CESNET Technical Report 8/2013

Enhanced management services in BESIP project (Bright Embedded Solution for IP telephony)

Jiří Šlachta, Lukáš Macura, Jan Rozhon, Fillip Řezáč, Jakub Šafařík and Miroslav Vozňák

CESNET z.s.p.o., Zikova 4, Praha, Czech Republic

{slachta, macura, rozhon, rezac, safarik, voznak}@cesnet.cz

Received 15.12.2013

Abstract

The article deals with recent advances in the development of multiplatform communication server based on OpenWrt Linux distribution. The solution is built as a universal open source modular system and the server has been developing within the framework of a BESIP project (Bright Embedded Solution for IP Telephony) since May 2011. The paper deals with overall concept of the developed solution, explains the architecture of BESIP project, describes all individual modules, presents the current state and the future intents.

Keywords: OpenWrt, SIP server, BESIP, Speech quality, VoIP security, NETCONF.

1 Introduction

The BESIP aims to become VoIP PBX system available for anybody; the users need not know complex software features and hidden internals of VoIP software. BESIP offers the prepared solution with integrated key components, the entire system is distributed as an image or individual packages can be installed from SVN and users do not care about dependencies, they just configure VoIP system which works. Every software solution includes own configuration and management. BESIP aims to be scalable solution with security and unified configuration in mind [1].

Several open-source applications were adopted and implemented into developed modules, however within the implementation many modifications were required, especially in the core module (OpenWrt) due to complicated porting of applications into OpenWrt Buildroot. Our patches were verified and accepted by OpenWrt community. The speech quality monitoring tool was developed from scratch and implemented in Java. BESIP can run on embedded devices as well as on high performance devices. It requires at least 32 MB RAM and runs on the majority of OpenWrt supported devices.

The way how to build an embedded IP-PBX (IP Private Branch exchange) running on an embedded Linux kernel is showed in [2], [3].

2 Selection of suitable Platform

The most important step which had to be done, was choosing right software distribution/platform. There was an idea to modify Debian distribution; this is probably the easiest way for developers. Debian includes many ports and packages which are available for many software services but Debian is not suitable for embedding. A modification of Debian, in order to be easily embedded into small device with read-only flash, is really a difficult task and the expected results of such work cannot lead to a source distribution.

Next solution was adopting some low-level distribution for embedding. There are several possibilities like FreeWrt, OpenWrt, DebWrt etc. After discussion and projects observations, we decided to select OpenWrt as a primary platform because of relatively well documentation, cooperation with community and its developers and its unified system of creating packages for it, even if the procedure of porting packages is not that easy (especially for packages without configure script and with special compilation procedures). After research and project discussion we made those decisions: OpenWrt for good scalability and simple embedding, Kamailio for reliability and high availability as SIP Proxy, Asterisk as B2BUA (Back-to-Back User Agent), YUMA as NETCONF server and OpenWrt UCI as configuration backend.

3 Build System

Before we describe the concept of BESIP system, it is necessary to introduce the BESIP build system which makes automation of creation system images much easier. As said above, BESIP 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 targeted device. Cross-compilation toolchain is compiled by host compilation system which is provided by any GNU/Linux distribution.

In the beginning of BESIP 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 pure, new system 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.

BESIP buildsystem is a set of scripts, Makefiles and definition files that makes an easy interface to OpenWrt buildroot. We can consider the main Makefile to be as a core of the BESIP 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.

For example following commands builds whole BESIP images and pushes them into the specified repository via SCP protocol or via RSYNC application.

```
# make all
# make download
```

For functional image testing we can use KVM or QEMU to virtualize or emulate environment for testing root filesystem images.

```
# make test-qemu
# make test-kvm
```

On the top of the core Makefile is autobuild script written. This script calls all atomic operations within more complex parametrized 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.

The following simplified example shows, how to create target file.

```
TARGET_CPU=x86
OWRT_NAME=trunk
TARGET_NAME=virtual_$(BESIP_VERSION)-owrt_$(OWRT_NAME)
TARGET_QEMU=i386
TARGET_QEMU_OPTS=-m 512
OWRT_IMG_DISK_NAME=openwrt-$(TARGET_CPU)-generic-combined-squashfs.img
BESIP_PACKAGES= gnugk=y suricata=y
OWRT_CONFIG_SET += TOOLCHAINOPTS=y TARGET_ROOTFS_ISO=y
EMBEDDED_MODULES += SATA_AHCI VMXNET3
```

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

```
#./autobuild.sh build virtual-x86-trunk
```

Once we can call the autobuild.sh script for specific target, we can put the specific script call into /etc/crontab and build platform images on regular basis.

4 Architecture and Technology

The BESIP architecture is depicted in Figure 1, it is created entirely from open source parts. This was main presumption for project management and developing. There are four basic modules: Core, Security, Monitoring and PBX. Core is divided into following parts:

- OpenWrt as build platform;
- NETCONF For administration of entire system, YUMA Free Toolkit was adopted;
- Web GUI for user-friendly configuration;

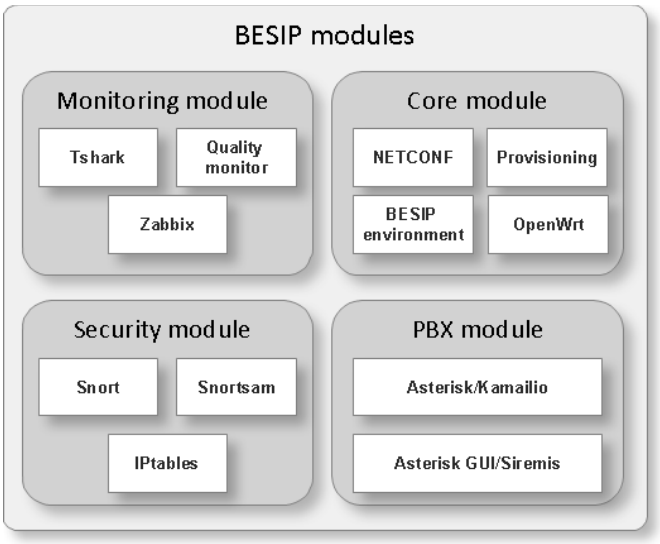

Figure 1. BESIP architecture.

The security module is based on SNORT, SNORTSam and iptables [4]. In addition to this, the Kamailio ratelimit and pike module is used for defending attacks. The monitoring module exploits a tshark package and our java code which interprets its results and gives information about particular speech quality. The Zabbix agent is used to report basic states of entire system and finally the PBX module is made from Kamailio in conjunction with Asterisk.

5 Core Module

BESIP core module consists of two mandatory parts. The first part is Configuration submodule that uses NETCONF protocol as its mandatory part. The second part is a module that provides whole BESIP environment. This module is a complete set of scripts that prepares OpenWrt environment during the "preinit" stage (while the booted device is running at first time) and provides simple interface for other existing BESIP modules.

I. BESIP Environment Submodule

BESIP environment handles several tasks which has to be performed at several stages in OpenWrt operating system. After the BESIP firmware image is built at this stage the operating system behaves as clean operating system with installed dependencies required by BESIP package and its submodules. On the first boot the init script is performed and it waits until the overlay filesystem is mounted. When the filesystem is mounted the "first_boot" procedure is performed. This procedure incorporates the initial setup of the system and preparation configuration files. The main advantage of this procedure is the applicability to any existing setup of OpenWrt system.

Another important part of BESIP system are helper functions that are included into a binary helper application. This application does the following procedures:

- Importing SIP gateways from SIP gateways databases,
- setting up a dial plan for internal SIP engine,
- resetting system image into factory defaults,
- performs system upgrade,
- controls internal security module,
- collects information for crash reporting to be used for debugging.

II. Provisioning Submodule

The impetus for development of provisioning tool arose during the period when firmware images created by BESIP 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 BESIP firmware image. The creation of such tool bring a question how should the target device should fetch and apply its configuration.

Before we discuss the implementation of provisioning sub module we should take a think about several questions. Those might be: "How the target device should receive the URI to server with device configuration? How should the device configuration be distributed? How should this configuration be applied? What should be distributed?" All those questions and issues were resolved in the UCI provisioning client that we developed.

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. How can this be useful? This information can be used for protocols which translates one kind of information to another. For this purpose we can use DNS protocol and its TXT records. The first question we can answered this way. The target configuration could be stored on a server within an URI in a variable from TXT record which is obtained from static URL provided by BESIP 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/"
```

The second and third question can be answered simply. If a device knows where to obtain configuration from, then the device can obtain configuration from ordinary transport protocols specified in provisioning URI. The benefits that BESIP 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 wget and immediately imported into UCI.

The client side of UCI provisioning currently has following stages:

1. Waiting for Ethernet adapter to be ready,
2. obtain UCI provisioning URI from specified protocols,
3. obtain UCI generic configuration,
4. obtain target device specific UCI configuration,
5. apply received configuration and perform actions associated with specific configuration.

The third stage appends to UCI provisioning URI static prefix to obtain configuration and the fourth stage appends MAC addresses to differentiate target device specific configuration.

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 sequence diagram depicted in Figure 2 to see how the UCI provisioning works.

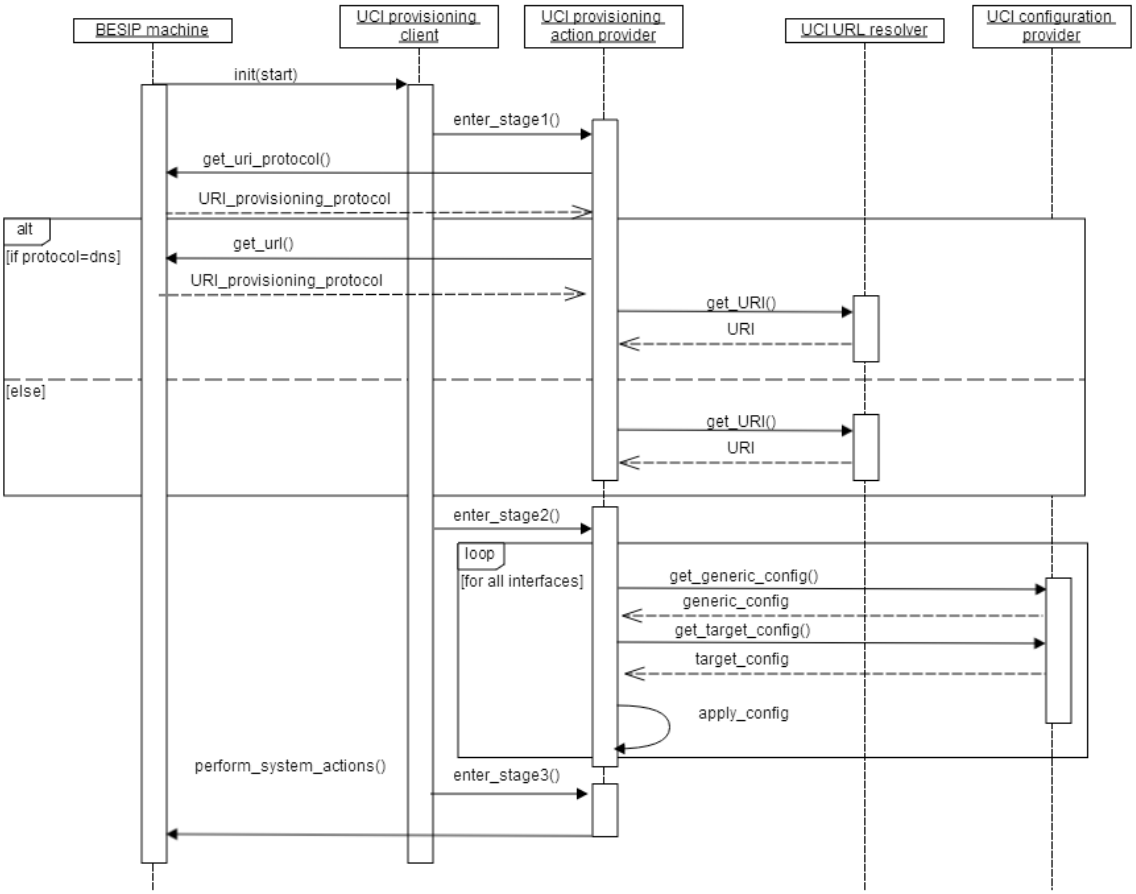

Figure 2. Sequence diagram of UCI provisioning client

III. 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 [5]. 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).

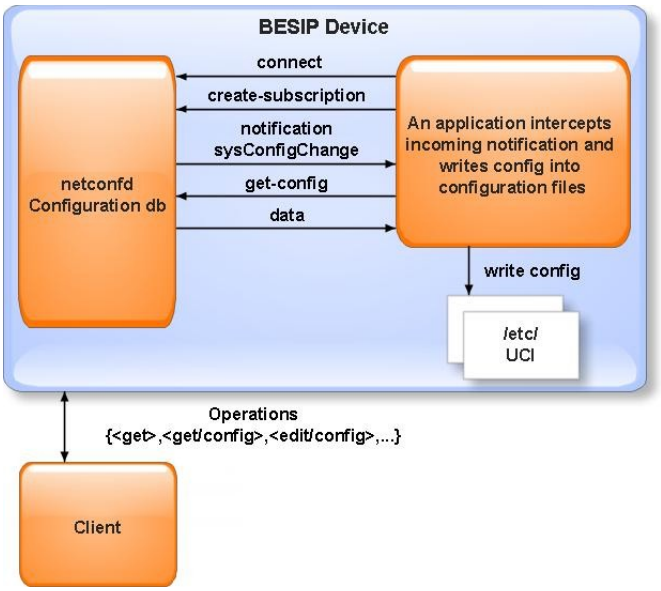

Figure 3. General concept of NETCONF communication in BESIP.

The structure of configuration data on NETCONF server in YUMA package (netconfd) is specified by the 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. Yuma is a package which provides 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.

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. We decided to provide glue between NETCONF and UCI as is depicted in Figure 3.

Current UCI implementation does not provide data validation, therefore for the data validation we used firmly defined YANG structures and mapped existing UCI configuration into YANG modules, where each module represents specific UCI configuration file stored in /etc/config. Current implementation, which is depicted in Figure 6, replicates configuration that is initiated by subsystem inotify into netconf database and on the other opposite way it transforms configurations asynchronously via subscriptions from NETCONF server.

Let's take a look at the simplified examples below, where is specified a list container which consists of leafs of items in YANG syntax, UCI syntax and configuration data in XML stored in NETCONF running configuration.

```
list interface {
  key name;
  leaf ifname { ... }
  leaf ipaddr {
```

```
        type inet:ip-address;
      }
      leaf netmask { ... }
    }
```

Figure 4. Simplified YANG syntax for UCI configuration file.

```
config interface lan
   option ifname   eth0
   option ipaddr   192.168.1.1
   option netmask  255.255.255.0
```

Figure 5. UCI syntax that is mapped by YANG module.

```
<interface name="lan">
   <ifname>eth0</ifname>
   <ipaddr>192.168.1.1</ipaddr>
   <netmask>255.255.255.0</netmask>
</interface>
```

Figure 6. Configuration data.

Today, many systems are configurable using web, ssh or telnet and each of them offers its own semantics and configuration file. BESIP project aims to change this situation, using NETCONF as defined communication and management protocol, configuration independent syntax will be available on all modules. At first stage of project, applications and libraries had been ported; afterwards we focused on implementation of NETCONF, UCI, PBX, Security and monitoring modules.

See Figure 7 to understand configuration data flow which has been defined in BESIP.

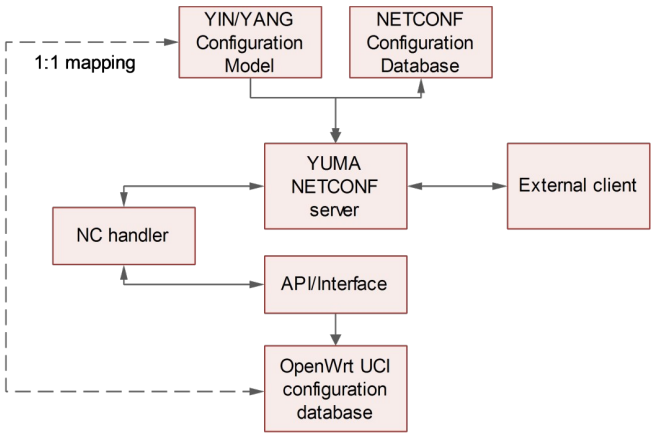

Figure 7. NETCONF usage in BESIP.

IV. Definition of Data Structure for Configuration Module

YANG module for BESIP 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 PBX Module

The PBX module is key part of the BESIP 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 functions. Kamailio is used for the proxying SIP requests, the traffic normalization and for the security [6]. 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 BESIP is able to adapt to the users requirements. 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]. The BESIP offers users an option to choose how system will work. From this reason, the BESIP includes both Kamailio and Asterisk. Today, only one of these engines can be configured but in future, both engines will work together and will be configured by common NETCONF server. Kamailio will route requests even if Asterisk will be out of order, only advanced PBX functions will be unavailable in such situation.

7 Security Module

Security module is very important part of BESIP and all the time, it was considered to make the developed system as secure as possible. Next to this, entire system has to be fault-tolerant, monitored and protected from attacks. It means that if the device is under attack, only attacker has to be blocked, not entire system or other users. If there is some security incident, BESIP immediately solves the situation and notifies this event in detailed report to the administrator.

The attack are recognized and processed by SNORT rules, the source IP address is automatically sent into firewall by SNORTSam and the intruder's IP is blocked. This is very flexible, reliable and effective implementation. 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 client sends REGISTER with proper credentials, it is not obviously security attack but the client attempt to register again and again, with every registration requires computing sources at SIP REGISTRAR server. Such attempts can be denoted and blocked for a time interval. Administrators can use Zabbix agent inside BESIP to gather all information directly into their monitoring system. The monitoring is very important part of the security module and BESIP team was already focused on the issue in early design [8]. Partially, BESIP is resistant to some kind of DoS attacks. It depends on hardware used. If hardware is strong enough to detect some security incidents on application level, the source IP is immediately dropped. But

for weak hardware it can be serious problem. In such case, it is better to stop DoS attacks before it reaches BESIP. For example, SNORT on a dedicated machine will be much more flexible than if is an integral part of VoIP system. Therefore, we recommend using an external IPS system to make VoIP service robust and secure. Nevertheless BESIP includes own IPS/IDS system [9], [10].

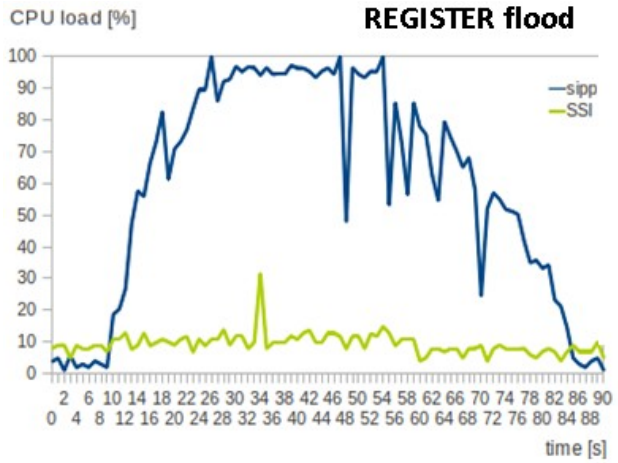

Figure 8. Attack effectiveness based on REGISTER flood.

The features of our security module were verified in test-bed and results are depicted in Figure 8 and 9. The CPU load was monitored during trivial SIP attacks. The line SSI (Snort, SnortSam, IPtables) represents the response in case of active security module in BESIP whereas next dependencies were measured without SSI. There were emulated only two types of DoS attacks, namely REGISTER flood and INVITE flood. In order to generate these attacks, we used sipp generator and in case of INVITE also inviteflood tool. The dependencies in both figures clearly prove the ability of security module to mitigate the performed attacks.

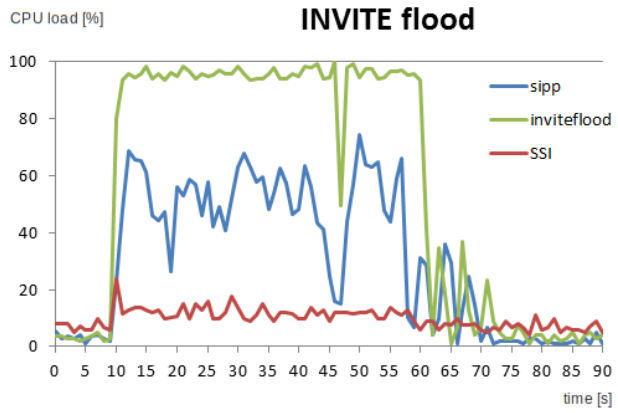

Figure 9. Attack effectiveness based on INVITE flood.

8 Monitoring Module

The overall solution of the monitoring system consists of several different open source components and also of the part that was directly developed for this purpose to meet the defined requirements. System structure is depicted in Figure 10.

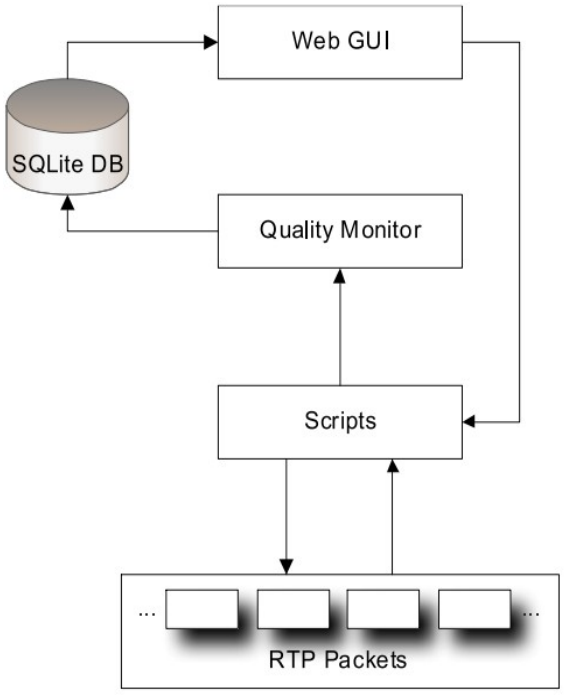

Figure 10. Overview of the logical structure of VQM.

We use a computational E-model [11], simplified for the purpose of the implementation. The computation can be split into several elements and is expressed by equation (1). The used values were adopted from recommendation ITU-T G.107 where the default values are listed and finally we need to express only two unknown parameters I_d and $I_{e\text{-}eff}$ which are described in equations (3) and (4).

$$R = 94.7688 - 1.4136 - I_d - I_{e-eff} + 0 \tag{1}$$

The simplified E-model only takes into account the effects from codec, packet loss and end-to-end delay [14]. The situation is depicted in Figure 11.

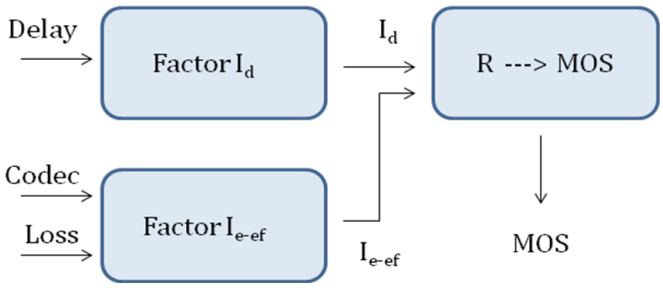

Figure 11. E-model in a simplified version.

The computed R is converted to MOS value. For this purpose, relation (2) was applied [11].

$$MOS = 1 \quad \text{for} \quad R < 6.5 \quad \text{and} \quad MOS = 4.5 \quad \text{for} \quad R > 100$$
$$MOS = 1 + 0.035 \cdot R + R \cdot (R - 60) \cdot (100 - R) \cdot 7 \cdot 10^{-6} \quad \text{for} \quad 6.5 \le R \le 100 \tag{2}$$

Factor I_d represents all impairments which are caused by different combinations of delays. We applied a linear regression to results gained in AT&T laboratories [12] and derived relation (3) which provides accurate results, with regression quality r=0.99 ranging from 0 to 400 ms.

$$I_d = \begin{cases} 0.0267 \cdot T & T < 175\,ms \\ 0.1194 \cdot T - 15.876 & 175\,ms < T < 400\,ms \end{cases} \quad (3)$$

Packet-loss dependent effective equipment impairment factor $I_{e\text{-}eff}$ is derived using the codec-specific value for the equipment impairment factor at zero packet-loss I_e and packet-loss robustness factor B_{pl}, both listed in Appendix I of ITU-T G.113 [13]. With packet-loss probability P_{pl}, $I_{e\text{-}eff}$ is calculated using the equation (4).

$$I_{e-eff} = I_e + (95 - I_e) \cdot \frac{P_{pl}}{\frac{P_{pl}}{BurstR} + B_{pl}} \quad (4)$$

BurstR is the so-called burst ratio, defined as the ratio between "Average length of observed bursts in an arrival sequence" and "Average length of bursts expected for the network under a random loss".

The system itself consists of three logical components, which are – web interface that serves the administrators (Web GUI), part of the script (Scripts) that controls the obtaining the information necessary to compute the speech quality in the simplified E-model, as is depicted in Figure 10. Last component is part of the Quality Monitor, which contains the logic for calculation itself and performs processing of data obtained by scripts. The SQLite3 database is used to store the results.

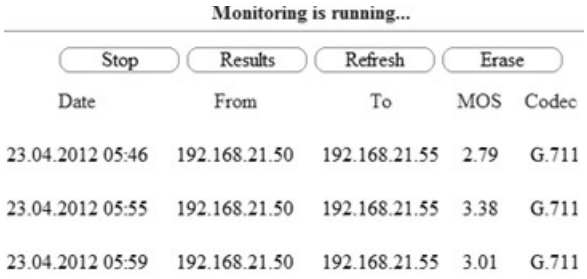

Figure 12. Sample of web interface of monitoring speech quality.

The developed application offers the comfort of management in a web application, the developed interface aggregates required functions. Web interface is the main part of user interaction with a monitoring tool. Monitoring tool is turned off in the default configuration and can be enabled using the intuitive main interface of BESIP any time. This part of the monitoring tools is also used as a mean to display the measured and computed results. Structure of the presented data is as follows: Time, Source IP, Destination IP, MOS and used Codec. An example of user interface is shown in Figure 12.

9 Performance Evaluation

In order to evaluate the performance of the BESIP system, we tested our BESIP running on low-end HW platform containing x86 Intel Atom D410, RAM 1GB/677MHz and 16GB SSD, see Figure 13.

Figure 13. Suitable HW platform

The performance test was carried in accordance with new standard RFC 6076 with our own benchmarking tool [15]. We found out that the tested platform with BESIP:

- is able to handle 256 registrations per second;
- is stable up to 6000 registrations;
- is able to process 90 simultaneously running calls.

When the limit of 90 simultaneous running calls was exceeded, we observed unsuccessful calls and a rapidly increasing SRD (Session request Delay); for example 75 % rate of successfully established sessions in case of 150 simultaneous calls and with SRD over 60 ms.

10 Conclusion

As we have mentioned, BESIP consists of several components which are distributed under GPL as an open-source solution. A few of them have been fully adopted such as the components in Security and PBX modules, some of them modified, concerning the CORE module and finally we have developed own tool for Speech quality assessment. The contribution of our work is not only few hundreds of hours spent on the development, on the coding BESIP system, we bring a new idea of the unified configuration management, with unified CLI syntax which enables to configure different systems, Asterisk and Kamailio in our case. We perceive that we need to solve a lot of issues, individual packages are working and after several pre-releases, the version 1.2 was released in April 2013.

BESIP is distributed as a functional image for several platforms, mainly for x86 platform which is also possible to run it on any virtualization x86 software. Configuration is available through web-browser or SSH client. Today, there is a trunk version in SVN which is actively developed and individual improvements are included in next subversions. After testing, version 2.0 will be released; new release 2.0 will be based completely on NETCONF with one API to configure entire system. Next to this, CLI syntax is developed and will be connected to NETCONF. 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 simple task. Users will modify only one configuration file to manage entire box. Project pages are available at [16], binary images from auto-build system can be downloaded from [17] and source codes can be checked out via SVN from the same page as well [17].

11 Acknowledgment

This work has been supported by the Ministry of Education of the Czech Republic within the project LM2010005.

12 References

[1] M. Voznak, F. Rezac, "Threats to voice over IP communications systems," WSEAS Transactions on Computers, Volume 9, Issue 11, 2010, pp. 1348-1358.

[2] F. Abid, N. Izeboudjen, M. Bakiri, S. Titri, F. Louiz, D. Lazib, "Embedded implementation of an IP-PBX /VoIP gateway," 24th International Conference on Microelectronics, December 2012, IEEE, Article number6471377.

[3] N. Titri, F. Louiz, M. Bakiri, F. Abid, D. Lazib, L. Rekab, "Opencores /Opensource Based Embedded System-on-Chip Platform for Voice over Internet," INTECH: VOIP Technologies, pp. 145-172.

[4] J. Safarik, F. Rezac, M. Voznak, Monitoring of Malicious Traffic in IP Telephony Infrastructure, Technical Report, 10p., December 2012.

[5] R.C. Joshi, A. Sardana, Honeypots: A New Paradigm to Information Security, Science Publishers,2011.

[6] M. Voznak, J. Safarik, "DoS attacks targeting SIP server and improvements of robustness, " International Journal of Mathematics and Computers in Simulation, Volume 6, Issue 1, 2012, pp. 177-184.

[7] J. K. Prasad, B. A. Kumar, "Analysis of SIP and realization of advanced IP-PBX features," 3rd International Conference on Electronics Computer Technology, Volume 6, 2011, IEEE Article number5942085, pp. 218-222.

[8] M. Voznak, K. Tomala, J. Vychodil, J. Slachta, "Advanced concept of voice communication server on embedded platform, " Przeglad Elektrotechniczny, Volume 89, Issue 2 B, 2013, pp. 228-233.

[9] D. Endler, M. Collier, Hacking Exposed VoIP, McGraw-Hill Osborne Media, 2009.

[10] D. Sisalem, J. Kuthan, T.S. Elhert, F. Fraunhofer, "Denial of Service Attacks Targeting SIP VoIP Infrastructure: Attack Scenarios and Prevention Mechanisms." IEEE Network, 2006.

[11] ITU-T, "The E-model: a computational model for use in transmission planning," Recommendation ITU-T G.107. International Telecommunication Union, 2011.

[12] G. Cole, H. Rosenbluth, "Voice over IP performance monitoring," ACM SIGCOMM Computer Communication, Volume 31, Issue 2, 2001, pp. 9-24.

[13] ITU-T, "Transmission impairments due to speech processing," Recommendation ITU-T G.113. International Telecommunication Union, 2007.

[14] M. Kavacky, E. Chromy, L. Krulikovska, P. Pavlovic, "Quality of Service Issues for Multiservice IP Networks, " In Proc. SIGMAP 2009 International Conference on Signal Processing and Multimedia Applications, Milan, Italy, July 2009, pp. 185 – 188.

[15] M. Voznak, J. Rozhon, "Approach to stress tests in SIP environment based on marginal analysis," Telecommunication Systems, Volume 52, Issue 3, March 2013, Pages 1583-1593.

[16] Management of BESIP Project, LipTel Team, 2013. URL https://besip.cesnet.cz

[17] Project BESIP, URL https://homeproj.cesnet.cz/projects/besip/wiki/Download