Dibbler – a portable DHCPv6
User's guide

Tomasz Mrugalski
thomson(at)klub.com.pl

2006-10-05

0.5.0

Contents

1 Intro **4**
1.1 Overview . . . 4
1.2 Supported parameters . . . 5
1.3 Not supported features . . . 6
1.4 Requirements . . . 6
1.5 Supported platforms . . . 6

2 Installation and usage **7**
2.1 Linux installation . . . 7
2.2 Windows installation . . . 8
2.3 IPv6 support . . . 8
2.3.1 Setting up IPv6 in Linux . . . 8
2.3.2 Setting up IPv6 in WindowsXP and 2003 . . . 8
2.3.3 Setting up IPv6 in Windows 2000 . . . 9
2.3.4 Setting up IPv6 in Windows NT4 . . . 9
2.4 Compilation . . . 10
2.4.1 Linux compilation . . . 10
2.4.2 Windows XP/2003 compilation . . . 10
2.4.3 Windows NT/2000 compilation . . . 10

3 Features HOWTO **11**
3.1 Stateless vs stateful and IA, TA options . . . 11
3.2 DNS Update . . . 12
3.2.1 Example BIND configuration . . . 13
3.2.2 Dynamic DNS Testing and tips . . . 15
3.3 Server address caching . . . 16
3.4 Relays . . . 17
3.5 XML files . . . 17

4 Configuration files **17**
4.1 Data types . . . 18
4.2 Scopes . . . 18
4.3 Comments . . . 18
4.4 Client configuration file . . . 18
4.4.1 Global parameters . . . 19
4.4.2 Interface declaration . . . 19
4.4.3 IA declaration . . . 19
4.4.4 Address declaration . . . 20
4.4.5 Standard options . . . 20
4.4.6 Extension options . . . 21
4.4.7 Stateless configuration . . . 23
4.4.8 Relay support . . . 23
4.5 Client configuration examples . . . 23
4.5.1 Example 1: Default . . . 23
4.5.2 Example 2: DNS . . . 24
4.5.3 Example 3: Timeouts and specific address . . . 24
4.5.4 Example 4: Unicast, more than one address . . . 24
4.5.5 Example 5: Quick configuration using Rapid-commit . . . 25
4.5.6 Example 6: Stateless mode . . . 25

4.5.7 Example 7: Dynamic DNS (FQDN) . . . 25
4.5.8 Example 8: Interface indexes . . . 26
4.6 Server configuration file . . . 27
4.6.1 Global scope . . . 27
4.6.2 Interface declaration . . . 27
4.6.3 Class scope . . . 27
4.6.4 Standard options . . . 28
4.6.5 Addional options . . . 29
4.7 Server configuration examples . . . 30
4.7.1 Example 1: Simple . . . 31
4.7.2 Example 2: Timeouts . . . 31
4.7.3 Example 3: Limiting amount of addresses . . . 31
4.7.4 Example 4: Unicast communication . . . 32
4.7.5 Example 5: Rapid-commit . . . 32
4.7.6 Example 6: Access control . . . 33
4.7.7 Example 7: Multiple classes . . . 33
4.7.8 Example 8: Relay support . . . 34
4.7.9 Example 9: 2 relays . . . 34
4.7.10 Example 10: Dynamic DNS (FQDN) . . . 35
4.8 Relay configuration file . . . 35
4.8.1 Global scope . . . 36
4.8.2 Interface declaration . . . 36
4.8.3 Options . . . 36
4.9 Relay configuration examples . . . 36
4.9.1 Example 1: Unicast/multicast . . . 37
4.9.2 Example 2: Multiple interfaces . . . 37
4.9.3 Example 3: 2 relays . . . 38

5 Frequently Asked Questions 38
5.1 Common Questions . . . 38
5.2 Linux specific questions . . . 39
5.3 Windows specific questions . . . 39

6 Miscellaneous topics 40
6.1 History . . . 40
6.2 Contact . . . 40
6.3 Thanks and greetings . . . 40

Bibliography 42

1 Intro

First of all, as an author I would like to thank you for your interest in this DHCPv6 implementation. If this documentation doesn't answer your questions or you have any suggestions, feel free to contact me. See Contact section for details. Also be sure to check out Dibbler website: http://klub.com.pl/dhcpv6/.

Tomasz Mrugalski

1.1 Overview

Dibbler is a portable DHCPv6 solution, which features server, client and relay. Currently there are ports available for Windows XP and 2003 (support for NT4 and 2000 is considered experimental) and Linux 2.4/2.6 systems. It supports both stateful (i.e. IPv6 address granting) and stateless (i.e. options granting) autoconfiguration. Besides basic functionality (specified in basic DHCPv6 spec, RFC3315 [5]), it also offers serveral enhancements, e.g. DNS servers and domain names configuration.

Dibbler is an open source software, distributed under GNU GPL licence. It means that it is freely available, free of charge and can be used by anyone (including commercial users). Source code is also provided, so anyone skilled enough can fix bugs, add new features and release his/her own version.

As for now, Dibbler support following features:

- Basic server discovery and address assignment (*SOLICIT*, *ADVERTISE*, *REQUEST* and *REPLY* messages) – This is a most common case: client discovers servers available in the local network, then asks for an address (and possibly additional options like DNS configuration), which is granted by a server.
- Best server discovery – when client detects more than one server available (by receiving more than one *ADVERTISE* message), it chooses the best one and remembers remaining ones as a backup.
- Multiple servers support – Client is capable of discovering and maintaning communication with several servers. For example, client would like to have 5 addresses configured. Prefered server can only grant 3, so client send request for remaining 2 addresses to one of the remaining servers.
- Relay support – In a larger network, which contains several Ethernet segments and/or wireless areas, sometimes centrally located DHCPv6 server might not be directly reachable. In such cace, additional proxies, so called relays, might be deployed to relay communication between clients and a remote server. Dibbler server supports indirect communication with clients via relays. Stand-alone, lightweight relay implementation is also available. Clients are capable of talking to the server directly or via relays.
- Address renewal – After receiving address from a server, client might be instructed to renew its address at regular intervals. Client periodically sends *RENEW* messege to a server, which granted its address. In case of communication failure, client is also able to attempt emergency address renewal (i.e. it sends *REBIND* message to any server).
- Unicast communication – if specific conditions are met, client could send messages directly to a server's unicast address, so additional servers does not need to process those messages. It also improves effciency, as all nodes present in LAN segment receive multicast packets.[1]
- Duplicate address detection – Client is able to detect and properly handle faulty situation, when server grants an address which is illegaly used by some other host. It will inform server of such circumstances (using *DECLINE* message), and request another address. Server will mark this address as used by unknown host, and will assign another address to a client.

[1] Nodes, which do not belong to specific multicast group, drop those packets silently. However, determining if host belongs or not to a group must be performed on each node. Also using multicast communication increases the network load.

- Power failure/crash support – After client recovers from a crash or a power failure, it still can have valid addresses assigned. In such circumstances, client uses *CONFIRM* message, to config if those addresses are still valid. [2].

- Normal and temporary addresses – Depending on its purpose, client can be configured to ask for normal (*IA_NA* option) or temporary (*IA_TA* option). Although use of temporary addresses is rather uncommon, both dibbler server and client support it.

- Hint system – Client can be configured to send various parameters and addresses in the *REQUEST* message. It will be treated as a hint by the server. If such hint is valid, it will be granted for this client.

- Server caching – Server can cache granted addresses, so the same client will receive the same address each time it asks. Size of this cache can be configured.

- Stateless mode – Client can be configured to not ask for any addresses, but the configuration options only. In such case, when no addresses are granted, such configuration is called stateless (*INFORMATION-REQUEST* message is used instead of normal *REQUEST*).

- Rapid Commit – Sometimes it is desirable to quicken configuration process. If both client and server are configured to use rapid commit, address assignment procedure can be shortened to 2 messages, instead of usual 4. Major advantage is lesser network usage and quicker client startup time.

1.2 Supported parameters

Except RFC3315-specified behavior [5], Dibbler also supports several enhancements:

- DNS Servers – During normal operation, almost all hosts require constant use of the DNS servers. It is necessary for event basic operations, like web surfing. DHCPv6 client can ask for information about DNS servers and DHCPv6 server will provide necessary information. [9]

- Domain Name – Client might be interested in obtaining information about its domain. Properly configured domain allow reference to a different hosts in the same domain using hostname only, not the full domain name, e.g. alice.example.com with properly configured domain can refer to another host in the same domain by using 'bob' only, instead of full name bob.example.com. [9]

- NTP Servers – To prevent clock misconfiguration and drift, NTP protocol [1] can be used to synchronize clocks. However, to successful use it, location of near NTP servers must be known. Dibbler is able to configure this information. [13]

- Time Zone – To avoid time-related ambiguation, each host should have timezone set properly. Dibbler is able to pass this parameter to all clients, who request it. [16]

- SIP Servers – Session Initiation Protocol (SIP) [4] is commonly used in VoIP solutions. One of the necessary information is SIP server addresses. This information can be passed to the clients. [6]

- SIP Domain Name – SIP domain name is another important parameter of the VoIP capable nodes. This parameter can be passed to all clients, who ask for it. [6]

- NIS, NIS+ Server – Network Information Service is a protocol for sharing authentication parameters between multiple Unix or Linux nodes. Both NIS and NIS+ server adresses can be passed to the clients. [11]

[2] As for 0.5.0 version, this functionality works on the server side only, client side support will be available in future releases.

- NIS, NIS+ Domain Name – NIS or NIS+ domain name is another necessary parameter for NIS or NIS+. It can be obtained from the DHCPv6 server to all clients, who require it. [11]
- Option Renewal Mechanism (Lifetime option)– All of the options mentioned on this list can be refreshed periodically. This might be handy if one of those parameters change. [14]
- Dynamic DNS Updates – Server can assign a fully qualified domain name for a client. To make such name useful, DNS servers must be informed that such name is bound to a specific IPv6 address. This procedure is called DNS Update. There are two kinds of the DNS Updates: forward and reverse. First is used to translate domain name to an address. The second one is used to obtain full domain name of a known address. See section 3.2 for details. [15]

1.3 Not supported features

Although list of the supported features increases with each release, some parts of the spec are not implemented yet. Below is a list of such features:

- Prefix Delegation [8]
- Authorization [5]
- Reconfigure mechanism [5]
- DNSSec (DNS Security Extensions) [12] in DNS Updates [15]

1.4 Requirements

Dibbler can be run on Linux systems with kernels from 2.4 and 2.6 series. IPv6 (compiled into kernel or as module) support is necessary to run dibbler. DHCPv6 uses UDP ports below 1024, so root privileges are required. They're also required to add, modify and delete various system parameters, e.g. IPv6 addresses.

Dibbler also runs on Windows XP and 2003. In XP systems, at least Service Pack 1 is required. To install various Dibbler parts (server, client or relay) as services, administrator privileges might be required.

Support for Windows NT4 and 2000 is limited and considered experimental. Due to lack of support and any kind of informations from Microsoft, this is not expected to change.

See RELEASE-NOTES for details about version-specific upgrades, fixes and features.

1.5 Supported platforms

Although Dibbler was developed on the i386 architecture, there are ports available for other architectures: IA64, AMD64, PowerPC, HPPA, Sparc, MIPS, S/390 and Alpha. They are available in the PLD, Gentoo and Debian Linux distributions. You can download system and distribution specficic packages from `http://www.pld-linux.org/`, `http://www.gentoo.org` or `http://www.debian.org`. Keep in mind that author has not tested those ports, so there might be some unknown issues present. If this is the case, be sure to notify package maintainers and possibly the author.

If your system is not on the list, don't despair. Dibbler is fully portable. Core logic is system independent and coded in C++ language. There are also several low-level functions, which are system speficic. They're used for adding addresses, retriving information about interfaces, setting DNS servers and so on. Porting Dibbler to other systems (and even other architectures) would require implementic only those serveral system-specific functions. See Developer's Guide for details.

2 Installation and usage

Client, server and relay are installed in the same way. Installation method is different in Windows and Linux systems, so each system installation is described separately. To simplify installation, it assumes that binary versions are used[3].

2.1 Linux installation

Starting with 0.4.0, Dibbler consists of 3 different elements: client, server and relay. During writing this documentation, Dibbler is already present in following Linux distributions:

Debian GNU/Linux – use standard tools (apt-get, aptitude) to install dibbler-client, dibbler-server, dibbler-relay or dibbler-doc packages (e.g. apt-get install dibbler-client)

Gentoo Linux – use emerge to install dibbler (e.g. emerge dibbler).

PLD GNU/Linux – use standard PLD's poldek tool to install dibbler package.

If you are using other Linux distribution, obtain (e.g. download from http://klub.com.pl/dhcpv6/) an archive, which suits your needs. Currently there are available RPM packages (which can be used in RedHat, Fedora Core, Mandrake or PLD distribution), DEB packages (suitable for Debian, Ubuntu or Knoppix) and ebuild (for Gentoo users). To install rpm package, execute `rpm -i archive.rpm` command. For example, to install dibbler 0.4.1, issue following command:

```
rpm -i dibbler-0.4.1-1.i386.rpm
```

To install Dibbler on Debian or other system with dpkg management system, run *dpkg – iarchive.deb* command. For example, to install server, issue following command:

```
dpkg -i dibbler-server_0.4.1-1_i386.deb
```

To install Dibbler in Gentoo systems, just type:

```
emerge dibbler
```

If you would like to install Dibbler from sources, please download tar.gz source archive, extract it, type make followed by target (e.g. server, client or relay[4]). After successful compilation type make install. For example, to build server and relay, type:

```
tar zxvf dibbler-0.4.0-src.tar.gz
make server relay
make install
mkdir -p /var/lib/dibbler
```

Depending what functionality do you want to use (server,client or relay), you should edit configuration file (`client.conf` for client, `server.conf` for server and `relay.conf` for relay). All configuration files should be placed in the `/etc/dibbler` directory. Also make sure that `/var/lib/dibbler` directory is present and is writeable. After editing configuration files, issue one of the following commands:

```
dibbler-server start
dibbler-client start
dibbler-relay start
```

[3] Compilation is not required, usually binary version can be used. Compilation should be performed by advanced users only, see *Compilation* section for details.

[4] To get full target list, type: make help.

`start` parameter requires little explanation. It instructs Dibbler to run in daemon mode – detach from console and run in the background. During configuration files fine-tuning, it is ofter better to watch Dibbler's bahavior instantly. In this case, use `run` instead of `start` parameter. Dibbler will present its messages on your console instead of log files. To finish it, press ctrl-c.

To stop server, client or relay running in daemon mode, type:

```
dibbler-server stop
dibbler-client stop
dibbler-relay stop
```

To see, if client, server or relay are running, type:

```
dibbler-server status
dibbler-client status
dibbler-relay status
```

To see full list of available commands, type `dibbler-server`, `dibbler-client` or `dibbler-relay` without any parameters.

2.2 Windows installation

Since the 0.2.1-RC1 release, Dibbler supports Windows XP and 2003. In version 0.4.1 exprimental support for Windows NT4 and 2000 was added. The easiest way of Windows installation is to download clickable Windows installer. It can be downloaded from http://klub.com.pl/dhcpv6/. After downloading, click on it and follow on screen instructions. Dibbler will be installed and all required links will be placed in the Start menu. Note that there are two Windows versions: one for XP/2003 and one for NT4/2000. Make sure to use proper port. If you haven't set up IPv6 support, see following sections for details.

2.3 IPv6 support

Some systems does not have IPv6 enabled by default. In that is the case, you can skip following subsections safely. If you are not sure, here is an easy way to check it. To verify if you have IPv6 support, execute following command: `ping6 ::1` (Linux) or `ping ::1` (Windows). If you get replies, you have IPv6 already installed.

2.3.1 Setting up IPv6 in Linux

IPv6 can be enabled in Linux systems in two ways: compiled directly into kernel or as a module. If you don't have IPv6 enabled, try to load IPv6 module: `modprobe ipv6` (command executed as root) and try ping6 once more. If that fails, you have to recompile kernel to support IPv6. There are numerous descriptions how to recompile kernel available on the web, just type "kernel compilation howto" in Google.

2.3.2 Setting up IPv6 in WindowsXP and 2003

If you have already working IPv6 support, you can safely skip this section. The easiest way to enable IPv6 support is to right click on the `My network place` on the desktop, select `Properties`, then locate your network interface, right click it and select `Properties`. Then click `Install...`, choose protocol and then IPv6 (its naming is somewhat diffrent depending on what Service Pack you have installed). In XP, there's much quicker way to install IPv6. Simply run command `ipv6 install` (i.e. hit Start..., choose run... and then type `ipv6 install`). Also make sure that you have built-in firewall disabled. See *Frequently Asked Question* section for details.

2.3.3 Setting up IPv6 in Windows 2000

If you have already working IPv6 support, you can safely skip this section. The following description was provided by Sob ((sob(at)hisoftware.cz). Thanks. This description assumes that ServicePack 4 is already installed.

1. Download the file tpipv6-001205.exe from: http://msdn.microsoft.com/downloads/sdks/platform/tpipv6.asp and save it to a local folder (for example, `C:\IPv6TP`).
2. From the local folder (`C:\IPv6TP`), run `Tpipv6-001205.exe` and extract the files to the same location.
3. From the local folder (`C:\IPv6TP`), run `Setup.exe -x` and extract the files to a subfolder of the current folder (for example, `C:\IPv6TP\files`).
4. From the folder containing the extracted files (`C:\IPv6TP\files`), open the file `Hotfix.inf` in a text editor.
5. In the [Version] section of the Hotfix.inf file, change the line NTServicePackVersion=256 to NTServicePackVersion=1024, and then save changes. [5]
6. From the folder containing the extracted files (`C:\IPv6TP\files`), run `Hotfix.exe`.
7. Restart the computer when prompted.
8. After the computer is restarted, from the Windows 2000 desktop, click Start, point to Settings, and then click Network and Dial-up Connections. As an alternative, you can right-click My Network Places, and then click Properties.
9. Right-click the Ethernet-based network interface to which you want to add the IPv6 protocol, and then click Properties. Typically, this network interface is named Local Area Connection.
10. Click Install.
11. In the Select Network Component Type dialog box, click Protocol, and then click Add.
12. In the Select Network Protocol dialog box, click Microsoft IPv6 Protocol and then click OK.
13. Click Close to close the Local Area Connection Properties dialog box.

2.3.4 Setting up IPv6 in Windows NT4

If you have already working IPv6 support, you can safely skip this section. The following description was provided by The following description was provided by Sob (sob(at)hisoftware.cz). Thanks.

1. Download the file msripv6-bin-1-4.exe from: http://research.microsoft.com/msripv6/msripv6.htmMicrosoft and save it to a local folder (for example, `C:\IPv6Kit`).
2. From the local folder (`C:\IPv6Kit`), run `msripv6-bin-1-4.exe` and extract the files to the same location.
3. Start the Control Panel's "Network" applet (an alternative way to do this is to right-click on "Network Neighborhood" and select "Properties") and select the "Protocols" tab.

[5] This defines Service Pack requirement. NTServicePackVersion is a ServicePack version multiplied by 256. If there would be SP5 available, this value should have been changed to the 1280.

4. Click the "Add..." button and then "Have Disk...". When it asks you for a disk, give it the full pathname to where you downloaded the binary distribution kit (`C:\IPv6Kit`).

5. IPv6 is now installed.

2.4 Compilation

Dibbler is distributed in 2 versions: binary and source code. For most users, binary version is better choice. Compilation is performed by more experienced users, preferably with programming knowledge. It does not offer significant advantages over binary version, only allows to understand internal Dibbler workings. You probably want just install and use Dibbler. If that is your case, read section named *Installation*. However, if you are skilled enough, you might want to tune several Dibbler aspects during compilation. See *Dibbler Developer's Guide* for information about various compilation parameters.

2.4.1 Linux compilation

Compilation in most cases is not necessary and should be performed only by experienced users. To compile dibbler, issue following commands:

```
tar zxvf dibbler-0.4.0-src.tar.gz
cd dibbler
make server client relay doc
```

That's it. You can also install it in the system by issuing command:

```
make install
```

If there are problems with missing/different compiler version, take a look at the beginning of the Makefile.inc file. Dibbler was compiled using gcc 2.95, 3.0, 3.2, 3.3, 3.4, 4.0 and 4.1 versions. Note that 2.95 is now considered obsolete and was not tested for some time. Lexer files were generated using flex 2.5.33. Parser file were created using bison++ 1.21.9[6].

If there are problems with `SrvLexer.cpp` and `ClntLexer.cpp` files, please use FlexLexer.h in Port-linux/ directory. Most simple way to do this is to copy this file to `/usr/include` directory.

2.4.2 Windows XP/2003 compilation

Download dibbler-0.5.0 -src.tar.gz and extract it. In `Port-win32` there are several project files (for server, client and relay) for MS Visual Studio 2003. Previous dibbler releases were compiled using MS Visual Studio .NET (sometimes called 2002). It might work with newest dibber version, but there are no guarantee. Open one of the project files and click Build command. That should start compilation. After a while, binary exe files will be stored in the `Debug/` directory.

2.4.3 Windows NT/2000 compilation

Windows NT4/2000 port is considered experimental, but there are reports that it works just fine. To compile it, you should download dev-cpp (http://www.bloodshed.net/dev/devcpp.html), a free IDE for Windows utilising minGW port of the gcc for Windows. Run dev-cpp, click ,,open project...", and open one of the `*.dev` files located in the Port-winnt2k directory, then click compile. You also should take a look at `Port-winnt2k/INFO` file for details.

[6] flex and bison++ tools are not required to compile Dibbler. Generated files are placed in CVS and in tar.gz archives

3 Features HOWTO

This section contains information about setting up various Dibbler features. Since this section was added recently, it is not yet comprehensive. That is expected to change.

3.1 Stateless vs stateful and IA, TA options

This section explains the difference between stateless and stateful configurations. IA and TA options usage is also described.

Usually, normal stateful configuration based on non-temporary addresses should be used. If you don't know, what temporary addresses are, you don't need them.

There are two kinds of configurations in DHCPv6 ([5], [10]):

stateful – it assumes that addresses (and possibly other parameters) are assigned to a client. To perform this kind of configuration, four messages are exchanged: *SOLICIT*, *ADVERTISE*, *REQUEST* and *REPLY*.

stateless – when only parameters are configured (without assigning addresses to a client). During execution of this type of configuration, only two messages are exchanged: *INF-REQUEST* and *REPLY*.

During normal operation, client works in a stateful mode. If not instructed otherwise, it will request one or more normal (i.e. non-temporary) address. It will use *IA* option (Identity Association for Non-temporary Addresses, see [5] for details) to request and retrieve addresses. Since this is a default behavior, it does not have to be mentioned in the client configuration file. Nevertheless, it can be provided:

```
# client.conf
iface eth0 {
  ia
  option dns-server
}
```

In a specific circustances, client might be interested in obtaining only temporary addresses. Although this is still a stateful mode, its configuration is sligtly different. There is a special option called *TA* (Identity Association for Temporary Addresses, see [5] for details). This option will be used to request and receive temporary addresses from the client. To force client to request temporary addresses instead of permanent ones, `ta` keyword must be used in client.conf file. If this option is defined, only temporary address will be requested. Keep in mind that temporary addresses are not renewed.

```
# client.conf
iface eth0 {
  ta
  option dns-server
}
```

It is also possible to instruct client to work in a stateless mode. It will not ask for any type of addresses, but will ask for specific non-adress related configuration parameters, e.g. DNS Servers information. This can be achieved by using `stateless` keyword. Since this is a global parameter, it is not defined on any interface, but as a global option.

```
# client.conf
stateless
iface eth0
{
  option dns-server
}
```

Some of the cases mentioned above can be used together. However, several combinations are illegal. Here is a complete list:

none – When no option is specified, client will assume one IA with one address should be requested. Client will send `ia` option (stateful autoconfiguration).

ia – Client will send `ia` option (stateful autoconfiguration).

ia,ta – When both options are specified, client will request for both - Non-temporary as well as Temporary addresses (stateful autoconfiguration).

stateless – Client will request additional configration parameters only and will not ask for addresses (stateless autoconfiguration).

stateless,ia – This combination is not allowed.

stateless,ta – This combination is not allowed.

stateless,ia,ta – This combination is not allowed.

3.2 DNS Update

Note: In this section, we will assume that hostnames will be used from the example.com domain and that addresses will be provided from the 2000::/64 pool.

During normal operation, DHCPv6 client receives one or more IPv6 address(es) from DHCPv6 server. If configured to do so, it can also receive information about DNS server addresses. As an additional service, DNS Update can be performed. This feature, sometimes known as Dynamic DNS, keeps DNS entries up to date. When client boots, it gets its fully qualified domain name and this name can be used to reach this particular client by other nodes. Details of this mechanism is described in [15].

There are two types of the DNS Updates. First is a so called forward resolving. It allows to change a node's name into its address, e.g. malcolm.example.com can be translated into 2000::123. Other kind of record, which can be updated is a so called reverse resolving. It allows to obtain full name of a node with know address, e.g. 2000::124 can be translated into zoe.example.com.

To configure this feature, following steps must be performed:

1. Configure DNS server. DNS server supporting IPv6 and dynamic updates must be configured. One example of such server is a BIND 9.3. It is necessary to allow listening on the IPv6 sockets and define that specific domain can be updated. See example below.

2. Configure Dibbler server to provide DNS server informations for clients. DNS Updates will be sent to the first DNS server on the list of available servers.

3. Configure Dibbler server to work in stateful mode, i.e. that it can provide addresses for the clients. This is a default mode, so unless configuration was altered, this step is already done. Make sure that there is no ,,stateless" keyword in the `server.conf` file.

4. Define list of the available names in the server configuration file. Make sure to use fully qualified domain names (e.g. malcolm.example.com), not the hostnames only.

5. Configure dibbler client to request for DNS Update. Use ,,option fqdn" to achieve this.

6. Server can be configured to execute

   - both (AAAA and PTR) updates by itself
   - execute PTR only by itself and let client execute AAAA update
   - don't perform any updates and let client perform AAAA update.

Note that only server is allowed to execute PTR updates. After configuration, client and/or server should log following line, which informs that Dynamic DNS Update was completed successfully.

```
2006.07.24 01:52:51 Client Notice    FQDN Configured successfully !
```

3.2.1 Example BIND configuration

Below are example configuration files for the BIND 9.3. First is a relevant part of the /etc/bind/named.conf configuration file. Generally, support for IPv6 in BIND is enabled (listen-on-v6) and there are two zones added: example.com (normal domain) and 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.ip6.arpa (reverse mapping). Corresponding files are stored in `example.com` and `rev-2000` files. For details about meaning of those directives, please consult *BIND 9 Administrator Reference Manual*.

Note: Provided configuration is not safe from the security point of view. See next subsection for details.

```
// part of the /etc/bind/named.conf configuration file
options {
    listen-on-v6 { any; };
    listen-on    { any; };

    // other global options here
    // ...
};

zone "example.com" {
    type master;
    file "example.com";
    allow-update   { any; };
    allow-transfer { any; };
    allow-query    { any; };

    // other example.com domain-specific
    // options follow
    // ...
};

zone "0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.ip6.arpa" {
    type master;
    file "rev-2000";
    allow-update   { any; };
```

```
    allow-transfer { any; };
    allow-query    { any; };

   // other 2000::/64 reverse domain related
   // options follow
   // ...
};
```

Below are examples of two files: forward and reverse zone. First example presents how to configure normal domain. As an example there is entry provided for zoe.example.com host, which has 2000::123 address. Note that you do not have to manually configure such entries – dibbler will do this automatically. It was merely provided as an example, what kind of mapping will be done in this zone.

```
;
$ORIGIN .
$TTL 86400      ; 1 day
example.com             IN SOA  v13.klub.com.pl. root.v13.klub.com.pl. (
                                129         ; serial
                                7200        ; refresh (2 hours)
                                3600        ; retry (1 hour)
                                604800      ; expire (1 week)
                                86400       ; minimum (1 day)
                                )
                        NS      v13.klub.com.pl.
                        A       1.2.3.4
                        TXT     "Fake domain used for Dibbler tests."
$ORIGIN example.com.
$TTL 7200       ; 2 hours
zoe                     AAAA    2000::123
```

Second example presents zone file for reverse mapping. It contains entries for a special zone called 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.ip6.arpa. This zone represents 2000::/64 address space. As an example there is a static entry, which maps address 2000::999 to a canonical name kaylee.example.com. Note that you do not have to manually configure such entries – dibbler will do this automatically. It was merely provided as an example, what kind of mapping will be done in this zone.

```
; rev-2000 example file
$ORIGIN .
$TTL 259200     ; 3 days

; this line below is split in two due to page with limitation
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.ip6.arpa IN
      SOA 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.ip6.arpa. hostmaster.ep.net. (
; this line above is split in two due to page with limitation
                                200608268  ; serial
                                86400      ; refresh (1 day)
                                1800       ; retry (30 minutes)
                                172800     ; expire (2 days)
```

```
                              259200     ; minimum (3 days)
                              )
                         NS      klub.com.pl.
$ORIGIN 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.ip6.arpa.
$TTL 86200      ; 23 hours 56 minutes 40 seconds
3.2.1                    PTR     picard.example.com.

; this line below is split in two due to page with limitation
9.9.9                    PTR     kaylee.example.com.
$ORIGIN 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.ip6.arpa.

; example entry: 2000::999 -> troi.example.com.
; this line below is split in two due to page with limitation
9.9.9.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.ip6.arpa
      PTR troi.example.com.
; this line above is split in two due to page with limitation
```

Note: Due to page width limitation, if the example above, two lines were split.

3.2.2 Dynamic DNS Testing and tips

Proper configuration of the DNS Update mechanism is not an easy task. Therefore this section provides description of several methods of testing and tuning BIND configuration. Please review following steps before reporting issues to the author or on the mailing list.

- See example server and client configuration files described in a sections 4.5.7 and 4.7.10. Also note that Dibbler distribution should be accompanied with several example configuration files. Some of them include FQDN usage examples.
- Make sure that unix user, which runs BIND, is able to create and write file example.com.jnl. When BIND is unable to create this journal file, it will fail to accept updates from dibbler and will report failure. Check BIND log files, which are usually stored in the `/var/log/` directory.
- Make sure that you have routing configured properly on a host, which will attempt to perform DNS Update. Use ping6 command to verify that DNS server is reachable from this host.
- Make sure that your DNS server is configured properly. To do so, you might want to use `nsupdate` tool. It is part of the BIND distribution, but it is sometimes distributed separated as part of the dnsutils package. After executing nsupdate tool, specify address of the DNS server (`server` command), specify update parameters (`update` command) and then type `send`. If nsupdate return a command prompt, then the update was successful. Otherwise nsupdate will print DNS server's response, e.g. NOTAUTH of SRVFAIL. See below for examples of successful forward (AAAA record) and reverse (PTR record) updates.
- After DNS Update is performed, DNS records can be verified using dig command line tool (a part of the dnsutils package). Command syntax is: `dig @(dns-server-address) name record-type`. In the following example, this query checks for name jayne.example.com at a server located at 2000::1 address. Record type AAAA (standard record for resolving name into IPv6 address) is requested. dig tool provides server's response in the `ANSWER SECTION:`. See example log below.
- In example BIND configuration above, zone transfers, queries and updates are allowed from anywhere. To make this configuration more secure, it might be a good idea to allow updates only from a certain range of addresses or even one (DHCPv6 server's) address only.

To manually make AAAA record update, type:

```
nsupdate
>server 2000::1
>update add worf.example.com 7200 IN AAAA 2000::567
>send
```

To manually make PTR record update, type:

```
nsupdate
>server 2000::1
>update add
3.2.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.ip6.arpa.
86200 IN PTR picard.example.com.
>send
```

Note: Everything between "update" and "picard.example.com" must be typed in one line.
And here is an example dig session:

```
v13:/var# dig @2000::1 jayne.example.com AAAA
; <<>> DiG 9.3.2 <<>> @2000::1 jayne.example.com AAAA
; (1 server found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33416
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; QUESTION SECTION:
;jayne.example.com.             IN      AAAA

;; ANSWER SECTION:
jayne.example.com.      7200    IN      AAAA    2001::e4

;; AUTHORITY SECTION:
example.com.            86400   IN      NS      v13.klub.com.pl.

;; Query time: 6 msec
;; SERVER: 2000::1#53(2000::1)
;; WHEN: Mon Jul 24 01:38:13 2006
;; MSG SIZE  rcvd: 136
```

3.3 Server address caching

Previous Dibbler versions assigned a random address from the available address pool, so the same client received different address each time it asked for one. In the 0.5.0 release, new mechanism was introduced to make sure that the same client gets the same address each time. It is called *Server caching*.

Below is the algorithm used by the server to assign an address to the client.

- if the client provided hint, it is valid (i.e. is part of the supported address pool) and not used, then assign requested address.
- if the client provided hint, it is valid (i.e. is part of the supported address pool) but used, then assign free address from the same pool.
- if the client provided hint, but it is not valid (i.e. is not part of the supported address pool, is link-local or a multicast address), then ignore the hint completety.
- if the did not provide valid hint (or provided invalid one), try to assign address previously assigned to this client (address caching)
- if this is the first time the client is seen, assign any address available.

3.4 Relays

In small networks, all nodes (server, hosts and routers) are connected to the same network segment – usually Ethernet segment or a single access point or hotspot. This is very convenient as all clients can reach server directly. However, larger networks usually are connected via routers, so direct communication is not always possible. On the other hand it is useful to have one server, which supports multiple links – some connected directly and some remotely.

FIXME: Provide description and maybe a figure.

3.5 XML files

During its execution, all dibbler components (client, server and relay) store its internal information in the XML files. In Linux systems, they are stored in the `/var/lib/dibbler` directory. In Windows, current directory (i.e. directory where exe files are located) is used instead. There are several xml files generated. Since they are similar for each component, following list provides description for server only:

- server-CfgMgr.xml – Represents information read from a configuration file, e.g. available address pool or DNS server configuration.
- server-IfaceMgr.xml – Represens detected interfaces in the operating system, as well as bound sockets and similar information.
- server-AddrMgr.xml – This is database, which contains identity associations with associated addresses.
- server-cache.xml – Since caching is implemented by the server only, this file is only created by the server. It contains information about previously assigned addresses.

4 Configuration files

This section describes Dibbler server, relay and client configuration. Square brackets denotes optional values: mandatory [optional]. Alternative is marked as |. A | B means A or B. Parsers are case-insensitive, so Iface, IfAcE, iface and IFACE mean the same. This does not apply to interface names. eth0 and ETH0 are dwo diffrent interfaces.

4.1 Data types

Config file parsing is token-based. Token can be considered a keyword or a specific phrase. Here's list of tokens used:

IPv6 address – IPv6 address, e.g. 2000:dead:beef::789

32-bit decimal integer – string containing only numbers, e.g. 123456

string – string of arbitrary characters enclosed in single or double quotes, e.g. 'this is a string'. If string contains only a-z, A-Z and 0-9 characters, quotes can be omited, e.g. beeblebrox

DUID identifier – hex number starting with 0x, e.g. 0x12abcd.

IPv6 address list – IPv6 addresses separated with commas, e.g. 2000::123, 2000::456

DUID list – DUIDs separated with commas, e.g. 0x0123456,0x0789abcd

string list – strings separated with comas, e.g. tealc,jackson,carter,oneill

boolean – YES, NO, TRUE, FALSE, 0 or 1. Each of them can be used, when user must enable or disable specific option.

4.2 Scopes

There are four scopes, in which options can be specified: global, inteface, IA and address. Every option is specific for one scope. Each option is only applied to a scope and all subscopes in which it is defined.

For example, T1 is defined for IA scope. However, it can be also used in more common scopes. In this case – in interface or global. Defining T1 in interface scope means: ,,for this interface the default T1 value is ...". The same applies to global scope. Options can be used multiple times. In that case value defined later is used.

Global scope is the largest. It covers the whole config file and applies to all intefaces, IAs, and addresses, unless some lower scope options override it. Next comes inteface scope. Options defined there are inteface-specific and apply to this interface, all IAs in this interface and addresses in those IAs. Next is IA scope. Options defined there are IA-specific and apply to this IA and to addresses it contains. Least significant scope is address.

4.3 Comments

Comments are also allowed. All common comment styles are supported:

- C++ style one-line comments: // this is comment
- C style multi-line comments: /* this is multiline comment */
- bash style one-line comments: # this is one-line comment

4.4 Client configuration file

Client configuration file should be named `client.conf`. It should be placed in the `/etc/dibbler/` directory (Linux system) or in the current directory (Windows systems). After successful startup, old version of this file is stored as `client.conf-old`. One of design requirements for client was ,,out of the box" usage. To achieve this, simply use empty `client.conf` file. Client will try to get one address for each up and running interface [7].

[7] Exactly: Client tries to configure each up, multicast-capable and running interface, which has link address at least 6 bytes long. So it will not configure tunnels (which usually have IPv4 address (4bytes long) as their link address. It should

4.4.1 Global parameters

There are several global options. Those options can be used in the global scope. However, inteface, IA and address scoped options can be used in the global scope, too. Client configuration file has following syntax:

```
global-options
interface-options
IA-options
address-options
interface-declaration
```

4.4.2 Interface declaration

Each system interface, which should be configured, must be mentioned in the configureation file. Interfaces can be declared with this syntax:

```
iface interface-name
{
  interface-options
  IA-options
  address-options
}
```

or

```
iface interface-number
{
  interface-options
  IA-options
  address-options
}
```

In the latter case, interface-number denotes interface number. It can be extracted from ,,ip l" (Linux) or ,,ipv6 if" (Windows). `interface-name` is an interface name. Name of the interface does not have to be enclosed in single or double quotes. It is necessary only in Windows systems, where interface names sometimes contain spaces, e.g. "local network connection". Interface scoped options can be used here. IA-scoped as well as address scoped options can also be used. They will be treated as a default values for future definitions of the IA and address instantations.

4.4.3 IA declaration

IA is an acronym for Identity Association. It is a logical entity representing address or addresses used to perform some functions. IA-options can be defined, e.g. T1. IPv6 addresses can be defined here. All those values will be used as hints for a server. Almost always, each DHCPv6 client will have exactly one IA on each interface. IA is declared using following syntax:

```
ia
{
  IA-options
  address-options
  address-declaration
}
```

configure all Ethernet and 802.11 interfaces. The latter was not tested by author due to lack of access to 802.11 equipment.

It is also possible to define multiple IA at once. To do so, following syntax might be used:

```
ia number
{
  IA-options
  address-options
}
```

Number is an optional number, which describes how many such IAs should be requested. Number is optional. If it is not specified, 1 is used. If this number is not equal 1, then address options are not allowed. That could come in handy when someone need serveral IAs with the same parameters. If IA contains no addresses, client assumes that one address should be configured. IA scoped as well as address options can be defined here. IA scoped options will be applied directly, while address scoped options will be used as default values for all addresses that will be defined in this IA.

4.4.4 Address declaration

When IA is defined, it is sometimes useful to define its address. Its value will be used as a hint for the server. Address is declared in the following way:

```
address number
{
  address-options
  IPv6-address
}
```

where number denotes how many addresses with those values should be requested. If it is diffrent than 1, then IPv6 address options are not allowed. Only address scoped options can be used here.

4.4.5 Standard options

So called standard options are defined by the base DHCPv6 specification, a so called RFC 3315 document [5]. Those options are called standard, because all DHCPv6 implementations, should properly handle them. Standard options are declared in the following way:

```
OptionName option-value
```

Every option has a scope it can be used in, default value and sometimes allowed range.

work-dir – (scope: global, type: string, default: .) Defines working directory.

log-level – (scope: global, type: integer, default: 7) Defines verbose level of the log messages. The valid range if from 1 (Emergency) to 8 (Debug). The higher the logging level is set, the more messages dibbler will print.

log-name – (scope: global, type: string, default: Client). Defines name, which should be used during logging.

log-mode – (scope: global, type: short, full or precise, default value: full) Defines logging mode. In the default, full mode, name, date and time in the h:m:s format will be printed. In short mode, only minutes and seconds will be printed (this mode is useful on terminals with limited width). Recently added precise mode logs information with seconds and microsecond precision. It is a useful for finding bottlenecks in the DHCPv6 autoconfiguration process.

strict-rfc-no-routing – (scope: global, type: none, default: not defined). During normal operation, DHCPv6 client should add IPv6 address only, without configuring routing, because this should be done with other means, i.e. router advertisements [2]. However, this behavior is confusing and lots of users complained about it, so since the 0.5.0-RC1 release, this has been changed in dibbler. Right now when dibbler client configures address, it also configures routing, so every host is able to communicate with other hosts, which have obtained address from the same server. If you don't like this behavior, you might want to use this option.

rapid-commit – (scope: interface, type: boolean, default: 0). This option allows rapid commit procedure to be performed. Note that enabling rapid commit on the client side is not enough. server must be configured to allow rapid commit, too.

unicast – (scope: interface, type: boolean, default: 0). This option specifies if client should request unicast communication from the server. If server is configured to allow it, it will add unicast option to its replies. It will allow client to communicate with server via unicast addresses instead of usual multicast.

prefered-servers – (scope: interface, type: address or duid list, default: empty). This list defines, which servers are prefered. When client sends *SOLICIT* message, all servers available in the local network will respond. When client receives multiple *ADVERTISE* messages, it will choose those sent by servers mentioned on the perfered-server list.

reject-servers – (scope: interface, type: address or duid list, default: empty) This list defines which server must be ignored. It has negative meaning to the prefered-servers list.

T1 – (scope: IA, type: integer: default: $2^{32}-1$). This value defines after what time client should start renew process. This is only a hint for the server. Actual value will be provided by the server.

T2 – (scope: IA, type: integer, default:$2^{32}-1$). This value defines after what time client will start emergency rebind procedure if renew process fails. This is only a hint for the server. Actual value will be provided by the server.

valid-lifetime – (scope:address, type: integer, default:$2^{32}-1$) This parameter defines valid lifetime of an address. It will be used as a hint for a server, when the client will send requests.

prefered-lifetime – (scope:address, type: integer, default:$2^{32}-1$) This parameter defines prefered lifetime of an address. It will be used as a hint for a server, when there client will send requests.

4.4.6 Extension options

Extension options are the options specified in external drafts and RFC documents, but not in the base spec [5]. To easily distinguish if an option is part of the base standard or one of the multiple extensions, `option` keyword was added in the extension options declaration. Therfore extension options are declared as follows:

```
option option-name
```

or

```
option option-name option-value
```

where option-name is name of the options. First approach instructs dibbler client to just ask for this particular option. Second approach includes requested values. When sent by the client, server will use those values as hints during those options assignment. Since those options are defined per interface, thus every extension option has an interface scope, i.e. it is defined once per interface. As for the 0.5.0 release, currently supported options are:

dns-server – (scope: interface, type: address list, default: none). This option conveys information about DNS servers available. After retriving this information, client will be able to resolve domain names into IP (both IPv4 and IPv6) addresses. Without setting up DNS servers, host's network capability is greatly reduced, as user can't use domain names (e.g. http://wp.pl/), but must use IP addresses directly (e.g. http://212.77.100.101/ or http://3ffe:1234::456/). Defined in [7].

domain – (scope: interface, type: domain list, default: none). This option is used for retriving domain or domains names, which the client is connected in. For example, if client's hostname is `alice.mylab.example.com` and it wants to contact `bob.mylab.example.com` it can simply refer to it as `bob`. Without domain name configured, it would have to use full domain name. After successful configuration, this useful shortcut is being used by all services available: web browsing, mail sending, news reading etc. Defined in [7].

ntp-server – (scope: interface, type: address list, default: none). This option defines information about available NTP servers. Network Time Protocol [1] is a protocol used for time synchronisation, so all hosts in the network has the same proper time set. Defined in [13].

time-zone – (scope: interface, type: timezone, default: none). It is possible to retrieve timezone from the server. If client is interested in this information, it should ask for this option. Note that this option is considered obsolete as it is mentioned in draft version only [16]. Work on this draft seems to be abandoned as similar functionality is provided in now standard [13].

sip-server – (scope: interface, type: address list, default: none). Session Initiation Protocol [4] is an control protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences. Its most common usage is VoIP. Format of this option is defined in [6].

sip-domain – (scope: interface, type: domain list, default: none). It is possible to define domain names for Session Initiation Protocol [4]. Configuration of this parameter will ease usage of domain names in the SIP protocol. Format of this option is defined in [6].

nis-server – (scope: interface, type: address list, default: none). Network Information Service (NIS) is a Unix-based system designed to use common login and user information on multiple systems, e.g. universities, where students can log on to ther accounts from any host. To use this functionality, a host needs information about NIS server's address. This can be retrieved with this option. Its format is defined in [11].

nis-domain – (scope: interface, type: domain list, default: none). Network Information Service (NIS) can albo specify domain names. It can be configured with this option. It is defined in [11].

nis+-server – (scope: interface, type: address list, default: none). Network Information Service Plus (NIS+) is an improved version of the NIS protocol. This option is defined in [11].

nis+-domain – (scope: interface, type: domain list, default: none). Similar to nis-domain, it defines domains for NIS+. This option is defined in [11].

lifetime – (scope: interface, type: boolean, default: no). Base spec of the DHCPv6 protocol does offers way of refreshing addresses only, but not the options. Lifetime defines, how often client would like to renew all its options. By default client will not send such option, but it will accept it and act accordingly if the server sends it on its own. Format of this option is defined in [14].

Note that timezone format is described in file `draft-ietf-dhc-dhcpv6-opt-tz-00.txt` and domain format is described in RFC 3646. After receiving options values from a server, client stores values of those options in separate files in the working directory (`/var/lib/dibbler` in Linux and current directory in

Windows). File names start with the option word, e.g. `option-dns-server`. Several options are also processed and set up in the system. Options supported in Linux and Windows environments are presented in the table below.

Option	Linux	WinXP/2003	WinNT/2000
dns-server	system, file	system, file	system,file
domain	file	file	file
ntp-server	file	file	file
time-zone	file	file	file
sip-server	file	file	file
sip-domain	file	file	file
nis-server	file	file	file
nis-domain	file	file	file
nis+-server	file	file	file
nis+-domain	file	file	file

4.4.7 Stateless configuration

If interface does not contain `IA` or `TA` keywords, client will ask for one address (one IA with one address request will be sent). If client should not request any addresses on this interface, *stateless*[8] keyword must be used. In such circumstances, only specified options will be requested.

4.4.8 Relay support

Usage of the relays is not visible from the client's point of view: Client can't detect if it communicates via relay(s) or directly with the server. Therefore no special directives on the client side are required to use relays.

4.5 Client configuration examples

This subsection contains various examples of the most popular configurations. Several additional examples are provided with the source code. Please download it and look at `*.conf` files.

4.5.1 Example 1: Default

In the most simple case, client configuration file can be empty. Client will try to assign one address for every interface present in the system, except interfaces, which are:

- down (flag UP not set)
- loopback (flag LOOPBACK set)
- not running (flag RUNNING not set)
- not multicast capable (flag MULTICAST not set)
- have link-layer address less than 6 bytes long (this requirement should skip all tunnels and virtual interfaces)

If you must use DHCPv6 on one of such interfaces (which is not recommended and such attempt probably will fail), you must explicitly specify this interface in the configuration file.

[8] In the version 0.2.1-RC1 and earlier, this directive was called no-ia. This depreciated name is valid for now, but might be removed in future releases.

4.5.2 Example 2: DNS

Configuration mentioned in previous subsection is a minimal one and in a real life will be used rarely. The most common usage of the DHCPv6 protocol is to request for an address and DNS configuration. Client configuration file achieving those goals is presented below:

```
# client.conf
log-mode short
log-level 7
iface eth0 {
  ia
  option dns-server
}
```

4.5.3 Example 3: Timeouts and specific address

Automatic configuration is being driven by several timers, which define, what action should be performed at various intervals. Since all values are provided by the server, client can only define values, which will be sent to a server as hints. Server might take them into consideration, but might also ignore them completely. Following example shows how to ask for a specific address and provide hints for a server. Client would like to get 2000::1:2:3 address, it would like to renew addresses once in 30 minutes (T1 timer is set to 1800 seconds). Client also would like to have address, which is prefered for an hour and is valid for 2 hours.

```
# client.conf
log-mode short
log-level 7
iface eth0 {
  T1 1800
  T2 2000
  prefered-lifetime 3600
  valid-lifetime 7200
  ia {
    address {
      2000::1:2:3
    }
  }
}
```

4.5.4 Example 4: Unicast, more than one address

Another example: client like to obtain 2 addresses on ,,Local Area Connection" interface. Note quotation marks around interface name. They are necessary since this particular interface name contains spaces. Client also would like to accept Unicast communication if server supports it. User don't care for details, so keep those log very short. Take note that you won't be able see to what Dibbler is doing with such low log-level. (Usually log-level should be set to 7, which is also a default value).

```
# client.conf
log-mode short
```

```
log-level 5
iface "Local Area Connection" {
  unicast yes
  ia 2
}
```

4.5.5 Example 5: Quick configuration using Rapid-commit

Rapid-commit is a shortened exchange with server. It consists of only two messages, instead of the usual four. It is worth to know that both sides (client and server) must also support rapid-commit to use this fast configuration.

```
# client.conf
iface eth1 {
  rapid-commit yes
  ia
  option dns-server
}
```

4.5.6 Example 6: Stateless mode

Client can be configured to work in a stateless mode. It means that it will obtain only some configuration parameters, but no addresses. Let's assume we want all the details stored in a log file and we want to obtain all possible configuration parameters. Here is a configuration file:

```
# client.conf
log-level 8
log-mode full
stateless
iface eth0
{
  option dns-server
  option domain
  option ntp-server
  option time-zone
  option sip-server
  option sip-domain
  option nis-server
  option nis-domain
  option nis+-server
  option nis+-domain
}
```

4.5.7 Example 7: Dynamic DNS (FQDN)

Dibbler client is able to request fully qualified domain name, i.e. name, which is fully resolvable using DNS. After receiving such name, it can perform DNS Update procedure. Client can ask for any name, without any preferrence. Here is an example how to configure client to perform such task:

```
# client.conf
log-level 7
iface eth0 {
# ask for address
    ia

# ask for options
   option dns-server
   option domain
   option fqdn
}
```

In this case, client will mention that it is interested in FQDN by using Option Request and empty FQDN option, as specified in [15]. Server upon receiving such request (if it is configured to support it), will provide FQDN option containing domain name. Depending on the server's configuration, all DNS Updates will be performed by the server, forward will be performed by client and reverse by the server, or only forward will be done by a client.

It is also possible for client to provide its name as a hint for server. Server might take it into consideration when it will choose a name for this client. Example of a configuration file for such configuration is provided below:

```
# client.conf
log-level 7
iface eth0 {
    # ask for address
    ia

    # ask for options
    option dns-server
    option domain
    option fqdn zoe.example.com
}
```

Note that to successfully perform DNS Update, address must be assigned and dns server address must be known. So ,,ia" and ,,option dns-server" are required for ,,option fqdn" to work properly. Also if DHCPv6 server provides more than one DNS server address, update will be attempted only forthe first address on the list.

4.5.8 Example 8: Interface indexes

Usually, interface names are referred to by names, e.g. eth0 or Local Area Connection. Every system also provides unique number associated with each infterface, usually called ifindex or interface index. It is possible to read the number using `ip l` command (Linux) or `ipv6 ifx`. Below is an example, which demonstrate how to use interface indexes:

```
# client.conf
log-mode short
```

```
log-level 5
iface 5 {
  ia
}
```

4.6 Server configuration file

Server configuration is stored in `server.conf` file in the +/etc/dibbler+ (Linux systems) or in current (Windows systems) directory.

4.6.1 Global scope

Every option can be declared in a global scope. Global options can be defined here. Also options of a smaller scopes can be defined here – they will be used as a default values. Configuration file has following syntax:

```
global-options
interface-options
class-options
interface-declaration
```

4.6.2 Interface declaration

Each network interface, which should be serviced by the server, must be mentioned in the configuration file. Network interface is defined like this:

```
iface interface-name
{
  interface-options
  class-options
}
```

or

```
iface number
{
  interface-options
  class-options
}
```

where `interface-name` denotes name of the interface and `interface-number` denotes its number. Name no longer needs to be enclosed in single or double quotes (except Windows systems, when interface name contains spaces). Note that virtual interfaces, used to setup relay support are also declared in this way.

4.6.3 Class scope

Class is a smallest scope used in the server configuration file. It contains definition of the addresses, which will be provided to clients. Only class scoped parameters can be defined here. Address class is declared as follows:

```
class
{
  class-options
  address-pool
}
```

Address pool defines range of the addresses, which can be assigned to the clients. It can be defined in one of the following formats:

```
pool minaddress-maxaddress
pool address/prefix
```

4.6.4 Standard options

So called standard options are defined by the base DHCPv6 specification, a so called RFC 3315 document [5]. Those options are called standard, because all DHCPv6 implementations, should properly handle them. Each option has a specific scope it belongs to.

Standard options are declared in the following way:

```
OptionName option-value
```

work-dir – (scope: global, type: string, default: .) Defines working directory.

log-level – (scope: global, type: integer, default: 7) Defines verbose level of the log messages. The valid range if from 1 (Emergency) to 8 (Debug). The higher the logging level is set, the more messages dibbler will print.

log-name – (scope: global, type: string, default: Server). Defines name, which should be used during logging.

log-mode – (scope: global, type: short, full or precise, default value: full) Defines logging mode. In the default, full mode, name, date and time in the h:m:s format will be printed. In short mode, only minutes and seconds will be printed (this mode is useful on terminals with limited width). Recently added precise mode logs information with seconds and microsecond precision. It is a useful for finding bottlenecks in the DHCPv6 autoconfiguration process.

cache-size – (scope: global, type: integer, default: 1048576). It defines a size of the memory (specified in bytes) which can se used to store cached entries.

preference – (scope: interface, type: 0-255, default: none). Eech server can be configured to a specific preference level. When client receives several *ADVERTISE* messages, it should choose that server, which has the highest preference level. It is also worth noting that client, upon reception of the *ADVERTISE* message with preference set to 255 should skip wait phase for possible other *ADVERTISE* messages.

unicast – (scope: interface, type: address, default:none). Normally clients sends data to a well known multicast address. This is easy to achieve, but it wastes network resources as all nodes in the network must process such messages and also network load is increased. To prevent this, server might be configured to inform clients about its unicast address, so clients, which accept it, will switch to a unicast communication.

rapid-commit – (scope: interface, type: boolean, default: 0). This option allows rapid commit procedure to be performed. Note that enabling rapid commit on the server side is not enough. Client must be configured to allow rapid commit, too.

iface-max-lease – (scope: interface, type: integer, default: $2^{32}-1$). This parameter defines, how many normal addresses can be granted on this interface.

client-max-lease – (scope: interface, type: interger, default:$2^{32}-1$). This parameter defines, how many addresses one client can get. Main purpose of this parameter is to limit number of used addresses by misbehaving (malicious or restarting) clients.

relay – (scope: interface, type: string, default: not defined). Used in relay definition. It specifies name of the physical (or name of another relay, if cascade relaying is used) interface, which is used to receive and transmit relayed data. See 3.4, 4.7.8 and 4.7.9 for details.

interface-id – (scope: interface, type: integer, default: not defined). Used in relay definition. Each relay interface should have defined its unique identified. It will be sent in the *interface-id* option. Note that this value must be the same as configured in the dibbler-relay. See 3.4, 4.7.8 and 4.9 for details.

T1 – (scope: class, type: integer or integer range: default: $2^{32}-1$). This value defines after what time client should start renew process. Exact value or accepted range can be specified. When exact value is defined, client's hints are ignored completely.

T2 – (scope: class, type: integer or integer range, default:$2^{32}-1$). This value defines after what time client will start emergency rebind procedure if renew process fails. Exact value or accepted range can be specified. When exact value is defined, client's hints are ignored completely.

valid-lifetime (scope: class, type: integer or integer range, default:$2^{32}-1$). This parameter defines valid lifetime of the granted addresses. If range is specified, client's hints from that range are accepted.

prefered-lifetime (scope: class, type: integer or integer range, default:$2^{32}-1$). This parameter defines prefered lifetime of the granted addresses. If range is specified, client's hits from that range will be accepted.

class-max-lease – (scope: interface, type: interger, default:$2^{32}-1$). This parameter defines, how many addresses can be assigned from that class.

reject-clients – (scope: class, type: address or DUID list, default: none). This parameter is sometimes called black-list. It is a list of a clients, which should not be supported. Clients can be identified by theirs link-local addresses or DUIDs.

accept-only – (scope: class, type: address or DUID list, default: none). This parameter is sometimes called white-list. It is a list of supported clients. When this list is not defined, by default all clients (except mentioned in reject-clients) are supported. When accept-only list is defined, only client from that list will be supported.

4.6.5 Addional options

Server supports additonal options, not specified in [5]. They have following generic form:

```
option OptionName OptionsValue
```

All supported options are specified below:

dns-server – (scope: interface, type: address list, default: none). This option conveys information about DNS servers available. After retriving this information, clients will be able to resolve domain names into IP (both IPv4 and IPv6) addresses. Defined in [7].

domain – (scope: interface, type: domain list, default: none). This option is used for configuring one or more domain names, which clients are connected in. For example, if client's hostname is `alice.mylab.example.com` and it wants to contact `bob.mylab.example.com`, it can simply refer to it as `bob`. Without domain name configured, it would have to use full domain name. Defined in [7].

ntp-server – (scope: interface, type: address list, default: none). This option defines information about available NTP servers. Network Time Protocol [1] is a protocol used for time synchronisation, so all hosts in the network has the same proper time set. Defined in [13].

time-zone – (scope: interface, type: timezone, default: none). It is possible to configure timezone, which is provided by the server. Note that this option is considered obsolete as it is mentioned in draft version only [16]. Work on this draft seems to be abandoned as similar functionality is provided by now standard [13].

sip-server – (scope: interface, type: address list, default: none). Session Initiation Protocol [4] is an control protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences. Its most common usage is VoIP. Format of this option is defined in [6].

sip-domain – (scope: interface, type: domain list, default: none). It is possible to define domain names for Session Initiation Protocol [4]. Configuration of this parameter will ease usage of domain names in the SIP protocol. Format of this option is defined in [6].

nis-server – (scope: interface, type: address list, default: none). Network Information Service (NIS) is a Unix-based system designed to use common login and user information on multiple systems, e.g. universities, where students can log on to ther accounts from any host. Its format is defined in [11].

nis-domain – (scope: interface, type: domain list, default: none). Network Information Service (NIS) can albo specify domain names. It can be configured with this option. It is defined in [11].

nis+-server – (scope: interface, type: address list, default: none). Network Information Service Plus (NIS+) is an improved version of the NIS protocol. This option is defined in [11].

nis+-domain – (scope: interface, type: domain list, default: none). Similar to nis-domain, it defines domains for NIS+. This option is defined in [11].

lifetime – (scope: interface, type: boolean, default: no). Base spec of the DHCPv6 protocol does offers way of refreshing addresses only, but not the options. Lifetime defines, how often client should renew all its options. When defined, lifetime option will be appended to all replies, which server sends to a client. If client does not support it, it should ignore this option. Format of this option is defined in [14].

Lifetime is a special case. It is not set up by client in a system configuration. It is, however, used by the client to know how long obtained values are correct.

4.7 Server configuration examples

This subsection contains various examples of the server configuration. If you are interested in additional examples, download source version and look at `*.conf` files.

4.7.1 Example 1: Simple

In opposite to client, server uses only interfaces described in config file. Let's examine this common situation: server has interface named *eth0* (which is fourth interface in the system) and is supposed to assign addresses from 2000::100/124 class. Simplest config file looks like that:

```
# server.conf
iface eth0
{
  class
  {
    pool 2000::100-2000::10f
  }
}
```

4.7.2 Example 2: Timeouts

Server should be configured to deliver specific timer values to the clients. This example shows how to instruct client to renew (T1 timer) addresses one in 10 minutes. In case of problems, ask other servers in 15 minutes (T2 timer), that allowe prefered lifetime range is from 30 minutes to 2 hours, and valid lifetime is from 1 hour to 1 day. DNS server parameter is also provided. Lifetime option is used to make clients renew all non-address related options renew once in 2 hours.

```
# server.conf
iface eth0
{
  T1 600
  T2 900
  prefered-lifetime 1800-3600
  valid-lifetime 3600-86400
  class
  {
    pool 2000::100/80
  }

  option dns-server 2000::1234
  option lifetime 7200
}
```

4.7.3 Example 3: Limiting amount of addresses

Another example: Server should support 2000::0/120 class on eth0 interface. It should not allow any client to obtain more than 5 addresses and should not grant more then 50 addresses in total. From this specific class only 20 addresses can be assigned. Server preference should be set to 7. This means that this server is more important than all server with preference set to 6 or less. Config file is presented below:

```
# server.conf
iface eth0
```

```
{
  iface-max-lease 50
  client-max-lease 5
  preference 7
  class
  {
    class-max-lease 20
    pool 2000::1-2000::100
  }
}
```

4.7.4 Example 4: Unicast communication

Here's modified previous example. Instead of specified limits, unicast communication should be supported and server should listen on 2000::1234 address. Note that default multicast address is still supported. You must have this unicast address already configured on server's interface.

```
# server.conf
log-level 7
iface eth0
{
  unicast 2000::1234
  class
  {
    pool 2000::1-2000::100
  }
}
```

4.7.5 Example 5: Rapid-commit

This configuration can be called quick. Rapid-commit is a way to shorten exchange to only two messages. It is quite useful in networks with heavy load. In case if client does not support rapid-commit, another trick is used. Preference is set to maximum possible value. 255 has a special meaning: it makes client to skip wait phase for possible advertise messages from other servers and quickly request addresses.

```
# server.conf
log-level 7
iface eth0
{
  rapid-commit yes
  preference 255
  class
  {
    pool 2000::1/112
  }
}
```

4.7.6 Example 6: Access control

Administrators can selectively allow certain client to use this server (white-list). On the other hand, some clients could be explicitly forbidden to use this server (black-list). Specific DUIDs, DUID ranges, link-local addresses or the whole address ranges are supported. Here is config file:

```
# server.conf
iface eth0
{
  class
  {
    # duid of the rejected client
    reject-clients ‘‘00001231200adeaaa’’
    2000::2f-2000::20  // it’s in reverse order, but it works.
                       // just a trick.
  }
}
iface eth1
{
  class
  {
    accept-only fe80::200:39ff:fe4b:1abc
    pool 2000::fe00-2000::feff
  }
}
```

4.7.7 Example 7: Multiple classes

Although this is not common, a few users have requested support for multiple classes on one interface. Dibbler server can be configured to use several classes. When client asks for an address, one of the classes is being choosen on a random basis. If not specified otherwise, all classes have equal probability of being chosen. However, this behavior can be modified using `share` parameter. In the following example, server supports 3 classes with different preference level: class 1 has 100, class 2 has 200 and class 3 has 300. This means that class 1 gets $\frac{100}{100+200+300} \approx 16\%$ of all requests, class 2 gets $\frac{200}{100+200+300} \approx 33\%$ and class 3 gets the rest ($\frac{300}{100+200+300} = 50\%$).

```
# server.conf
log-level 7
log-mode short

iface eth0 {
 T1 1000
 T2 2000

 class {
   share 100
   pool 4000::1/80
 }
 class {
```

```
    share 200
    pool 2000::1-2000::ff
  }

  class {
    share 300
    pool 3000::1234:5678/112
  }
}
```

4.7.8 Example 8: Relay support

To get more informations about relay configuration, see section 3.4. Following server configuration example explains how to use relays. There is some remote relay with will send encapsulated data over eth1 interface. It is configured to append interface-id option set to 5020 value. Let's allow all clients using this relay some addresses and information about DNS servers:

```
# server.conf
iface relay1 {
  relay eth1
  interface-id 5020
  class {
    pool 2000::1-2000::ff
  }
  option dns-server 2000::100,2000::101
}
```

4.7.9 Example 9: 2 relays

This is advanced configuration. It assumes that client sends data to relay1, which encapsulates it and forwards it to relay2, which eventually sends it to the server (after additional encapsulation). It assumes that first relay adds interface-id option set to 6011 and second one adds similar option set to 6021.

```
# server.conf
iface relay1
{
  relay eth0
  interface-id 6011
}

iface relay2
{
  relay relay1
  interface-id 6021
  T1 1000
  T2 2000
  class {
    pool 6020::20-6020::ff
```

```
  }
}
```

4.7.10 Example 10: Dynamic DNS (FQDN)

Support for Dynamid DNS Updates was added recently. To configure it on the server side, list of available names must be defined. Each name can be reserved for a certain address or DUID. When no reservation is specified, it will available to everyone, i.e. the first client asks for FQDN will get this name. In following example, name 'zebuline.example.com' is reserved for address 2000::1, kael.example.com is reserved for 2000::2 and test.example.com is reserved for client using DUID 00:01:00:00:43:ce:25:b4:00:13:d4:02:4b:f5.

Also note that is required to define, which side can perform updates. This is done using single number after ,,option fqdn" phrase. Server can perform two kinds of DNS Updates: AAAA (forward resolving, i.e. name to address) and PTR (reverse resolving, i.e. address to name). To configure server to execute both updates, specify 2. This is a default behavior. If this value will be skipped, server will attempt to perform both updates. When 1 will be specified, server will update PTR record only and will leave updating AAAA record to the client. When this value is set to 0, server will not perform any updates.

The last parameter (64 in the following example) is a prefix length of the reverse domain supported by the DNS server, i.e. if this is set to 64, and 2000::/64 addresses are used, DNS server must support 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.ip6.arpa. zone.

```
# server.conf
log-level 8
log-mode precise
iface "eth1" {
 prefered-lifetime 3600
 valid-lifetime 7200
 class {
   pool 2000::1-2000::ff
 }

 option dns-server 2000::100,2000::101
 option domain example.com, test1.example.com
 option fqdn 2 64
        zebuline.example.com - 2000::1,
        kael.example.com - 2000::2,
        test.example.com - 0x0001000043ce25b40013d4024bf5,
        zoe.example.com,
        malcolm.example.com,
        kaylee.example.com,
        jayne.example.com
}
```

4.8 Relay configuration file

Relay configuration is stored in +relay.conf+ file in the +/etc/dibbler/+ (Linux systems) or in current directory (Windows systems).

4.8.1 Global scope

Every option can be declared in global scope. Config file consists of global options and one or more inteface definitions. Note that reasonable minimum is 2 interfaces, as defining only one would mean to resend messages on the same interface.

4.8.2 Interface declaration

Interface can be declared this way:

```
iface name_of_the_interface
{
  interface options
}
```

or

```
iface number
{
  interface options
}
```

where name_of_the_interface denotes name of the interface and number denotes it's number. It does not need to be enclosed in single or double quotes (except windows cases, when interface name contains spaces).

4.8.3 Options

Every option has a scope it can be used in, default value and sometimes allowed range.

Name	Scope	Values (default)	default	Description
log-level	global	1-8	8	log-level (8 is most verbose)
log-name	global	string	Relay	Name, which appears in a log file
log-mode	global	short or full	full	logging mode: short (date and name suppressed) or full
client multicast	interface	boolean		Client's messages should be received on the multicast address.
client unicast	interface	address	not defined	Client's messages should be received on the specified multicast address.
server multicast	interface	boolean		Forwarded messages should be sent to the multicast address.
server unicast	interface	address	not defined	Forwarded messages should be send to the specified address.
interface-id	interface	integer	not defined	Identifier of that particular interface. Used for interface-id option.

It is worth mentioning that interface-id should be specified on the interface, which is used to receive messages from the clients, not the one used to forward packets to server.

4.9 Relay configuration examples

Relay configuration file is fairly simple. Relay forwards DHCPv6 messages between interfaces. Messages from client are encapsulated and forwarded as RELAY_FORW messages. Replies from server are received as RELAY_REPL message. After decapsulation, they are being sent back to clients.

It is vital to inform server, where this relayed message was received. DHCPv6 does this using interface-id option. This identifier must be unique. Otherwise relays will get confused when they will receive reply from server. Note that this id does not need to be alligned with system interface id (ifindex). Think about it as "ethernet segment identifier" if you are using Ethernet network or as "bss identifier" if you are using 802.11 network.

If you are interested in additional examples, download source version and look at `*.conf` files.

4.9.1 Example 1: Unicast/multicast

Let's assume this case: relay has 2 interfaces: eth0 and eth1. Clients are located on the eth1 network. Relay should receive data on that interface using well-known ALL_DHCP_RELAYS_AND_SERVER multicast address (ff02::1:2). Relay also listens on its global address 2000::123. Packets received on the eth1 should be forwarded on the eth0 interface, also using multicast address:

```
# relay.conf
log-level 8
log-mode short
iface eth0 {
  server multicast yes
}
iface eth1 {
  client multicast yes
  client unicast 2000::123
  interface-id 1000
}
```

4.9.2 Example 2: Multiple interfaces

Here is another example. This time messages should be forwarded from eth1 and eth3 to the eth0 interface (using multicast) and to the eth2 interface (using server's global address 2000::546). Also clients must use multicasts (the default approach):

```
# relay.conf
iface eth0 {
  server multicast yes
}
iface eth2 {
  server unicast 2000::456
}
iface eth1 {
  client multicast yes
  interface-id 1000
}
iface eth3 {
  client multicast yes
  interface-id 1001
}
```

4.9.3 Example 3: 2 relays

Those two configuration files correspond to the ,,2 relays" example provided in the server example 8. See 3.4 for details.

```
# relay.conf - relay 1
log-level 8
log-mode full

# messages will be forwarded on this interface using multicast
iface eth2 {
   server multicast yes    // relay messages on this interface to ff05::1:3
 # server unicast 6000::10 // relay messages on this interface to this global address
}

iface eth1 {
#  client multicast yes    // bind ff02::1:2
  client unicast 6011::1   // bind this address
  interface-id 6011
}
```

```
# relay.conf - relay 2
iface eth0 {
#   server multicast yes  // relay messages on this interface to ff05::1:3
  server unicast 6011::1  // relay messages on this interface to this global address
}

# client can send messages to multicast
# (or specific link-local addr) on this link
iface eth1 {
  client multicast yes    // bind ff02::1:2
# client unicast 6021::1  // bind this address
  interface-id 6021
}
```

5 Frequently Asked Questions

Soon after initial Dibbler version was released, feedback from user regarding various things started to appear. Some of the questions were common enough to get into this section.

5.1 Common Questions

Q: Why client does not configure routing after assigning addresses, so I cannot e.g. ping other hosts?
A: It's rather difficult problem. DHCP's job is to obtain address and it exactly does that. To ping any other host, routing should be configured. And this should be done using Router Advertisements. It's kinda odd, but that's the way it was meant to work. If there will be requests from users, I'll think about some enchancements.

Important note: This behaviour has changed in the 0.5.0 release. See *strict-rfc-no-routing* directive in the 4.4 section.

Q: Dibbler sends some options which have values not recognized by the Ethereal/Wireshark or by other implementations. What's wrong?

A: DHCPv6 is a relatively new protocol and additional options are in a specification phase. It means that until standarisation process is over, they do not have any officially assigned numbers. Once standarization process is over (and RFC document is released), this option gets an official number.

There's pretty good chance that different implementors may choose diffrent values for those not-yet officialy accepted options. To change those values in Dibbler, you have to modify file misc/DHCPConst.h and recompile server or client. See Developer's Guide, section *Option Values* for details.

Currently options with assigned values are:

- RFC3315: *CLIENT_ID*, *SERVER_ID*, *IA_NA*, *IAADDR*, *OPTION_REQUEST*, *PREFERENCE*, *ELAPSED*, *STATUS_CODE*, *RAPID-COMMIT*, *IA_TA*, *RELAY_MSG*, *AUTH_MSG*, *USER_CLASS*, *VENDOR_CLASS*, *VENDOR_OPTS*, *INTERFACE_ID*, *RECONF_MSG*, *RECONF_ACCEPT*;
- RFC3319: *SIP_SERVERS*, *SIP_DOMAINS*;
- RFC3646: *DNS_RESOLVERS*, *DOMAIN_LIST*;
- RFC3633: *IA_PD*, *IA_PREFIX*;
- RFC3898: *NIS_SERVERS*, *NIS+_SERVERS*, *NIS_DOMAIN*, *NIS+_DOMAIN*.

Take note that Dibbler does not support all of them. There are several options which currently does not have values assigned (in parenthesis are numbers used in Dibbler): *NTP_SERVERS* (40), *TIME_ZONE* (41), *LIFETIME* (42), *FQDN* (43).

Q: I can't get (insert your trouble causing feature here) to work. What's wrong?

A: Go to the project `http://klub.com.pl/dhcpv6/`homepage and browse list archives. If your problem was not reported before, please don't hesitate to write to the mailing list or contact author directly.

5.2 Linux specific questions

Q: I can't run client and server on the same host. What's wrong?

A: First of all, running client and server on the same host is just plain meaningless, except testing purposes only. There is a problem with sockets binding. To work around this problem, consult Developer's Guide, Tip section how to compile Dibbler with certain options.

Q: After enabling unicast communication, my client fails to send REQUEST messages. What's wrong?

A: This is a problem with certain kernels. My limited test capabilites allowed me to conclude that there's problem with 2.4.20 kernel. Everything works fine with 2.6.0 with USAGI patches. Patched kernels with enhanced IPv6 support can be downloaded from `http://www.linux-ipv6.org/`. Please let me know if your kernel works or not.

5.3 Windows specific questions

Q: After installing *Advanced Networking Pack* or *Windows XP ServicePack2* my DHCPv6 (or other IPv6 application) stopped working. Is Dibbler compatible with Windows XP SP2?

A: Both products (Advanced Networking Pack as well as Service Pack 2 for Windows XP) provide IPv6 firewall. It is configured by default to reject all incoming IPv6 traffic. You have to disable this firewall. To disable firewall on the ,,Local Area Connection" interface, issue following command in a console:

```
netsh firewall set adapter "Local Area Connection" filter=disable
```

Q: Server or client refuses to create DUID. What's wrong?

A: Make sure that you have at least one up and running interface with at least 6 bytes long MAC address. Simple ethernet or WIFI card matches those requirements. Note that network cable must be plugged (or in case of wifi card – associated with access point), otherwise interface is marked as down.

6 Miscellaneous topics

6.1 History

Dibbler project was started as master thesis by Tomasz Mrugalski and Marek Senderski on Computer Science faculty on Gdansk University of Technology. Both authors graduated in september 2003 and soon after started their jobs.

During master thesis writing, it came to my attention that there are other DHCPv6 implementations available, but none of them has been named properly. Refering to them was a bit silly: ,,DHCPv6 published on sourceforge.net has better support than DHCPv6 developed in KAME project, but our DHCPv6 implementation...". So I have decided that this implementation should have a name. Soon it was named Dibbler after famous CMOT Dibbler from Discworld series by Terry Pratchett.

Sadly, Marek does not have enough free time to develop Dibbler, so his involvement is non-existent at this time. However, that does not mean, that this project is abandoned. It is being actively developed by me (Tomek). Keep in mind that I work at full time and do Ph.D. studies, so my free time is also greatly limited.

6.2 Contact

There is an website located at http://klub.com.pl/dhcpv6. If you belive you have found a bug, please put it in Bugzilla – it is a bug tracking system located at http://klub.com.pl/bugzilla. If you are not familiar with that kind of system, don't worry. After simple registration, you will be asked for system and Dibbler version you are using and so on. Without feedback from users, author will not be aware of many bugs and so will not be able to fix them. That's why users feedback is very important. You can also send bug report directly using e-mail. Be sure to be as detailed as possible. Please include both server and client log files, both config and xml files. If you are familiar with tcpdump or ethereal, traffic dumps from this programs are also great help.

If you have used Dibbler and it worked ok, this documentation answered all you question and everything is in order (hmmm, wake up, it must be a dream, it isn't reality:), also send a short note to author. He can be contated at thomson(at)klub(dot)com(dot)pl (replace (at) with @ and dot with .). Be sure to include information which country do you live in. It's just author's curiosity to know where Dibbler is being used or tested.

6.3 Thanks and greetings

I would like to send my thanks and greetings to various persons. Without them, Dibbler would not be where it is today. For a full list of contributors, see AUTHORS file.

Marek Senderski – He's author of almost half of the Dibbler code. Without his efforts, Dibbler would be simple, long forgotten by now master thesis.

Jozef Wozniak – My master thesis' supervisor. He allowed me to see DHCP in a larger scope – as part of total automatisation process.

Jacek Swiatowiak – He's my master thesis consultant. He guided Marek and me to take first steps with DHCPv6 implementation.

Ania Szulc – Discworld fan and a great girl, too. She's the one who helped me to decide how to name this yet-untitled DHCPv6 implementation.

Christian Strauf – Without his queries and questions, Dibbler would be abandoned in late 2003.

Bartek Gajda – His interest convinced me that Dibbler is worth the effort to develop it further.

Artur Binczewski and Maciej Patelczyk – They both ensured that Dibbler is (and always will be) GNU GPL software. Open source community is grateful.

Josep Sole – His mails (directly and indirectly) resulted in various fixes and speeded up 0.2.0 release.

Sob – He has ported 0.4.0 back to Win2000 and NT. As a direct result, 0.4.1 was released for those platforms, too.

Guy "GMSoft" Martin – He has provided me with access to HPPA machine, so I was able to squish some little/big endian bugs. He also uploaded ebuild to the Gentoo portage.

Bartosz "fEnio" Fenski – He taught me how much work needs to be done, before deb packages are considered ok. It took me some time to understand that more pain for the package developer means less problems for the end user. Thanks to him, Dibbler is now part of the Debian GNU/Linux distribution.

Adrien Clerc and his team – Their contribution of the DNS Updates code is most welcome.

Krzysztof Wnuk – He has fixed, improved and extended DNS Updates support as well as provided initial support for prefix delegation.

References

[1] Mills, D., "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI", RFC2030, IETF, October 1996.

[2] T. Narten, E. Nordmark and W. Simpson "Neighbor Discovery for IP Version 6 (IPv6)", RFC2461, December 1998.

[3] S. Thomson, and T. Narten "IPv6 Stateless Address Autoconfiguration", RFC2462, IETF, December 1998.

[4] J.Rosenberg and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC3263, IETF, June 2002.

[5] R. Droms, Ed. "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC3315, IETF, July 2003.

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

[7] S. Thomson, C. Huitema, V. Ksinant and M. Souissi "DNS Extensions to Support IP Version 6", RFC3596, IETF, October 2003.

[8] O. Troan, and R. Droms "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC3633, IETF, December 2003.

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

[10] R. Droms, "Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6", RFC3736, IETF, April 2004.

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

[12] R. Arends, R. Austein, M. Larson, D. Massey and S. Rose "DNS Security Introduction and Requirements", RFC4033, IETF, March 2005

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

[14] S. Venaas, T. Chown, and B. Volz "Information Refresh Time Option for DHCPv6", work in progress, IETF, January 2005, draft-ietf-dhc-lifetime-03.txt.

[15] B. Volz "The DHCPv6 Client FQDN Option", work in progress, IETF, September 2005, draft-ietf-dhc-dhcpv6-fqdn-05.txt.

[16] A.K. Vijayabhaskar "Time Configuration Options for DHCPv6", work in progress, IETF, October 2003, draft-ietf-dhc-dhcpv6-opt-timeconfig-03.txt.