Dibbler – a portable DHCPv6
User's guide

Tomasz Mrugalski
thomson(at)klub.com.pl

2006-08-22

0.4.2-CVS_FQDN+TA_support

Contents

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 question or you have any suggestions, feel free to contact me. See *Contact* section for details. Also be sure to check out Dibbler website located at http://klub.com.pl/dhcpv6/.

2 Overview

Dibbler is a portable DHCPv6 solution. It 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[1], it also offers serveral enhancements, e.g. DNS servers and doman 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). Sources are also available, so anyone skilled enough can fix bugs, add new features and release his/her own version.

As for now, Dibbler offers these features:

- Basic server discovery and address assignment (SOLICIT, ADVERTISE, REQUEST and REPLY messages) – simplest case possible: client discovers server, then asks for an address, which is granted by a server.
- Best server discovery – when client receives more than one ADVERTISE messages from different servers, it chooses the best one and remembers remaining ones as a backup.
- Many servers support – client is capable of discovering and dealing with multiple 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 – Dibbler server supports indirect communication with clients via relays. Stand-alone relay implementation is also available. Clients can talk to the server directly or via relays.
- 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.[2]
- Address renewal (RENEW,REBIND and REPLY messages) – client renews addresses at certain time intervals, if server specified so.
- Duplicate address detection (DECLINE and REPLY messages) – client can detect and properly handle faulty situation, when server grants address which is illegaly used by some other host. It will inform server of such circumstances, and request for another address. Server will mark this address as used by unknown host, and will assign another address to a client.
- Power failure/crash support (CONFIRM and REPLY messages) – after client recovers from crash or power failure, it still can have assigned valid addresses. In such circumstances, client uses CONFIRM message, to config if those addresses are still valid[3].

[1] specified in RFC3315

[2] 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.

[3] As for 0.4.2 version, this functionality works on server side only, client side support will be available in future releases.

- IA Option – this option is used to carry addresses. Both server and client support multiple IAs in one message. Additional feature is client capability to ask for a specific address.
- TA Option – this option is used to carry temporary addresses. Both server and client supports this. Note that it is better to use non-temporary, normal (ia) addresses. If you don't know, what TA option is, you definetely don't need it.
- Rapid Commit Option (SOLICIT and REPLY messages) – if both client and server are configured to use rapid commit, address assignment procedure can be shortened to 2 messages. Major advantage is lesser network usage and quicker client startup time.

Except RFC3315-specified behavior, Dibbler also support several enhancements:

- DNS Servers Option – client can ask for information about DNS servers. DHCPv6 server will provide those addresses.
- Domain Name Option – client can ask for information about domain name it is connected in.
- Time Zone Option – client can ask for information about time zone it is currently in.
- NTP Servers Option – client can ask for Network Time Protocol Servers to synchronize its clock.
- SIP Servers Option – SIP servers IPv6 address information can be passed to clients.
- SIP domain name - SIP domain name can be passed to clients.
- NIS, NIS+ server Option – Both NIS and NIS+ server adresses can be passed to clients.
- NIS, NIS+ domain name Option – NIS or NIS+ domain names can be passed to clients.
- Option renewal mechanism (Lifetime Option) – options obtained from server can be updated periodically.
- Dynamic DNS Updates – server can assign a fully qualified domain name for a client. Client is able to perform DNS Update, i.e. inform DNS server about its name. Currently server is able to provide such names, but does not support the DNS Update procedure by itself and

And now some implementation specific details:

- Server, client and relay, after each action dumps state to disk in a XML format, so it can be easily processed in an automated manner. Simple example of this advantage is a script, which can generate reports about server usage (assigned addresses, clients configured and so on);
- 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.
- 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 Linux Distribution 2.0 as well as well as in Debian GNU/Linux. You can download them from http://www.pld-linux.org/ or http://www.debian.org. Keep in mind that author has not tested those ports, so there might be some unknown issues present.

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

3 Requirements

Dibbler can be run on Linux systems with kernels from 2.4 and 2.6 series. Obviously, IPv6 (compiled into kernel or as module) support is required to run. 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, don't expect this state to change.

4 Installation and usage

Client, server and relay are installed in the same way. Installation method is different in Windows and Linux systems, so they're described separately. To simplify installation, it assumes that binary versions are used[4].

4.1 Linux installation

Starting with 0.4.1, there will are 3 different packages: client, server and relay. For some architectures there will be also documentation package provided. During writing this documentation, Dibbler is already present in:

- the PLD Linux Distribution 2.0 (in the test section)
- Gentoo GNU/Linux

There are efforts under way to include Dibbler in Debian GNU/Linux distribution.

Obtain (e.g. download from http://klub.com.pl/dhcpv6/) an archive, which suits your needs. Currently there are provided RPM packages (which can be used in RedHat, Fedora Core, Mandrake or PLD distribution), DEB packages (suitable for Debian or Knoppix) and ebuild (for Gentoo users). To install rpm package, run *rpm – iarchive.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.rpm
```

To install Dibbler in Gentoo systems, just type:

```
emerge dibbler
```

If you would like to install Dibbler from sources, download tar.gz source archive, extract it, type make followed by target (e.g. server, client or relay[5]). After successful compilation type make install. For example, to build server and relay, type:

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

[5] To get full target list, type: *makehelp*

```
tar zxvf dibbler-0.1.0-src.tar.gz
make server relay
make install
```

Depending what functionality do you want to use (server,client or relay), you should edit config file (`client.conf` for client, `server.conf` for server and `relay.conf` for relay). All config files should be placed in the `/etc/dibbler` directory. After editing, issue one of the following commands:

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

`start` parameter needs a little comment. It instructs Dibbler to run in daemon mode – detach from console and run in the background. During config 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. 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
```

4.2 Windows installation

Starting at 0.2.1, Dibbler supports Windows XP and 2003. In version 0.4.1 exprimental support for Windows NT4 and 2000 is provided. The easiest way is to download clickable Windows installer. Download it 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.

4.3 Setting up IPv6 in Linux

IPv6 can be enabled in Linux systems in two ways: compiled directly into kernel or as a module. To verify if you have IPv6 support, issue following command: `ping6 ::1`. If you get replies, you have IPv6 already installed. If this fails, try to load IPv6 module: `modprobe ipv6` (issued 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 in the net, just type "kernel compilation howto" in http://www.google.com.

4.4 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`.

4.5 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: 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. [6]
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 connection to which you want to add the IPv6 protocol, and then click Properties. Typically, this connection 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.

4.6 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: 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.
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 should be installed.

[6] 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 2048.

5 Compilation

Dibbler is distributed in 2 versions: binary and source. For most users, binary version is better choice. Compilation is performed by more experienced users, preferably with programming knowledge. It does not offer any advances 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*.

5.1 Linux compilation

Compilation in most cases is not necessary and should be performed only by experienced users. Perferred method is to use binaries provided on Dibbler's website. 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 and 3.4 versions. Lexer files were generated using flex 2.5.31. Parser file were created using bison++ 1.21.9[7]. Everything was developed under Debian GNU/Linux system.

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. Additional information about compilation can be found in *Dibbler Developer's Guide*.

5.2 Windows XP/2003 compilation

Download dibbler-0.3.0-src.tar.gz and extract it. In Port-winxp there will be project files (for server, client and relay) for MS Visual Studio. Open one of them and click Build command. That should do the trick. Additional information about compilation can be found in *Dibbler Developer's Guide*.

5.3 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 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 Features HOWTO

This recently added section contains information about setting up various Dibbler features.

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

6.1 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.

Note: 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 ([2], [7]):

stateful – when 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 normal (i.e. non-temporary) addresses. It will use *IA* (Identity Association for Non-temporary Addresses, see [2] for details) to request and retrieve addresses. Since this is a default behavior, it does not have to be explicitely 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 [2] 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.

```
# 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 an 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 – Same as `ia`. 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.

6.2 DNS Update

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.

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 executed:

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.

```
options {
        listen-on-v6 { any; };
        listen-on { any; };

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

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

      // other options follow
      // ...
};
```

Note: Make sure that unix user, which runs BIND, is able to create and write file pri/example.com.jnl.

After configuration, client should log following line, which informs that Dynamic DNS Update was completed successfully.

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

DNS Update can now be verified using dig command line tool (a part of the dnsutils package). Command syntex is as follows: `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:` below:

```
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
```

See example server and client configuration files described in a sections 7.5.7 and 7.7.10. Also note that Dibbler distribution should be accompanied with several example configuration files. Some of them include FQDN usage examples.

6.3 Address caching

Previous Dibbler versions assigned a random address from a available address pool, so the same client received different address each time it asked for one. In the 0.5.0 release, two new mechanisms were introduced to make sure that the same client gets the same address each time.

FIXME:

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, 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.

6.4 Relays

FIXME: Provide description and maybe a figure.

6.5 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 client only:

- client-CfgMgr.xml - Represents information read from a configuration file as its associated status (e.g. CONFIGURED or FAILED).
- client-IfaceMgr.xml - Represens detected interfaces in the operating system, as well as bound sockets and similar information.
- client-AddrMgr.xml - This is database, which contains identity associations with associated addresses.

7 Configuration files

This section describes Dibbler server and (optional) 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, of course. eth0 and ETH0 are dwo diffrent interfaces.

7.1 Tokens and basic informations

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

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 string'. If string contains only a-z, A-Z and 0-9 characters, quotes can be omited.

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

IPv6 address list – IPv6 addresses separated with commas.

DUID list – DUIDs separated with commas.

string list – strings separated with comas.

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

7.2 Scopes

There are four scopes, in which options can be specified: global, inteface, IA and address.

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. Every option is specific for one scope. 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 default value for T1 is ...". The same applies to global scope. Options can be used multiple times. In that case value defined later is used.

7.3 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

7.4 Client configuration file

Client config 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 [8].

7.4.1 Global scope

Every option can be declared in global scope. Config file has this form:

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

7.4.2 Interface declaration

Interface can be declared this way:

```
iface interface_name
{
  interface options
  IA options
  address options
}
```

or

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

In every case, number denotes interface number. It can be extracted from ,,ip l" (Linux) or ,,ipv6 if" (Windows). `interface_name` is an interface name. Also take a note that name of the interface no longer needs 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".

7.4.3 IA declaration

IA is a short for Identity Association. It is a logical entity representing address or addresses used to perform some functions. Almost always, each DHCPv6 client will have exactly one IA. IA is declared this way:

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

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

where 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.

7.4.4 Address declaration

Addres is declared like this:

```
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.

7.4.5 Standard options

Standard options are... well, standard. This means that they have nothing to do with any extensions. Standard options are declared this way:

```
OptionName option-value
```

Every option has a scope it can be used in, default value and sometimes allowed range. Parameters denoted with (H) are used as hints for the server. Value of *work-dir* option is currently not used. In *log-mode* option, `short` and `full` values are supported. `syslog` and `eventlog` will be available in future releases. rapid-commit and unicast expect one boolean parameter. It can be TRUE, FALSE, YES, NO, 0 or 1. Setting log level to too low value (5 or less) can result in mysterious behavior. 6 or 7 is a recommended value.

Name	Scope	Values (default)	default	Description
valid-lifetime	address	integer	4294967296	valid lifetime for address (specified in seconds) (H)
prefered-lifetime	address	integer	4294967296	after this amount of time(in seconds) address becomes depreciated (H)
T1	IA	integer	4294967296	client should renew addresses after T1 seconds (H)
T2	IA	integer	4294967296	client should send REBIND after T2 seconds (H)
reject-servers	IA	addrs or DUID list	empty	list containing servers which should be discarded in configuration of this IA
prefered-servers	IA	addrs or DUID list	empty	Prefered servers list. ADVERTISE messages received by client are sorted according to this list.
rapid-commit	interface	0 or 1	0	should we use Rapid Commit?
unicast	interface	0 or 1	0	Is unicast communication allowed?
work-dir	global	string	empty	working directory
log-level	global	1-8	8	log-level (8 is most verbose)
log-name	global	string	Client	Name, which appears in a log file
log-mode	global	short or full	full	logging mode: short (date and name suppressed) or full.
strict-rfc-no-routing	global	none	none	if this option is present, routing will not be configured

7.4.6 Addional options

Additional options are the options specified in external drafts and in RFC documents. They are declared with `option` keyword:

```
option OptionName option-value
```

where OptionName is one of possible values listed below:

OptionName	Scope	Values (default)	default	Description
dns-server	interface	addrs list	not defined	preferred DNS servers list (H)
domain	interface	domains list	not defined	preferred domain (H)
ntp-server	interface	addrs list	not defined	preferred NTP servers list (H)
time-zone	interface	timezone	not defined	preferred time zone (H)
sip-server	interface	addrs list	not defined	preferred SIP servers list (H)
sip-domain	interface	domains list	not defined	preferred SIP domain (H)
nis-server	interface	addrs list	not defined	preferred NIS servers list (H)
nis-domain	interface	domain	not defined	preferred NIS domain (H)
nis+-server	interface	addrs list	not defined	preferred NIS+ servers list (H)
nis+-domain	interface	domain	not defined	preferred NIS+ domain (H)
lifetime	interface	YES/NO	no	Should client request lifetime option?

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 them in separate files in the working directory, e.g. `option-dns-server`. Several options are 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	system, 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

7.4.7 Stateless configuration

If interface does not contain `IA` keyword, one IA with one address is assumed. If client should not request for address on this interface, *stateless* [9] must be used. In such circumstances, only specified options will be requested.

7.4.8 Relay support

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

7.5 Client configuration examples

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

7.5.1 Example 1: Default

In simplest case, client config 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)
- which are not running (flag RUNNING not set)
- which are not multicast capable (flag MULTICAST not set)

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

[9] 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.

7.5.2 Example 2: DNS

Simple config config file requesting 1 address and DNS configuration on eth0 interface looks like that:

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

7.5.3 Example 3: Timeouts and specific address

Another example is presented below. Client asks for 1 address and would like it to be 2000::1:2:3. Rapid-commit is allowed and client would like to renew this address once in a 10 minutes.

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

7.5.4 Example 4: Unicast, more than one address

Here's yet another example. We would like to obtain 2 addresses on ,,Local Area Connection" interface. Note quotation marks around interface name. They're necessary since this particular interface name contains spaces. Client also would like to accept Unicast communication if server supports it. We don't care for details, so keep those log very short. Take note that you won't be able to what Dibbler is doing with such low log-level. (Preferred log-level is 7). Config file looks like that:

```
# client.conf
log-mode short
log-level 5
iface "Local Area Connection" {
  unicast yes
  ia 2
}
```

7.5.5 Exmaple 5: Rapid-commit

Rapid-commit is a shortened exchange with server. It consists of only two messages, instead of the usual four. Instead of using interface name, we provide its index. It is worth to know that server must also support rapid-commit.

```
# client.conf
iface 3 {
  rapid-commit yes
  ia
  option dns-server
}
```

7.5.6 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 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
}
```

7.5.7 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. Server upon receiving such request (if it is configured to support it), will provide FQDN option containing domain name.

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
}
```

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

7.6 Server configuration file

Server configuration is stored in `server.conf` file in the +/etc/dibbler+ (Linux systems) or in current (Windows systems) directory. After successful startup, old version of this file is stored as `server.conf-old`.

7.6.1 Global scope

Every option can be declared in global scope. Config file has this form:

```
interface declaration  |
global options         |
interface options      |
class options
```

7.6.2 Interface declaration

Interface can be declared this way:

```
iface name_of_this_interface
{
  interface options      |
  class options
}
```

or

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

where name_of_this_interface denotes name of the interface and number denotes it's number. It no longer needs to be enclosed in single or double quotes (except windows cases, when interface name contains spaces).

7.6.3 Class scope

Address class is declared as follows:

```
class
{
     class options |
     address pool
}
```

address pool can be defined in one of the following formats:

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

7.6.4 Options

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

Name	Scope	Values (default)	default	Description
work-dir	global	string	empty	working directory
log-level	global	1-8	8	log-level (8 is most verbose)
log-name	global	string	Client	Name, which appears in a log file
cache-size	global	integer	1048576	size of the address cache, specified in bytes
preference	interface	0-255	0	server preference value (higher is more prefered)
unicast	interface	address	empty	Specify which address should be used.
iface-max-lease	interface	integer	4294967296	how many addresses can be leased by all clients?
client-max-lease	interface	integer	4294967296	how many addresses can be leased by one client?
rapid-commit	interface	0 or 1	0	should we allow Rapid Commit (SOLICIT–REPLY)?
relay	interface	string	not defined	Name of the physical interface used to reach this relay
interface-id	interface	integer	not defined	ID of the relay interface. Must be unique
valid-lifetime	class	integer	4294967296	valid lifetime for address (specified in seconds)
prefered-lifetime	class	integer	4294967296	after this amount of time(in seconds) address becomes depreciated
T1	class	integer	4294967296	client should renew addresses after T1 seconds
T2	class	integer	4294967296	client should send REBIND after T2 seconds
reject-clients	class	addrs or DUID list	empty	list containing servers which should be discarded in configuration of this IA
accept-only	class	addrs or DUID list	empty	these are the only clients allowed to use this class
class-max-lease	class	integer	4294967296	how many addresses can be leased from this class?

log-mode – defines logging level. Currently supported options are: full (daemon name, date and time), short (minutes and seconds), precise (seconds and microseconds, used mainly for finding bottlenecks). In future releases, syslog (unix only) and eventlog (windows only) will be supported. Default is full.

7.6.5 Addional options

Server supports additonal options, not specified in RFC3315. They have generic form:

```
option OptionName OptionsValue
```

All supported options are specified in the table below:

OptionName	OptionsValue	Default	Description
dns-server	addrs list	empty	DNS servers list
domain	string list	empty	domain names list
ntp-server	addrs list	empty	NTP servers list
time-zone	timezone	empty	time zone
sip-server	addrs list	empty	SIP servers list
sip-domain	string list	empty	domain names list
nis-server	addrs list	empty	NIS servers list
nis-domain	string	empty	domain name
nisplus-server	addrs list	empty	NIS+ servers list
nis-domain	string	empty	domain name
lifetime	integer	empty	how often renew options?

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.

7.7 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.

7.7.1 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
  }
}
```

7.7.2 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
}
```

7.7.3 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
  }
}
```

7.7.4 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
  }
}
```

7.7.5 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
  }
}
```

7.7.6 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
  }
}
```

7.7.7 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
 }
}
```

7.7.8 Example 8: Relay support

To get more informations about relay configuration, see section 6.4. 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
}
```

7.7.9 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
  }
}
```

7.7.10 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' is reserved for address 2000::1, kael 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 possible 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.

```
# 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
         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
}
```

7.8 Relay configuration file

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

7.8.1 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.

7.8.2 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).

7.8.3 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	Client	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.

7.9 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.

7.9.1 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
}
```

7.9.2 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
}
```

7.9.3 Example 3: 2 relays

Those two configuration files correspond to the ,,2 relays" example provided in the server example 8. See 6.4 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
}
```

8 Frequently Asked Question

Soon after Dibbler was published, I started to receive questions from users. Some of them were common enough to get into this section.

8.1 Common

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.

Q: Dibbler sends some options which have values not recognized by the Ethereal 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).

8.2 Linux specific

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.

8.3 Windows specific

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 provide IPv6 firewall. It is configured by default to reject all incoming IPv6 traffic. You have to disable this firewall. To do so, 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 card matches those requirements. Note that network cable must be plugged, otherwise interface is marked as down.

9 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.

10 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.

11 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.

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.

References

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

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

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

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

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

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

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

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

[9] S. Venaas, T. Chown, and B. Volz "Information Refresh Time Option for DHCPv6", work in progress, IETF, January 2005

[10] B. Volz "The DHCPv6 Client FQDN Option", work in progress, IETF, September 2005