https://doi.org/10.26552/com.C.2016.1.48-55

Jiri Slachta - Miroslav Voznak - Dan Komosny - Homero Toral-Cruz - Peppino Fazio *

AUTOMATICALLY PROVISIONED EMBEDDED COMMUNICATION SYSTEM BASED ON OPENWRT PLATFORM

The article deals with a design of a system that provides tools for creation of automatically provisioned embedded communication system and its components. As the key feature of the BEESIP platform (Bright Efficient Embedded Solution for IP Telephony) 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 business in the vast range of services.

Keywords: *BEESIP, SIP, IP Telephony, RTP, Asterisk, Kamailio, Snort, OpenWrt.*

1. Introduction

In order to reduce costs for maintaining communication systems, both physical devices and services running on them, many companies make a decision on moving the service infrastructure to cloud service providers. Nevertheless, 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 have 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]. Among desired characteristics belongs an easy integration of such device into almost any computer network. In mid of 2011, a new project (BESIP) was established under strong support of the CESNET association (Association of Czech universities and Academy of Science), aiming at a robust and secure VoIP telephony infrastructure with additional key components that make 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 the unified configuration in mind [1]. The given name BESIP has had to be changed to BEESIP (The Bright Efficient Embedded Solution for IP Telephony) in 2014 because our BESIP trademark registration was rejected by the Czech Industrial Property Office due to the same existing trademark in field of public transport. In last two years, next important features have been implemented, and the automatic provisioning belongs to them. Soon after being ported Asterisk to OpenWrt Linux distribution within BEESIP project, we became responsible for maintenance of Telephony repository in OpenWrt, it includes any packages and patches which are connected with telephony.

2. Related work

As mentioned in the introduction, we discuss the implementation of a SIP communication server solution which would be an alternative to several current implementations. The main advantage of our solution is the ability to easily and quickly set up a full-featured PBX on almost any hardware. We can presume that almost all implementations are based on open-source Asterisk PBX, web-interface for Asterisk and with a GNU/Linux distribution on the base layer. At present, there are several projects that offer multipurpose IP telephony solutions for embedded devices and for household or enterprise platforms [2 and 3]. 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. The first version of this project was released on 29 April 2005.

* [1]Jiri Slachta, [1]Miroslav Voznak, [2]Dan Komosny, [3]Homero Toral-Cruz, [4]Peppino Fazio
[1]VSB-Technical University of Ostrava, Czech Republic
[2]Brno University of Technology, Czech Republic
[3]University of Quintana Roo, Col. del Bosque, Mexico
[4]University of Calabria, Arcavacata di Rende, Italy
E-mail: jiri.slachta@vsb.cz

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. On 3rd May 2006 the development of this project was discontinued and was replaced by its successor Trixbox. 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 [4 and 5]. The increasing popularity of embedded devices, such as Raspberry Pi, is the reason why the Micro Elastix distribution was born. However, all of those projects are either prepared for x86 machines only or for specific hardware. Micro Elastix only supports three platforms, namely PICO-SAM9G45, MCUZONE and Raspberry Pi [6].

None of the projects includes a security module that would offer a complete IPS and IDS system to prevent attacks against the SIP Registrar server. Also, there is no module that would monitor the quality of voice calls transmitted through an integrated PBX [2 and 7]. Thanks to the portability of the OpenWrt distribution we prepare a BEESIP bootable image for almost any device.

3. Platform 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 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. It 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

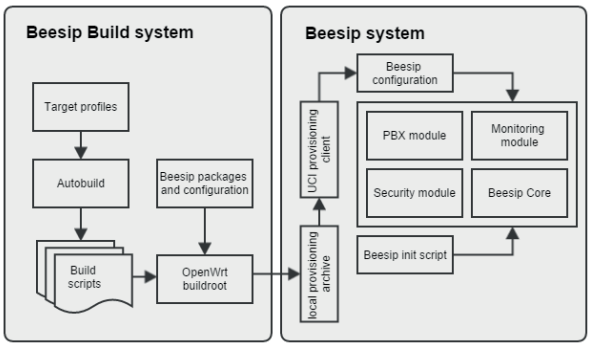

Fig. 1 Architecture of BeeSIP system

targeting on. The secondary part of BEESIP is the OpenWrt Linux distribution which uses packages that provide desired functionality. In this case to provide modules from packages that serve as PBX, monitoring system, security system, management system and the core connecting modules among themselves. In almost each home and small office there has been distributed and deployed embedded networking equipment for routing the Internet connection, providing multiple media services and securing the network behind the device. Despite the fact that there has been some focus on the security of network itself, those devices have received small attention to prevent the attackers to abuse the open vulnerabilities on such equipment. 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 and 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. The architecture of BEESIP system focuses mainly on providing multimedia services, such as IP telephony. Administrators of this service have to ensure if the content is delivered reliably, securely and the voice traffic should also follow given quality parameters as well due to an impact of background traffic on speech quality [10 and 11]. In the beginning of the project the first version of monitoring system was proposed [12]. 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 measurement gives information about particular speech quality. 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. Nowadays, the monitoring module works as an agent in the system which provides continuous monitoring evaluated immediately on the monitoring server and our work in this field received a best paper award at 22nd International

Science Conference on Computer Networks [13]. The rest of the monitoring functionality is handled by the Zabbix agent. The described modules deliver heterogeneous services which have to work in conjunction with each other. The last part of the system are core services that provide 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.

4. Build system

Before the concept of BEESIP system is described, it is necessary to introduce the build system which 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 codes 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. The 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. 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 telephony environment

Since IP telephony in the Czech national research and education network is highly developed and we interconnected via VoIP nearly all PBXs' of Czech universities 15 years ago [12], we keep information about academic network infrastructure (more than 50 VoIP Gateways and PBXs behind them). The project BEESIP 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 the Czech academic research and education network. The main focus of the system is to maintain and to monitor prefixes to the gateways. The data are provided from the database in the JSON format, our scripts in BEESIP automatically prepare configuration files for internal Asterisk and set up the outgoing traffic to establish trunks against these 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 internal Asterisk in the system, creates phone provisioning data according to the type of the phone connected to it.

5.2. 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 who builds the system, but on the network administrator, then configuration should lie outside of a BEESIP firmware image. The creation of such tool brings a question how 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 translate 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 build system. This solution is replicable for any protocol which allows distribution of 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 necessary when system administrator needs to differentiate configuration for devices which start up the first time. UCI provisioning client currently handles not only configuration files that are handled by UCI system (Unified Configuration Interface) for centralized configuration, but also the tar archives that consist of compressed overlay. 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 need to simplify 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 do 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 identify the hostname, domain, MAC address, IP address, release number and many others and finally, 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 the following specific uciprov stage call „uciprov_hook_add uciprov_stage custom-function-name“ in the script preamble. Thus we are able to upgrade the BEESIP system, distribute SSH keys or configuration files that do not comply with UCI syntax. Since we are working only with variables, we are able to move the application logic into scripts. This led to the implementation 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 consist of an export provided by UCI system. The flow diagram is depicted in Fig. 2 and the description, how UCI provisioning works, is following:

1. Waiting for the system to be ready to be provisioned.
2. Stage 1 (preinit):
   a. During the first stage the URIs are obtained.
   b. Uciprov macro functions are set up from UCI configuration file. The same applies to every variable.
   c. Subsequently the uciprov_geturi hook is called. This stage calls every URI resolver script.
   d. Call user hooked scripts.
3. Stage 2 (obtain configuration from URI):
   a. URI addresses are validated.
   b. Obtain configuration or files from user modules.
   c. Call user hooked scripts (preapply).

d. If obtaining configuration failed, retry stage 2.
4. Stage 3 - apply received configuration:
5. Call user hooked scripts (postapply, reboot).

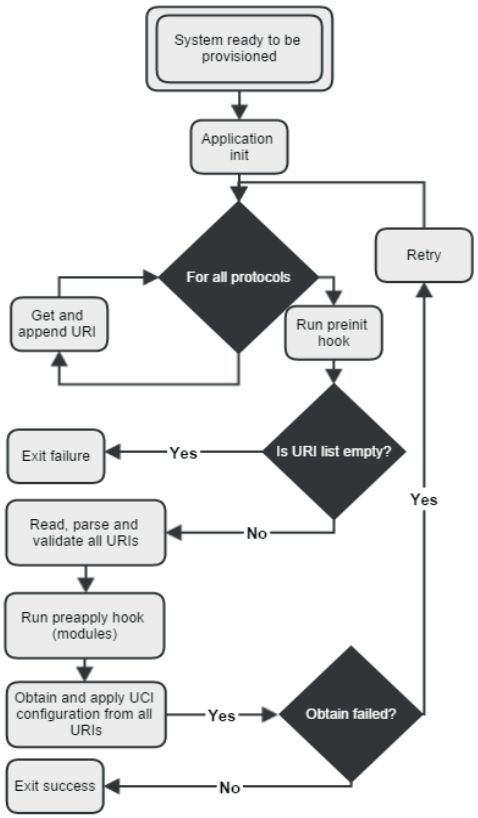

Fig. 2 Flow diagram of UCI provisioning client

6. PBX module

The PBX module is a key part of the BEESIP project. It operates as SIP proxy or SIP B2BUA, depending on configuration, and ensures a call routing. Asterisk is used for call manipulation and for the PBX services. Kamailio is used as a complementary part of PBX module for the proxying SIP requests, the traffic normalization and for the security. There are always two factors when developing VoIP solution. The first one is high availability and reliability, the second one is an issue of advanced functions. Many developers try to find a compromise, we have implemented both, and our BEESIP is able to adapt to the user requirements [12]. More complex system can handle many PBX functions such as a call recording or an interactive voice response but due to the bigger complexity it is more susceptible to fault. On the opposite side, pure SIP proxy is easier software, which can perform call routing, more fault tolerant, but it is more difficult to use the advanced PBX functions.

7. Security module

The security module is next element of BEESIP and all the time, it was considered to make the developed system as secure as possible [8]. Next to this, the entire system has to be fault-tolerant, monitored and protected from attacks. If a security incident is detected, BEESIP immediately solves the situation and notifies this event in a detailed report to administrator. The attacks are recognized and processed by SNORT rules, the source IP address is automatically sent into the firewall by SNORTSam and the intruder's IP is blocked. This is very flexible, reliable and efficient solution. Dropping attack based on IP directly in the Linux kernel is much more efficient than to check messages on the application level. Only first messages are going to SNORT filter. When SNORT identifies a suspicious traffic, next messages from the same IP are blocked. If more soft faults appear from some IP, it is blocked at the IPTABLES level; this approach can effectively block incorrectly configured clients and servers. For example, if a client sends REGISTER with proper credentials, it is not obviously security attack but the client attempts to register again and again, with every registration requires computing sources at SIP REGISTRAR server, see Fig. 3. Such attempts can be denoted and blocked for a time interval, the line IPS (Intrusion Protection System) represents the CPU load in case of active security module in BEESIP. The dependencies clearly prove the ability of security module to mitigate the performed attacks.

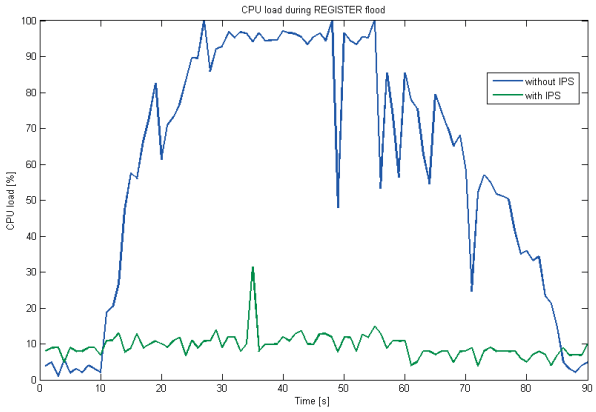

Fig. 3 Attack effectiveness based on REGISTER flood

Administrators can use Zabbix agent inside BEESIP to gather all information directly into their monitoring system. Partially, BEESIP is resistant to some kind of DoS attacks. It depends on hardware used. If the hardware is strong enough to detect some security incidents on application level, the source IP is immediately dropped. Low-performance hardware cannot handle such detection on application level. In such a case, it is more suitable to stop DoS attack before it reaches

BEESIP. Therefore, the SNORT running on a dedicated machine provides more flexible and robust solution than the SNORT as an integral part of VoIP system [8].

8. Monitoring module

Due to the nature of the BEESIP architecture, which is focused mainly on the embedded and low-profile devices, BEESIP can be deployed as the network monitoring probe as well, as is depicted in Fig. 4. For these purposes, libraries allowing continuous monitoring of speech quality have been developed and incorporated into the BEESIP system. The monitoring module takes the advantage of the Asterisk PBX, which is the part of the BEESIP's PBX module, and is designed to work as the network probe [13].

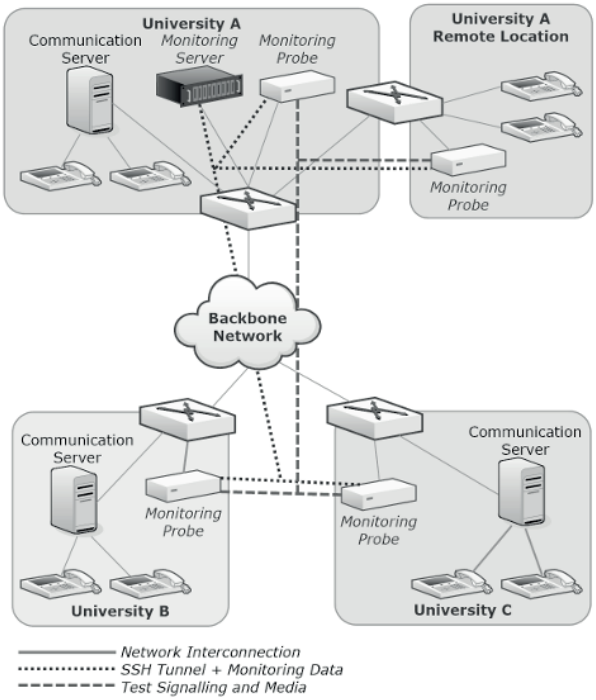

Fig. 4 The monitoring system architecture

For the BEESIP's monitoring module to work properly there are two necessary components. The first one is the BEESIP itself in the role of speech monitoring probe, the second one is the data collection and presentation server. While the former is fully implemented in the BEESIP, the latter requires only a mainstream platform capable of running HTTP server with Zabbix monitoring server and can therefore be run on any commonly used server platform. The server role is to collect the recorded phone calls and calculate the Mean Opinion Score for the sound files, the client part looks into the database of the partner probes, which is obtained from the Zabbix server, and then establishes a call to all these partner probes in regular intervals. The call media consist of sound samples that conform to the ITU-T P.862 recommendation, the media payload is identical for the both directions of the call and is recorded to the wav file. The resulting wav file is then sent to the server for analysis and presentation of the results. Due to the relatively simple implementation of the monitoring probe a highly scalable solution for speech quality monitoring can be deployed using the cost-effective hardware. The BEESIP ensures the automatic addition of new probes to the monitoring system, but it requires the access to the local DHCP (Dynamic Host Control Protocol) service, where the administrator needs to enter the records the probe requires to work properly. First of all, it is the probe's IP address and hostname, which are used to communicate with the rest of the network. Then it is the server's IP address or its resolvable domain name to allow the probe to communicate with the server and the last item (apart from standard ones, e.g. default gateway, etc.) is the path from which the probe can download its private key. The last item has to be kept secret and the communication must be secured using HTTPS protocol and restricted only for intended hosts. With all this information in place the probe then can register itself to the server without the user interaction, which is useful in large deployment scenarios. The BEESIP provides the secure communication based on public keys infrastructure, which is also used to provide access control, so that only authorized probes can join the system and fetch the sensitive data about the other probes. The transfer of network information from the server to the probes uses a module, which is built upon the Zabbix monitoring server [13]. By using the custom made tools and standard Zabbix communication application interface the direct link from the probes to network information database of the Zabbix server is created. The probe then reads the complete list of the probes, which is then used for testing. A standard Zabbix web interface is used together with visualization tool that allows to render a map of probes and the tested interconnections, so that the results of measurements are easily accessible and understandable. Together with the information about the partner probes, server sends the parameters of the test as well. The most important parameter carries the information about the time period between individual test rounds. All the test calls initiated by one probe are performed at once, which in case of precise time synchronization across the network will result in all calls to be performed at once. Due to relatively low bandwidth of a single call it is very difficult to reach measurable network load in the intended environment of backbone networks. When the call (or test round) is finished, the recorded wav files are then transferred to the server for analysis. The test rounds are then performed periodically and their frequency depends on the time interval set by the administrator of the network. This way

the compromise between network load and responsiveness of the system can be defined thus making the system usable in a vast number of different network environments.

9. Use cases

In this paper, main ideas of the BEESIP architecture are described, the BEESIP includes own provisioning and build tool that is able to build this OpenWrt-based system for almost any platform with desired characteristics. Up to this time there has been developed following use cases:

- Eduroam Access Points – Fully provisioned, adaptable and monitored Access Points (open-source and low-cost version supporting and spreading IP mobility and roaming within academic environment).
- Honeypots and network probes – fully provisioned honeypots and network probes that are able to evaluate network characteristics and collect network data.
- PBX and SBC (Session Border Controller) – Components for building an infrastructure that provides VoIP telephony.
- Speech quality monitoring probes – fully provisioned probes providing speech quality assessment with integration into the monitoring system Zabbix.

10. 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 modules for several areas, such as security, PBX, provisioning or management. Some of them have been developed from scratch and the rest of the components have been fully adopted. New tools for speech quality assessment and provisioning have been especially designed for the deployment on embedded devices. The contribution of our work is in the overall design, implementation and maintenance of open-source communication system BEESIP. We are aware of the fact that our work represents applied research and experimental development and is highly practically oriented, nevertheless with many positive comments from research and industrial community (e.g. technology director at Cisco on linkedin). All source codes are released under GPL license and available as open-source software. 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.

The approach used in the build system 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.

Acknowledgement

The research leading to these results received funding from the grant of SGS reg. no. SP2015/82 conducted at VSB-Technical University of Ostrava, Czech Republic and also was supported by the National Sustainability Program under grant LO1401. For the research, infrastructure of the SIX Center was used.

References

[1] VOZNAK, M., SLACHTA, J., MACURA, L., TOMALA, K.: Advanced Solution of SIP Communication Server with a New Approach to Management, *Telecommunication Systems*, vol. 59, No. 4, 2015, 541-549.

[2] SEGEC, P., KOVACIKOVA, T.: A Survey of Open Source Products for Building a SIP Communication Platform, *Advances in Multimedia*, no. 372591, 2011.

[3] ABID, F., IZEBOUDJEN, N., BAKIRI, M., TITRI, S., LOUIZ, F., LAZIB, D.: *Embedded Implementation of an IP-PBX/VOIP Gateway*, Proc. of 24th Intern. conference on microelectronics, No. 6471377, 2012.

[4] 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, 2011, 145-172.

[5] PRASAD, J., KUMAR, B.: *Analysis of SIP and Realization of Advanced IP-PBX Features*, Proc. 3rd Intern. conference on electronics computer technology, vol. 6, no. 5942085, 2011, 218-222.
[6] SENTHIL, S. K., DHIVYALEKSHMI, B. S., PREETHI, S., PERUMALRAJA, R.: PBX Implementation in Lan Using Asterisk Open Source Software, *Intern. J. of Applied Engineering Research*, vol. 10, no. 55, 2015, 66-69.
[7] ALAM, M., BOSE, S., RAHMAN, M., AL-MUMIN, M.: *Small Office Pbx Using Voice Over Internet Protocol (VOIP)*, Intern. conference Advanced communication technology, No. 4195481, 2007, 1618-1622.
[8] SAFARIK, J., VOZNAK, M., REZAC, F., MACURA, L.: IP Telephony Server Emulation for Monitoring and Analysis of Malicious Activity in VOIP Network, *Communications - Scientific Letters of the University of Zilina*, vol. 15, No. 2a, 2013, 191-196.
[9] CHI, R.: Intrusion Detection System Based on Snort, Lecture Notes in Electrical Engineering, *272 LNEE*, vol. 3, 2014, 657-664.
[10] POCTA, P., KORTIS, P., VACULIK, M.: Impact of Background Traffic on Speech Quality in VOWLAN, *Advances in Multimedia*, vol. 2007, 2007, No. 57423.
[11] MRVOVA, M., POCTA, P.: A Quality Estimation of Synthesized Speech Transmitted over IP Networks, *Communications - Scientific Letters of the University of Zilina*, vol. 16, No. 1, 2014, 121-126.
[12] VOZNAK, M., TOMALA, K., VYCHODIL, J., SLACHTA, J.: Advanced Concept of Voice Communication Server on Embedded Platform, *Przeglad Elektrotechniczny*, vol. 89, No. 2b, 2013, 228-233.
[13] REZAC, F., ROZHON, J., SLACHTA, J., VOZNAK, M.: Speech Quality Measurement in IP Telephony Networks by Using The Modular Probes, *Communications in Computer and Information Science*, vol. 522, 2015, 172-181.
[14] Project beesip, repository with source codes, available on url: https://homeproj.cesnet.cz/ projects/besip/wiki/ download.