Design and Implementation of an extensible Router Configuration Management Program

Richard Hartmann
(richih@net.in.tum.de)

System Development Project
Technische Universitaet Muenchen
Lehrstuhl fuer Netzwerkarchitektur

SS 2006 (Version of December 18, 2006)

Abstract

In this postmortem, the planning, design and implementation of an extensible, modularized system to manage configurations of various devices is shown. Problems with the implementation, documentation to the system and ideas for future extensions are also included. The system is in production use at the router laboratory for the chair of Network Architectures at Technical University of Munich.
# Contents

1 Motivation 3

2 Design 3
   2.1 Requirements .................................................. 3
   2.2 Design decisions ............................................ 3

3 Background information 4
   3.1 The normal way to interact with the routers ................. 4
      3.1.1 Showing and setting up a configuration on a Cisco router ... 5
      3.1.2 Showing and setting up a configuration on a Juniper router ... 7

4 Implementation 10
   4.1 General overview of the programs .............................. 10
   4.2 Problems encountered during implementation .................. 10
   4.3 Changes of the requirements during implementation .......... 12
   4.4 Database layout ................................................ 13
   4.5 A closer look at the programs and modules .................... 14
      4.5.1 install.sh .................................................. 14
      4.5.2 lab.schema ............................................... 14
      4.5.3 bin/lab.pl ............................................... 15
      4.5.4 bin/labd.pl .............................................. 15
      4.5.5 bin/labconfd.pl ......................................... 15
      4.5.6 bin/labconf.pl .......................................... 16
      4.5.7 etc/labd.conf ........................................... 16
      4.5.8 lib/globals.pm .......................................... 16
      4.5.9 sbin/labadmin.pl ........................................ 16
      4.5.10 scripts/labd ............................................ 16
      4.5.11 lib/LabLib.pm ......................................... 16
      4.5.12 lib/LabTools.pm ....................................... 17
      4.5.13 lib/InteractTerminalserver.pm .......................... 19
      4.5.14 lib/InteractCisco.pm ................................... 20
      4.5.15 lib/InteractJuniper.pm ................................ 20
      4.5.16 lib/LabDatabase.pm .................................... 21

5 User Documentation 21
   5.1 Use Cases ...................................................... 21
      5.1.1 Reserving, saving and loading ............................ 21
      5.1.2 More details and options on reservations ................. 23
      5.1.3 Other advanced use cases ................................ 25
   5.2 Help output .................................................... 27

6 References 29
1 Motivation

Simulations can only go to a certain point when trying to test new hypotheses. Having a real life system to test against is always desirable. As networking hardware, especially routers, can be very expensive, there is a need to share resources. This need to share results in conflicting setups of the available hardware, making reconfiguration a daily chore. Setting up configurations of network devices can be a tedious task, especially when it needs to be repeated on a regular basis. Of course, one can store text files with the relevant configurations and re-enable them using copy and paste. While doing this can save a considerable amount time and work, it is still repetitive work with a clearly defined course of action, making it an ideal target for automation. The task of the author was to design and implement a system doing exactly this. It is called the labtool.

2 Design

2.1 Requirements

After initial evaluation, the following requirements were agreed upon:

1. ease of use
2. no human interaction necessary
3. automation
4. graceful handling of special circumstances (e.g. unknown password for enable mode)
5. long-term storage of configurations
6. extensibility for other/new models or devices
7. must work even if the test-net itself is left in a non-working state (e.g. setup of the routers, switches, etc.)
8. full integration with the reservation management implemented in parallel by Matthias Vallentin.

2.2 Design decisions

For storage of user and device information as well as all configurations and relations between those entities, a relational database would be used. The various advantages of databases over plain text or CSV\(^1\) should be well known and are not in the scope of this postmortem. Viable choices for the actual database implementation were PostgreSQL and MySQL, both Open Source databases available free of charge and widely supported in various languages. The database chosen was PostgreSQL, which won over MySQL for three reasons:

- At the time the project was started, MySQL 4 was still current. MySQL 4 only features MyISAM tables which supports neither referential integrity nor transactions. This puts lots of effort for verification and validation into the code level, where it does not belong.
- Stability and maintainability of PostgreSQL are better than of MySQL.
- The web interface available for PostgreSQL, phpPgAdmin, is faster, more consistent and thus more effective than MySQL’s counterpart, phpMyAdmin.

\(^1\)Comma Separated Values, a plain text with basic structuring to make parsing easier.
Even though it was planned to use Expect [Expect] in the beginning, this choice was proven wrong very early in the project. Instead, Perl was used for its various advantages.

- Availability of a vast repository of modules through CPAN, the Comprehensive Perl Archive Network. The modules provided by CPAN are generally of high quality.
- Ease of use. Although it is easy to write illegible Perl code, even basic discipline will ensure code that is almost as easy to read as normal text.
- Perl was designed to work on strings. As the configurations are nothing but strings, manipulating them is easy in Perl.
- The author already had a good working knowledge of Perl and did not need to learn a new language.

Also, it was decided to avoid a monolithic design so as not to hamper future extensions to the system. A modularized system allows both for easier debugging and adaption of new parts.

3 Background information

Why was there a need for a labtool? In an environment where many people want the exclusive use of few resources, synchronization is a must. Also, storing, naming and retrieving lots of plain text files takes up a lot of time while never being a good and easy to use solution. The integration of a fully automated system to handle configurations into the resource management was an obvious decision. The challenges this task posed will be discussed here.

3.1 The normal way to interact with the routers

Normally, one would use either ssh or telnet to connect to the routers in the lab. This would usually be done over one of the network interfaces built into the routers. Being an experimental test-lab for a university, the router-lab will often be used for new setups and by inexperienced users. It follows that the router-lab is not a network that can be expected to be fully functional at all times and is often fragmented on top of that. Due to these circumstances, there must be a means of access that is guaranteed to work no matter what the state of the router-lab is. The choice of an access method fell to the serial consoles the routers, switches and load-gens in the router-lab have. Each of these serial consoles is connected to a tty on a serial port-server. This terminal server, in turn, offers access to these ttys via ssh or telnet on the completely separate control network of the router-lab.

To access a device on the router lab, one simply connects to the terminal server and provides a password if necessary. At this point, one is presented with a command line interface which is also called a CLI\(^2\). Everything a router supports can be set, changed, adapted or viewed via commands on this CLI. Even though GUIs\(^3\) are viewed as more intuitive by some, CLI commands are easier to script, i.e. automate than having to emulate mouse movements etc. To edit any configuration items or to view possibly sensitive data, one needs to get into privileged mode. Now, it is possible to display the full configuration. To overwrite an old configuration, one has to get the router to forget the old configuration prior to loading the new one. It is important to note that the old configuration must be overridden and that it is not enough to just append to the old configuration.

---

\(^2\)Command Line Interface, the most common examples of which are DOS and Bash.

\(^3\)Graphical User Interface, the most common examples of which are the interfaces Windows and Mac OS X use as default.
Else, there might be fragments of the old configuration left over that have unforeseen effects or even make the device unusable for the desired setup completely. After loading the new configuration one still needs to mark it as the active one. This will now be shown for both a Cisco and a Juniper router along with all commands that are necessary.

As easy as following these steps might sound, a human being will evaluate a lot of extra information without even thinking about it. He or she will notice that a router is not in the desired mode and adapt his behavior accordingly. For example, if one is on configuration mode on a Juniper when connecting already, one would simply leave, enter it again and paste the configuration. Having a program decide on its own what to do next to get the router into the desired state, needs a complex set of demission points. Generally speaking, the challenge is to convert an interactive task into a batch task.

The most common of the commands needed to make sure nothing unexpected happens will be printed in *italics*. So, whenever you see something printed in *italics* in this section, you will know that normally, a human being would not need to do this every time it goes through these steps, but that a computer program needs to execute the commands to make sure it does not get into a state where it does not know what to do any more and has to time out and thus exit abnormally. A human might or might not do these steps, but it is generally able to decide if they will be necessary at this point.

### 3.1.1 Showing and setting up a configuration on a Cisco router

To get at the configuration of a Cisco, we first connect to the router via telnet. After connecting, we will first go into enabled mode, then display the configuration. Then, we will set a new configuration which will require rebooting of the router and write back the changes.

1. We connect to the to the router.
   
   *When in batch mode, we need to send CTRL-C before starting to work on the device. This is necessary as it is the only way to be absolutely certain that one would leave configuration mode if the last user left the device in this state. Apart from a timing problem, this does not affect any other state the router might be in.*

   ```
   richih@cheetah ~ % lab -c c1
   connecting to ts1 port 2001...
   use <CTRL>+[] q <RETURN> to exit telnet client...
   Trying 10.0.254.10...
   Connected to ts1.
   Escape character is ‘^[’.
   ^C
   c1>
   ```

2. Next, we go into enable mode and show the running configuration. Each time we see the ‘–More–’ prompt, we have to hit space to make the router continue printing its configuration.

   ```
   c1>enable
   c1#show running-config
   <lots of text>
   --More--
   <more text>
   --More--
   c1#
   ```

---

4See section 4.2 on page 12.
When in batch mode, we will set the terminal length to infinite. This will make pressing of
space unnecessary. Also, all output of debug, error or other messages will be suppressed so
it does not get mixed into the configuration dump. As a side note, when using Net::Telnet
to execute these steps and then save the resulting configuration into the database, we need to
remove the first line of the configuration as that is merely an information about how many
bytes the configuration uses, but without an exclamation mark prepended\(^5\). The last six lines
also need to be removed as Net::Telnet will append six blank lines to the configuration each
time it dumps the configuration\(^6\).

c1>term length 0

c1>enable

c1#undebug all

All possible debugging has been turned off

```text
configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
```

c1(config)#logging console 0

c1(config)#exit

c1#show running-config

<lots of text>

c1#

In any case, we will need to provide a password if there is an enable password set.
When in batch mode, the labtool will try a set of default passwords and also try all passwords
stored in the database for that user. If none of the passwords work, the labtool is forced to
restart the router via a SNMP command to the power modules.

3. After having had a look at the configuration, we decide to overwrite it with our
own. Unfortunately, IOS only offers cumulative updates of its configuration with-
out any way to really replace\(^7\) a configuration. The only way to start with a truly
empty configuration is to reboot the router and send a BREAK sequence to it over
the terminal console while it is booting. After sending the BREAK, one gets into
the ROMMON of IOS. The ROMMON can be used to configure and override every
aspect of the configuration including, but not limited to, resetting the whole con-
figuration of the router. When resuming the boot process, the router will ask if it
should enter a guided configuration mode which we negate as we already have a
configuration we want to use. First, we need to erase the nvram.

```
c1#erase nvram:
Erasing the nvram filesystem will remove all configuration files! Continue? [confirm]
[OK]
Erase of nvram: complete
```

c1#

4. Following this, we have to tell the router not to load the running configuration the
next time it boots.

```
c1#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
```

c1(config)#config-register 0x2142

c1(config)#exit

c1#

5. Now we need to reboot the router. As we have a configuration we want to use,
already, we do not need to walk through the setup wizard.

---

\(^5\)An exclamation mark is the comment character for Cisco’s IOS.

\(^6\)See section 4.2 on page 10.

\(^7\)Replacing would mean that any parts the old configuration which are not touched in the new configuration
would need to be deleted completely instead of leaving them intact as IOS does.
Having rebooted the router, we tell the router to load the stored configuration the next time it reboots. Having set the new configuration, we press CTRL-Z to exit configuration mode. When pasting the configuration, it is important to note that each network interface that is supposed to be active needs to be activated separately. This is necessary as IOS will shut down all interfaces by default.

Router> enable
Router# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)# config-register 0x2102
<lots of configuration pasted in here>
Router(config)# ^Z
Router(config)# write
Building configuration...
[OK]

3.1.2 Showing and setting up a configuration on a Juniper router

We will go through the same routine with a Juniper router. As you will be able to see, the interaction with a Juniper is easier and has, at least in the opinion of the author, a more consistent feel. A large plus is that one can always log in as root on a Juniper router when one has access to the serial console.

1. The first step is, of course, to log onto the router.

   richih@cheetah ~/work/labtools/lib % lab -c j1
   connecting to ts1 port 2006...
   use <CTRL>+] q <RETURN> to exit telnet client...
   Trying 10.0.254.10...
   Connected to ts1.
   Escape character is '^[']'.

   j1 (ttyd0)
If we happen to be in configuration mode already, we need to leave it. This means making sure we are at the root level of the configuration tree, exiting the config mode and, if we are asked if we want to leave without committing the changes, answering yes. We will assume we are in edit system mode.

```plaintext
[edit system]
root@j1# top
```

```plaintext
[edit]
root@j1# exit
```

The configuration has been changed but not committed
Exit with uncommitted changes? [yes,no] (yes) yes

Exiting configuration mode

```plaintext
root@j1> exit
```

2. Next, we need to go into configuration mode.
   When in batch mode, we will need to put the router into a mode where we do not need to press space every few lines.

```plaintext
root@j1% cli
root@j1> set cli screen-length 0
Screen length set to 0
```

```plaintext
root@j1> edit
Entering configuration mode
```

```plaintext
[edit]
root@j1#
```

3. Getting the configuration is done very easily. As the default config is very short, it is included in this listing.

```plaintext
root@j1# show
version 7.4R1.7;
system {
    host-name j1;
}
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 127.0.0.1/32;
            }
        }
    }
}
```
4. To load a completely new configuration on a Juniper router, we only have to issue a single command and then commit the changes. Rebooting the device is not necessary, which saves both time and effort.

```
root@j1# load override terminal
[Type `D at a new line to end input]
version 7.4R1.7;
system {
    host-name j1;
}
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 127.0.0.1/32;
            }
        }
    }
}
^D
load complete
```

```
[edit]
root@j1# commit
commit complete
```

5. To leave the router again, we exit configuration mode and quit the telnet connection. If we only showed the configuration, we might be asked if we want to exit without committing the changes. In this scenario, we will assume this is the case.

```
root@j1# exit
The configuration has been changed but not committed
Exit with uncommitted changes? [yes,no] (yes) yes
Exiting configuration mode
```

```
root@j1>
telnet> quit
Connection closed.
```

richih@cheetah ~ %
4 Implementation

Figure 1: Interdependencies of the program files.

4.1 General overview of the programs

Figure 1 shows the layout and the interdependencies of the programs. Only the more important files are shown as, for example, LabServer.pm only loads the other modules and does nothing else. Links between .pl files and .pm files and in between .pm files are not shown as the resulting mesh does not really help the understandability while looking untidy. Let it be enough to say that labd.pl, labconfd.pl and labconf.pl all load all of their various functions from the modules as needed. The user will only see and interact with lab.pl. It acts as the interface to all client functionality to the labtool. labd.pl will receive and handle all client requests for both reservation and configuration management and communicate with labconfd.pl about scheduled configuration actions and their return values via the database. Basically, both daemons can write into the queue which labconfd.pl will work off by starting one instance of labconf.pl for each task. labd.pl can then tell the user about the status of the individual tasks he has. All of the files will be described in closer detail in section 4.5 on page 14.

4.2 Problems encountered during implementation

Of course, the first choice to make with every program is the language one wants to write the program in. As every router, switch and loadgen\(^8\) is connected to a terminal server on a separate control network, the decision which method of access to use was very easy. These terminal servers support both ssh and telnet access to the serial interface of the hardware. Expect, a script language based on TCL [TCL], is specifically designed to act on STDIN and STDOUT of other programs, which seems to make it a good choice for a program depending heavily on interaction with ssh or telnet. Initial tests showed

\(^8\)By ‘loadgen’ the author refers to server machines connected to the test network designated to generate or sniff traffic, run various programs or computer simulations.
good performance and behavior of the language. Another large plus was the availability of RANCID [RANCID], a program detecting and displaying configuration changes on Cisco machines. To achieve this end, it implements full interaction with any Cisco device of any make or build date. The initial plan was to adapt this existing code to cut down development time. Unfortunately, the Expect compiler started to show erratic behavior. Extensive debugging of the program showed that Expect has major issues with stability and predictability of what code would actually do, regardless the specification and documentation of the language. When looking for a solution to this problem, the author was told by a developer of Expect that ‘this happens sometimes’ and was a known bug, albeit the causes for the bug and reliable workarounds that were in accordance with the syntax of Expect were not known. It was decided to use Perl [Perl] instead.

This change in programming language required the use of external modules from CPAN [CPAN] to interact with the routers as Perl does not have the ability to directly interact with console programs by itself. There is a CPAN module that enhances Perl accordingly, but as it is based on Expect also, it was decided not to use this, either. As the resulting solution would be not be agnostic to the communication program used to access the routers any more, this made a decision for either ssh or telnet necessary. Naturally, ssh access was the preferred option, thus this path was pursued first. During initial tests, various broken dependencies for the CPAN modules needed for ssh support in Perl required a lot of effort to make even basic prototyping of ssh access possible. Of the two ssh implementations available in CPAN, the one that was in a working state did not run without errors either so that after some discussion, it was decided to use telnet access, instead. The inherent insecurity of telnet, in authorization, secrecy as well as in integrity, would make it an impossible choice in a real-world setup. In the special situation the router lab is in however, those down-sides can be accepted. Still, the situation this creates is far from desirable. Per policy, no secret data may be stored, transmitted or processed in the router lab, so there is no risk of transmitting sensitive data in clear text, anyway. Also, people having access to the router lab are well aware of their responsibilities. Still, ssh would have been preferred if it had been possible to use it.

The third major problem during implementation was a peculiarity with CPAN’s Net::Telnet [Net::Telnet] module. In certain situations, Net::Telnet does not process the output of programs as fast as it should. When there is a lot of text returned by the router immediately after a command is sent, the regular expression matching Net::Telnet->cmd() uses for detecting a successfully executed command does not work any more. This means that the program will wait for the command to be processed until the internal timeout is hit. At this point, the program dies, printing to STDOUT that it did not receive any results. When trying to debug via the internal functions provided by Net::Telnet, one sees the same effect, namely an empty input buffer. Finally, tcpdump showed that the routers did send back the expected output, the deciding hint was given. The error disappeared when executing the program step by step in Perl’s debugger, which hinted strongly towards a timing issue. A workaround was found using a combination of Net::Telnet->print() and Net::Telnet->waitfor() which makes the code ugly and more cumbersome, but works, at least. Although the fix is very trivial in looking back, finding the cause of the effects was complicated.

Also, the need for concurrent execution of commands and interaction posed a bigger challenge than anticipated. Perl 5 only gained thread support recently, which means that various CPAN modules are not thread-safe, yet. Most notably, DBD and DBI::PgSQL, the modules required to connect and send commands to the PostgreSQL in Perl, do not support threading at this point. As the error messages produced by the modules when trying to instantiate or use objects from them are misleading and do not mention threading at all, debugging this behavior became a tedious task. When the reason for the errors Perl was throwing was discovered, it was decided to use different Perl scripts started as processes which can naturally execute concurrently on any multitasking system.
The final challenge encountered was a strange error which sometimes prevented the execution of tasks on the Cisco routers to run correctly. This behavior was never shown in test mode, but at some point during the production phase, it started to show. The error never occurred when using the step-by-step execution of Perl’s debugger, which hinted at yet another timing issue. After adding sleep(1) between each command in Perl and systematically removing these lines again, the author found out that the ‘pressing’ of CTRL-C on the IOS console was the cause of the problems. This had been added only recently to get out of configure mode, because some students participating in the router-lab course left the Cisco routers in this state. Manual testing showed that pressing CTRL-C and Enter immediately afterwards is too fast for the Cisco to react on. If the Cisco was not fast enough, the script waited for the return value of the command sent, which would never come. Again, this made the script die.

4.3 Changes of the requirements during implementation

Even though every user has access to the home directory of every other user in the router lab and the usage of telnet for interaction with the routers, there is still data that needs to be protected from normal users. This is especially true for database passwords and other credentials such as community strings (even though one could argue that SNMP can be sniffed). As every script needs to be readable in order to be executed, the normal safety measure for binary executables of not making them world-readable can not be used. The solution to the dilemma is to use a client-server model in which the server accepts commands from the client. In this scenario, all credentials can be stored in the server to which normal users do not need to have read access. This client-server system, based on RPC calls, was only introduced into the system at a point when the author’s implementation was fairly advanced.

While moving to the new system, most of the implementation was also enhanced by object-oriented design techniques. Similarly to C and C++ it is possible to mix imperative and object-oriented styles in Perl. Due to the merging of the two branches reservation management and configuration management, the scope of the project had reached a size where modularization and object-orientation were a significant advantage, anyway.

Shortly afterwards, it was decided to keep any passwords and credentials out of the subversion repository. In the initial phase, the program grew from test snippets which still had all access codes stored directly in them. Of course, this is not acceptable for security reasons. Thus, the system was reworked to use config files which were stored outside of the subversion repository. A further advantage of this approach was the unification of all passwords which helped to eliminate redundancies a lot. This approach further enabled the author to use global variables holding possible states of database fields. Checking if the return value is equal to $TASK_RUNNING instead of ‘1’ for example makes the code much more readable for both the author and any potential future developers.

Also, in the initial layout of the database, the hardware classes were simply a text field in the table lab.devices which held the model name and number of the device. When the reservation system began to grow in complexity, a feature was added enabling the user to reserve a class of device, e.g. ‘Router’, ‘Cisco’, ‘Juniper’, ‘M7i’ and others. Thus, the introduction of a new table, lab.device_classes became necessary. This self-referencing table contains a tree of various device classes and their interrelations. As the old method

\(^9\)See section 4.5 on page 14.
to check which commands and Interact\textsuperscript{10} module to use was a simple text comparison, this system needed to be adapted to a tree-walk to find out what device the program was dealing with in various places of the program.

4.4 Database layout

![Database diagram]

\textbf{Figure 2:} An overview of the database layout of the labtool.

For the purpose of this document, there is no need to lay down the complete database scheme. Only a general overview of the tables, their content and the interrelations will be provided. An overview of the entity relationship model is included to make understanding the interdependencies easier. Please see figure 2 on page 13.

The table ‘devices’ holds all info needed about any device in the router lab. The field class is a foreign key to the table ‘class’. This table is self-referencing, building a tree of devices, for example, the router c3 is a model c2691 which is a router_cisco which in turn is a router. Using this scheme, one is able to get every router with a single query and a lot more.

Configurations are indexed by what reservation they were stored in. To access a certain configuration, one has to provide the unique triple of router id, reservation id and

\textsuperscript{10}See section 4.5 on page 14.
version number. For example, c3.100.3 would refer to the third manually stored configuration c3 had during reservation 100. Version 0 is reserved for the configuration which is saved at the end of the reservation automatically. For each tuple of reservation and version, there needs to exist an entry in ‘config_checkpoints’ which stores the comments one can assign to each configuration dump.

Users of the labtool are stored in ‘users’ along with all contact data and the password. A user in itself does not have the rights to do anything requiring authentication. Necessary permissions are stored in ‘permissions’. Those permissions are mapped towards groups in the table ‘groups_permissions’. These groups, in turn are stored in ‘groups’ and associated with users in ‘users_groups’. Of course, a user can be a member of any number of groups, but should be a member of the group users at the least to be useful in any way. Rights are cumulative which means that the largest right set a user’s groups allow is used for this user. Every user also needs an entry in ‘magic_cookies’ to enable sudo-like permission caching.

4.5 A closer look at the programs and modules

4.5.1 install.sh

If called as root, install.sh will install the current version of the labtool on the system. scripts/labd will need to be adapted if the system the labtool is being installed on is not Gentoo.

If necessary, it creates $INSTALL_DIR, all required subdirectories and creates the log and pid files and then copies the following files:

- bin/lab.pl $INSTALL_DIR/bin
- bin/labd.pl to $INSTALL_DIR/bin
- bin/labconfd.pl to $INSTALL_DIR/bin
- bin/labconf.pl to $INSTALL_DIR/bin
- lib/globals.pm to $INSTALL_DIR/lib
- lib/InteractCisco.pm to $INSTALL_DIR/lib
- lib/InteractJuniper.pm to $INSTALL_DIR/lib
- lib/InteractTerminalserver.pm to $INSTALL_DIR/lib
- lib/LabDatabase.pm to $INSTALL_DIR/lib
- lib/LabLib.pm to $INSTALL_DIR/lib
- lib/LabReservations.pm to $INSTALL_DIR/lib
- lib/LabServer.pm to $INSTALL_DIR/lib
- lib/LabTools.pm to $INSTALL_DIR/lib
- lib/LabConfd.pm to $INSTALL_DIR/lib
- sbin/labadmin.pl to /sbin/labadmin.pl

It will also make sure the permissions are set correctly after copying/creating those files.

4.5.2 lab.schema

lab.schema is a SQL structure dump of the database used for the labtool.
4.5.3  bin/lab.pl

lab.pl is the program used to access all client functionality provided by the labtool. Its only function is to pass on options to labd.pl after a basic check for syntax errors. It will print any strings returned to it by the server either STDOUT or STDERR, depending on what options were used. For example, when exporting a configuration, all interaction like password checking will go to STDERR while the output requested, the configuration in question, will be printed to STDOUT to make piping into a file easier. All communication between lab.pl and labd.pl is done via RPC calls. lab.pl will display all open, running and erroneous tasks to STDERR after everything else is done each time it is called with by authenticated user. Once a user has authenticated with the same name as his effective Unix user, he will receive a magic cookie that will be stored locally in /var/run/labtools/$USER. This magic cookie is valid for 30 minutes and will be renewed each time the user runs lab.pl with authentication again. Of course, every user can only read his own cookie.

Its parameters will be listed in section 5.2 on page 27.

4.5.4  bin/labd.pl

labd.pl is one of the two daemons included in the labtool. Its purpose is to accept RPC calls from lab.pl and execute the requested actions after authentication, if necessary. All interaction between the user using lab.pl and the labtool is passed through this daemon before reaching any other part of the system. It handles authentication, authorization, sanity and safety checks of any options and will then call the according function from the libraries or put tasks onto the task list for labconfd.pl to handle. Reservation management is implemented completely in this daemon and its libraries, while the configuration management also utilizes labconfd.pl and labconf.pl. Without this lab.pl running, the labtool is practically non-functional from the user’s perspective even though configuration management could continue to work in the background. labd.pl accepts the following parameters:


4.5.5  bin/labconfd.pl

labconfd.pl is a daemon process serving three purposes.

- Checking for upcoming changes in the reservation table in the next ten seconds. If it finds an ending reservation, a task for saving the configuration on all routers in the reservation is added to the task list. In case there are starting reservations for which loading of a certain configuration were requested, those are added to the task list, as well.

- Controlling the sequential processing of items on the task list. Obviously, only one instance of labconf.pl may interact with any given device through the terminal servers at the same time, lest they should interfere with each other. Thus, no new jobs for a device may be started until the running one has finished. labconfd.pl takes care of exactly that.

- Marking any jobs that have been running for more than 10 minutes as erroneous, clearing the queue for other jobs to begin.

It loads it’s main function from lib/LabConfd.pm which serves no other purpose.
4.5.6  bin/labconf.pl

labconf.pl is called by labconfd.pl and stores or retrieves exactly one configuration from exactly one router. If labconfd.pl needs to deploy or retrieve several configurations, it will call the according number of separate instances of labconf.pl. Naturally, there may only be one labconf.pl in execution for each router at any given time. All necessary parameters are passed as command line parameters which can be useful for debugging. Database credentials and SNMP community strings are read from /usr/local/labtools/etc/labd.conf via globals.pm for security purposes.

4.5.7  etc/labd.conf

labd.conf is the configuration file holding all access control credentials like database user name and password, SNMP community strings, pid and cookie files, the port to listen on etc. It also holds global variables which are used to improve readability of the code and to define log levels and such. It is readable by group labd only to prevent normal users from gaining this information.

4.5.8  lib/globals.pm

globals.pm holds all global parameters used in the labtool. These globals are treated as constants and are never written to in the actual program. Their primary purpose is to make reading the code easier and to control log levels. globals.pm will also take care of parsing labd.conf and provide SNMP community strings, telnet server passwords, database host information and credentials to any program loading globals.pm. This means that the context of the program loading globals.pm must have sufficient rights to read labd.conf. Else, the module will die with an error.

4.5.9  sbin/labadmin.pl

labadmin.pl is used to create new users. It takes care of generating the Unix users, deploying their data throughout the lab onto all machines and system images. The program will also create the labtool user including putting it into the users group and generating the magic cookie store file. All information about the new user needs to be provided on the command line making it easier to script batch generation of new users. The expected parameters are user, real-name, email, primary_phone, secondary_phone and comment. All parameters containing white-spaces need to be quoted. To generate a new user 'bunny', one would do the following as root:

labadmin.pl bunny 'Bugs Bunny' bunny@carroteater.org +155512345 +155554321 'He likes carrots. A lot.'

4.5.10  scripts/labd

labd is the init.d script used to start and stop both labd.pl and labconfd.pl. It is written to work on Gentoo-based systems. As Gentoo has a special framework for init scripts, this script will not run on other distributions without modifications.

4.5.11  lib/LabLib.pm

LabLib.pm is the core library for labd.pl. Most of its functions are explained in the paper by Matthias Vallentin [?]. Only functions in which the author took substantial part will be listed here.
authenticate_via_cookie()
Expects: LabObject, user name, magic cookie
Returns: 1 if cookie still valid, 0 else
This function will check if the magic cookie provided by the user is the same as the one
saved in the database. If that cookie is less than 30 minutes old, it will be extended to a
new full 30 minutes.

s_info()
Expects: LabObject, verbosity level, message
Returns: void info string
This function will decide if the message should be displayed or discarded. It will re-
format the message to fit into the standard output format of the labtool along with the
appropriate prefix and colors for this verbosity level. It will also log all messages to
a log file on disk. Mailing of errors and critical errors is possible in conjunction with
mail_message()

mail_message()
Expects: LabObject, recipient, subject, message
Returns: void
This function will email any message to the recipients.

device_is_router()
Expects: LabObject, device id
Returns: 1 if device is a router, 0 else
This function will check if the device in question is a router.

usage()
Expects: LabObject, filename of client program
Returns: help output of the labtool
This function will return the help output for the labtool.

4.5.12 lib/LabTools.pm

LabTools.pm is the main module for Configuration management.
The various functions it holds are partially for outside calls from lab.pl, the client pro-
gram end user are interacting with, and partially for internal calls by other parts of the
server.

A few general notes of interest:

- The authenticated user that is frequently referred to in the following section can be
  changed by using the ‘switch user’ parameter of lab.pl

- With info string, the author refers to formatted messages including color codes and
  the appropriate information about which actions were taken, warnings and errors.

- When a function returns 0 or 1, it will always return 0 in case of an error and 1 in
  case of success.
powercycle_device()
Expects: LabObject, device name
Returns: info string
After checking the permission of the authenticated user, this function looks up the power module a device is connected to and power-cycles its power port via SNMP. Afterwards, it checks if the device reset successfully, also via SNMP.

dump_device_configuration()
Expects: LabObject, [device name[, comment]]+
Returns: info string
After checking the permission of the authenticated user, this function connects to a device, gets its configuration and puts it into the database. It creates config checkpoints for the appropriate reservation and stores a standard comment along with the time stamp if no comment was provided.

load_default_device_configuration()
Expects: LabObject, [device name]+
Returns: info string
After checking the permission of the authenticated user, this function loads the default configuration for the device from the database onto the device.

load_device_configuration()
Expects: LabObject, [device name, reservation id, version[, device destination name]]+
Returns: info string
After checking the permission of the authenticated user, this function loads the configuration specified by reservation id and version from the database onto the device. The configuration is loaded onto ‘device destination name’ if it is specified.

export_device_configuration()
Expects: LabObject, [device name, reservation id, version]+
Returns: exported configuration if successful, info string if not
After checking the permission of the authenticated user, this function loads the configuration specified by reservation id and version from the database and returns it to the calling function.

owns_config()
Expects: LabObject, device name, reservation id, version
Returns: 1 if authenticated user is owner, 0 if not
This function checks if the authenticated user owns a configuration specified by reservation id and version.

kill_telnet_sessions()
Expects: LabObject, device name
Returns: info string
After checking the permission of the authenticated user, this function ends all telnet connections to a device forcefully.

list_configs_by_user()
Expects: LabObject, limit
Returns: configurations list if successful, info string if not
This function lists the last <limit> configurations the authenticated user owns.
change_password()
Expects: LabObject, user, password
Returns: info string
After checking the permission of the authenticated user, this function changes the password of the user supplied as argument. Of course, this will require superuser rights.

get_open_tasks_by_user()
Expects: LabObject, user
Returns: info string
After checking the permission of the authenticated user, this function will return all open, started and erroneous tasks owned by the user.

4.5.13 lib/InteractTerminalserver.pm

Generally speaking, the Interact* modules make the connection between the Perl daemons on the one side and the physical interfaces on the other side. At the moment, there are three Interact* modules: InteractCisco.pm, InteractJuniper.pm and InteractTerminalserver.pm. Of these, InteractTerminalserver.pm has a special role in that it may get invoked before or after every other Interact* module. Every other Interact* module which is supposed to talk to a class of devices must offer four functions to labconf.pl if it is to be integrated in the general scheme of handling devices. Additionally, one command to output the settings in question can be defined and will be passed as parameter to the loading function. The names of the functions do not matter as long as the functionality is as expected. Still, for reasons of maintainability, it is suggested that the names of the functions follow the general theme of the ones which are already implemented. The four actions each module must offer to the outside are building up a connection to the device (preferably over the terminal servers), getting the configuration in question from the device, setting the configuration to the device and disconnecting from the device.

InteractTerminalserver.pm is the module interfacing with the terminal servers deployed throughout the lab. It is offering the following functions:

connect_telnet_terminalserver()
Expects: DB handle, device name, user name, password
Returns: Telnet object, TTY to to act on
As the name implies, this function will create a connection to the terminal server and return the Net::Telnet object.

enable_exclusive_use()
Expects: Telnet object, TTY to to act on
Returns: 1 if successful, 0 if not
This function will kill all connections to a certain device and set the number of allowed connections to one.

disable_exclusive_use()
Expects: Telnet object, TTY to to act on
Returns: 1 if successful, 0 if not
This function will set the number of allowed connections to four, the maximum permitted by the terminal servers.
**disconnect_telnet_terminalserver()**
*Expects: Telnet object, TTY to act on*
*Returns: 1 if successful, 0 if not*
This function will close the telnet session gracefully and leave the router in a usable state.

### 4.5.14 lib/InteractCisco.pm

**connect_telnet_cisco()**
*Expects: host, port, DB Handle, login, device*
*Returns: Net::Telnet object of the connection.*
This function connects to a Cisco router via a terminal server.

**get_settings_cisco()**
*Expects: Net::Telnet object of the connection, command to show the configuration.*
*Returns: The configuration of the router.*
This function gets the configuration of a Cisco router and return it to the caller.

**set_settings_cisco()**
*Expects: Net::Telnet object of the connection, configuration for the device.*
*Returns: void*
This function saves the configuration onto the device.

**disconnect_telnet_cisco()**
*Expects: Net::Telnet object of the connection.*
*Returns: void*
This function disconnects from the terminal server after putting the router into enabled mode for the convenience of the user.

**restart_cisco()**
*Expects: Net::Telnet object of the connection, DB Handle, device*
*Returns: void*
This function restarts the device via snmp_reset_cisco().

**snmp_reset_cisco()**
*Expects: DB Handle, device id*
*Returns: void*
This function restarts the device and checks if the reset was done successfully, both via SNMP.

**logit_without_db()**
*Expects: line, device, info*
*Returns: void*
This function is meant for module-internal use only and logs both to STDOUT and into a log file. It is used both for debugging and logging in production mode.

### 4.5.15 lib/InteractJuniper.pm

**connect_telnet_juniper()**
*Expects: host, port*
*Returns: Net::Telnet object of the connection*
This function will connect to a Juniper router via a terminal server.
get_settings_juniper()
Expects: Net::Telnet object of the connection, command to show the configuration
Returns: The configuration of the router.
This function will get the configuration of a Juniper router and return it to the caller.

set_settings_juniper()
Expects: Net::Telnet object of the connection, configuration for the device
Returns: void
This function will save the configuration onto the device.

exit_telnet_juniper()
Expects: Net::Telnet object of the connection.
Returns: void
This function will put the router into enabled mode and disconnect from the terminal server.

4.5.16 lib/LabDatabase.pm

LabDatabase.pm was planned as to hold all database queries in a single centralized location. Over time, it became obvious that the overhead introduced was not worth any potential benefits and the plan was abandoned. The module still holds two functions which are used by both InteractCisco.pm and InteractJuniper.pm. Any future Interact*.pm modules should use these functions to avoid code duplication.

save_settings()
Expects: DBI object of the database connection, configuration of the device, user name, group name, device id, reservation id, version id
Returns: void or DBI error
This function will save a configuration into the database.

load_settings()
Expects: DBI object of the database connection, configuration id
Returns: void or DBI error
This function will retrieve a configuration from the database.

5 User Documentation

5.1 Use Cases

The best of documentation lacks if there are no examples of how to actually use the program. This section will show the handling of basic and not-so-basic tasks with the labtool.

For completeness, this part of the documentation will not be limited to functionality implemented by the author, but also make use of the parts written by Matthias Vallentin and Stefan Kornexl.

5.1.1 Reserving, saving and loading

Let us start with an easy example and assume a new user named Paul wants to reserve a Cisco router, work on it and save his configuration into the database. After working for some time, he misconfigures the router and needs to revert to the saved configuration. When he finishes, he logs out and walks away. Paul would go through the following steps:
1. He lists all currently active reservations to see if there is a Cisco router available.

```
paul@cheetah ~ % lab -sA
```

+--------------------------------+
<table>
<thead>
<tr>
<th>routerlab reservations</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```
paul@cheetah ~ %
```

2. As none of the Cisco routers are taken, he randomly decides to use c3 and reserves it for the next hour.

```
paul@cheetah ~ % lab -u paul -a now,'in 1 hour' -D c3
[ ?? ] could not authenticate via magic cookie
Password:
[ ok ] authenticated successfully
[ ok ] meta-data for reservation [2443] added successfully
[ ok ] grabbing device [c3]
[ ok ] committing DB changes!
[ ok ] your reservation [2443] has been added successfully!
paul@cheetah ~ %
```

3. Paul now wants to open a telnet sessions to c3. As he authenticated a few seconds before, his magic cookie is still valid. Thus, he does not need to provide a password again.

```
paul@cheetah ~ % lab -u paul -c c3
[ ok ] authenticated via magic cookie
connecting to ts1 port 2003...
use <CTRL>+] q <RETURN> to exit telnet client...
Trying 10.0.254.10...
Connected to ts1.
Escape character is ']''.
c3>
```
4. After working for some time, he needs to save the configuration which is currently on c3.

```
paul@cheetah ~ % lab -u paul -d c3,'this configuration works'
[ ok ] authenticated via magic cookie
[ ok ] added dump of device c3 to reservation: 2443 version: 1 to reservation task list
paul@cheetah ~ %
```

5. After working for some time, Paul unfortunately leaves c3 in a non-working state. Having read the documentation, he knows that the labtool needs to reboot the Cisco routers to load a configuration, anyway. Thus, he just tells the labtool to load his old configuration from half an hour ago.

```
paul@cheetah ~ % lab -u paul -l c3.2443.1
[ ok ] authenticated via magic cookie
[ ok ] added loading of configuration for device c3 to task list
paul@cheetah ~ %
```

6. As Paul knows loading a configuration onto a Cisco router will take some time, he checks if the task is still running from time to time.

```
paul@cheetah ~ % lab -u paul
[ ok ] authenticated via magic cookie
[ ok ] Task 3359 to load configuration for device c3 (2443,1) running
paul@cheetah ~ %
```

7. After working again for some time he is finished with his tasks logs out and lets the labtool save his configuration when the reservation ends automatically. To end his telnet session, he presses `CTRL-;` to get into telnet’s control mode and enters `quit`.

```
c3>
telnet> quit
Connection closed.
paul@cheetah ~ %
```

Note that the labtool will create c3.2443.0 at the end of Paul’s time slot automatically.

5.1.2 More details and options on reservations

The next time Paul gets back, he already has a reservation of which he knows nothing yet as his partner created it for him. He will look up which routers are reserved for him, get to work on them and realize he needs another router to work with. Shortly before his reservation ends, he thinks that he will need an hour more to finish all assignments. Thus, he extends his reservation by half an hour. When he finishes his work a few minutes later, he ends the reservation immediately to free up the routers for the others.

1. Knowing nothing about his reservation, Paul lists all reservations to get an overview.

```
paul@cheetah ~ % lab -sa
```

```
+--------------------------------+---------------------+---------------------+----------------+
  | id               | start_time | end_time | login     |
+--------------------------------+---------------------+---------------------+----------------+
```

Note that the labtool will create c3.2443.0 at the end of Paul’s time slot automatically.
2. Paul now wants to see what devices are in the reservation. **There must be no space between -sR and 2444!**

   ```
   paul@cheetah ~ % lab -sR 2444
   ```

   +---------------------------------------------------------+
   | details for routerlab reservation no. [ 2444 ]          |
   +---------------------------------------------------------+
   | id          | start_time      | end_time      | login     |
   | +---------------------+---------------------+----------------+|
   | devices        |                |                |            |
   | c1            |                |                |            |
   | j1            |                |                |            |
   | loadgen103    |                |                |            |
   +---------------------------------------------------------+

3. Soon, Paul realizes he will need another router for his experiments. He decides to add j3 to his reservation.

   ```
   paul@cheetah ~ % lab -u paul -A 2444 -D j3
   [ ok ] authenticated via magic cookie
   [ ok ] grabbing device [j3]
   [ ok ] committing DB changes!
   [ ok ] your reservation [2444] has been added successfully!
   ```

4. Having had something to eat in between, Paul lost half an hour and decides to make up for the lost time by extending his reservation.

   ```
   paul@cheetah ~ % lab -u paul -e 2444,'in 30 minutes'
   [ ok ] authenticated via magic cookie
   [ ok ] reservation [2444] extended until: 2006-07-31 01:48:00!
   ```

5. After a few minutes, Paul is finished with his assignments and decides to free up his reserved devices.

   ```
   paul@cheetah ~ % lab -u paul -r 2444
   [ ok ] authenticated via magic cookie
   [ !!! ] WARNING: all your device configurations associated with this reservation \ will be deleted! (yes/no)
   no
   ```

Paul decides that this is not what he wants to do and looks into the documentation again.
6. He now thinks he knows what to do to end his time slot in advance.

```
paul@cheetah ~ % lab -u paul -y 2444
[ ok ] authenticated via magic cookie
[ ok ] reservation yielded, will end at 2006-07-31 01:22:54.474589+02
paul@cheetah ~ %
```

5.1.3 Other advanced use cases

As Paul feels more confident, he now tries out more of the options, but makes the occasional error, too. He will try to reserve a few devices, but encounter problems because some of them are already reserved for other people. While trying to view a configuration, he accidentally mistypes the reservation id and is prevented from seeing the configuration of another user.

1. Paul literally needs a few devices and does not much care which they are so he decides to just grab a few. Of course, he needs an overview of which classes are available.

```
paul@cheetah ~ % lab -s
  router
  router_cisco
    c2691
c2621
c7500
  lightstream 1010
  C1605
  router_juniper
    M7i
powermodule
controlsystem
loadgen
athlon
dual_opteron
switch
switch_cisco
  C2950
  C3750
  C3500XL
  C3550
paul@cheetah ~ %
```

2. He thinks he will need three routers and three load-gens at 2 am for two hours. The labtool will automatically decide which devices to use. Each device has an internal weight which ensures the least powerful devices will be used first if the user did not specify which class of device he needs.

```
paul@cheetah ~ % lab -u paul -a '02:00:00','04:00:00' -C 3:router,3:loadgen
[ ok ] authenticated via magic cookie
[ ok ] meta-data for reservation [2449] added successfully
[ ok ] grabbing device [c1]
[ ok ] grabbing device [c2]
[ ok ] grabbing device [loadgen201]
[ ok ] grabbing device [loadgen202]
[ ok ] grabbing device [loadgen203]
```
3. Realizing he would need some data from the local disk of loadgen104, he wants to add it to his reservation, too.

```
paul@cheetah ~ % lab -u paul -A 2449 -D loadgen104
[ ok ] authenticated via magic cookie
[ !! ] reservation conflict: device [loadgen104]
[ !! ] conflict with rids: 2429
[ !! ] stopping reservation! use -f to continue reservation omitting conflicting items
[ !! ] rolling back...
paul@cheetah ~ %
```

He looks up the details for reservation 2429, emails the user in question who sends back the files Paul needs.

4. Now Paul wants to append his configuration from reservation 2444 to a report he has to write and wants to export it from the database.

```
paul@cheetah ~ % lab -u paul -x c3.2445.1
[ ok ] authenticated via magic cookie
[ !! ] you do not own configuration c3.2445.1
paul@cheetah ~ %
```

5. Paul is not quite sure what he did wrong, he decides to list his 10 most current configurations.

```
paul@cheetah ~ % lab -u paul -sc10
2443.0 c3 2006-07-31 01:55:09+02 ""
2444.0 c1 2006-07-31 01:22:59+02 ""
2444.0 j3 2006-07-31 01:22:59+02 ""
2444.0 j1 2006-07-31 01:22:56+02 ""
2443.1 c3 2006-07-31 00:58:49+02 "this configuration works"
2287.0 c1 2006-07-19 16:00:03+02 ""
2287.0 j2 2006-07-19 16:00:01+02 ""
2287.0 c2 2006-07-19 16:00:01+02 ""
2287.0 j1 2006-07-19 15:59:59+02 ""
2280.0 c1 2006-07-18 12:00:02+02 ""
paul@cheetah ~ %
```

6. Spotting his error, Paul tries again with the correct reservation id. This time, he redirects the output to a file. As you can see, all status messages are printed to STDERR while all output that might be of long-term interest will be put onto STDOUT.

```
paul@cheetah ~ % lab -u paul -x c3.2443.1 > my_config
[ ok ] authenticated via magic cookie
paul@cheetah ~ % ls -l my_config
-rw-r--r-- 1 paul users 757 Jul 31 01:59 my_config
paul@cheetah ~ %
```

7. While working on the load-gens, he accidentally locks up a machine completely. As he does not have physical access to the router lab to press the reset button he needs to use the network-switchable power modules deployed throughout the lab.
8. Having a bad day, Paul also messes up the configuration of c2 completely. As he wanted to start out with a fresh configuration anyway, he decides to just load the default onto the router.

```
paul@cheetah ~ % lab -u paul -L c2
[ ok ] authenticated via magic cookie
[ ok ] added loading of default for device c2 to task list
paul@cheetah ~ 
```

9. After a few minutes, Paul’s laptop freezes. When he is finished rebooting his computer, he is unable to reconnect as the old telnet sessions have not timed out, yet. Thus, he kills the existing connections, in this case to c3.

```
paul@cheetah ~ % lab -u paul -k c3
[ ok ] authenticated via magic cookie
[ ok ] killed all connections to device c3
paul@cheetah ~ 
```

10. Suddenly, Paul realizes he is still using the preset password. Of course, he wants to change it immediately.

```
paul@cheetah ~ % lab -u paul -m paul
Password: 
[ ok ] authenticated successfully
enter new password: 
enter password again: 
[ ok ] Password modified
paul@cheetah ~ 
```

As you can see, the magic cookie did not work this time. This is, of course, to prevent user from changing the password of someone else who might have gone away from his workstation for a few minutes.

5.2 Help output

This is a verbatim copy of the online help provided by lab.pl. As you can see, the options are explained with the help of a few examples. The meaning of the parameters in angle brackets is explained last and in EBNF.

```
usage: /usr/local/labtools/bin/lab [options], where options are:

-a <metaData> - Add a reservation
-A <rid> - Append to an existing reservation (instead of -a)
-c <did> - login to device <did> via Console
-C <class> - specify a list of devices from a certain Class you like to have
-d <dump> - Dump router configuration and save it to the database
-D <details> - specify a list of Devices you like to have
-e <ext> - Extend a reservation
-f - Force the reservation even if not all devices/ports are free
-h - displays this Help output :) 
-k <did> - Kill all telnet sessions to the device
```

27
-l <load> - Load a specific configuration from database
-L <did> - Load default config onto device <did>
-m <login> - Modify user password
-p <did> - Powercycle device with <did>
-P <port> - specify a custom server Port
-r <rid> - Remove a reservation
   combined with -D <details>: remove parts of reservation
-sr - Show all reservations
-sa - Show all currently active reservations
-sA - Show all currently active reservations and the reserved objects
-sR<rid> - Show details for a reservation <rid> (no space between R and <rid>!)
-sc<limit> - Show your last <number configurations (no space between c and <limit>!)
-sC - Show all available <class_names>
-t - enables the 'Test-mode' does a dry-run (no database changes)
-u <login> - authenticate with your routerlab login (Username)
-U <login> - sudo to <login> (after authenticating with -u, order matters)
-v - turns Verbose mode on (detailed information)
-V <vlevel> - specify Verbosity level manually (0 = critical, 4 = debug)
-x <load> - eXport a router configuration to stdout
-y <rid> - make the reservation <rid> end right now (Yield)

<metaData> ::= <start_time>,<end_time>,[,<login>]
<start_time> ::= flexible perl Date::Manip time-string (see below)
<end_time> ::= flexible perl Date::Manip time-string
<login> ::= your router-lab login

<rid> ::= unique identification of your reservation:(reservation id)
<load> ::= <did>,<rid>,<version>,<did>,<rid>,<version>...
<dump> ::= <did>,[<comment>],<did>,[<comment>]...
<port> ::= custom port number where the server listens
<ext> ::= <rid>,<end_time>
<vlevel> ::= 0 | 1 | 2 | 3 | 4 (0 = critical, 4 = debug)
<class> ::= [<count>:]<class_name>[/s][,[<count>:]<class_name>[/s]]
            (s is the shared bit
            if count is omitted, we assume count == 1)
<count> ::= the number of devices you like to have from this class
<class_name> ::= name of the class (an alphabetic string) (see -sc)
<details> ::= <did>[/s xor .<pid>][.<did>[.<pid>]]
<did> ::= hostname of the requested device (device identifier)
<pid> ::= <port> | <port-range>
<port-range> ::= <port>-<port> (range, e.g. 3-7)
<port> ::= unique identification number of the requested port
.getVersion> ::= version number of a device configuration
<comment> ::= comment for your personal use
<limit> ::= an integer

examples: /usr/local/labtools/bin/lab -u foo -a now,‘in 5 days at 12:00’,
        foo -D c1,c2,loadgen5,c4.3-8,c5.1-2 -fvt
/usr/local/labtools/bin/lab -u foo -A 42 -D 2,5.25,6.2 -vt
        (appends to reservations with id 42)
/usr/local/labtools/bin/lab -u foo -r 42 -D 2,5.25,6.2 -V 4 -t
flexible time string: some examples: (from Date::Manip)
"first sunday in june 2042 at 14:00", "sunday week 22 2042",
"22nd sunday at noon", "sunday 22nd week in 2042", "next friday at noon",
"next month", "in 3 weeks at 12:00", "3 weeks later", "Friday in 2 weeks"
"last day of October", "Friday" (Friday of current week)

>>> www.net.in.tum.de/routerlab

6 References

[CPAN] Comprehensive Perl Archive Network
     http://www.cpan.org
[TCL] Tool Control Language
     http://www.tcl.tk
[RANCID] Really Awesome New Config Differ
     http://www.shrubbery.net/rancid/
[Net::Telnet] http://search.cpan.org/~jrogers/Net-Telnet-3.03/lib/Net/Telnet.pm

List of Figures

<table>
<thead>
<tr>
<th></th>
<th>Interdependencies of the program files.</th>
<th>An overview of the database layout of the labtool.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>