifier value=""/>
```

```
<algorithmtype value="0"/>
```

```
<secval value="1"/>
```

```
<GIP value=""/>
```

```
<oGIP value=""/>
```

```
<Keys value=""/>
```

```
</CGATSIG>
```

```
</Details>
```

XML file contains the cached DATA

15. Acknowledgements

The author would like to thank all those people who directly helped in improving this draft and all supporters of this draft, especially Ralph Droms, Andrew Sullivan, Ted Lemon, Brian Haberman.

16. References

16.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)," RFC 3972, March 2005.
- [RFC3971] Arkko, J., Kempf, J., Zill, B., and P. Nikander, "Secure Neighbor Discovery (SEND)", RFC 3971, March 2005.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

- [RFC2930] Eastlake 3rd, D., "Secret Key Establishment for DNS (TKEY RR)", RFC 2930, September 2000.
- [RFC1035] Mockapetris, P., "Domain Names - Implementation And Specification", RFC 1035, November 1987.
- [RFC4941] Narten, T., Draves, R., Krishnan, S., "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, September 2007.
- [RFC2136] Vixie, P. (Editor), Thomson, S., Rekhter, Y., Bound, J., "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997.
- [RFC2845] Vixie, P., Gudmundsson, O., Eastlake 3rd, D., Wellington, B., "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, May 2000.

16.2. Informative References

- [1] Aura, T., "Cryptographically Generated Addresses (CGA)", Lecture Notes in Computer Science, Springer, vol. 2851/2003, pp. 29-43, 2003.
- [2] Montenegro, G. and Castelluccia, C., "Statistically Unique and Cryptographically Verifiable (SUCV) Identifiers and Addresses," ISOC Symposium on Network and Distributed System Security (NDSS 2002), the Internet Society, 2002.
- [3] AlSa'deh, A., Rafiee, H., Meinel, C., "IPv6 Stateless Address Autoconfiguration: Balancing Between Security, Privacy and Usability". Lecture Notes in Computer Science, Springer(5th International Symposium on Foundations & Practice of Security (FPS). October 25 - 26, 2012 Montreal, QC, Canada), 2012.
- [4] Rafiee, H., Meinel, C., "A Simple Secure Addressing Generation Scheme for IPv6 AutoConfiguration (SSAS)". Work in progress, <http://tools.ietf.org/html/draft-rafi-6man-ssas>, 2013.
- [5] Rafiee, H., Meinel, C., "A Simple Secure Addressing Scheme for IPv6 AutoConfiguration (SSAS)", 11th International conference on Privacy, Security and Trust (IEEE PST), 2013.
- [6] AlSa'deh, A., Rafiee, H., Meinel, C., "Cryptographically Generated Addresses (CGAs): Possible Attacks and Proposed Mitigation Approaches," in proceedings of 12th IEEE International Conference on Computer and Information Technology (IEEE CIT'12), pp.332-339, 2012.
- [7] Rafiee, H., Meinel, C., "A Secure, Flexible Framework for DNS Authentication in IPv6 Autoconfiguration" in proceedings of The 12th IEEE International Symposium on Network Computing and

Applications (IEEE NCA13), 2013.

Authors' Addresses

Hosnieh Rafiee
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Str. 2-3
Potsdam, Germany
Phone: +49 (0)331-5509-546
Email: ietf@rozanak.com

Dr. Christoph Meinel
(Professor)
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Str. 2-3
Potsdam, Germany
Email: meinel@hpi.uni-potsdam.de

Dr. Martin von Loewis
(Professor)
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Str. 2-3
Potsdam, Germany

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 24, 2014

Q. Sun
China Telecom
W. Liu
C. Zhou
Huawei Technologies
October 21, 2013

Problem Statement for Openv6 Scheme
draft-sun-openv6-problem-statement-00

Abstract

The IPv6 transition has been an ongoing process throughout the world due to the exhaustion of the IPv4 address space. However, this transition leads to costly end-to-end network upgrades and poses new challenges of managing a large number of devices with a variety of transitioning protocols. While IPv6 transition tools exist, there are still new issues exist. Operators may need various types of IPv6 transition technologies depending on performance requirements, deployment scenarios, etc.

To address these difficulties, a unifying approach would provide a unified way to deploy IPv6 in a cost-effective, flexible manner.

This document describes issues for deploying multiple transition technologies during the whole IPv6 transition period. It also discusses potential technical gaps to achieve this work.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 24, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Unified Openv6 Use case	3
3.1. Evolve from one Scenario to Another	3
3.2. Multiple Transition Mechanisms Co-Exist	4
3.3. Scattered Address Pool Management	5
3.4. Extensibility	5
4. Coexistence of IPv6 Transition Technologies	6
5. Security Considerations	7
6. Acknowledgements	7
7. References	7
7.1. Normative References	7
7.2. Informative References	7
Authors' Addresses	9

1. Introduction

The exhaustion of the IPv4 address space has been a practical problem that providers are facing today. Network address migration to IPv6 is ongoing or upcoming throughout the world. However, IPv6 requires costly end-to-end network upgrades and different network scenarios will co-exist during IPv6 transition. Therefore, operators have to upgrade network devices to support different transition tools, or buy new devices for different scenarios.

It is more efficient to unify the transition technologies thanks to a Transition Data Plane (TDP). The TDP deals with flow-table mapping and is unaware of specific transition technologies. The service-awareness is provided by a Transition Management Server (TMS). The TMS uses IPv6 Transition Service Modules (ITSM) to know the characteristics of each technology. Finally, the TMS provides a Northbound Interface (NBI) that enables the ITSM to manipulate the traffic for different scenarios. This approach unifies the variety of IPv6 transition mechanisms in a cost-effective, flexible manner.

This document describes issues arising from deploying multiple transition technologies during the whole IPv6 transition period. It also discusses potential technical gaps for the unified solution.

2. Terminology

3. Unified Openv6 Use case

3.1. Evolve from one Scenario to Another

During the IPv6 transition period, the network needs three stages of IPv4-only, dual-stack and IPv6-only. The networks should support both IPv4 services and IPv6 services at each stage.[One-vision-for-IPv6]

There are multiple IPv6 transition technologies for different network scenarios (e.g. IPv4 network for IPv4/IPv6 user access, IPv6 network for IPv4/IPv6 user access, IPv4 servers for IPv6 visitors, etc.). Different network scenarios will co-exist during IPv6 transition which means the IPv6 transition device should support multiple IPv6 transition technologies. The following are possible scenarios in the whole IPv6 transition period.

- 1)Scenario 1: IPv6 host visit IPv6 servers via IPv4 access network
- 2)Scenario 2: IPv4 host visit IPv4 servers via IPv4 NAT Dual-stack network
- 3)Scenario 3: IPv6 host visit IPv6 servers via IPv6 network
- 4)Scenario 4: IPv4 host visit IPv4 servers via IPv6 access network
- 5)Scenario 5: IPv6 host visit IPv6 servers via IPv4 access network
- 6)Scenario 6: IPv4 host and IPv6 host interaction

It is not necessary that every operator will go through each scenario one by one. For example, some operators may start from scenario 1,

and some may start directly from scenario 2 or scenario 4. However, since the final stage (target) is the IPv6-only access network, one still need to undergo multiple scenarios from the long term.

In such a case, the operator should either upgrade existing devices to support new features, or replace them with new ones. In particular, when the operator's network consists of devices from different vendors, it is hard to guarantee that all the legacy devices can be upgraded at the same time. This is costly and complicated.

The issues are:

1. How to manipulate Transition Data Plane(TDP) with different modes?
2. How to identify the capabilities of different transition devices ?
3. How does the Transition Data Plane (TDP) identify different modes in the unified platform ?

3.2. Multiple Transition Mechanisms Co-Exist

In transition from one scenario to another, different transition mechanisms may have different impacts on user experience. For example, DS-Lite would have some impact due to address sharing compared to 6rd mechanisms, and NAT64 would have extra impact due to ALG issue. Operators may prepare a fallback mechanism to guarantee the same level of user experience when there are complaints from subscribers. Therefore, it is required to support multiple transition mechanisms in the same area.

Another use case is that multiple scenarios may exist in the same stage. For example, if there are both IPv6-only devices and IPv4-only host in the same area with limited public IPv4 address, both NAT64 and NAT44 (or DS-Lite) are required to achieve IPv4 service connectivity.

Current implementations normally use a separate instance for each mechanism,, and additional policies need to be applied when running multiple mechanisms in one device. Some have a limitation on the number of policies which can be configured in one device, while some have restrictions regarding the resource occupation (e.g. one transition instance will occupy a static amount of memory). The major challenges of IPv6 deployment mainly lie in two aspects:

The need to implement different IPv6 transition technologies in the same hardware and the need to support this by upgrading network devices as little as possible.

The need to hop over legacy infrastructures which are not IPv6 enabled, costly or impossible to upgrade.

The issues are:

1. How to support multiple transition mechanisms in a cost-efficient and flexible way ?
2. How to easily identify the transition type of different subscribers ?

3.3. Scattered Address Pool Management

When operators are facing with address shortage problem, the remaining IPv4 address pools are usually quite scattered. It is quite complicated for an operator to manage scattered address pools in many transition devices. The situation will become even worse when multiple transition mechanisms in the same device need to be configured with different address pools. Besides, since the occupation of the address pools may vary during different transition periods, (e.g. when there is not many IPv6-enabled services and IPv6-enabled devices, IPv4 traffic will still occupy a great portion of the total traffic, while in the later stage of IPv6 transition, IPv4 traffic will decrease and the amount of IPv4 address pools will decrease accordingly.

The ideal way is to manage the address pools centrally. Different transition mechanisms can require the address pools on-demand. For example, when one transition mechanism is running out of the current address pools, it may request a additional address pool. It can also release the address pools that it is not using anymore. In this way, operators do not need to configure the address pools one by one manually and it also helps using the address pools more efficiently.

The issues are:

1. How to configure the address pools for different mechanisms ?
2. How to collect the current status of address pool usage ?

3.4. Extensibility

In migration from IPv4 to IPv6, different scenarios usually need different solutions. Although IETF has already invented some

mechanisms including DS-Lite[RFC6333], XLate[RFC6146], BIH[RFC6535], NAT64[RFC6146], etc., However, the current solutions have solved the following scenarios only in a limited way:

*IPv4 client communicates to IPv6 server

*IPv4 client communicates to IPv6 peer

It is possible that new technologies will be invented in the future. In addition, some mechanism are still evolving from a session-based solution (e.g. DS-Lite) to a more scalable way (e.g. lw4over6, MAP). It might be possible that operators who have already deployed one solution may upgrade to a better one in the future. Besides, IPv6 transition can also be regarded as a virtualized network function which can be offered to a third-party.

Therefore, it is required to offer an open and programmable way to easily add new features without modifying existing device hardware.

The issues are :

1. How to add a new module to the transition data plane ?
 2. How to offer the capability for the possible future technologies?
4. Coexistence of IPv6 Transition Technologies

The different network environments (architecture, scale, services deployed, varying IP traffic) cause a variety of IPv6 transition technologies for different operators. This section analyzes the current and future coexistence of IPv6 transition technologies situation as well as the issues behind it.

Since IPv6 was proposed, there have been a couple of RFCs and on-going documents in IETF, as listed in the table below.

status	number	documents
RFC	8 or more	[RFC5571], [RFC6333], [RFC6674], [RFC5969], [RFC6219], [RFC6535], [RFC6654], [RFC6145], ...
WG draft	6 or more	[I-D.ietf-softwire-4rd], [I-D.ietf-softwire-map], [I-D.ietf-softwire-map-t], [I-D.ietf-softwire-public-4over6], [I-D.ietf-softwire-lw4over6], [I-D.ietf-v6ops-464xlat], ...
Active draft	several	...

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

Table 1: A Table of IPv6 Transition Technologies @ IETF

The situation described above depicts the difficulty of selecting appropriate IPv6 transition technologies for the carriers. Moreover, according to [SD-NAT], there are multiple stages during the whole IPv6 transition period, and a variety of technologies and equipments are used during different IPv6 transition stages. To protect the user experience and the early investment, an operator will not upgrade its network directly to the final stage of IPv6 transition. During different IPv6 transition stages, an operator needs different technologies in different stages. Thus, a method that is able to implement different IPv6 transition technologies in the same hardware is crucial, to avoid repeated investments.

5. Security Considerations

6. Acknowledgements

N/A.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

7.2. Informative References

[I-D.ietf-softwire-4rd]
Despres, R., Jiang, S., Penno, R., Lee, Y., Chen, G., and M. Chen, "IPv4 Residual Deployment via IPv6 - a Stateless Solution (4rd)", draft-ietf-softwire-4rd-07 (work in progress), October 2013.

[I-D.ietf-softwire-lw4over6]
Cui, Y., Qiong, Q., Boucadair, M., Tsou, T., Lee, Y., and I. Farrer, "Lightweight 4over6: An Extension to the DS-Lite Architecture", draft-ietf-softwire-lw4over6-01 (work in progress), July 2013.

[I-D.ietf-softwire-map-t]
Li, X., Bao, C., Dec, W., Troan, O., Matsushima, S., and T. Murakami, "Mapping of Address and Port using Translation (MAP-T)", draft-ietf-softwire-map-t-04 (work in progress), September 2013.

- [I-D.ietf-softwire-map]
Troan, O., Dec, W., Li, X., Bao, C., Matsushima, S.,
Murakami, T., and T. Taylor, "Mapping of Address and Port
with Encapsulation (MAP)", draft-ietf-softwire-map-08
(work in progress), August 2013.
- [I-D.ietf-softwire-public-4over6]
Cui, Y., Wu, J., Wu, P., Vautrin, O., and Y. Lee, "Public
IPv4 over IPv6 Access Network", draft-ietf-softwire-
public-4over6-10 (work in progress), July 2013.
- [I-D.ietf-v6ops-464xlat]
Mawatari, M., Kawashima, M., and C. Byrne, "464XLAT:
Combination of Stateful and Stateless Translation", draft-
ietf-v6ops-464xlat-10 (work in progress), February 2013.
- [One-vision-for-IPv6]
Mark Townsley, "One vision for IPv6", .
- [RFC5571] Storer, B., Pignataro, C., Dos Santos, M., Stevant, B.,
Toutain, L., and J. Tremblay, "Softwire Hub and Spoke
Deployment Framework with Layer Two Tunneling Protocol
Version 2 (L2TPv2)", RFC 5571, June 2009.
- [RFC5969] Townsley, W. and O. Troan, "IPv6 Rapid Deployment on IPv4
Infrastructures (6rd) -- Protocol Specification", RFC
5969, August 2010.
- [RFC6145] Li, X., Bao, C., and F. Baker, "IP/ICMP Translation
Algorithm", RFC 6145, April 2011.
- [RFC6219] Li, X., Bao, C., Chen, M., Zhang, H., and J. Wu, "The
China Education and Research Network (CERNET) IVI
Translation Design and Deployment for the IPv4/IPv6
Coexistence and Transition", RFC 6219, May 2011.
- [RFC6333] Durand, A., Droms, R., Woodyatt, J., and Y. Lee, "Dual-
Stack Lite Broadband Deployments Following IPv4
Exhaustion", RFC 6333, August 2011.
- [RFC6535] Huang, B., Deng, H., and T. Savolainen, "Dual-Stack Hosts
Using "Bump-in-the-Host" (BIH)", RFC 6535, February 2012.
- [RFC6654] Tsou, T., Zhou, C., Taylor, T., and Q. Chen, "Gateway-
Initiated IPv6 Rapid Deployment on IPv4 Infrastructures
(GI 6rd)", RFC 6654, July 2012.

- [RFC6674] Brockners, F., Gundavelli, S., Speicher, S., and D. Ward,
"Gateway-Initiated Dual-Stack Lite Deployment", RFC 6674,
July 2012.
- [RFC6674] Brockners, F., Gundavelli, S., Speicher, S., and D. Ward,
"Gateway-Initiated Dual-Stack Lite Deployment", RFC 6674,
July 2012.
- [SD-NAT] Alain Durand, "SD-NAT", ,
<<http://www.ietf.org/proceedings/82/slides/behave-10.pdf>>.

Authors' Addresses

Qiong Sun
China Telecom
No.118 Xizhimennei street, Xicheng District
Beijing 100035
P.R. China

Email: sunqiong@ctbri.com.cn

Will(Shucheng) Liu
Huawei Technologies
Bantian, Longgang District
Shenzhen 518129
P.R. China

Email: liushucheng@huawei.com

Cathy Zhou
Huawei Technologies
Bantian, Longgang District
Shenzhen 518129
P.R. China

Email: cathy.zhou@huawei.com