Automatically Provisioned Embedded Systems in Managed Networks

Jiri Slachta[a], Miroslav Voznak[a,*], Homero Toral-Cruz[b], Peppino Fazio[c]

[a]VSB-Technical University of Ostrava, 17. Listopadu 15/2172, Ostrava 708 33, Czech Republic
[b]University of Quintana Roo, Boulevard Bahía s/n Esq. Ignacio Comonfort, Col. del Bosque 77019, México
[c]University of Calabria, Via P. Bucci 41/C, Arcavacata di Rende (CS) 87036, Italy

---

Abstract

The article deals with a design of a new automatically provisioned embedded system. Through the years of our active development a highly advanced platform has been created. This platform, called BEESIP, is meant for the embedded network devices, and allows them to act as telephony exchanges, secured access points, VPN concentrators, etc. As the key feature of the BEESIP platform a unique building and provisioning system of the network devices has been developed allowing the administrators to fully control the firmware and configuration of the devices even in the remote and inaccessible locations. The process of custom firmware building and device provisioning eases the mass deployment of the BEESIP based hardware to cover the needs of small to medium businesses in vast range of services.This paper describes the latest progress in development and the proposition of the BEESIP system, which is based on one of the popular Linux distributions for the embedded devices. The developed system is fully adoptable and each component is reusable on any other Linux distribution. This system introduces several components for several areas, such as security, PBX, provisioning or management. Several components have been developed from scratch and the rest of the components have been fully adopted.

1. Introduction

Nowadays, there is a trend to simplify management of network devices by moving the service infrastructure to cloud service providers. Such steps are considered appropriate in decisions to reduce costs for maintaining the devices and also the services running on them. However, during the movement of infrastructure to third party several questions might arise. Services in cloud infrastructure were designed to be used for everyone, but they usually lack the broad configurability. In addition to these issues the security questions has to be resolved, since the infrastructure is not under the control.

During the years of development, a platform to address the mentioned issues has been created [1], [2]. Among desired characteristics belongs an easy integration of such device into almost any computer network. The main intention of this open-source platform called BEESIP is to provide easily integrated multimedia services. This project serves as a robust and secure VoIP telephony infrastructure with additional key components that makes this solution easily adaptable and configurable even without the deep knowledge of the technologies used by the components. It also aims to be a scalable solution with unified configuration in mind [1].

2. State of the Art

As mentioned in the introduction, we discuss the implementation of a SIP communication server solution which would be an alternative to several current implementations. At present, there are several projects that offer multipurpose IP telephony solutions for embedded devices and for household or enterprise platforms [3], [4].

The initial project of a GNU/Linux distribution which offers an easy set-up of IP telephony in a few steps is the Asterisk@Home project. This project integrated a web interface for Asterisk, Flash Operators Panel to control and monitor PBX in real-time and also offered a full FAX support within one bootable image for almost any x86 PC. The development of this project was discontinued and was replaced by its successor Trixbox in 2006, however, the development of Trixbox does not seem to continue any more. Two existing projects - AsteriskNOW and Elastix – now offer an alternative to Trixbox. The former, AsteriskNOW appears to be similar to Trixbox – a packed GNU/Linux distribution with Asterisk with a FreePBX web interface on top of it. The latter, Elastix, is a bit more modular. Compared to any other project, it offers a slightly more modular hierarchy to facilitate the applicability to a multiple service server [5], [6], [7].

3. Platfrom Architecture

One of the biggest challenges during BEESIP development was to create or modify any existing Linux distribution to serve our expectations. We needed to create an environment that would be fully customizable to any purpose and also to be easily maintainable through the time the BEESIP would be developed. The advantage of portability to any platform was also welcomed. The choice of Linux distribution we wanted to modify fell on OpenWrt Linux distribution. The reason why we chose that system was the approach for building firmware, the toolchain, cross-compiler and all applications are downloaded, patched and built by scratch. This means that OpenWrt does not contain any source code, it does only have its build system with templates, patches and Makefiles with procedures how to build a system and its packages for targeted device. This approach allows us to create custom procedures for build system and packages that can be modified at any stage.

A simplified view on BEESIP architecture is depicted in Fig. 1 which describes how the architecture is designed. The first block, the build system, is a wrapper on the top of the OpenWrt build system. It is designed for easy creation of firmware images within the single text file which describes what should be built for specific architecture and device we are targeting on. The secondary part of BEESIP is the OpenWrt Linux distribution which uses packages that provides desired functionality. In this case to provide modules from packages that serves as PBX, monitoring system, security system, management system and the core connecting modules among themselves.

In almost each modern home and small office there has been distributed and deployed embedded networking equipment for routing the Internet connection, providing multiple media services or to secure the network behind the device. Despite the fact that there has been some focus on the security of network itself, those devices has received small attention to prevent the attackers to abuse the open vulnerabilities on such devices. The absence of

computational capacity on such devices is also another fact that has to be resolved to prevent denial of service. The solution of security in BEESIP is based on SNORT application with cooperation of SNORTSam and iptables [8], [9]. In addition to preventing multimedia systems being unusable during DoS attacks the system has to protect itself. For limiting and blocking the attacks over VoIP traffic the ratelimit and pike modules from Kamailio package are used [10].

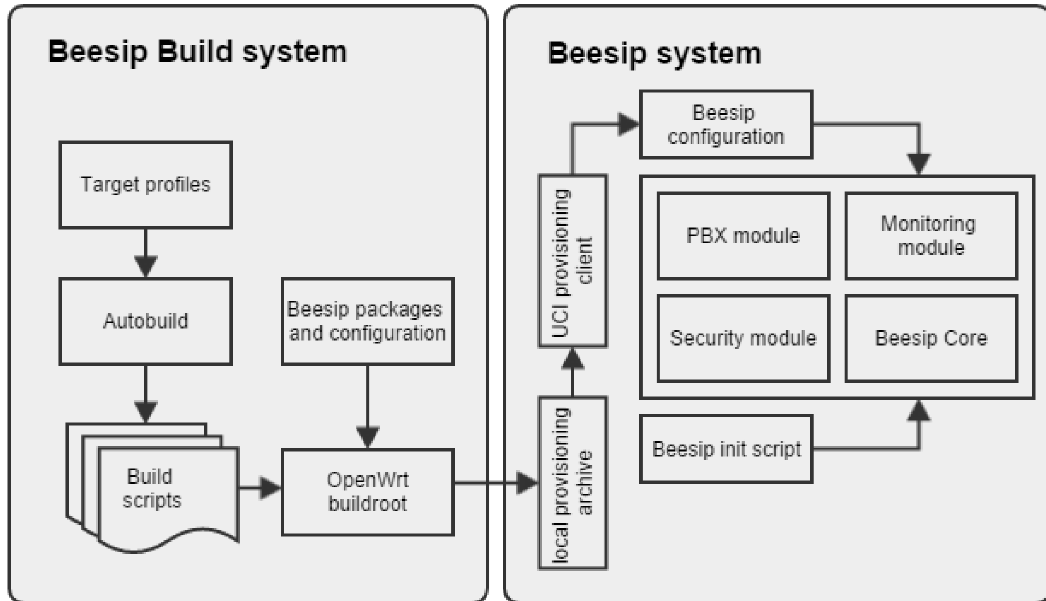

Fig. 1. Architecture of BEESIP system.

The architecture of BEESIP system focuses mainly on providing multimedia services, such as VoIP telephony. Administrators of this services have to ensure if the content is delivered reliably, securely and the data should also follow given quality parameters as well. In the beginning of the project the first version of monitoring system was proposed [11].

For monitoring purposes the measurements of IP telephony traffic are achieved directly on the device. This solution exploited a tshark package and our java application interpreting the results from tshark. This one-time measurements gives information about particular speech quality [12]. However, this solution providing one-time measurements was not robust enough. To address the anomalies in the network infrastructure the successor of the previous application has been made. The monitoring module works as an agent in the system which provides continuous monitoring evaluated immediately on the monitoring server. The rest of the monitoring functionality is handled by the Zabbix agent.

The described modules delivers heterogeneous services which has to work in conjunction with each other. The last part of the system are core services that provides abstraction layer on the top of the modules. It is represented by a shell library (providing functions for scripts) with executable files to make the system working. As a configuration provider we developed UCI provisioning client that prepares the configuration for the system.

We achieved significant advances especially in the BEESIP build system and the core module which have been recently newly designed and impelemented, therefore, we focused our intent in this paper on these news in BEESIP, whereas all modules are described in detail in our last article published in [1].

4. Build System

The build system reduces the building procedure into one script call. As said above, BEESIP is based on GNU/Linux distribution OpenWrt which is built on top of the OpenWrt Buildroot. Buildroot is a set of Makefiles

and files that allows to compile cross-compilation toolchain and to generate by that toolchain resulting cross-compiled applications into a root filesystem image to be used in the targeting device. Cross-compilation toolchain is compiled by host compilation system which is provided by any GNU/Linux distribution.

In the beginning of BEESIP development we met issues that were holding us back. We could not test all changes immediately, we had to recompile all code and generate images nearly always when we ported new application, modified post installation scripts or when cross-compilation toolchain has changed. Also, the system behaves differently during testing if it is new root filesystem image, or modified root filesystem that has been run more than once. At least those issues led us to create an easy interface that will ease the creation, automation and functional testing for system images.

BEESIP build system is a set of scripts, Makefiles and definition files that make an easy interface to OpenWrt Buildroot. We can consider the main Makefile to be as a core of the BEESIP build system. It performs all atomic operations with OpenWrt Buildroot, works with source code management systems (to update/revert/any operation with local copies of OpenWrt source codes), patches OpenWrt Buildroot and executes images as virtual machines. Those commands might be used by any user or by autobuild scripts, which will be described after.

On the top of the core Makefile is autobuild.sh script. This script calls all atomic operations within more complex parameterized operations whose variables are defined in specific target files. Those target files are user defined and on the basis of those files are configuration files for OpenWrt Buildroot created. Once we have configuration files the system images could be created by calling autobuild.sh script with command build and parameter containing the name of the target file.

```
TARGET_CPU=ar71xx
OWRT_NAME=trunk
TARGET_NAME= eduroamap_wr841nd-ar71xx-trunk
TARGET_QEMU=mips
TARGET_QEMU_OPTS=-m64

$(eval $(call BesipDefaults,eduroam))

OWRT_IMG_FACTORY_NAME=openwrt-ar71xx-generic-tl-wr841n-v8-squashfs-factory.bin
OWRT_IMG_PROFILE=TLWR841
OWRT_IMG_KERNEL_NAME=openwrt-$(TARGET_CPU)-generic-vmlinux.elf
OWRT_CONFIG_SET += \
     TARGET_ar71xx=y \
     TARGET_ar71xx_generic_TLWR841=y
```

The following example shows how to build system images based on the target file.

```
#./autobuild.sh build eduroamap-wr841nd-trunk
```

Such techniques can be used for any purpose of automated building system images for any device or platform supported by OpenWrt. Those could be firmware images for campus access points, specialized network probes, virtualized multimedia servers or any other devices.

5. Core Module

The role of the Core module is to provide a glue among all services that served by all BEESIP modules. The most important part of the Core module is the BEESIP shell library that provides functions for all utilities and scripts used by BEESIP system. Functionality of a Core module complements utilities for configuration management and for simplified configuration of system image. With all those utilities comes along also default configuration which prepares all module services into fully functional state with all BEESIP modules running and operational. Also, the

role of this module is to switch any existing OpenWrt environment to BEESIP environment while the device is booted the first time or the BEESIP environment is used and ran the first time.

5.1. BEESIP Environment

BEESIP environment handles tasks which has to be performed at several stages in OpenWrt operating system. After the BEESIP firmware image is built at this stage the operating system behaves as clean operating system with installed dependencies required by BEESIP package and its services. On the first boot the init script is performed and it waits until the overlay filesystem is mounted. When the filesystem is mounted subsequently the UCI provisioning client (uciprov) prepares the configuration for the system and services. The details about the techniques used in the provisioning client are described in the paper later. The main advantage of this procedure is the applicability to any existing setup of OpenWrt system.

Each component of the system uses the centralized configuration of OpenWrt, known under abbreviation UCI. The main executable application includes the most of the OpenWrt functionality and BEESIP functionality into one directly available set of commands. In addition to the basic functionality, such as system upgrades and resetting system to the factory defaults, it also prepares the system for the situation when it becomes unstable and unusable. The executable script collects information for crash reporting used for application debugging.

5.2. BEESIP Telephony Environment

Even though almost every part of the environment was described in the previous paragraphs, the additional value on the top of the PBX system should be introduced. Since the CESNET, which is the national research and education network that interconnects academic institutions, has information about academic network infrastructure, this project should draw benefits from its nature. Each participant of this network stores the data about their VoIP gateways in the IPTelix system, which is the database for the VoIP gateways connected to CESNET. The main focus of the system is to maintain and to monitor prefixes to the gateways.

The data that are obtained from the database are in the JSON format or in format that used for the extensions.conf and the sip.conf. The script environment that prepares the configuration of internal Asterisk, sets up the outgoing traffic to create trunks against the gateways. To identify the VoIP traffic the data from BEESIP UCI configuration file are obtained and subsequently the IP telephony prefixes and the numbering plan is set up.

Phone provisioning tool, which is connected to local Asterisk PBX in the system, creates phone provisioning data in according to the type of the phone connected to it. For the purpose of phone configuration the schema of the specific phone model has to be provided for this tool.

5.3. Provisioning Client

The impetus for development of provisioning tool arose during the period when firmware images created by BEESIP build system were deployed to computers, routers and wireless access points. Those machines were not configured for target networks, which were supposed to be deployed on. Because the target configuration does not depend on a person which builds the system, but on the network administrator, then configuration should lay outside of a BEESIP firmware image. The creation of such tool bring a question how should the target device should fetch and apply its configuration.

In the build system, we can pass static information about our provisioning server which provides configuration (during build time). We can also change this information in firmware image. This information can be used for protocols which translates one kind of information to another. As an example we can use DNS protocol and its TXT records. The target configuration could be stored on a server designated within an URI in a variable from TXT record which is obtained from static URL provided by BEESIP build system. This solution is replicable for any protocol which allows distribution that kind of information (LLDP, DHCP or any other else).

An example how to resolve UCI provisioning URI:

```
host -t txt provdomain \
provdomain descriptive text \ "provuri=http://12.34.56.78/uciprov/"
```

If a device knows where to obtain configuration from then the device can construct all provisioning URI addresses for each device state it needs. This approach is needed when system administrator needs to differentiate configuration for devices which starts up the first time, if those devices are refreshing its common device configuration on a regular basis or if it is the configuration that is obtained after device startup. UCI provisioning client written for BEESIP currently handles only configuration files that are handled by UCI system (Unified Configuration Interface) for centralized configuration. If a device knows where to obtain configuration from, then it can obtain configuration data from ordinary transport protocols designated in provisioning URI. The benefits that BEESIP draws from OpenWrt builds upon the UCI configuration system which is based on plain text configuration files with firmly defined structure. This configuration is obtained using software for file retrieval from network resources, e.g. wget, and immediately imported into UCI.

From the introductory part of motivation for the techniques it is clear why provisioning is a needed component for configuration deployment on higher number of such devices. During the development of any application or any system the developers needs the ease up the process of deployment of applications and its configuration, thus the UCI provisioning tool was developed, known under abbreviation uciprov.

The architecture of the uciprov tool stands on the two separate parts. The first part of the uciprov tool is located in the build system of OpenWrt. The package itself supports the selection of used protocols for discovery of provisioning URI, and also offers user to add specific variables during the build time. Within the package we can also work with macros, thus all variables does not have to be static at all. This can be used when the domain has to be resolved, but the URL structure is known. There are several macros that mostly identifies the hostname, domain, MAC address, IP address, release number and many other else.

On the basis of information from previous paragraphs we are able to make system upgrades or automatic system reconfiguration without administrator intervention on the targeting device.

An example of used macros in the build system for the uciprov tool can be seen below where the URI to image for the system upgrade is distributed.

```
string "Static URI for sysupgrade"
  depends on UCIPROV_USE_STATIC
  default "{base_uri}/image{fd}.bin"
```

The second side of UCI provisioning tool is an installable package to OpenWrt system. Despite the fact that the tool was designed for the distribution of UCI configuration among all desired devices, it can call any function that we hook to any stage of uciprov tool. The modules for this tool must hook their functions with following specific uciprov stage by “book_hook_add uciprov-stage function-name” call in the script preamble. Thus we are able to do system upgrades, distribute SSH keys or configuration files that does not comply with UCI syntax. Since we are working only with variables, we can move the application logic into scripts, where handling with those variables is handled. This led to creation of URI resolver scripts (DNSSEC, DNS, HTTPS...) and also to scripts for handling the constructed URIs (system upgrade, public key distribution…).

The server side of UCI provisioning is currently solved by providing static file structure with files which consists of export provided by UCI system. See flow diagram depicted in Fig. 2 to see how the UCI provisioning works.

The client side has several stages:

1. *Waiting for the system to be ready to be provisioned.*
2. *Stage 1 (preinit):*
   - *During the first stage the URIs are obtained.*
   - *Uciprov macros are set up from UCI configuration file. The same applies to every variable.*
   - *Subsequently the uciprov_geturi hook is called. This stage calls every URI resolver script.*

- *Call user hooked scripts.*

3. *Stage 2 (obtain configuration from URI):*
   - *URI addresses are validated.*
   - *Obtain configuration or files from user modules.*
   - *Call user hooked scripts (preapply).*
   - *If obtaining configuration failed, retry stage 2.*
4. *Stage 3 - apply received configuration.*
   - *Call user hooked scripts (postapply, reboot)*

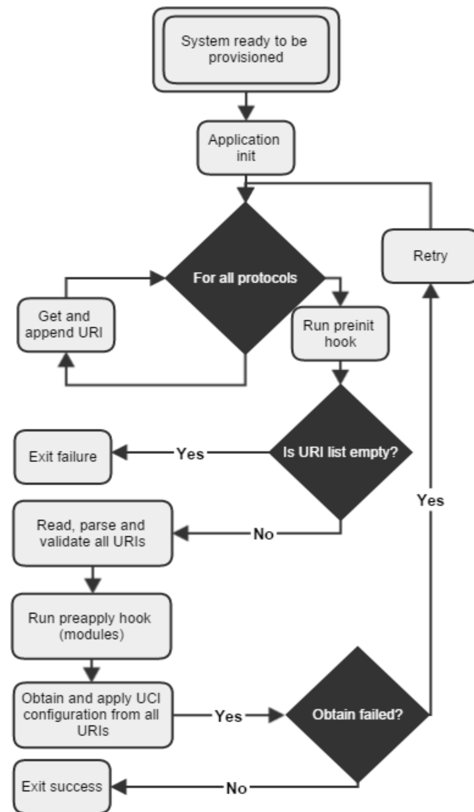

Fig. 2. Flow diagram of UCI provisioning client.

5.4. Configuration Submodule

The NETCONF protocol exploits a specified mechanism for exchanging the configuration data among an administrator and network devices. This protocol allows the device to send and receive configuration data through XML documents using the RPC paradigm [13]. These XML documents are handed over the RPC calls; the RPC request is initiated by a client that requests the configuration data or a command to be performed on the server.

While these requests are being performed, the client is blocked until he receives the RPC reply from NETCONF server. That replies consists of a configuration that is complete or a partial. Another reply is a message informing us if a command was successfully performed on the server or not. This communication is transferred over a transport protocol which has to be secured and to allow an authentication and authorization. The most probable and secure way, how to communicate with the NETCONF server, is to use SSH2 protocol (RPC calls over SSH subsystem).

The structure of configuration data on NETCONF server is specified by a YANG module which defines the semantics and syntax of a management feature. It provides complex data structures which allow design any data structures that will meet the requirements of developers. The early design of the NETCONF functionality was based on the Yuma project, which is a package that was used for providing tools for the network management. It consists of a NETCONF client yangcli, server netconfd, validation tools and netconf-subsystem, which allows us to communicate with NETCONF server through a SSH2 subsystem. However this development of this solution was discontinued and the proposed solution is currently being converted to Netopeer server, which uses the libnetconf library. This library with the application is developed as well as the BEESIP project under CESNET organization.

The configuration data are stored securely on the NETCONF server and all requests and responses must comply with firmly defined structure, specified by YANG modules. Next, global database of all the configurable parameters is required and it is ensured by NETCONF server. The configuration parameters are inserted by the user and stored in the NETCONF server. Consequently, the stored configuration data are available through simple queries. It makes the device quickly configurable, therefore a backup or a restore of configuration can be simply and quickly performed.

OpenWrt uses UCI as configuration backend, it is a group of configuration files which can be read or modified by common UCI API. The YANG module for BEESIP is divided into containers and definitions. There was maximum effort to create simple but working schema for VoIP communication system. All containers and data types were glued from others VoIP software configuration options. We examined many configuration types and scenarios and extracted best practices from there. Core point of configuration is SIP interface. All messages are routed through this object. There can be more SIP interfaces in configuration, each of them can have another transport protocol, IP address or port. Next to this, there is SIP routing container which is used to specify next hop addresses, filtering and message mangling.

We defined following containers:

- **Interfaces container**
  System IP interfaces are listed here. Both IPv4 and IPv6 are supported. IP addresses are glued later to other modules. All traffic from and to BESIP can flow only across interfaces defined here. For simplicity and clear definition we use ietf- ip RFC draft for this container.

- **Dynamic maps container**
  For dynamic mapping of objects, we defined this container. Dynamic map is entity where we construct several outputs from external inputs and for modifying external data by device. Dynamic means that they are not statically bound to configuration file, but target engine will ask for resolving just during call. For example, we can create dynamic map, which will take telephone number as parameter, asks LDAP server and returns caller name. Another example is dynamic map which can get or set credit of user to use prepaid accounting. Dynamic map will be evaluated for every call. Opposite of dynamic maps are static maps created by preprocessor module. Today we prepare two basic types of dynamic maps - LDAP and SQL. Later, there can be more types like DNS. Dynamic maps have nothing to do with internal data structures of configured system.

- **Realms**
  Realms are group of domains with same security considerations. Instead of writing same routing rules for any domain, it is better to use this concept. Best practice is to create realm internal, external, operator and trusted. Next to this, each realm has own security rules. Like maximum number of simultaneous calls or maximum number of errors per second. We are preparing basic YANG definitions hierarchical. So configuration data, if they are same for more contexts, will be inherited in this order: globals, realms, domain, sip-interface.

- **Domains**
  Domains container explicitly defines parameters for DNS mapping to real domains. There are three common types of domains - served by system, known to system and unknown to system. When domain is server by system, features of domain are derived from configuration file. Known domains are not directly served by system, but from some reason (like nonstandard routing) have to be defined in configuration. Unknown domains are served by DNS lookups by default. For example, it is possible to define SIP services, sip interfaces or methods running on served domain. For not served but known domain, it is possible to explicitly set outgoing SIP gateway. Domain container is designed very carefully because our motivation is to use standard Internet technologies for call routing whenever possible. For well configured domains with NAPTR and SRV records, no configuration is needed at all. Except local security consideration like putting domain into internal realm. Domains can be glued to dynamic map.

- **Users**
  Users inside system are only in one container. Their rights are checked against roles. Users can be mapped to dynamic map or they can be generated by preprocessor module in phase of configuration. Everything is authenticated and authorized against this container. This means NETCONF, CLI, Web-GUI, SIP and all other services have common place for authentication and authorization. This is crucial because user can register to SIP or change user parameter of his line across Web-GUI with same credentials. We will support user certificates for authentication as well. Users can be interconnected with dynamic map. So system will ask external entity for resolving user parameters or set them.

- **SIP Interfaces**
  SIP interfaces are interconnected with interfaces, domains and realms. SIP interface can be TCP, UDP, and TLS on both IPv4 and IPv6. Each SIP interface has defined engine. Engine is software/hardware which does SIP proxying or B2BUA. Configuration file processor will generate configuration files for each kind of engine. Today, we plan Asterisk, FreeSwitch, Kamailio and OpenSIPS. Each engine has set of features. For example, Asterisk cannot act as SIP proxy and Kamailio cannot act as B2BUA. We suffer from YANG definitions where it is possible to block configuration parts which are not possible if given feature is not enabled.

6. Conclusion

This paper describes the idea and the proposition of the BEESIP system, which is based on one of the popular Linux distributions for the embedded devices. The developed system is fully adoptable and each component is reusable on any other Linux distribution. This system introduces several components for several areas, such as security, PBX, provisioning or management. Several components have been developed from scratch and the rest of the components have been fully adopted. The systems for voice quality measurement and for system provisioning have been especially designed for the deployment on embedded devices where the applications takes effect. The contribution of our work lays also on a new idea of the unified configuration management with specific syntax which enables user to configure system without the knowledge of specific engine on the base of the system, Asterisk and Kamailio in our case.

Fully functional platform images are distributed and prepared mainly for x86 platform, which is also possible easily virtualized for testing or deployment purposes. Nowadays there are platform targets created for x86 platform and access points based on MIPS architecture (ar71xx platform). Binary images from the auto-build system can be downloaded from [14] and source codes can be cloned from GIT repository from the same page as well [14].

There are several example firmware images for several target devices, such as TP-Link access points or Raspberry PI computer. Configuration is available through web-browser, SSH client or to be provisioned using supported provisioning protocols. Next to this, CLI syntax has been developing and will be connected to the NETCONF server using also the UCI configuration as the frontend. CLI will be independent of internal software so

if some internal software is modified, there will be no change in configuration. Even more, CLI and NETCONF configuration will be independent on hardware and version. To export configuration from one box and to import it to the next one will be a simple task. Users will modify only one configuration file to manage entire box.

The contribution of this work lies in the overall concept of the BEESIP and the approach used in the build system which can help networkers to maintain increasing amount of devices that are under their administration. Moreover, we became responsible for maintenance of Telephony repository in OpenWrt and the trust given us by OpenWrt community is the real appreciation for our work in the BEESIP project. Now, we check and help to improve every patch and package which is submitted by developers in Telephony OpenWrt repository. We are closer to the needs of networkers and connected with OpenWrt community.

Acknowledgements

This research was funded by the grant SGS reg. no. SP2015/82 conducted at VSB-Technical University of Ostrava, Czech Republic and by the Ministry of Education of the Czech Republic within the project LM2010005.Achieved results were partially supported by the project No. CZ.1.07/2.3.00/20.0217 "The Development of Excellence of the Telecommunication Research Team in Relation to International Cooperation" within the frame of the operation programme "Education for competitiveness" financed by the Structural Funds and from the state budget of the Czech Republic.

References

1. Voznak M, Slachta J, Macura L, Tomala K. Advanced solution of SIP communication server with a new approach to management. *Telecommunication Systems*, 9 p. (Article in Press), 2014.
2. Voznak M, Slachta J, Macura L. Development of advanced concept of voice communication server on embedded platform. *International Journal of Mathematical Models and Methods in Applied Sciences*, vol. 7, no. 2, pp. 103-110, 2013.
3. Segec P, Kovacikova T. A survey of open source products for building a SIP communication platform. *Advances in Multimedia*, art. no. 372591, 2011.
4. Abid F, Izeboudjen N, Bakiri M, Titri S, Louiz F, Lazib D. Embedded implementation of an IP-PBX/VoIP gateway. *In Proc. 24th International Conference on Microelectronics*, Article number 6471377, 2012.
5. Titri N, Louiz F, Bakiri M, Abid F, Lazib D, Rekab L. An Opencores/Open-source Based Embedded System-on-Chip Platform for Voice over Internet. INTECH: *VOIP Technologies*, pp. 145-172, 2011.
6. Prasad JK, Kumar BA. Analysis of SIP and realization of advanced IP-PBX features. *In Proc. 3rd International Conference on Electronics Computer Technology*, Volume 6, Article number 5942085, pp. 218-222, 2011.
7. Alam MZ, Bose S, Rahman MM, Al-Mumin MA. Small office PBX using Voice over internet protocol (VOIP). In *International Conference on Advanced Communication Technology*, art. no. 4195481, pp. 1618-1622, 2007.
8. Voznak M, Safarik J, Rezac F. Threat prevention and intrusion detection in VoIP infrastructures. *International Journal of Mathematics and Computers in Simulation*, vol. 7, no. 1, pp. 69-76, 2013.
9. Chi R. Intrusion detection system based on snort. *Lecture Notes in Electrical Engineering*, 272 LNEE (VOL. 3), pp. 657-664, 2014.
10. De Cicco L, Cofano G, Mascolo S. Local SIP overload control. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7889 LNCS, pp. 204-215, 2013.
11. Voznak M, Tomala K, Vychodil J, Slachta J. Advanced concept of voice communication server on embedded platform. *Przeglad Elektrotechniczny*, vol. 89, no. 2 B, pp. 228-233, 2013.
12. Estrada L, Torres D, Toral H. Analytical description of a parameter-based optimization of the quality of service for voIP communications. *WSEAS Transactions on Communications*,vol. 8, no. 9, pp. 1042-1052, 2009.
13. Schönwälder J, Björklund M, Shafer P. Network configuration management using NETCONF and YANG. *IEEE Communications Magazine*, 48 (9), art. no. 5560601, pp. 166-173, 2010.
14. Project BEESIP, Online Available onURL: https://homeproj.cesnet.cz/projects/besip/wiki/Download