



BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

CONFIGURATION OF OPENWRT SYSTEM USING NETCONF PROTOCOL

KONFIGURACE OPENWRT SYSTÉMU POMOCÍ PROTOKOLU NETCONF

BACHELOR'S THESIS

BAKALÁŘSKÁ PRÁCE

AUTHOR

AUTOR PRÁCE

PETER NAGY

SUPERVISOR

VEDOUČÍ PRÁCE

Ing. LUKÁŠ KEKELY

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů

Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Nagy Peter**

Obor: Informační technologie

Téma: **Konfigurace OpenWRT systému pomocí protokolu NETCONF
Configuration of OpenWRT System Using NETCONF Protocol**

Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se s platformou OpenWRT a způsobem tvorby softwaru pro ni.
2. Nastudujte protokol NETCONF a jeho implementaci Netopeer.
3. Jako pluginy do serveru Netopeer navrhnete a implementujete konfiguraci základních systémových parametrů podle datových modelů ietf-system, ietf-interfaces a ietf-ip pro platformu OpenWRT.
4. Vytvořené pluginy otestujte na skutečném zařízení.
5. Porovnejte standardní způsoby konfigurace OpenWRT systému s pluginy vytvořenými podle datových modelů IETF a diskutujte možné rozšíření datových modelů.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese <http://www.fit.vutbr.cz/info/szz/>

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kekely Lukáš, Ing.**, UPSY FIT VUT

Datum zadání: 1. listopadu 2015

Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2



doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstract

The aim of this thesis is OpenWrt platform configuration using the NETCONF protocol. Existing tools such as libnetconf library and Netopeer toolset were used for the communication using the NETCONF protocol. Implementation part deals with the development of modules for system and network interfaces configuration.

Abstrakt

Cílem práce je konfigurace platformy OpenWrt s využitím protokolu NETCONF. Na komunikaci pomocí protokolu NETCONF byly použity stávající nástroje ve formě knihovny libnetconf a sady nástrojů Netopeer. Implementační část se zabývá vývojem modulů na konfiguraci systému a síťových rozhraní.

Keywords

OpenWrt, NETCONF, YANG, Netopeer, libnetconf, configuration

Klíčová slova

OpenWrt, NETCONF, YANG, Netopeer, libnetconf, konfigurace

Reference

NAGY, Peter. *Configuration of OpenWRT System Using NETCONF Protocol*. Brno, 2016. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Kekely Lukáš.

Configuration of OpenWRT System Using NETCONF Protocol

Declaration

Hereby I declare that this thesis is my original authorial work, which I have worked out on my own under the leadership of Ing. Lukáš Kekely. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

.....
Peter Nagy
May 17, 2016

Acknowledgements

I would like to thank my supervisor Ing. Lukáš Kekely and RNDr. Radek Krejčí from CESNET for lot of valuable advices and help.

© Peter Nagy, 2016.

This thesis was created as a school work at the Brno University of Technology, Faculty of Information Technology. The thesis is protected by copyright law and its use without author's explicit consent is illegal, except for cases defined by law.

Contents

1	Introduction	3
2	OpenWrt	4
2.1	History	4
2.2	Build system	4
2.3	Packages	5
2.4	Image installation methods	6
2.5	OpenWrt configuration	7
2.6	First login	8
3	NETCONF	10
3.1	Architecture	10
3.2	RPC Messages	10
3.3	Datastores	11
3.4	YANG	11
3.5	Extensions	12
3.6	Implementations	12
4	Design and implementation	14
4.1	Configuration files	14
4.2	Data models	16
4.3	System	16
4.4	Interfaces management	21
4.5	IP management	23
4.6	DHCP	25
4.7	Wireless	26
5	Testing	27
5.1	Hardware	27
5.2	NETCONF clients	28
5.3	Test cases	29
6	Conclusion	34
6.1	Future work	35
	Bibliography	36
	Appendices	38
	List of Appendices	39

A CD Content	40
B Virtual test environment	41
C YANG data model for password encryption	42
D YANG data model for DHCP configuration	44
E YANG data model for Wireless configuration	50

Chapter 1

Introduction

Computer networks are nowadays complex and widely used. This raises the problem of the increased number of network elements to be configured. Equipment can be from different vendors, their configuration may vary. To configure such a network efficiently can be a problem. On the other hand, configuration automation would bring the network operators' benefits, especially it would save the money.

In order to streamline network configuration process, Internet Engineering Task Force (IETF) standardized protocol called Simple Network Management Protocol (SNMP). It became early apparent that SNMP is not used as intended. In most cases, SNMP was not being used to configure network equipment, but for network monitoring. Network equipment manufacturers created their own user-friendly configuration interfaces, usually in the form of a command line interface (CLI). Most of them still support SNMP, but it is not maintained as their own command line interface. The result is that some equipment features cannot be configured through SNMP. It no longer scales as an effective way for performing many network management functions in such a complex environment [1].

New network management protocols have been designed such as Network Configuration Protocol (NETCONF) [8]. NETCONF could meet the future needs, as a scalable, efficient, and effective method for performing configuration [1]. NETCONF provides mechanisms to install, manipulate and delete the configuration of the devices. Communication takes place via simple remote procedure calls (RPCs), encoded using an Extensible Markup Language (XML). In order to help with the network configuration automation, configuration can be applied for a range of devices and can be performed automatically, while keeping flexibility and vendor independence. These features could make configuration efficient, in network scenarios where thousands of devices need to be configured.

The goal of this thesis is to make possible OpenWrt devices configuration using NETCONF protocol. OpenWrt provides fully manageable operating system, not a strict firmware, for Small office/home office (SOHO) routers [12]. In this thesis, modules for OpenWrt configuration will be created, libnetconf [6] implementation of NETCONF protocol and Netopeer [7] toolset will take care of using the NETCONF protocol as well as storing configuration data. Device configuration is divided into two modules, system and network configuration. These modules are based on standardized YANG data models for system (RFC 7317 [2]), interface (RFC 7223 [4]) and IP (RFC 7277 [5]) management. My implementation is based on previous implementation for other Linux platforms developed by Czech Education and Scientific NETwork (CESNET), which can be found in Netopeer repository [7].

Chapter 2

OpenWrt

OpenWrt is open source GNU/Linux distribution for embedded devices. It provides a fully writable file system and package management tool. The main components are Linux kernel, *musl* and BusyBox. It is dedicated mainly for the network embedded devices [23]. It provides many customisation options such as installing custom packages and completely configuring the system by users needs. The main focus of OpenWrt developers is to support a new platforms, improve stability and performance.

2.1 History

OpenWrt project started in January 2004. First stable version was for Linksys devices from WRT54 series. From 2005 OpenWrt is using GNU/Linux kernel and only add patches for the system and network interface drivers. From 2007 OpenWrt release names are inspired by cocktails. After the system starts, the banner shows the preparation formula [22].

Nowadays OpenWrt has a large user's base. Many other platforms like dd-wrt are based on OpenWrt. OpenWrt supports most wireless chipsets and architectures like mips, arm, powerpc and x86.

2.2 Build system

The equipment for which is OpenWrt designed has limited computing power. Compiling on OpenWrt would be slow and lengthy. The package has to be compiled on the host (PC), but for the embedded device. This mechanism is called cross compiling [21]. OpenWrt build system is a collection of Makefiles and patches allowing user to generate root filesystem and to cross compile packages for embedded devices. The compilation runs on Linux, BSD or OS X operating systems [27]. First step is to download build system git repository:

```
$ git clone git://git.openwrt.org/openwrt.git
```

The actual trunk¹ (May 17, 2016) version is called bleeding edge. The trunk is changing frequently and contains some experimental patches, which do not have to be stable. The second step is to download and install all available packages:

```
$ ./scripts/feeds update -a  
$ ./scripts/feeds install -a
```

¹The development branch.

To build system configuration run *make menuconfig* script, figure 2.1 shows menuconfig configuration menu. It handles configuration of target platform, included packages, filesystems, etc. Menuconfig is a simple, but powerful tool for creating OpenWrt system images. To run compilation execute *make* command. First compilation takes about 1-2 hours.

The result of compilation is an image. Images can be in various formats, depending on used filesystem. The most recent feature is the ability to compile images for virtual machines. This helps developer with testing, without need of having actual OpenWrt device. More information about virtual machines can be found in appendix B.



Figure 2.1: Menuconfig configuration menu

2.3 Packages

As mentioned in previous section, OpenWrt can be built as whole system and distributed as image, which can be installed on device. However, packages for OpenWrt can be compiled also separately. Example of compiling the Netopeer package:

```
$ make package/admin/netopeer/{clean,compile}
```

Every package in OpenWrt typically contains three types of components:

- Makefile
- package/patches
- package/files

Files and patches directories are optional. Patches directory contains bug fixes and optimisations for reducing size of the package. Files directory contains default config or init files. Makefile defines package dependencies and installation as well as removal of the package [20].

Package makefile structure

Package makefile provides steps to download, compile and install the package.

Configure section defines how the package should be compiled.

```
define Build/Configure
    $(call Build/Configure/Default,--with-linux-headers=$(LINUX_DIR))
endef
```

After cross compiling, the package should be copied to destination image.

```
define Package/helloworld/install
    $(INSTALL_DIR) $(1)/usr/sbin
    $(INSTALL_BIN) $(PKG_BUILD_DIR)/helloworld $(1)/usr/sbin/
endef
```

Package manager

OpenWrt uses `opkg` as a package management tool. The `opkg` is lightweight package management tool used to download, install and remove packages from local package repositories or ones located on the internet [26]. It can be understood as an alternative to `apt(8)` or `yum(8)` widely used in Linux distributions.

Before starting the installation of packages from official repository, it is necessary to download their list. The path to the repository is listed in `/etc/opkg.conf`. To update list of available packages, use:

```
opkg update
```

To get all available packages, use:

```
opkg list
```

A list of installed packages, including dependencies and other details can be found in `/usr/lib/opkg/status`. To install additional packages run:

```
opkg install package_name
```

2.4 Image installation methods

Before every installation, it is required to check which installation methods are supported by currently used device. OpenWrt installation is device specific. In general there are four installation methods [19]:

- **via OEM firmware** - WebUI of the OEM firmware is used for firmware upload,
- **via Bootloader and an Ethernet port** - The firmware is uploaded via TFTP or FTP,
- **via Bootloader and Serial port** - The firmware is uploaded via Serial port,
- **via JTAG** - JTAG interface is used to upload firmware.

2.5 OpenWrt configuration

Linux system configuration files are usually located in `/etc/` directory. However, OpenWrt decided to unify format of these configuration files, the reason is to make configuration easier and more centralized. The process of unifying configuration has other benefits, for example providing application programming interface (API) for the web interface. Described configuration interface is called Unified configuration interface (UCI). Every UCI based configuration file is located in `/etc/config/` directory. UCI is successor to NV-RAM based configuration found in older OpenWrt versions. Most applications have been made UCI compatible by writing original configuration file to corresponding UCI file [24]. Some of them have even made UCI configuration files.

Basic OpenWrt configuration is split into several files. The basic configuration files are:

- **system** - system configuration like hostname, timezone,
- **network** - device network interface configuration,
- **dhcp** - dns and dhcp settings,
- **wireless** - wireless interfaces settings and wifi network definition.

Configuration files are divided into sections. Each section contains a config statement line which divides file into sections. Items do not need to be quoted, quotation is required if the value contains spaces or tabs. Single or double quotes can be used. Section config defines start of the section with type “example” and name “test”. The option defines configuration with data type and value. The list keyword is used for multiple values definition, same name (collection in our example) will be used for every list item. The only difference is the value [24].

Simple configuration file example:

```
config 'example' 'test'
    option 'string' 'some value'
    option 'boolean' '1'
    list 'collection' 'first item'
    list 'collection' "second item"
```

There are two main ways how OpenWrt configuration can be modified. Configuration can be edited manually by editing files located in `/etc/config/` directory. However, after editing the file, corresponding service must be reloaded or restarted. Configuration can be also changed via various APIs like Lua, C and Shell, provided by *libuci*. LuCI² also makes changes in configuration files via Lua API. UCI command line utility uses Shell API to edit configuration.

LuCI

Project LuCI started in 2008 as a part of the OpenWrt Kamikaze release. The reason to start this project was the absence of free, extensible and easily maintainable web interface for embedded devices. LuCI uses the Lua programming language while other web interfaces make heavy use of the shell-scripting languages. LuCI splits the interfaces into

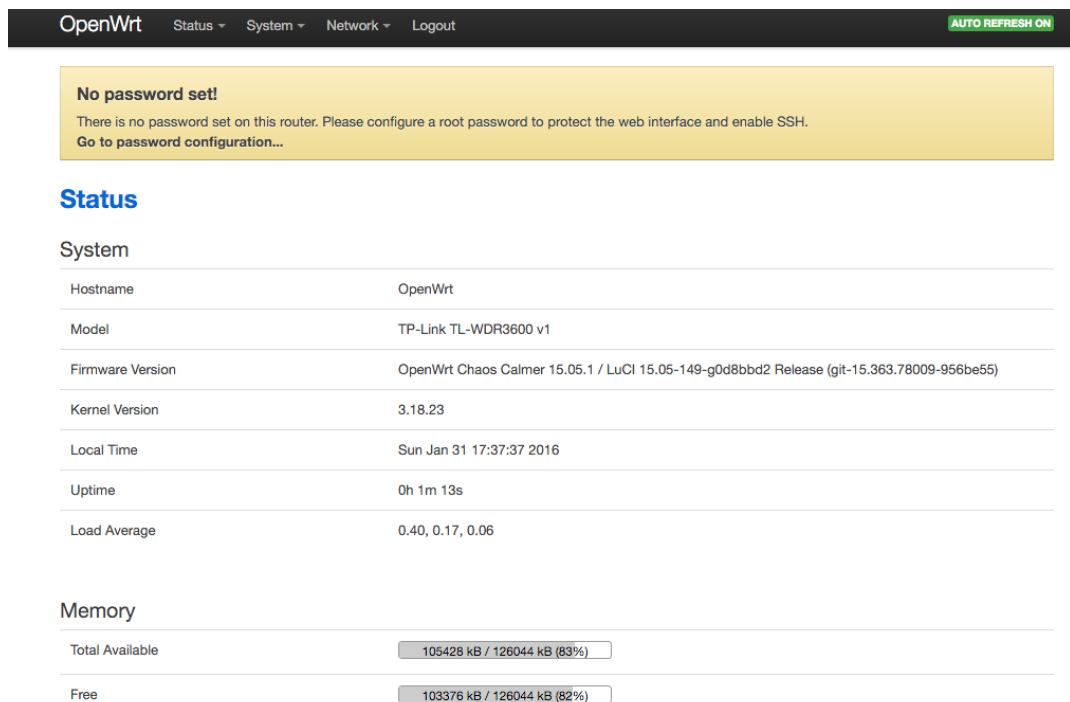


Figure 2.2: LuCI web interface

models and views, uses object-oriented templates and libraries [17]. That ensures better maintainability, higher performance and smaller size.

To be able to use LuCI, device must have UCI installed. LuCI uses UCI Lua API to communicate with embedded device. Every new release tries to make the device easier to configure and use. More web interfaces based on UCI API are available like project JUCI, shown on figure 2.3, which is based on using HTML5, angular.js and fast Lua backend.

2.6 First login

After installing OpenWrt on device, the user can log in for the first time. The first login is different, because user have to login through command line interface using telnet. Device must be connected with the computer via UTP cable. Default IP address of the device is 192.168.1.1. To activate SSH access, a root password must be set using `passwd(1)` utility [25]. Password setup will block the telnet daemon. Access via SSH and HTTPS, if LuCI is installed, will be granted without restarting the device.

²OpenWrt default WebUI

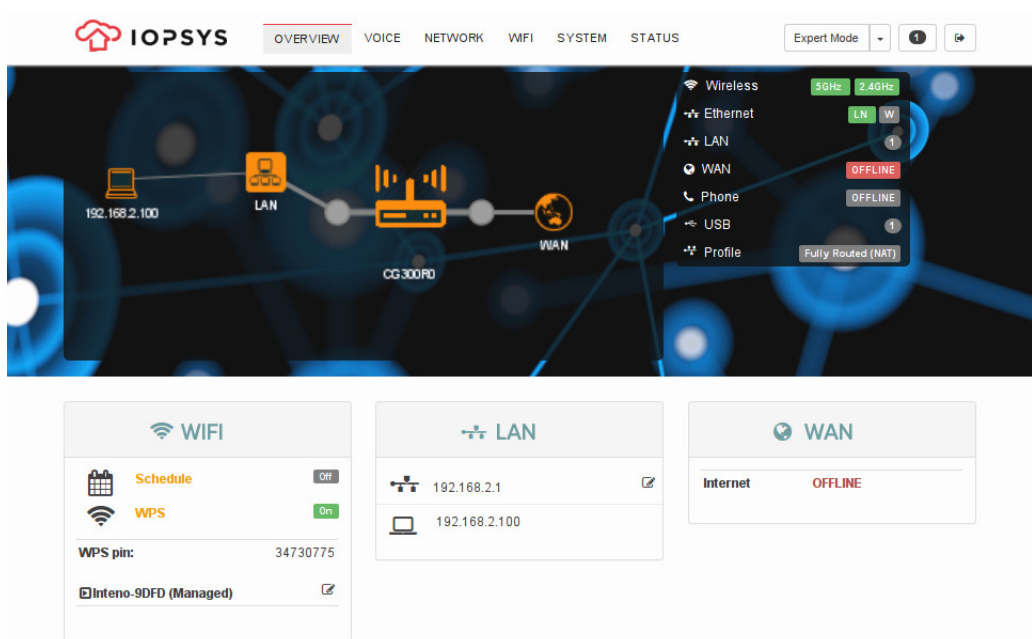


Figure 2.3: JUCI web interface

Chapter 3

NETCONF

The Network configuration Protocol (NETCONF) is a protocol standardised as RFC 4741. Later it was revised and published as RFC 6241 [8]. It is being adopted by major network equipment providers as SNMP successor.

NETCONF provides mechanisms to install, change and delete device configuration. Operations are realized on the top of the remote procedure call (RPC) layer. NETCONF recognizes difference between configuration data which can be modified and state data which is read only. The NETCONF protocol is using XML based data encoding for configuration data as well as protocol messages. Data is modeled using YANG, data modelling language created for network configuration, described in section 3.4. NETCONF is designed to replace proprietary configuration interfaces, in many ways it mimics the proprietary configuration interface. However, it provides structured error information, which proprietary interfaces usually cannot provide [14]. NETCONF has the concept of a logical datastores such as *running*, *startup* and *candidate*.

Connection between a client and a server must be secured. Most used protocols to establish secure connection are SSH and TLS. A transport layer protocol is responsible for client server authentication. NETCONF peer assumes that connection was secured by an underlying protocol.

3.1 Architecture

NETCONF is using client-server communication model. Each peer advertises its capabilities during the initial capability exchange. Based on these capabilities, peers behavior can be modified. This ensures that new functionality can be added to the protocol without any problems. At the same time, the client knows which operations are supported by server and not asking for the unsupported operations.

3.2 RPC Messages

Remote procedure call (RPC) is analogous to function call. Arguments are passed like function argument to remote procedure and caller waits for a response to be received from the remote procedure. RPC uses client server model [10]. The requesting program is client and service provider is server. NETCONF is using RPC based communication. RPC messages are encoded using XML. RPC messages and their attributes are defined in the RFC 6241 [8]. There is a list of basic operations:

- **copy-config** - Copy one configuration datastore to another,
- **delete-config** - Delete a configuration datastore,
- **edit-config** - Change the contents of a configuration datastore,
- **get-config** - Retrieve the whole or a part of a configuration datastore,
- **get** - Retrieve the configuration and state data,
- **lock** - Prevent changes to a datastore from another session,
- **unlock** - Release a lock on a datastore.

Additional RPC operations can be defined and implemented. For example, operation for reboot the device is described as RPC operation in one of the data model and will be discussed later.

Communication via RPC messages is synchronous. NETCONF defines also an asynchronous way of communication called notifications.

Example of RPC message for retrieving running device configuration:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
  </get-config>
</rpc>
```

3.3 Datastores

As it was mentioned before, data which can be retrieved from a running system is divided into two categories, configuration and state data. Configuration data is a writable data that can be set to the device. On the contrary, state data is read-only data such as number of packets received on interface. There are two types of operations for getting the configuration, *get-config* for retrieving configuration data only and *get* for configuration and state data. NETCONF divides configuration data into three categories:

- **running** - data representing actual device configuration,
- **startup** - data representing device configuration which will be set after system start,
- **candidate** - data representing device configuration which is ready to use.

3.4 YANG

The conceptual configuration data found on device should be understood by an operators. Since NETCONF needs to support special features, which are not found in other languages, it needs its own data modeling language, like SNMP needs SMIv2 [15]. YANG is a data modelling language created for a network configuration protocol. It was created by IETF and standardised as RFC 6020 [3]. YANG is used to model configuration and state data

accessed by NETCONF. YANG can be translated into an alternative XML-based syntax called YANG Independent Notation (YIN). Advantage of YANG is NETCONF-specific features support, including notifications and RPC operations. Although YANG has a limited scope of usage, being applied only to NETCONF.

YANG model specifies a document structure as well as allowed values. IETF defined a few YANG models to support NETCONF. For example *ietf-system* model for device system configuration, *ietf-interfaces* and *ietf-ip* for device network interfaces configuration.

One of the most important YANG features is a possibility to add augment without modifying the original data model. An augment allows to insert additional nodes into data model, for current or an external module. This feature is useful for vendors to add vendor-specific information to data model. The next important feature is a deviation. In the real world, devices are not able to implement the whole model as written. Deviations are not part of the published standard, it describes how implementations vary from the standard. Deviation should be used as last resort when device can not implement the model faithfully [3].

3.5 Extensions

Call Home

Call Home is a mechanism when the peer acting as NETCONF server actively opens connection to NETCONF client. After establishing the connection successfully, client takes initiative. If the connection is dropped, server tries to reconnect depending on configuration [13]. Call home was created to help the common network scenarios which would be hard to implement. It can help in case when a NETCONF client does not know the address of the server. This mechanism could be used for autoconfiguration in case of new device installation to configure itself without any user interaction. Most common scenario appears in a local networks when device is behind Network address translation (NAT) and client is not able to access server. In a local networks, OpenWrt is often used on SOHO devices and Call Home can be a useful feature.

Notifications

Notifications are defined as optional NETCONF capability defined in RFC 5277. As mentioned, it is a asynchronous way of communication. NETCONF server sends notification when a certain event has been recognised by the server. Clients have to subscribe to receive a notification from the server [8].

3.6 Implementations

In this section, some of the NETCONF implementations will be discussed.

libnetconf

The libnetconf is an open source NETCONF library developed in C, currently under development of CESNET. It provides basic functions such as connecting client and server via SSH, sending NETCONF messages and working with configuration data stored in datastore [6]. Libnetconf implements NETCONF protocol according to RFC 6241, 6242, 6243, 5277, etc.

Transaction API (TransAPI) is a libnetconf framework that helps developers to focus on

configuring and managing device without deep understanding of NETCONF protocol. It allows to choose a part of the configuration by the developer that can be easily configured as a block. It is based on “sensitive paths” generator which creates single function for every sensitive path. Whenever something changes in a datastore, corresponding callback function is called which reflects configuration file changes into a device behaviour.

TransAPI provides an opportunity to implement NETCONF RPC behaviour defined in the data model. The callback is generated for each RPC definition. Whenever a server calls RPC function with RPC message which contains RPC operation, libnetconf calls callback function implemented in the module.

TransAPI has a mechanism for watching files for changes called file callbacks. Developers can create callback functions to watch for changes in several files. This mechanism is helpful when device is configured by some other method. The configuration files can be manually edited by the device administrator and these changes will be automatically written into libnetconf configuration datastore.

Netopeer

Netopeer is an open source set of utilities and tools built for remote network configuration implemented and maintained by CESNET. It is based on libnetconf library providing SSH and TLS transport [7]. The `netopeer-cli(8)` is a NETCONF client which allows user to connect to NETCONF enabled device and manipulate with configuration data. Server capabilities are implemented in `netopeer-server(8)`. The `netopeer-server(8)` runs as service daemon integrating SSH/TLS server.

YumaPro

YumaPro¹ is a proprietary NETCONF toolset forked from open source Yuma project. After Yuma went proprietary, OpenYuma² project was created, which continues to develop and maintain the original open source Yuma project. YumaPro added many new features including notification support, performance and stability improvements. It provides a complete professional solution for a network configuration automation. YumaPro claims to be over 900 times faster than OpenYuma, this result was measured in the case of loading large amount of entries at the boot-time [16].

freenetconf

The freenetconf project is implementing a NETCONF server called `freenetconfd`³, focusing on low memory usage to be run on any platform and architecture. It is developed in C and optimized for OpenWrt [18]. Plugins can be used to extend functionality. Nowadays, the `freenetconfd` is not maintained and not providing support for new OpenWrt versions.

¹<https://www.yumaworks.com/yumapro-sdk/>

²<https://github.com/OpenClovis/OpenYuma>

³<https://github.com/freenetconf/freenetconfd>

Chapter 4

Design and implementation

The goal of this thesis is the ability to configure the basic OpenWrt device settings using NETCONF protocol. It is necessary to cover the basic configuration of all the system components. The configuration can be divided into a system and a network interfaces configuration. The basic data models have been created by NETCONF Data Modeling Language (NETMOD) group to describe this configuration. The data model for network interfaces configuration does not support dynamic host configuration protocol (DHCP) and wireless configuration. These parts are important, their configuration will be necessary to design. Data models will be added to the original model as augments. Not all received configuration data can be applied to the device, some data nodes have no corresponding items in the device configuration. Implementation part will provide more information about these problems.

The application is built on the libnetconf library in version 0.10.0. Netopeer toolkit was also used, actual version is 0.8.0. To work with the configuration files in XML, libxml2 library was used. As the standard C language library *glibc* was chosen, which is standard in GNU-Linux systems. However, during the development, the library was changed to *musl*. Its main advantage is the compactness, focus on embedded systems as OpenWrt and not least the speed. The OpenWrt system was used with additional dependencies such as `useradd(8)` to add users, `usermod(8)` to modify existing users and `ip(8)` utility to work with network interfaces. Developed transAPI modules are part of the Netopeer server implementation, figure 4.1 shows the application architecture. The application can be distributed and installed from the package “netopeer.ipk” by `opkg`, standard package management system on OpenWrt, or compiled into system. How to compile and install the package to the OpenWrt system was described in chapter 2. This project is developed for CESNET, source codes can be found in Netopeer repository in branch openwrt [7].

4.1 Configuration files

As mentioned in section 2.5, OpenWrt contains UCI configuration files. These files have predefined syntax. The syntax of these files is also defined above. The format of the configuration files was created by UCI, due to simplify and standardize the configuration of OpenWrt system. Models implemented by author keep the system in a consistent state. Consistent state means that the same configuration data are included in the running data-store, the currently configured system, and other configuration files. In case of calling callback, transAPI module in any of these modules will set the current system configura-

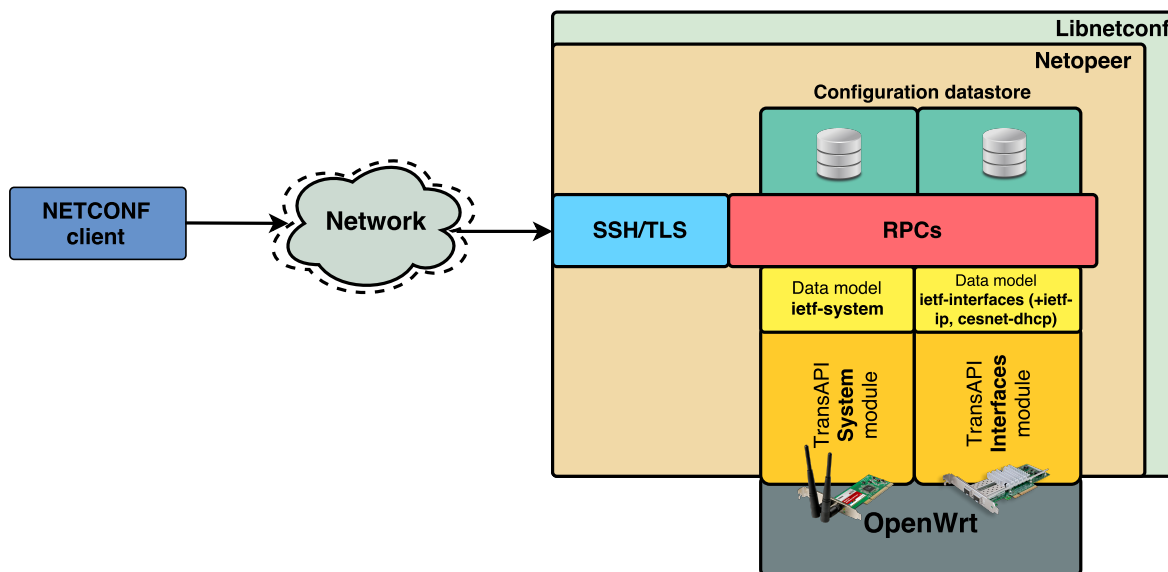


Figure 4.1: Application architecture

tion and also modify the configuration file.

The *libuci* library was originally planned to use for parsing and modifying configuration files. It implements the parser of the UCI configuration files. After unsuccessful attempts to use this library, due to the lack of API documentation, the decision to implement own simplified configuration parser was made. It contains functions for modifying, deleting, and obtaining configuration.

The change of the configuration files does not have to be caused by Netopeer, due to a datastore change. Users can manually modify configuration files or another configuration management system can be used (LuCI), that communicates via UCI configuration interface. In these cases, the problem of inconsistency of configuration files to a Netopeer transAPI data storage occurs. The solution is to monitor configuration files. Libnetconf ability to monitor external configuration files was used for this purpose. The callback can be defined to the configuration file. It will take care of the modifying datastores in case of a file change.

Not all configuration files are located in the */etc/config* directory. For DNS configuration, it is necessary to modify */etc/resolv.conf* configuration file, which is an alias to */tmp/resolv.conf*. This file contains IP addresses of the DNS servers, which client queries, and a list of default domains used to complete the fully qualified domain name. There is an example of *resolv.conf* configuration file:

```
search fit.vutbr.cz
nameserver 8.8.8.8
nameserver 8.8.4.4
```

Exact syntax of this file can be found in Linux `resolv.conf(5)` manual page.

4.2 Data models

As mentioned in section 3.4, data models are in YANG format. In this thesis, data models are presented in a tree structure, converted using `pyang`¹. The `pyang` tool can convert between various YANG language formats. For example, transfer between YIN and YANG format is possible.

Tree schema definition

For demonstration purposes YANG data models are presented as tree structure, which is a simple understandable notation. Prefix “rw” before data node names stands for configuration data. Nodes with this prefix can be configured and are presented in the configuration datastore. Prefix “ro” stands for read only state data. Symbols after data node names “?” means an optional node. Character “!” means a presence container. Character “*” denotes list or leaf-list. The brackets “[” and “]” enclose list keys.

4.3 System

As it was discussed previously, system configuration model is defined in YANG Data Model for System Management, RFC 7317 [2]. It can be divided into few sections which will be discussed later. Data model contains system identification for basic system information, clock, domain name and authentication information. OpenWrt uses `/etc/config/system` file for system configuration. Radius configuration also covered by the model was not implemented.

System identification

This part of model provides basic system information. The node `contact` defines the administrator contact information, this information is useful when a problem with device occurs. System location can be defined with `location` node. The `hostname` provides basic device identification on the network. System-state group identifies the platform and operating system.

```
+--rw system
| +--rw contact?          string
| +--rw hostname?        inet:domain-name
| +--rw location?        string
+--ro system-state
  +--ro platform
    +--ro os-name?        string
    +--ro os-release?     string
    +--ro os-version?     string
    +--ro machine?        string
```

Contact as well as location were not implemented. There are no equivalents for these items in OpenWrt. The contact and location have only informational purpose for system administrator. Although data of these nodes are still available in `ietf-system` transAPI module configuration datastore, user or developer can further use these pieces of information.

¹<https://github.com/mbj4668/pyang>

To configure hostname, it is written to `/proc/sys/kernel/hostname` file. Within every reboot, this file is overwritten by hostname configuration from system configuration file. Because of this behavior, `netopeer-server(8)` have to update as well as the system configuration file.

To get platform information, IEEE Std 1003.1-2008 standard defines to use functions from `sys/utsname.h` defined in POSIX C library. Data structure `utsname` is filled using `uname()` function, which provides needed information.

Clock

There are two ways how to define timezone, only one of them can be defined at time. Timezone-name defines name of the timezone according to IANA Timezone Database YANG Module. The second way defines timezone-utc-offset. It is a minute offset which is added to UTC time to identify system time zone.

```

+--rw system
  +--rw clock
    +--rw (timezone)?
      +--:(timezone-name)
      | +--rw timezone-name?    timezone-name
      +--:(timezone-utc-offset)
      | +--rw timezone-utc-offset?  int16

```

Received timezone must be converted into internal OpenWrt notation called “*TZ string*”. This notation is used in all OpenWrt configuration files to setup timezone. After converting the timezone, it is written to `/etc/TZ file`. Timezone in system configuration file is also changed.

Timezone-name could be directly written to the UCI configuration file. However, this method requires to have `zoneinfo` packages installed. These additional dependencies have about 2.3 MB. This implementation is focused on minimizing the number of dependencies, so the described method is not supported and timezone-name is always converted into OpenWrt internal notation.

NTP

Network time protocol (NTP) is time synchronization protocol [11]. This model can provide functionality for NTP client. Node `enabled` activates synchronization. Read only system-state clock obtains information about current datetime and boot datetime.

```

+--rw system
| +--rw ntp!
|   +--rw enabled?    boolean
|   +--rw server* [name]
|     +--rw name      string
|     +--rw (transport)
|       | +--:(udp)
|       | +--rw udp
|       |   +--rw address    inet:host
|       |   +--rw port?     inet:port-number
|       +--rw association-type?  enumeration
|       +--rw iburst?        boolean

```

```

|           +--rw prefer?           boolean
+--ro system-state
  +--ro clock
    +--ro current-datetime?         yang:date-and-time
    +--ro boot-datetime?            yang:date-and-time

```

The default NTP OpenWrt implementation is used as NTP client, which is a part of busybox called *ntpd*. The *iburst* and *prefer* options were not implemented, there is no option for setting these features in OpenWrt system configuration file. However, *ntpd* supports these option, they cannot be set through UCI configuration file. After updating the configuration file, NTP service has to load the new configuration with command:

```
./etc/init.d/sysntpd reload
```

For getting current datetime, `time(NULL)` function is used. Boot datetime is saved after boot within transAPI module initialization, `transapi_init()` function. However these timestamps have to be converted into YANG date-and-time format. For this purpose `nc_time2datetime()` function is used. Example of getting time in YANG *date-and-time* format:

```

<clock>
  <current-datetime>2016-04-07T18:55:27Z</current-datetime>
  <boot-datetime>2016-04-07T18:54:55Z</boot-datetime>
</clock>

```

DNS Resolver

Domain name system (DNS) resolver, also known as DNS client. Resolver is responsible for full domain resolution, resolving domain names to IP addresses. This subtree includes list of servers IP addresses used to query during the resolution as well as domains, which should be searched when resolving a hostname.

```

+--rw system
  +--rw dns-resolver
    +--rw search*    inet:domain-name
    +--rw server* [name]
      | +--rw name    string
      | +--rw (transport)
      |   +--:(udp-and-tcp)
      |     +--udp-and-tcp
      |       +--rw address    inet:ip-address
      |       +--rw port?      inet:port-number
    +--rw options
      +--rw timeout?    uint8
      +--rw attempts?   uint8

```

In OpenWrt, DNS and DHCP configuration are based on the same daemon called `dnsmasq(8)`, with `/etc/config/dhcp` configuration file. However, list of servers used to resolution as well as the search domains are located in `/etc/resolv.conf` as mentioned earlier. Definition of port is not supported, default port 53 is used. There are additional options for timeout and attempts, which are also implemented.

The author's implementation directly edits the *resolv.conf* file. DNS configuration defines another resolv file used by DHCP DNS autoconfiguration, usually set to */tmp/resolv.conf.auto*. When these two files are defined, `dnsmasq(8)` queries servers from each file and uses the first response.

User management

This subtree is responsible for a user management. The user passwords and SSH public keys can be used to connect to the system. Every OpenWrt device has a default root user.

```
+--rw system
  +--rw authentication
    +--rw user-authentication-order*  identityref
    +--rw user* [name]
      +--rw name                      string
      +--rw password?                 ianach:crypt-hash
      +--rw authorized-key* [name]
        +--rw name                    string
        +--rw algorithm               string
        +--rw key-data                binary
```

Node `user-authentication-order` defines a user defined sequence of authentication into the system. It is not implemented, my implementation supports only local password authentication method which also includes authentication using SSH keys.

Users are being added to system by calling `useradd(8)` tool with the parameter *name* taken from the */system/authentication/user/name* node. There is an example of adding a new user to the system:

```
# useradd -m [name] -s [default_shell] -p [encrypted_password]
```

If the password in clear text format is given, program creates a hash of the password by using `crypt()` function from the *musl* library. The received passwords in encrypted form are directly stored in a data store as well as */etc/shadow*. Passwords are stored in data store only in encrypted form. For identification of the type of encryption, every encryption has its own *id*.

To increase password security, random characters can be added and used as additional input to a hashing function, it is called salt. It makes more time-consuming to crack a password using typical brute force or dictionary attacks.

Different types of encryption are defined in data model as features. That means its implementation is optional. OpenWrt supports MD5 authentication, but clear text passwords are encrypted to DES, which is a default password encryption method. However, the *ietf-system* data model does not support DES encryption algorithm. This type of encryption was added using the *cesnet-system-authentication* model, where “des” keyword is used as encryption *id*. Definition of described data model can be found in appendix C in YANG format. Identification of different encryption types follows:

- `<id> = des` - DES
- `<id> = 1` - MD5
- `<id> = 5` - SHA-256

- `<id> = 6 - SHA-512`

The default password encryption method can be changed in the `/etc/login.defs` file after the keyword `ENCRYPT_METHOD`. SHA encryption methods like SHA-256 and SHA-512 are not supported by default. However, busybox parameter `CONFIG_BUSYBOX_DEFAULT_USE_BB_CRYPT_SHA` can be configured to `true`, before the system compilation, to support SHA encryption methods.

Possible password formats defined by data model:

- `0<clear text password>`
- `$<id>$<salt>$<password hash>`
- `$<id>$<parameter>$<salt>$<password hash>`

The password is set in the `/etc/shadow` file. At first author used a set of functions for working with the shadow file, `lckpwn()`, `getspent()`, etc. When library changed to `musl`, the functions related to working with shadow file were no longer working. The functions are not implemented, only empty function bodies are present. Because of this, author needed to change the way how to work with the shadow file. Currently the passwords are adjusted using `useradd(8)` or `usermod(8)` tools as mentioned above. The only drawback is that the encrypted password will be visible to users listing active processes.

SSH keys ensure secured connection to the system. It is based on asymmetric cryptography, pair of keys - public and private key. Every user has its own file to store keys. Keys are usually stored in `~/.ssh/authorized_keys` file. However this configuration can be changed in `/etc/ssh/sshd_config` after `AuthorizedKeysFile` keyword. The root has a special place to store keys. If default SSH server `dropbear(8)` is used, public key for root must be stored in `/etc/dropbear/authorized_keys` file.

System RPC operations

Model defines a few RPC operations to set the current time, restart and shutdown the system:

```
+---x set-current-datetime
|   +---w input
|       +---w current-datetime yang:date-and-time
+---x system-restart
+---x system-shutdown
```

The time that is taken from RPC messages need to be converted. It is stored in the data model format `yang:date-and-time`. After receiving the RPC messages, time is converted using `nc_datetime2time()` function, which takes time as his first parameter. The time zone is set to the same as described in section clock.

Further, the system can be restarted or shut down. In both operations, the delay is set to one second. Example of RPC message setting the system time:

```
<set-current-datetime xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
  <current-datetime>2015-12-19T16:39:57-08:00</current-datetime>
</set-current-datetime>
```

4.4 Interfaces management

To work with network interfaces, IETF defines YANG data model *ietf-interfaces*, RFC 7223 [4]. This model is a sort of basic model to work with the network interfaces. It is expected, that developers will add an augment to a specific type of interface. As it will be shown, *ietf-ip* data model will be added.

The model contains several configurable items such as interface name, type and interface status. However, most of the data are read only statistics data. From the model, users can get information about physical address, interface speed, etc. Subtree statistics provides statistics from the interface. This is particularly the amount of transferred data, the number of errors on the interface, etc. Statistics are recorded in both directions.

Interface

This subtree represents basic configurable data on interface.

```
+--rw interfaces
  +--rw interface* [name]
    +--rw name                string
    +--rw description?       string
    +--rw type                identityref
    +--rw enabled?           boolean
    +--rw link-up-down-trap-enable? enumeration
```

The callback for working with interface name is implemented, but interface name cannot be modified, because it is a data model key. Also the interface names in OpenWrt are defined and cannot be changed.

The description has only informational character for the system administrator. It has no equivalent in OpenWrt system, but it can be found in *ietf-interfaces* datastore.

Interface type is defined as *iana-if-type*, which is specified as IANA Interface Type YANG Module. Large number of interface types are defined, but most of them are not used. OpenWrt as well as other Unix based systems define interface type as number in kernel path `/sys/class/net/[if_name]/type`, where *if_name* refers to interface name. This number must be converted to *iana-if-type*. The kernel header file `include/linux/if_arp.h` defines meaning of interface type numbers and function `iface_get_type()` gets the type number from the kernel and converts it to *iana-if-type*. The most used interface types are:

- **softwareLoopback** - loopback interface
- **ethernetCsmacd** - Ethernet type interface
- **ieee80211** - wireless interface

The purpose of the *enabled* node is to setup an interface state. The interface can be in enabled or disabled state. Changing state is done by `ip(8)` utility. The configuration is also changed when interface is in disabled state, configuration will take effect after the interface is enabled. The following example shows enabling one of the interfaces:

```
# ip link set dev eth0 up
```

Node for generating SNMP notifications about interface state change, *link-up-down-trap-enable* was not implemented. OpenWrt has no SNMP installed by default.

State information

Most of the state information is collected by parsing files in the */sys/class/net/[if_name]* directory. The only exception is *last-change* node. It refers to the time when operational status was changed for the last time. For this purpose, `stat()` function on the */sys/class/net/[if_name]/operstate* file is used to get the time of last change.

```
+--ro interfaces-state
  +--ro interface* [name]
    +--ro name                string
    +--ro type                identityref
    +--ro admin-status        enumeration {if-mib}?
    +--ro oper-status         enumeration
    +--ro last-change?        yang:date-and-time
    +--ro if-index            int32 {if-mib}?
    +--ro phys-address?       yang:phys-address
    +--ro higher-layer-if*    interface-state-ref
    +--ro lower-layer-if*    interface-state-ref
    +--ro speed?              yang:gauge64
```

Interface statistics

Each interface stores all statistical data into */proc/net/dev* file. This approach allows to efficiently parse only one file to get all needed data. Even though *discontinuity-time* is an exception, it refers to the most recent time when one or more counters suffered a discontinuity.

```
+--ro interfaces-state
  +--ro interface* [name]
    +--ro statistics
      +--ro discontinuity-time yang:date-and-time
      +--ro in-octets?         yang:counter64
      +--ro in-unicast-pkts?   yang:counter64
      +--ro in-broadcast-pkts? yang:counter64
      +--ro in-multicast-pkts? yang:counter64
      +--ro in-discards?       yang:counter32
      +--ro in-errors?         yang:counter32
      +--ro in-unknown-protos? yang:counter32
      +--ro out-octets?        yang:counter64
      +--ro out-unicast-pkts?   yang:counter64
      +--ro out-broadcast-pkts? yang:counter64
      +--ro out-multicast-pkts? yang:counter64
      +--ro out-discards?      yang:counter32
      +--ro out-errors?        yang:counter32
```

4.5 IP management

IP configuration is based on *ietf-ip* data model. This model is described in RFC 7277 [5]. The purpose of this model is to extend the interface configuration by IP protocol. Model contains two subtrees, one for IPv4 and IPv6 protocol. This approach allows to enable or disable each protocol as needed.

Model contains maximum transmission unit (MTU), packet forwarding and IP address configuration. Network prefix in IPv4 address can be defined using prefix length or netmask. However, in IPv6 protocol, prefix length must be defined. Model also defines address resolution protocol (ARP) cache. IPv6 have a subtree for autoconfiguration parameters.

Configuration is modified in the interface kernel configuration files located in */proc/sys/net* and */sys/class/net* paths, or using *ip(8)* utility.

```
augment /if:interfaces/if:interface:
  +---rw ipv4!
  |   +---rw enabled?          boolean
  |   +---rw forwarding?      boolean
  |   +---rw mtu?             uint16
  |   +---rw address* [ip]
  |   |   +---rw ip            inet:ipv4-address-no-zone
  |   |   +---rw (subnet)
  |   |       +---:(prefix-length)
  |   |       |   +---rw prefix-length?  uint8
  |   |       +---:(netmask)
  |   |       +---rw netmask?          yang:dotted-quad
  |   +---rw neighbor* [ip]
  |       +---rw ip                inet:ipv4-address-no-zone
  |       +---rw link-layer-address yang:phys-address
  +---rw ipv6!
  |   +---rw enabled?          boolean
  |   +---rw forwarding?      boolean
  |   +---rw mtu?             uint32
  |   +---rw address* [ip]
  |   |   +---rw ip            inet:ipv6-address-no-zone
  |   |   +---rw prefix-length  uint8
  |   +---rw neighbor* [ip]
  |   |   +---rw ip            inet:ipv6-address-no-zone
  |   |   +---rw link-layer-address yang:phys-address
  |   +---rw dup-addr-detect-transmits?  uint32
  +---rw autoconf
  |   +---rw create-global-addresses?    boolean
  |   +---rw create-temporary-addresses?  boolean
  |   +---rw temporary-valid-lifetime?    uint32
  |   +---rw temporary-preferred-lifetime? uint32
```

IP Address setup

To set the IP address, *ip(8)* utility is used. When node */ipv4/enabled* is set to false, IP address is ignored, it will not be added to the interface. Whether subnet mask or prefix is set, created transAPI modules will always convert to have data in both formats. Prefix

length is required for `ip(8)`, which is used to set the IP address, and subnet mask is needed for modifying UCI configuration file `/etc/config/network`.

Editing UCI configuration files may cause problems. As mentioned above, each configuration element is part of section. There is an example of interface configuration file for demonstration purposes:

```
config interface '192_168_1_1'
    option ifname 'eth0'
    option proto 'static'
    option ipaddr '192.168.1.1'
    option netmask '255.255.255.0'
```

The section configuration name is set to `192_168_1_1`, it may seem unnecessary to use such an identifier. An IP address had to be chosen, because of its uniqueness. The IP address is defined in a model as a key which means that it cannot appear more than once. In the name of the section, it is not permitted to use the character “.”, it is replaced by “_”. The name of the section could be generated automatically, but it is not possible because additional configuration can be bind to the section. Settings from other configuration files do not bind to the actual network interface, but to the specific section. It also provides the opportunity to configure multiple IP addresses on a single interface using configuration files. The same rules are applied for IPv6 protocol.

An example of the interface configuration:

```
<?xml version="1.0" encoding="utf-8"?>
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
            xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <enabled>true</enabled>
    <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
      <enabled>true</enabled>
      <address>
        <ip>192.168.1.1</ip>
        <netmask>255.255.255.0</netmask>
      </address>
    </ipv4>
  </interface>
</interfaces>
```

An example of consequent configuration change using `ip(8)` utility:

```
# ip addr add [ip_address]/[subnet_mask] dev [interface]
```

ARP Cache

A subtree neighbor manually adds records to convert IP addresses to physical (MAC) address. These records are also known as ARP cache. To work with the ARP entries `ip(8)` tool is used. An example of adding the ARP record:

```
# ip neigh add [ip_address] lladdr [mac_address] dev [interface]
```

In the case of interface restart, previously added neighbors are lost. Although the configuration has not changed and has already been set once, the application must re-add the neighbors according to the configuration contained in the datastore.

IPv6 Autoconfiguration

Although OpenWrt supports IPv6, in UCI configuration files autoconfiguration cannot be set as IP address obtaining the method. When global addresses are enabled (option create-global-addresses) in the kernel file `/proc/sys/net/[if_name]/autoconfiguration` on a given interface (`if_name`) is set to “1”. Other options in this subtree are set similarly. Based on these settings, the interface will automatically add the IPv6 address.

4.6 DHCP

Referring to the presented models, the interface cannot be set to gain IP address from a DHCP server. The most used equipment by OpenWrt (SOHO routers) usually needs to provide DHCP server service to connected clients. Due to these arguments, author decided to design a model to augment the *ietf-ip* model. It will be used to configure the DHCP client and server.

```

module: cesnet-dhcp
augment /if:interfaces/if:interface/ip:ipv4:
  +--rw origin?          identityref
  +--rw dhcp-server
    +--rw start?         inet:ipv4-address-no-zone
    +--rw stop?          inet:ipv4-address-no-zone
    +--rw leasetime?     string
    +--rw default-gateway? inet:ipv4-address-no-zone
augment /if:interfaces-state/if:interface/ip:ipv4:
  +--ro dhcp-config
    +--ro ip-address?    inet:ipv4-address-no-zone
    +--ro prefix-length? uint8
    +--ro default-gateway? inet:ipv4-address-no-zone
    +--ro dns-server*    inet:ipv4-address-no-zone
    +--ro dns-search*    inet:host

```

The model defines an item called the origin, which may take the value *manual* or *dhcp*. If *dhcp* is selected, DHCP client will be switched on. Further the IP addresses may be manually set on the interface. On the contrary, if *manual* is selected, the IP address has to be defined manually. The IP addresses between the start and stop node are leased to clients. Options for setting the lease time as well as the default gateway are also present.

Default `dnsmasq(8)` daemon is used as DHCP server. Configuration items are set in `/etc/dnsmasq.conf` file. However, UCI configuration file `/etc/config/dhcp` can be also used.

```

augment /if:interfaces/if:interface/ip:ipv6:
  +--rw origin?          identityref
  +--rw dhcp-server
    +--rw enabled?       boolean
    +--rw router-advertisements? enumeration

```



```

    +---rw ndp?                boolean
augment /if:interfaces-state/if:interface/ip:ipv6:
  +---ro dhcp-config
    +---ro ip-address?        inet:ipv6-address-no-zone
    +---ro prefix-length?     uint8
    +---ro default-gateway?   inet:ipv6-address-no-zone
    +---ro dns-server*        inet:ipv6-address-no-zone
    +---ro dns-search*        inet:host

```

Configuration options are available for IPv4 as well as for IPv6 protocol. The IPv6 configuration is almost the same as IPv4, except the DHCP server. DHCPv6 server can work in *server* or *relay* mode. Router advertisement as well as network discovery protocol can be also configured to *server*, *relay*, or *disabled* mode. These option are set in `/etc/config/dhcp` file. Whole model definition in YANG format can be found in appendix [D](#).

4.7 Wireless

As it was mentioned, wireless SOHO devices are the most used OpenWrt equipment. Author decided to create *cesnet-wireless* model to be able to configure wireless interfaces on the device. This model provides only basic wireless configuration.

```

module: cesnet-wireless
augment /if:interfaces/if:interface:
  +---rw wireless!
    +---rw enabled?          boolean
    +---rw device?           string
    +---rw ssid?             string
    +---rw mode?             wireless-mode
    +---rw hidden?           boolean
    +---rw encryption-method
      +---rw algorithm?     identityref
      +---rw password?      string

```

Model defines the basic operation with the wireless interface. It is possible to enable/disable, set ssid and wireless mode. Supported modes depend on used hardware, the most common usage is in access point (AP) mode. The configuration can be modified in `/etc/config/wireless` file. After modifying the configuration, a network service needs to be reloaded. The Model definition can be found in appendix [E](#).

The wireless network also supports encryption. OpenWrt supports a number of different encryption modes. The commonly used encryption modes are defined in the data model. There is a list of supported encryption modes, more encryption methods can be added without changing the current data model:

- **wep** - Simple WEP encryption with one key (not recommended to use)
- **psk** - WPA Personal encryption
- **psk2** - WPA2 Personal encryption (recommended to use)

Chapter 5

Testing

Elementary testing was always done after some new features were implemented just to verify the basic functionality. Once the modules were considered as finished, more complex tests and test cases were designed. This process is called iterative development. Some of these test cases are demonstrated in the section 5.3.

The `valgrind(1)` utility was used to verify the memory usage. Because of OpenWrt issue, `valgrind(1)` cannot be used on OpenWrt platform. It was used on x86 host PC to verify some parts of the code to detect any invalid memory access and unfreed dynamically allocated memory. When an issue was found, the `gdb(1)` utility was used to debug.

The test cases were divided into two categories, basic system configuration and interfaces configuration. The basic system configuration was tested according to the *ietf-system* model. Whenever the configuration changes, the UCI configuration files should be also updated. The same task was done on the interface configuration module based on *ietf-interfaces* and *ietf-ip* model. The test cases were tried on two different environments, real hardware device and virtual environment.

The virtual environment was used for fast verifying the application functionality. It is easier and faster to start a virtual machine than to install the image on real hardware device. On the other hand, real hardware was also used for testing purposes. For example, wireless configuration cannot be tested in the virtual environment. More information about the virtual machine can be found in the appendix B.

The feedback received from the testing was valuable and several issues were discovered mainly in the interfaces configuration. Also a few memory leaks were discovered. This chapter will provide more information about used hardware, NETCONF clients used for testing and test cases.

5.1 Hardware

Tests on real hardware device was one of the test scenarios. OpenWrt supports many platforms and devices from different vendors. A few aspects must be considered when choosing a proper hardware. The OpenWrt support is one of them, which can be checked in OpenWrt hardware database¹.

The second aspect is a performance of the device. Enough RAM memory when running multiple applications can be crucial. Some devices can use additional memory as SWAP

¹<https://wiki.openwrt.org/toh/start>

from flash memory. It is recommended to have about 128 MB for running multiple applications. Devices for home use usually have from 4 to 16 MB flash memory. Nowadays 8 MB is standard, it makes about 5 MB space for user installed packages. However, it depends on OpenWrt version. Technical parameters [9] of OpenWrt hardware can be found in table 5.1.

Device	TP-Link WR841	TP-Link WDR3600	Turris Omnia
CPU	400 MHz	560 MHz	2x 1,6 GHz
RAM	32 MB	128 MB	1 GB
Flash	4 MB	8 MB	4 GB
USB	no	2x 2.0	2x 3.0
LAN	4x 100 Mbps	4x 1000 Mbps	5x 1000 Mbps
WAN	100 Mbps	1000 Mbps	1000 Mbps
Price ²	≈ 20 €	≈ 60 €	≈ 250 €

Table 5.1: Table of devices

The `netopeer-server(8)` depends on a few other libraries and utilities. A table of dependencies with the size of each dependency and `netopeer-server(8)` itself can be found in table 5.2.

Application	Size
<code>libxml2</code>	1.0 MB
<code>libnetconf</code>	0.7 MB
<code>libssh</code>	0.38 MB
<code>ip-full</code>	0.28 MB
<code>shadow-utils</code>	0.17 MB
<code>netopeer-server</code>	0.05 MB
Total	2.58 MB

Table 5.2: Dependencies

The `netopeer-server(8)` with all dependencies, including `libnetconf`, requires about 2.5 MB of space on the flash. Running application requires about 8-11 MB of RAM memory. Device with 32 MB of RAM and 8 MB of flash should be enough to run `netopeer-server(8)` with developed modules on OpenWrt. Based on these requirements, ordinary mid range device should meet all requirements. For testing purposes in this thesis, TP-Link TL-WDR3600 was chosen. It is affordable, meets all requirements, fully supported and recommended by OpenWrt community.

5.2 NETCONF clients

Based on application architecture shown in figure 4.1, NETCONF clients were used for testing purposes. One command line interface (CLI) and one graphical user interface

²Average prices in May 17, 2016

(GUI) application was chosen. As it was explained in section 3.6, Netopeer implements `netopeer-cli(8)` as CLI client and NetopeerGUI as web GUI for configuration. These tools were chosen, because they are open source and a part of the Netopeer project. Later in this section, these NETCONF clients will be presented.

netopeer-cli

This application is a NETCONF client, which is a part of Netopeer toolset. It is a powerful command line interface, which supports all operations described in section 3.2. Additional custom rpc operations can be sent using `user-rpc` command. The `netopeer-cli(8)` was developed as a part of the Netopeer project and primary used for testing purposes. However, it is a fully featured NETCONF client. Commands used to connect and edit device configuration:

```
connect --login root 192.168.1.1
```

```
edit-config [--config <file>] running|startup|candidate
```

NetopeerGUI

The NetopeerGUI³ is a web user interface for the network devices configuration, which uses `mod_netconf` apache module as back-end. NetopeerGUI can communicate with any device which supports NETCONF protocol. It is implemented in PHP as Symfony2 application using technologies such as jQuery and SQLite. Following figure 5.1 shows NetopeerGUI user interface.

NetopeerGUI is installed on server and can connect remotely to any device capable of NETCONF protocol. This method has an advantage in connecting to more than one device and managing devices from one location. However, to use NetopeerGUI as a default user interface for configuration in OpenWrt, it must be installed locally and additional dependencies such as PHP and apache must be added. Testing locally installed NetopeerGUI as default user interface for configuration was successful only in virtual environment. Memory requirements are too high, NetopeerGUI requires about 200 MB on flash and 30 MB of RAM. Only few devices, such as Turris Omnia⁴ have hardware to run such an application. However, it can be resolved using USB flash drive. Most of the devices have at least one USB port. USB flash drive can be used to extend memory on the device. Flash drive speed is a disadvantage, but this mechanism can be used to run NetopeerGUI locally.

5.3 Test cases

In order to demonstrate the application functionality, a few basic configuration test cases are described in this section. The `netopeer-cli(8)` is used as NETCONF client. Complete test kit can be found on attached removable media described in the appendix A. Each test case has a description, configuration data, `netopeer-cli(8)` commands used to send the configuration to the device, and verification part. Before each test, `netopeer-cli(8)` was used to connect to `netopeer-server(8)` running on OpenWrt device:

```
> connect --login root 192.168.1.1
```

³<https://github.com/CESNET/Netopeer-GUI>

⁴<https://omnia.turris.cz>

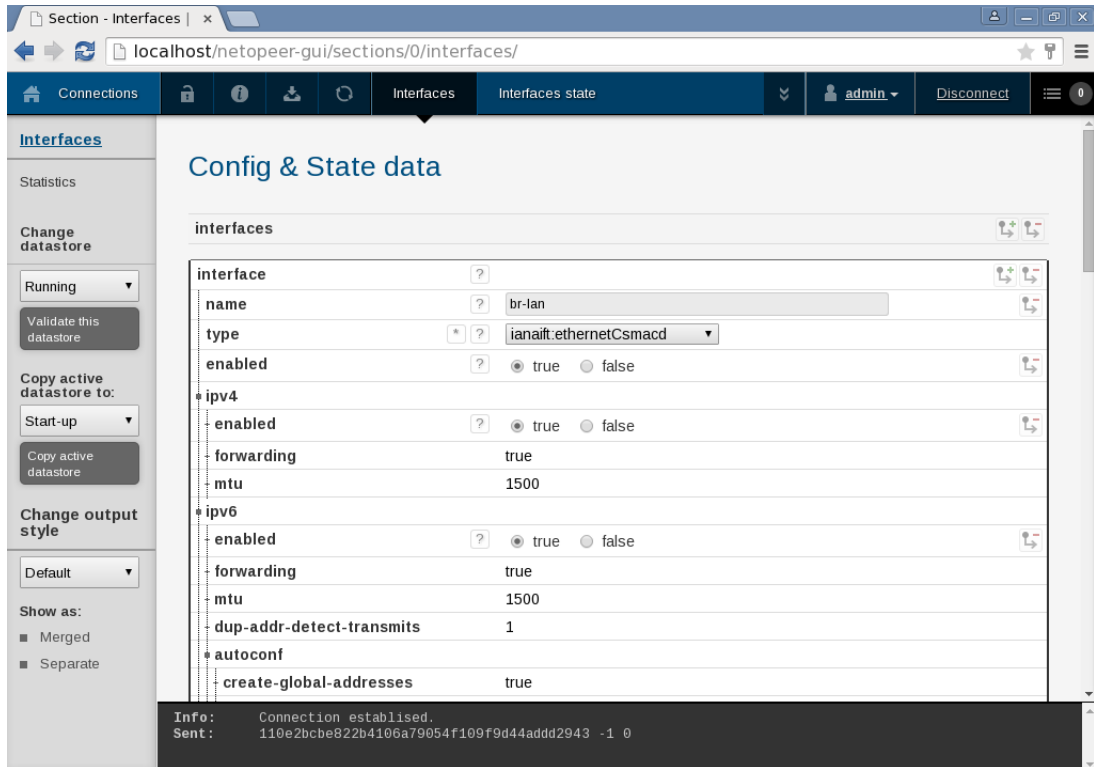


Figure 5.1: NetopeerGUI web interface

It is not possible to describe all the test cases, due to the extent of this work. Some of them are presented in this section.

Basic system configuration

This test case provides a basic system configuration. Usually, this type of configuration, such as configuration of the hostname and timezone, is done after the first system start. Following configuration data is sent to the device:

```
<?xml version="1.0" encoding="utf-8"?>
<system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
  <hostname>testingDevice</hostname>
  <clock>
    <timezone-utc-offset>-240</timezone-utc-offset>
  </clock>
</system>
```

The configuration is applied using `netopeer-cli(8)` with `edit-config` operation and shown configuration data affect running datastore. Merge is used as default operation, the configuration will be merged with target datastore. In case of error, server will stop processing `edit-config` operation and will restore the configuration to its state before this operation. Here is a `netopeer-cli(8)` command to edit device configuration:

```
> edit-config --defop merge --error rollback --config conf.xml running
```

When the configuration is applied to the device, verification is needed. Device hostname can be easily verified by connecting remotely to the device using SSH. The hostname is shown on command line after the user logged in:

1. Before: `root@OpenWrt:/#`
2. After: `root@testingDevice:/#`

For verifying the time configuration `date(1)` utility can be used. The next output demonstrates the date before and after the configuration was modified.

1. Before: `Mon May 2 20:47:52 UTC 2016`
2. After: `Mon May 2 16:51:15 AST 2016`

User configuration

OpenWrt has no password set for root user by default, so it is recommended to set a password.

```
<?xml version="1.0" encoding="utf-8"?>
<system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
  <authentication>
    <user>
      <name>root</name>
      <password>$0$password1</password>
    </user>
  </authentication>
</system>
```

The configuration is saved to startup datastore. It will be applied within the next start of `netopeer-server(8)`. Apply configuration using `netopeer-cli(8)`:

```
> edit-config --config conf.xml startup
```

To verify the configuration, the output from `/etc/shadow` is demonstrated before and after the configuration change.

1. Before: `root::0:0:99999:7:::`
2. After: `root:93.yRQP.MGcwg:16923:0:99999:7:::`

Content of the startup datastore is also verified by using `get-config` operation. The password is saved in encrypted form as it is demonstrated in following output of the `get-config` operation:

```
<authentication>
  <user>
    <name>root</name>
    <password>$des$N6n9JU.wmMGUk</password>
  </user>
</authentication>
```

Reboot the device

SOHO devices running on OpenWrt may require rebooting the device sometimes. It can be helpful in case of slow network or when any other issue occurs. There are the configuration data used for rebooting the device:

```
<system-restart xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
</system-restart>
```

To reboot the device, user defined *rpc* has to be sent instead of *edit-config*. The *user-rpc* command provides sending custom user's defined *rpc* operation, this behaviour is *netopeer-cli(8)* specific.

```
> user-rpc --file reboot.xml
```

Device should be rebooted one second after *rpc* operation is received.

IP address configuration

Some parts of the configuration need to be removed from the device. In the following configuration data, *remove* operation is used to remove the IP address from the interface.

```
<?xml version="1.0" encoding="utf-8"?>
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
  xmlns:op="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interface>
    <name>eth1</name>
    <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
      <address op:operation="remove">
        <ip>192.168.3.1</ip>
      </address>
    </ipv4>
  </interface>
</interfaces>
```

The configuration data are sent using *netopeer-cli(8)* utility to the candidate configuration datastore. From the candidate datastore the configuration is copied to the running datastore and applied to the device. There are commands providing the described process:

```
> edit-config --config conf.xml candidate
```

```
> copy-config --source candidate running
```

The configuration can be verified using *ip(8)* utility. There is an output of the *ip(8)* utility showing IP addresses assigned to the interface:

1. Before:

```
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP>
    link/ether 08:00:27:52:2d:c5 brd ff:ff:ff:ff:ff:ff
    inet 10.0.3.15/24 brd 10.0.3.255 scope global eth1
        valid_lft forever preferred_lft forever
    inet 192.168.3.1/24 scope global eth1
        valid_lft forever preferred_lft forever
```


2. After:

```
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP>
    link/ether 08:00:27:52:2d:c5 brd ff:ff:ff:ff:ff:ff
    inet 10.0.3.15/24 brd 10.0.3.255 scope global eth1
        valid_lft forever preferred_lft forever
```

Wireless configuration

Wireless configuration is defined as a part of the interface configuration. The `create` operation is used for creating a new access point on the defined interface.

```
<?xml version="1.0" encoding="utf-8"?>
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
            xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
            xmlns:op="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <enabled>>true</enabled>
    <wireless op:operation="create"
            xmlns="urn:cesnet:params:xml:ns:yang:cesnet-wireless">
      <device>radio0</device>
      <enabled>true</enabled>
      <ssid>OpenWrt</ssid>
      <mode>ap</mode>
    </wireless>
  </interface>
</interfaces>
```

Applying configuration to a running datastore using `netopeer-cli(8)`:

```
> edit-config --config conf.xml running
```

After updating the running configuration, wireless network is visible to nearby wireless clients. There is an output of `wifi` utility used for verifying wireless configuration:

```
"interfaces":
  [
    {
      "section": "@wifi-iface[0]",
      "config": {
        "mode": "ap",
        "ssid": "OpenWrt",
        "encryption": "none",
        "network": [
          "lan"
        ],
        "mode": "ap"
      }
    }
  ]
```

Chapter 6

Conclusion

This thesis provides an overview of the NETCONF protocol, its implementation in Netopeer and OpenWrt system. Studying this topic was followed by a configuration modules design, and their implementation. The main requirement was to provide a reliable basic system and network interfaces configuration of OpenWrt system. Due to the hardware limitations based on the real equipment, low memory demand was an important aspect.

The first step was to get acquainted with the Netopeer toolset and libnetconf library. It is particularly important to know the principle, how the transAPI modules operate. In the first part of this work there are some general principles, a description of OpenWrt and its configuration using UCI interface and the description of a web extension LuCI. Although I had a previous user experience with OpenWrt, I needed to learn how to develop for this platform. I also learned the principle of NTP, DNS, and SSH keys in OpenWrt system. Furthermore, to work with the network interfaces, I got to know how to work with the `ip(8)` utility and how to configure network interfaces in Linux systems.

After studying, it was time to design. I came to the need of implementing my own UCI configuration files parser. My proposal of developed transAPI modules is based on standardized data models with custom augments, which focus on DHCP and wireless network interface configuration.

The next phase was the implementation of the proposed solution. The whole work is implemented in C. I used the libnetconf library to work with the NETCONF protocol. The configuration data are transmitted using XML for creating and parsing the documents. The implementation is divided into two parts. The first part are transAPI modules, which actually has the capability of the basic system and network interfaces configuration, as well as other augmented data models such as DHCP and wireless configuration. The second part is the configuration parser, which was used in both modules.

The last part of the work is devoted to the results of the testing and the checking of functionality in individual parts of the configuration. Tests indicate that the developed transAPI modules can be used for basic OpenWrt configuration on real hardware device. However, DHCP and wireless configuration is still in development and can contain bugs. Compared to the UCI interface, it provides better remote configuration opportunities as well as it is more capable of network configuration automation. Drawback is the absence of some important configuration features that UCI provides, for example firewall and routing configuration.

6.1 Future work

The configuration modules for OpenWrt can be improved in a lot of ways. The improvements can be done in each module. I see the most crucial development for the future use in the replacement of my configuration parser for the parser from *libuci* library. Most of the tools, which work with UCI configuration files use *libuci* via its C API. The problem with lack of the documentation should be resolved with help from the OpenWrt community.

A new versions of *libnetconf*¹ library and *Netopeer*² are currently under the development. The configuration modules should be ported after stable versions are released.

The system module could be improved by adding support to upgrade the device operating system. This feature is important for security reasons, because devices should have up to date software. Operating system upgrade should be implemented as RPC operation and added to *ietf-system* module as augment. A new firmware could be provided as url, downloaded using `wget(1)` and installed to the device.

The wireless module could provide more statistic information about registred users and signal strength. Although these features require only minor changes, their implementation is not part of this thesis.

There are some new data models that could be implemented. For example the model providing routing management information is available as a draft. Next, there is a model for Quality of Service (QoS) management also available as a draft. These modules are still changing frequently, they should be standardized in the near future. Nowadays, network equipment security is a significant aspect of modern computer networks. Module for controlling firewall could be designed and implemented.

Currently, there is an initiative for using these *Netopeer* modules on OpenWrt devices for testing in a small internet service provider (ISP) company in Southern Slovakia. The devices should be placed in the clients homes as an ordinary home wireless router. These devices are often placed behind the NAT, the main objective is to provide basic device configuration without unnecessary visiting the clients home, which can cost a lot of money.

¹<https://github.com/CESNET/libnetconf2>

²<https://github.com/CESNET/Netopeer2>

Bibliography

- [1] B. Hedstrom, A. W.; Sakthidharan, S.: Protocol Efficiencies of NETCONF versus SNMP for Configuration Management Functions [online]. May 2011.
Available at: <http://morse.colorado.edu/~tlen5710/11s/11NETCONFvsSNMP.pdf>
- [2] Bierman, A.; Bjorklund, M.: A YANG Data Model for System Management. RFC 7317, RFC Editor, August 2014.
Available at: <https://tools.ietf.org/html/rfc7317>
- [3] Bjorklund, M.: YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, RFC Editor, October 2010.
Available at: <https://tools.ietf.org/html/rfc6020>
- [4] Bjorklund, M.: A YANG Data Model for Interface Management. RFC 7223, RFC Editor, May 2014.
Available at: <https://tools.ietf.org/html/rfc7223>
- [5] Bjorklund, M.: A YANG Data Model for IP Management. RFC 7277, RFC Editor, June 2014.
Available at: <https://tools.ietf.org/html/rfc7277>
- [6] Cesnet z.s.p.o.: libnetconf. 2015 [cit. 2016-05-02].
Available at: <https://github.com/CESNET/libnetconf>
- [7] Cesnet z.s.p.o.: Netopeer. 2015 [cit. 2016-05-02].
Available at: <https://github.com/CESNET/netopeer>
- [8] Enns, R.; Bjorklund, M.; Schoenwaelder, J.; aj.: Network Configuration Protocol (NETCONF). RFC 6241, RFC Editor, June 2011.
Available at: <https://tools.ietf.org/html/rfc6241>
- [9] Krejčí, R.; Hájek, J.: Overview of the Local Network Monitoring Projects and Tools [online]. 2015.
Available at:
<https://www.cesnet.cz/wp-content/uploads/2015/03/sohomonitoring.pdf>
- [10] Marshall, D.: Remote Procedure Calls [online]. 1999 [cit. 2016-04-15].
Available at: <https://www.cs.cf.ac.uk/Dave/C/node33.html>
- [11] Mills, D. L.: Network Time Protocol (NTP). RFC 958, RFC Editor, September 1985.
Available at: <https://tools.ietf.org/html/rfc958>
- [12] Tavares, D. M.; aj.: Access Point Reconfiguration Using OpenWrt [online]. 2014.
Available at: <http://worldcomp-proceedings.com/proc/p2014/ICW7101.pdf>

- [13] Vaško, M.: *Integrace SSH/TLS do Netopeer Netconf serveru [online]*. Master's thesis, Masaryk university, Faculty of Informatics, Brno, 2015 [cit. 2016-05-04]. Available at: <http://theses.cz/id/gx7cgj>
- [14] Wallin, S.; Wikström, C.: Automating Network and Service Configuration Using NETCONF and YANG. In *Proceedings of the 25th International Conference on Large Installation System Administration, LISA'11, Berkeley, CA, USA: USENIX Association, 2011*, p. 22–22. Available at: <http://dl.acm.org/citation.cfm?id=2208488.2208510>
- [15] Xu, H.; Xiao, D.: *Challenges for Next Generation Network Operations and Service Management: 11th Asia-Pacific Network Operations and Management Symposium, APNOMS 2008, Beijing, China, October 22-24, 2008. Proceedings*, chapter Considerations on NETCONF-Based Data Modeling. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ISBN 978-3-540-88623-5, p. 167–176.
- [16] YumaWorks: Transaction Performance [online]. 2016 [cit. 2016-04-06]. Available at: <https://www.yumaworks.com/netconfd-pro/transaction-performance>
- [17] LuCI – Technical Reference [online]. 2015 [cit. 2016-04-06]. Available at: <https://wiki.openwrt.org/doc/techref/luci>
- [18] Freenetconf [online]. 2015 [cit. 2016-04-18]. Available at: <http://www.freenetconf.org/>
- [19] Installing OpenWrt [online]. 2016 [cit. 2016-04-06]. Available at: <https://wiki.openwrt.org/doc/howto/generic.flashing>
- [20] Creating packages [online]. 2016 [cit. 2016-04-15]. Available at: <https://wiki.openwrt.org/doc/devel/packages>
- [21] Cross Compile [online]. February 2016 [cit. 2016-04-15]. Available at: <https://wiki.openwrt.org/doc/devel/crosscompile>
- [22] OpenWrt Version History [online]. 2016 [cit. 2016-04-15]. Available at: <https://wiki.openwrt.org/about/history>
- [23] OpenWrt Wireless freedom [online]. 2016 [cit. 2016-04-15]. Available at: <https://openwrt.org>
- [24] The UCI System [online]. 2016 [cit. 2016-04-15]. Available at: <https://wiki.openwrt.org/doc/uci>
- [25] OpenWrt – First Login [online]. 2016 [cit. 2016-04-18]. Available at: <https://wiki.openwrt.org/doc/howto/firstlogin>
- [26] OPKG Package Manager [online]. 2016 [cit. 2016-04-18]. Available at: <https://wiki.openwrt.org/doc/techref/opkg>
- [27] OpenWrt's build system – About [online]. 2016 [cit. 2016-04-23]. Available at: <https://wiki.openwrt.org/about/toolchain>

Appendices

List of Appendices

A CD Content	40
B Virtual test environment	41
C YANG data model for password encryption	42
D YANG data model for DHCP configuration	44
E YANG data model for Wireless configuration	50

Appendix A

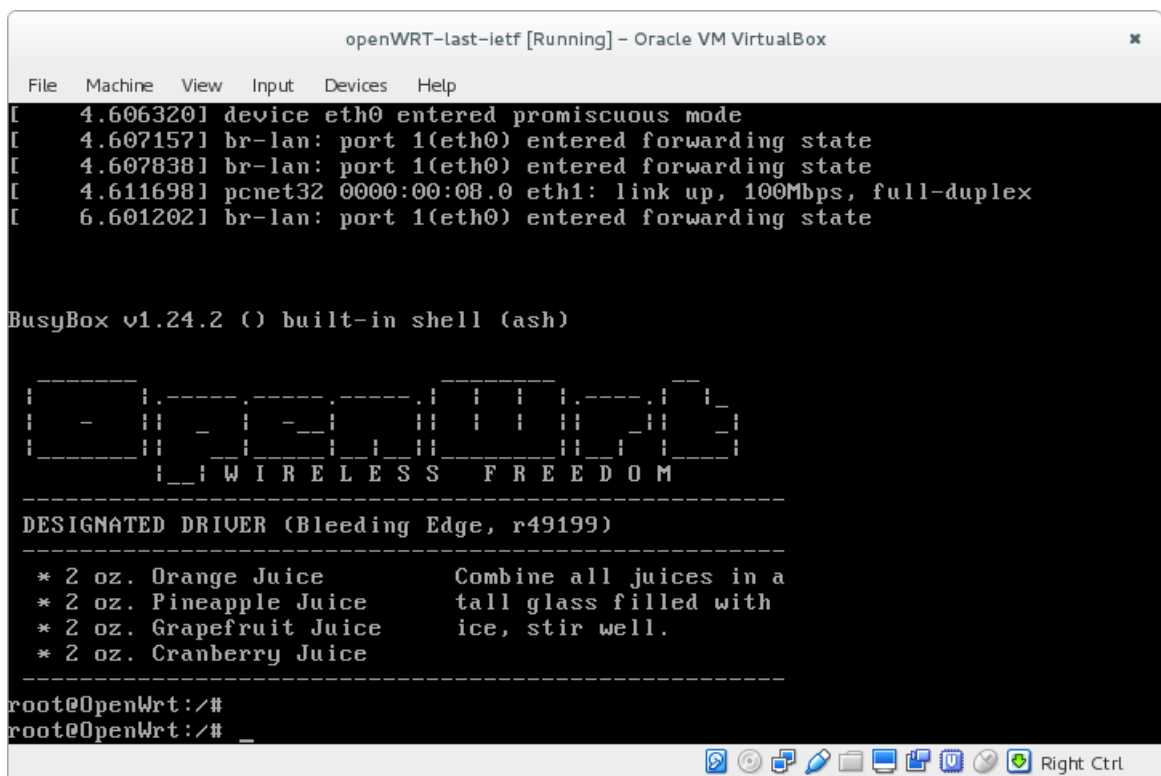
CD Content

- **src/** - Application source codes,
- **test-cases/** - Test cases used to test the application,
- **virtual/** - OpenWrt image for VirtualBox with installed configuration modules,
- **thesis.pdf** - Thesis in pdf format,
- **src-latex/** - Source files of the thesis in latex.

Appendix B

Virtual test environment

OpenWrt can be compiled for x86 platform. The build root has an option to create VirtualBox or VMware image, it can be helpful for rapid application development. In this thesis, VirtualBox is used for testing the application on x86 virtual environment. Images created by build root are using GRUB as bootloader. Figure B.1 shows running OpenWrt in VirtualBox.



```
openWRT-last-ietf [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
[ 4.606320] device eth0 entered promiscuous mode
[ 4.607157] br-lan: port 1(eth0) entered forwarding state
[ 4.607838] br-lan: port 1(eth0) entered forwarding state
[ 4.611698] pcnet32 0000:00:08.0 eth1: link up, 100Mbps, full-duplex
[ 6.601202] br-lan: port 1(eth0) entered forwarding state

BusyBox v1.24.2 () built-in shell (ash)

-----
|_ | W I R E L E S S   F R E E D O M
-----
DESIGNATED DRIVER (Bleeding Edge, r49199)
-----
* 2 oz. Orange Juice      Combine all juices in a
* 2 oz. Pineapple Juice   tall glass filled with
* 2 oz. Grapefruit Juice  ice, stir well.
* 2 oz. Cranberry Juice
-----

root@OpenWrt:~#
root@OpenWrt:~#
```

Figure B.1: OpenWrt in VirtualBox

VirtualBox image is included on attached media described in appendix A.

Appendix C

YANG data model for password encryption

```
module cesnet-system-authentication {

    namespace "urn:cesnet:yang:system-authentication";
    prefix sys-auth;

    import iana-crypt-hash {
        prefix ianach;
    }

    import ietf-system {
        prefix sys;
    }

    contact
        "Peter Nagy <xnagyp01@stud.fit.vutbr.cz>";

    description
        "Add support for des encryption used in OpenWrt

    Copyright (C) 2016 CESNET, z.s.p.o."
    ;

    revision 2016-06-11 {
        description
            "Initial revision.";
    }

    deviation "/sys:system/sys:authentication/sys:user/sys:password" {
        deviate replace {
            type union {
                type ianach:crypt-hash;
                type string {
```

```
    pattern
    '$des$[a-zA-Z0-9./]*';
  }
}
}
}
```

Appendix D

YANG data model for DHCP configuration

```
module cesnet-dhcp {

    namespace "urn:cesnet:yang:dhcp";
    prefix dhcp;

    import ietf-interfaces {
        prefix if;
    }
    import ietf-ip {
        prefix ip;
    }
    import ietf-inet-types {
        prefix inet;
    }

    contact
    "Peter Nagy <xnagyp01@stud.fit.vutbr.cz>";

    description
    "This module contains a data model for
    the configuration of dhcp configuration.

    Copyright (C) 2016 CESNET, z.s.p.o.

    TODO: License
    ";

    revision 2016-02-02 {
        description
        "Added feature to configure dhcp server";
    }
}
```

```

revision 2013-07-02 {
    description
    "Initial revision.";
}

/*
 * Identities
 */

identity origin {
    description
    "Base identity for address origin";
}

identity manual {
    base origin;
    description
    "Manual IP address origin";
}

identity dhcp {
    base origin;
    description
    "DHCP IP address origin";
}

identity linklayer {
    base origin;
    description
    "Linklayer IP address address origin";
}

/* Data nodes */

grouping dhcp-status-ipv4 {
    description
    "DHCP state info for interface with enabled DHCP,
    IPv4 protocol.";

    leaf ip-address {
        config false;
        type inet:ipv4-address-no-zone;
        description
        "The IP address on the interface.";
    }
    leaf prefix-length {

```

```

    config false;
    type uint8 {
        range "0..128";
    }
    description
    "The length of the subnet prefix.";
}
leaf default-gateway {
    config false;
    type inet:ipv4-address-no-zone;
    description
    "The default gateway of an interface.";
}
leaf-list dns-server {
    config false;
    type inet:ipv4-address-no-zone;
    description
    "The DNS server addresses obtained by DHCP.";
}
leaf-list dns-search {
    config false;
    type inet:host;
    ordered-by user;
    description
    "List of domains obtained by DHCP to search
    when resolving a hostname .";
}
}

grouping dhcp-status-ipv6 {
    description
    "DHCP state info for interface with enabled DHCP,
    IPv6 protocol.";

    leaf ip-address {
        config false;
        type inet:ipv6-address-no-zone;
        description
        "The IP address on the interface.";
    }
    leaf prefix-length {
        config false;
        type uint8 {
            range "0..128";
        }
        description
        "The length of the subnet prefix.";
    }
}

```

```

leaf default-gateway {
    config false;
    type inet:ipv6-address-no-zone;
    description
    "The default gateway of an interface.";
}
leaf-list dns-server {
    config false;
    type inet:ipv6-address-no-zone;
    description
    "The DNS server addresses obtained by DHCP.";
}
leaf-list dns-search {
    config false;
    type inet:host;
    ordered-by user;
    description
    "List of domains obtained by DHCP to search
    when resolving a hostname .";
}
}

grouping dhcp-server-ipv4 {
    description
    "DHCP server on interface configuration.";

    leaf start {
        type inet:ipv4-address-no-zone;
        description
        "Specifies the start network address, the minimum
        address that may be leased to clients";
    }
    leaf stop {
        type inet:ipv4-address-no-zone;
        description
        "Specifies the last network address, the maximum address
        that may be leased to clients";
    }
    leaf leasetime {
        type string;
        description
        "Specifies the lease time of addresses handed out to clients,
        for example 12h or 30m";
    }
    leaf default-gateway {
        type inet:ipv4-address-no-zone;
        description
        "Specifies alternative default gateway";
    }
}

```

```

    }
}

grouping dhcp-server-ipv6 {
    description
    "DHCP server on interface configuration.";

    leaf enabled {
        type boolean;
        description
        "Specifies if dhcpv6 server is enabled";
    }
    leaf router-advertisements {
        type enumeration {
            enum server {
                description
                "Specifies router advertisement to server mode";
            }
            enum relay {
                description
                "Specifies router advertisement to relay mode";
            }
            enum disabled {
                description
                "Specifies the router advertisement to be disabled";
            }
        }
        description
        "Specifies the router advertisement type";
    }
    leaf ndp {
        type boolean;
        description
        "Specifies Neighbor Discovery Protocol";
    }
}

augment "/if:interfaces/if:interface/ip:ipv4" {
    leaf origin {
        type identityref {
            base origin;
        }
    }
    container dhcp-server {
        when "../origin = 'manual'";
        uses dhcp-server-ipv4;
    }
}

```



```

augment "/if:interfaces-state/if:interface/ip:ipv4" {
  container dhcp-config {
    config false;
    when "/if:interfaces/if:interface/ip:ipv4/origin = 'dhcp'";
    uses dhcp-status-ipv4;
  }
}

augment "/if:interfaces/if:interface/ip:ipv6" {
  leaf origin {
    type identityref {
      base origin;
    }
  }
  container dhcp-server {
    when "../origin = 'manual'";
    uses dhcp-server-ipv6;
  }
}

augment "/if:interfaces-state/if:interface/ip:ipv6" {
  container dhcp-config {
    config false;
    when "/if:interfaces/if:interface/ip:ipv6/origin = 'dhcp'";
    uses dhcp-status-ipv6;
  }
}
}

```

Appendix E

YANG data model for Wireless configuration

```
module cesnet-wireless {

    namespace "urn:cesnet:yang:wireless";
    prefix wifi;

    import ietf-interfaces {
        prefix if;
    }

    contact
    "Peter Nagy <xnagyp01@stud.fit.vutbr.cz>";

    description
    "This module contains a data model for
    the configuration of wireless network.

    Copyright (C) 2016 CESNET, z.s.p.o.
    ";

    revision 2016-02-02 {
        description
        "Initial revision.";
    }

    typedef wireless-mode {
        type enumeration {
            enum ap {
                description "Access point mode";
            }
            enum sta {
                description "STA for managed - client mode";
            }
        }
    }
}
```

```

    enum adhoc {
        description "Wireless Ad-Hoc mode";
    }
    enum wds {
        description "Static WDS mode";
    }
    enum monitor {
        description "Monitoring mode";
    }
    enum mesh {
        description "Mesh mode";
    }
}
description "Wireless interface mode";
}

identity wireless-encryption-algorithm {
    description
    "Base identity for wireless encryption algorithm";
}

identity wep {
    base wireless-encryption-algorithm;
    description
    "WEP encryption";
}

identity psk {
    base wireless-encryption-algorithm;
    description
    "WPA-PSK encryption";
}

identity psk2 {
    base wireless-encryption-algorithm;
    description
    "WPA2-PSK encryption";
}

/* Configuration data nodes */
augment "/if:interfaces/if:interface" {
    list wireless {
        key "device";
        description
        "Wireless parameters";

        leaf enabled {
            type boolean;

```

```
        default true;
        description
        "Controls whether wireless is enabled of disabled
        on this interface.";
    }
    leaf device {
        type string;
        description
        "Controls wireless hardware device";
    }
    leaf ssid {
        type string;
        default "OpenWrt";
        description
        "Wireless SSID set on this interface";
    }
    leaf mode {
        type wireless-mode;
        description
        "Interface wireless mode";
    }
    leaf hidden {
        type boolean;
        default false;
        description
        "Controls wireless SSID broadcasting";
    }
}

container encryption-method {
    description
    "Wireless encryption mode";

    leaf algorithm {
        type identityref {
            base wireless-encryption-algorithm;
        }
        description
        "Wireless encryption algorithm";
    }
    leaf password {
        type string;
        description
        "Wireless encryption password";
    }
}
}
}
}
```