Dibbler – a portable Dynamic Host Configuration Protocol for IPv6 implementation

T. Mrugalski*
*Gdansk University of Technology, Faculty of Electronics, Telecommunications and Informatics, Gdansk, Poland
tomasz.mrugalski@eti.pg.gda.pl

***Abstract*— One of the major advantages of the IPv6 protocol family is an extensive set of autoconfiguration mechanisms. One of them is a stateful autoconfiguration called Dynamic Host Configuration Protocol for IPv6, often abbreviated as DHCPv6. This paper presents Dibbler, a highly portable DHCPv6 implementation. Design goals, current state of implementation and future improvement areas are thoroughly discussed. During two years of development, implementation, testing and usage various deficiencies of the DHCPv6 protocol were discovered. In the latter part of the paper several improvement suggestions and extensions are proposed: a method to use DHCPv6 on routerless networks and a way to bypass DNS update mechanism during Fully Qualified Domain Name retrieval for improved efficiency on clients with limited capabilities. A new proposal regarding booting disk-less IPv6 capable devices using proposed DHCPv6 enhancement is also described.**

I. Introduction

One of the major improvements in the IPv6, compared to the IPv4 protocol, is the ability to perform various tasks in an automated manner. Dynamic Host Configuration Protocol for IPv6, ofter abbreviated as DHCPv6, is a stateful autoconfiguration protocol for IPv6 networks. After several years of discussions and 28 draft revisions[1] it was published in June 2003 by the Internet Engineering Task Force as RFC3315 document [1]. This protocol take advantage of various IPv6 features, such as multicast [4] or IPv6 address state [2]. Stateful autoconfiguration is considered a more advanced in comparison to the stateless address autoconfiguration [2]. It offers several IPv6 address lease and renewal mechanisms as well as additional configuration options distribution. This later capability is considered the most powerful aspect of the whole protocol.

DHCPv6 is expected to be widely deployed in almost all IPv6 local area networks, especially when automatic configuration is expected. It is worth noting that even in the simple case of a host being used to browse web sites, stateless autoconfiguration is not sufficient. Missing component is a DNS server address. Without DHCPv6, it must be configured statically. This approach raises number of disadvantages. Before entering normal service, manual configuration is required. This also applies to the visiting mobile nodes. Should DNS server fail or change address, all hosts in the whole network must be reconfigured manually. All visiting nodes also have to be reconfigured manually, which appears as additional difficulty. Therefore providing addresses of the available DNS servers is being expected the most common usage of the DHCPv6.

Although stateless address autoconfiguration (SAA) supported by all IPv6 hosts provides methods for obtaining IPv6 addresses, stateful configuration offers additional advantages for administrators. SAA does not offer any authentication mechanisms. It is also fully autonomous without any possibility to manage of gather statistics. Network administrators cannot reject unwanted nodes nor keep track of hosts being configured. Both of this disadvantages are solved in the stateful configuration.

Since DHCPv6 standard publication, various improvements and enhancements were proposed. Some of them (e.g. DNS and domain name distribution mechanism) are published as RFC documents, while others are being actively discussed (e.g. option renewal mechanism). Although first draft of the DHCPv6 specification is dated back to 1995, there are surprisingly few implementations. Even fewer are up to date and pretend to be compatible with official standard. One of them is Dibbler.

II. Design requirements

The Dibbler project has been started as master thesis of the two students of the Faculty of Electronics, Telecommunications and Informatics of Gdansk University of Technology in Poland. After graduation in September 2003, Dibbler is being developed by Tomasz Mrugalski as part of his Ph.D. studies.

Before implementation phase, intensive design study was performed. Several key requirements were defined:

- Portable. DHCPv6 support on multiple platforms is poor and often non-existent, e.g. in MS Windows environment. Limiting implementation to only one or several chosen environments would greatly reduce importance of this project. Therefore produced code must be as portable as possible.
- Layered. To meet portability requirements, code is split to two layers. Higher, system independent layer is responsible for core logic of the protocol. It is implemented in C++. This language was chosen due to good object oriented mechanisms support as well as good portability. Lower layer is very small and is system dependent. It is responsible for network interface and system specific tasks, e.g. running as daemon in Linux or running as Windows service in Windows environment. This layer is coded in C.

[1]During DHCPv6 standard publication, it was the highest revision number in the IETF history.

- Extensible. The option concept in the DHCPv6 protocol was designed with future extensions in mind. Therefore implementation must be easily extensible. This was accomplished by using object oriented features such as inheritance or polymorphism. This further strengthens C++ as a language of choice.
- Open. Author strongly believes in an open source development model. Among key advantages of this approach is easy access to the source code by all interested persons. Possibility to modify the code to suit specific individual needs is another advantage. In the author's opinion, it is a duty of every scientist to share the results of his work. In computer science this can be easily accomplished by publishing source code and allowing other to exploit it. It is interesting to note that several students, as well as researchers and large industry employees from around the world have contacted author regarding various aspects of Dibbler implementation.
- Well documented. Lack of proper documentation is a common flaw among open source projects. To avoid this, Dibbler should provide extensive manuals accompanied with multiple examples,User's Guide for regular users and Developer's Guide for potential developers to ease internal structure learning process.

III. Implementation

Practical implementation has began in the second quarter of the 2003. Currently Dibbler consists of three major elements: client, server and relay. It has the capability to perform major part of all actions specified in the standard ([1]) as well as numerous([5], [10], [11], [12])
extensions:

- Server discovery. Client is capable of server detection. Should more than one server be detected, client will choose the one that suits his needs best.
- Address lease. Client can obtain one or more addresses from server. Identity Association (IA) concept is fully supported. Server as well as client can use multiple IAs.
- Multiserver support. Client can maintain simultaneous communication with with multiple servers. Such need arise when client requires C addresses, but each present server can provide only $S_i < C$ addresses.
- Address renewal. Both (renew and rebind) address renewal mechanisms are supported.
- 11 extension options. Various configuration parameters defined in drafts and standards are supported: DNS addresses, domain name [10], NTP servers, time zone, SIP domains and servers [5] and many other [14], [12].
- Stateless mode. Although DHCPv6 is considered a stateful autoconfiguration, it can be run in the stateless mode. This mode does not cover any address related tasks, so state of the client need not to be tracked. Common usage of this mode is a configuration parameters retrieval such as DNS server location or domain name. This behavior is conformant to the Stateless DHCPv6 specification [11].
- Rapid commit mode. Normal DHCPv6 operation requires four message transmissions before address and options are finally granted to the client. In some circumstances (e.g. client assumes that only one server or relay is present in the network), a rapid mode can be used. It is two times faster than normal operation.
- Unicast communication. Usually client communicates with server using multicast packets. Such packets are delivered to all nodes in the network, processed and eventually dropped by a non-members of the DHCP_ALL_RELAY_AGENTS_AND_SERVERS group. Frequent communication by multiple clients can appear as unneeded burden to all nodes present is the network. To avoid this, server can instruct client to use unicast communication.
- Relay support. Since DHCPv6 uses link-scoped multicast address, receiver of the client messages must be present on-link. Deploying servers is a larger network on all links is a major scalability issue. To deal with this problem, a relay mechanism was developed. Only a simple relay agent must be present in each network. After receiving client-originated message, data is encapsulated and forwarded to the remote server. Reply is decapsulated and sent back to the client. This advanced scenario is fully supported by the Dibbler.

One of the project goals is to speed up IPv6 adoption by providing reliable DHCPv6 solution on as many platforms as possible. Currently Windows XP/2003 and Linux 2.4 and 2.6 series are supported. Dibbler is the only one DHCPv6 solution for Windows systems publicly available and one of two available for Linux systems. It is also the only one with provides both stand-alone relay and relay support. It is worth noting that there are binaries available for non-Intel architectures. Currently Dibbler is included in the PLD Linux Distribution[2], which provides RPM packages for AMD64, Sparc, PowerPC and Alpha processors. BSD port is also planned in the near future.

First version was released in September 2003. Since November 2003, number of binary and source code downloads is being observed. It appears that downloads per month is slowly, but steadily increasing. This dependency was depicted on Fig. 1. There are two exceptions of this rule, observed in July and November 2004. There are two factors which affected those months:

- 0.2.0 and 0.3.0 versions were released on July and November 2004, respectively.
- Dibbler was presented on the kick-off meeting of Polish IPv6 Task Force. This event took place at the beginning of the November 2004.

Due to server crash, log files for January till April 2004 period were lost. Number of downloads was was linearly interpolated.

[2]PLD Linux Distribution website is available at http://www.pld-linux.org. Dibbler RPM packages are available at ftp://ftp.pld-linux.org/dists/2.0/test/

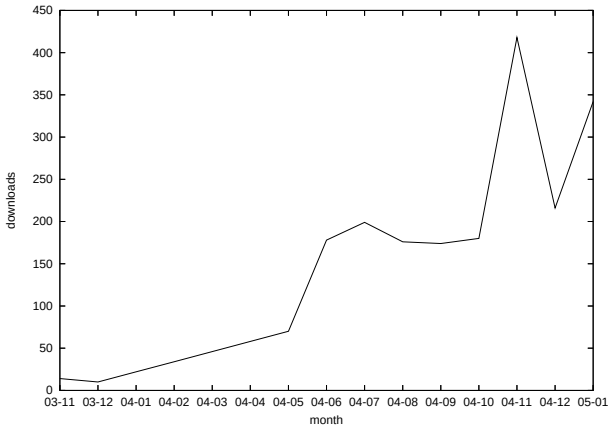

Fig. 1. Downloads per month.

Download counts are based on the analysis of the web server logs. There is no way to figure exact number of downloads, because binary and source code can be downloaded from various places, e.g. PLD repository.

Dibbler is being accepted as a mainstream solution. During the time of writing this paper[3], it was already present in the PLD Linux Distribution. Inclusion in the Debian and Gentoo distributions are also in advanced stage. It was tested by the 6NET project team. Test report [15] described Dibbler as "stable", although several deficiencies were found. All of them were fixed in the following Dibbler release.

Several features are planned in future releases. Most frequently asked functionality is security support. Other important, not yet implemented feature is an extension to provide to prefix delegations, defined in the [9]. Recently proposed FQDN retrieval/DNS Update mechanism will be implemented as well as all enhancements mentioned in the following section.

IV. Proposed features

During development, testing and usage, various deficiencies of the DHCPv6 protocol were detected. Although both protocol and its implementation is fully usable, there are several areas, which can be improved.

The major issue is lack of completeness. DHCPv6 cannot be fully operational by itself. This is best explained by example. Let's assume there are 3 hosts located in the same physical network, which want to interact. Each of them obtain unique address with the same prefix. Although this may not seem obvious, those hosts are still unable to communicate. Each of them has an address assigned to the network interface. To successfully exchange data, proper routing entries have to be set up in each host's routing table. This is usually done via Router Advertisement messages sent by the router. This appears to be a major flaw. Protocol designers assumed that DHCPv6 must coexist with a router. There is no way to use stateful autoconfiguration in the router-less network.

[3]February 2005

To overcome this problem, new option is proposed. It is being tentatively called *OPTION_IANAPREFIX*. Server can provide it as *IA_NA* suboption. It will be used to carry information about prefix. Client, after receiving *IA_NA* option, processes it in the usual way. If the *OPTION_IANAPREFIX* is provided, except normal operations, client builds proper router entry consisting of IPv6address with less significant $128-prefix$ bit masked by zeroes. For example:

```
IAADDRESS   3ffe:8320:210::1:2
IANAPREFIX  64
Route entry 3ffe:8320:210::/64
```

This option should only be allowed in the REPLY message, which is sent as response to the REQUEST message. This enhancement conforms to the general idea of DHCPv6 protocol: clients, which do not support this feature, will threat it as unknown option and will ignore it.

This option should not to be confuse with OPTION_IAPREFIX, defined in the "IA Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6" [9] document. IAPREFIX option is used in the prefix delegation mechanism used to remotely delegate prefixed to routers.

Second proposed improvement is related to the embedded device startup. Numerous embedded devices do not have permanent data storage capabilities such as hard drive or flash memory. During boot process, such device must obtain information about server capable of providing kernel images. Kernel parameters should also be provided. After retrieving those informations, client downloads kernel via TFTP protocol [3] (or other similar file transfer mechanism). Next step is to boot downloaded kernel and mount remotely a root filesystem. This is usually done using NFS protocol [8]. Details regarding root filesystem are passed as normal kernel boot options. This option is tentatively called *OPTION_BOOTPARAMS*. Its proposed format is presented on the Fig. 2.

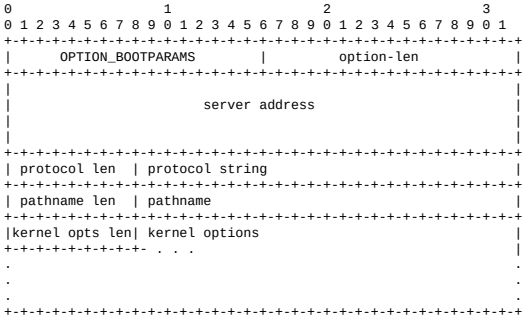

Fig. 2. BOOTPARAMS option structure.

There are several fields in this option:

- Server address – 128 bit length field containing IPv6 address of a server, which provides kernel images.
- Protocol length – 8 bit length field. It defines protocol string length.

- Protocol string – a string, which describes protocol used to retrieve kernel image, e.g. „tftp”.
- Pathname length – 8 bit length field. It defines pathname string length.
- Pathname – a string containing full path to the kernel image, e.g. „/tftpboot/images/bzImage-2.6.10”.
- Kernel opts length – 8 bit length field. It defines kernel options string length.
- Kernel options – a string containing all options passed to the kernel during boot, e.g. „ro single”

This option provides uniform mechanism to retrieve and run kernels with proper boot parameters. Referring to the protocol used to download kernel with string containing name (e.g. „tftp”) is a way to handle future, not yet defined protocols.

Third proposed enhancement is related to the mechanism proposed by Bernard Volz in a draft document entitled “The DHCPv6 Client FQDN Option” [13]. DNS ([6]) maintains (among other things) the information about mapping between hosts’ Fully Qualified Domain Names (FQDNs) and IP addresses assigned to the hosts. Mentioned draft introduced mechanism to provide FQDN for client and clarify, which side (server or client) should perform DNS update ([7]).

One case is not foreseen, however. DNS mechanism is quite complicated and its implementation can appear as major problem in devices with limited capabilities (e.g. embedded device). Sometimes there is a need to obtain fully qualified domain name. One example of such need is a array of IPv6 capable sensors, which log certain informations on central logging server. Log entry should contain logging entity’s name. Updating DNS is not necessary as there is no need to remotely access such sensors by their names. Therefore mechanism for obtaining FQDN without performing DNS update is necessary.

To enable such mechanism, additional field must be added to the option format. It is similar to the format proposed in [13], except flags field. New flags format is presented in the Fig. 3.

MBZ is a abbreviation for *Must Be Zero*. N,O and S bits meaning has not changed and is explained in the original proposal. Additional D (short for *Disable*) bit is used to completely disable DNS update mechanism. It is important to note that in the original D bit was part of the MBZ field and had to have 0 value. Therefore enhanced clients will be able to communicate with server which implements basic proposal only (and vice versa). When client sends the Client FQDN option, it sets the D bit to indicate that it will not perform any DNS updates and wishes that DHCPv6 also should not perform it. If server supports D bit, it will return Client DHCPv6 option with D bit set. Otherwise server simply ignore it, and will clear that bit. This will inform client, that its desire to not perform DNS update was ignored.

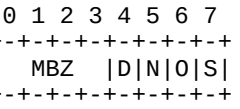

Fig. 3. Flags structure in the FQDN option.

V. Conclusions

Since official publication of the IPv6 standards in the late 1998, the rate of IPv6 adoption is very slow. It is often pointed out that there is a „vicious circle” effect. Network administrators do not provide IPv6 connectivity, because there is minor interest among end users. On the other hand, due to limited access and lack of applications, there is a low demand for IPv6 solutions. Limited awareness is also an important factor. First steps should be taken by application developers and network administrators. It is necessary to provide IPv6 support for as wide area of applications as possible. Although works on DHCPv6 protocol were started a decade ago, the most spectacular developments – both theoretical as well as practical – were made after official protocol publication.

Most of the research being done is focused on seamless IPv4 to IPv6 transition and coexistence of both protocol. IPv6-only solutions should also be examined with great scrutiny. Example of such important areas are various aspects of autoconfiguration. In a near future, number of network capable devices will rapidly increase. Such devices are expected to take full advantage of advanced IPv6 mechanisms such as stateful autoconfiguration. It is also worth noting that IPv4 protocol lasted for over two decades and IPv6 is expected to be used for at least the same amount of time. Therefore protocols and solutions designed and developed today will have a great impact of future Internet.

References

[1] R. Droms, Ed. “Dynamic Host Configuration Protocol for IPv6 (DHCPv6)”, RFC3315, IETF, July 2003
[2] S. Thomson, and T. Narten “IPv6 Stateless Address Autoconfiguration”, RFC2462, IETF, December 1998
[3] K. Sollins, “The TFTP protocol (Revision 2)”, RFC1350, IETF, July 1992
[4] S. Deering, W. Fenner, and B. Haberman “Multicast Listener Discovery (MLD) for IPv6”, RFC2710, IETF, October 1999
[5] H. Schulzrinne, and B. Volz “Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol (SIP) Servers”, RFC3319, IETF, July 2003
[6] S. Thomson, C. Huitema, V. Ksinant and M. Souissi “DNS Extensions to Support IP Version 6”, RFC3596, IETF, October 2003
[7] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound “Dynamic Updates in the Domain Name System (DNS UPDATE)”, RFC2136, IETF, April 1997
[8] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, et al. “NFS version 4 Protocol”, RFC3010, IETF, December 2000
[9] O. Troan, and R. Droms “IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6”, RFC3633, IETF, December 2003
[10] R. Droms, Ed. “DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)”, RFC3646, IETF, December 2003
[11] R. Droms, “Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6”, RFC3736, IETF, April 2004
[12] V. Kalusivalingam “Network Information Service (NIS) Configuration Options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)”, RFC3898, IETF, October 2004
[13] B. Volz, “The DHCPv6 Client FQDN Option”, work in progress, IETF, September 2004
[14] S. Venaas, T. Chown, and B. Volz “Information Refresh Time Option for DHCPv6”, work in progress, IETF, January 2005
[15] C. Schild, and Andre Stolze “DHCPv6 implementation and test report version 2”, 6NET, Deliverable 3.2.3v2, November 2004