Dibbler – a portable DHCPv6
User's guide

Tomasz Mrugalski
thomson(at)klub.com.pl

2005-03-16

0.4.0

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 and Linux 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 or add new features.

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. 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].
- 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.

[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 0.4.0 version, this functionality works on server side only, client side support will be available in future releases.

- 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.
- 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.

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: AMD64, PowerPC, Sparc and Alpha. They are available in the PLD Linux Disctribution 2.0 in test section. You can download them from `ftp://ftp.pld-linux.org/dists/2.0/test/`. Keep in mind 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. As DHCPv6 uses UDP ports below 1024, root privileges are required.

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.

4 Installation and usage

Client, server and relay are installed in the same way. Installation method is different in WindowsXP 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 0.4.0, 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. There are efforts under way to include Dibbler in Debian GNU/Linux distribution as well as in Gentoo 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 0.4.0, issue following command:

```
rpm -i dibbler-0.4.0-1.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_0.4.0-1_i386.rpm
```

To install Dibbler in Gentoo systems, download ebuild script and issue command:

```
emerge dibbler-0.3.1.ebuild
```

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:

```
tar zxvf dibbler-0.4.0-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:

[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*

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

To see, if client, server or relay are running[6], 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. The easiest way it 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.

5 Compilation

Dibbler is distributed in 2 versions: binary and source files. 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 WindowsXP/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 C++ 2003. 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*.

[6] Running status is based on /var/lib/dibbler/*.pid files. In rare occasions, when server crashes, this status will show server status as running.

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

6 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.

6.1 Tokens and basic informations

Config file parsing is token-based. 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.

6.2 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, until 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.

6.3 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

6.4 Client configuration file

Client config file should be named `client.conf`. 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].

6.4.1 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
```

6.4.2 Interface declaration

Interface can be declared this way:

```
iface name_of_this_interface
{
  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). `name_of_this_interface` 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".

6.4.3 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.

6.4.4 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.

6.4.5 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.

6.4.6 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	Support (Linux)	Support (winXP/2003)
dns-server	system, file	system, file
domain	file	system, file
ntp-server	file	file
time-zone	file	file
sip-server	file	file
sip-domain	file	file
nis-server	file	file
nis-domain	file	file
nis+-server	file	file
nis+-domain	file	file

6.4.7 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.

6.4.8 Client configuration file examples

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.

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

```
log-mode short
log-level 7
iface eth0 {
  option dns-server
  ia {
  }
}
```

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:

[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.

```
log-mode short
log-level 7
iface eth0 {
  rapid-commit YES
  ia {
    address {
      2000::1:2:3
    }
  }
}
```

Here's yet another example. We would like to obtain 2 addresses on ,,Local Area Connection" interface. Unicast communication is ok in this scenario. We don't care for details, so keep those log very short. Config file looks like that:

```
log-mode short
log-level 3
iface "Local Area Connection" {
  ia {
    address
    address
  }
}
```

Here is a more coplicated case. Let's say there are 4 interfaces, numbered 1 thru 4. Interfaces 1,2 and 3 are not to be configured. Interface 4, named eth0 should have 3 IAs. Two of them are supposed to contain one address each. Third IA should contain 3 addresses. Addresses assigned to first and second IA should have prefered-lifetime 1 hour and valid-lifetime 2 hours. This IA should have 3 specific addresses: 2000::1, 2000::2 and 2000::3. Information about NTP servers,our current timezone, available DNS servers and our domain should also be retrived. Here's config file:

```
iface eth0
{
  valid-lifetime 7200    // default value - 2 hours
  prefered-lifetime 3600 // default value - 1 hour
  T1 600                 // request 10 minutes interval
  T2 1200  /* trouble begins after 20 minutes
                           of server's silence */
  IA 2                   // 2 IAs with just specified values (1h/2h/10min/20min)
  IA  // third IA is more specific
  {
    valid-lifetime 3600    // valid lifetime changed to 1 hour
    prefered-lifetime 1800 // prefered lifetime changed to 30min
    address
    {
      2000::1   // request those addresses
      2000::2
      2000::3
    }
  }
  option ntp-server        // ask for NTP servers
```

```
  option time-zone          // ask for timezone
  option dns-server         // ask for DNS servers
  option domain             // ask for domain
}
```

6.5 Server configuration file

Server configuration is stored in `server.conf` file. After successful startup, old version of this file is stored as `server.conf-old`.

6.5.1 Global scope

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

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

6.5.2 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).

6.5.3 Class scope

Address class is declared as follows:

```
class
{
     class options |
     address poll
}
```

address poll has this format:

```
poll minaddress-maxaddress
```

6.5.4 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
log-mode	global	short or full	full	logging mode: short (date and name suppressed) or full
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?

6.5.5 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.

6.5.6 Server configuration file examples

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:

```
iface eth0
{
  class
  {
    pool 2000::100-2000::10f
  }
}
```

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. And it should have prefence set to 7, accept T1 from 1 to 2 minutes and so on. Config file is presented below:

```
log-mode short
log-level 7
iface eth0
{
   iface-max-lease 50
   client-max-lease 5
   preference 7
   class
   {
       pool 2000::1-2000::100
   }
}
```

Here's modified previous example. Instead of specified limits, unicast communication should be supported.

```
log-level 7
```

```
iface eth0
{
   unicast 2000::1234
   class
   {
       pool 2000::1-2000::100
   }
}
```

Ok, now let's present last config file with various requirements:

- Add new class on fifth interface named *eth1*, for example 2000::fe00/120.
- Rapit commit is allowed on eth1.
- Server preference on this interface is set to maximum (255).
- Assign 2000::20/124 class on fourth interface named eth0. Preference value for this interface should be 0.
- Ignore client with DUID "00001231200adeaaa" in the class 2000::20/124.
- Valid and prefered lifetimes are 1 hour and 30 minutes respectively. T1 and T2 set to 10 minutes and 20 minutes.
- For class 2000::100/124 valid and prefered lifetimes are 2 hours and 1 hour.
- In class 2000::100/124 one client can request up to two addresses.
- Do not assign more than 10 addresses from 2000::20/124 class.
- Client with fe80::200:39ff:fe4b:1abc link-local address should get his static address 2000::2f.
- We shall support DNS and NTP servers on interface 5.
- Log level is set to omit debug messages.

Here's config to do all this stuff:

```
log-level 7
valid-lifetime 3600
prefered-lifetime 1800
T1 600
T2 1200
iface eth0
{
  preference 0
  class
  {
    reject-clients ‘‘00001231200adeaaa’’
    2000::2f-2000::20  // it’s in reverse order, but it works.
                       // just a trick.
  }
  class
  {
```

```
    accept-only fe80::200:39ff:fe4b:1abc
    pool 2000::2f
  }
}
iface 5
{
  dns-server 2000::123:456,2000::456:1234
  ntp-server 2000::1111:2222
  rapid-commit 1
  preference 255
  class
  {
    pool 2000::fe00-2000::feff
    class-max-lease 10
  }

  class
  {
    valid-lifetime 7200
    prefered-lifetime 3600
    pool 2000::100-2000::10f
    client-max-lease 2
  }
}
```

The last server configuration example explains how to use relays. There is some remote relay with will send encapsulated 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:

```
iface relay1 {
 relay eth1
 interface-id 5020

 class {
   pool 2000::1-2000::ff
 }
 option dns-server 2000::100,2000::101
}
```

6.6 Relay configuration file

Relay configuration is stored in `/etc/dibbler/relay.conf` file.

6.6.1 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.

6.6.2 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).

6.6.3 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.

6.6.4 Relay configuration file 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.

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 mul-

ticast 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:

```
log-level 8
log-mode short

iface eth0 {
  server multicast yes
}

iface eth1 {
  client multicast yes
  client unicast 2000::123
  interface-id 1000
}
```

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):

```
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 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.

7.1 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).

7.2 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.

7.3 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: In 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 commands 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.

8 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 involvment is very limited at this time. However, that does not mean, that this project is abadoned. 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.

9 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.

10 Thanks and greetings

I would like to send my thanks and greetings to various persons. Without them, Dibbler would not be where is it 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 abadoned half a year or so ago.

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 greateful.

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