Technical Report 2/2016

Automatically Provisioned Embedded Communications System Based on OpenWRT Platform

Filip Řezáč, Jan Rozhon, Jakub Šafařík, Miroslav Vozňák and Erik Grešák

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

filip@cesnet.cz, rozhon@cesnet.cz, safarik@cesnet.cz, voznak@cesnet.cz, gresak@cesnet.cz

Received 16. 11. 2016

Abstract

The technical report deals with a design of an automatically provisioned embedded communication system. Through the years of an active development a highly advanced platform has been created. This platform, called SeeSIP, is meant for the embedded network devices, and allows them to act as telephony exchanges, secured access points, voice quality measuring probes, honeypots, etc. Since it is a common situation that there are many such devices throughout the network, the difficulties related to distributed management and provisioning arise. As the key feature of the SeeSIP 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 SeeSIP based hardware to cover the needs of small to medium businesses in vast range of services.

Keywords: VoIP, Configuration provisioning, OpenWRT, Embedded Systems, Use Cases, SIP

1 Introduction

In order to reduce costs for maintaining communication systems, both physical devices and services running on them, many companies make 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 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. 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 on 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. The given name BESIP has had to be changed to SeeSIP (The Smart Efficient Embedded Solution for IP Telephony) because registration of our BESIP trademark was rejected by the Czech Industrial Property Office due to the same existing trademark in field of public transport. The entire project is documented on the website of the CESNET association [3].

2 Platform Architecture

One of the biggest challenges during SeeSIP 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 SeeSIP 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.

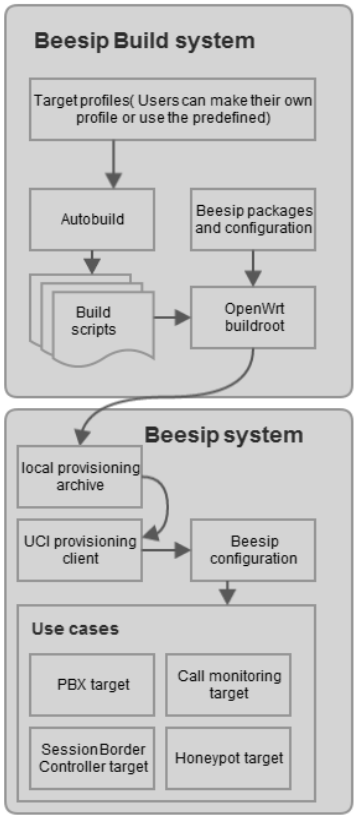

Fig. 1. Architecture of SeeSIP system.

A simplified view on SeeSIP 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 SeeSIP is the OpenWRT Linux distribution which uses packages that provides desired functionality. In this case to provide modules from packages that serves as PBX, call quality monitoring probe, SBC (Session Border Controller) target and Honeypot agent.

The architecture of SeeSIP 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 [4].

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. 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 [5]. 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.

3 Build System

Before the concept of SeeSIP system is described, it is necessary to introduce the build system which reduces the building procedure into one script call. As said above, SeeSIP 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.

At the beginning of SeeSIP 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.

SeeSIP 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 SeeSIP 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 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=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.

4 Core Module

The role of the Core module is to provide a glue among all services that served by all SeeSIP use

cases. The most important part of the Core module is the SeeSIP shell library that provides functions for all utilities and scripts used by SeeSIP 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 SeeSIP use cases running and operational. Also, the role of this module is to switch any existing OpenWRT environment to SeeSIP environment while the device is booted the first time or the SeeSIP environment is used and ran the first time.

4.1 SeeSIP Environment

SeeSIP environment handles tasks which has to be performed at several stages in OpenWRT operating system. After the SeeSIP firmware image is built at this stage the operating system behaves as clean operating system with installed dependencies required by SeeSIP 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 report 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 SeeSIP 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.

4.2 Provisioning Client

The impetus for development of provisioning tool arose during the period when firmware images created by SeeSIP 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 SeeSIP firmware image. The creation of such tool brings a question how should the target device 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 SeeSIP 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 SeeSIP 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 SeeSIP 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.

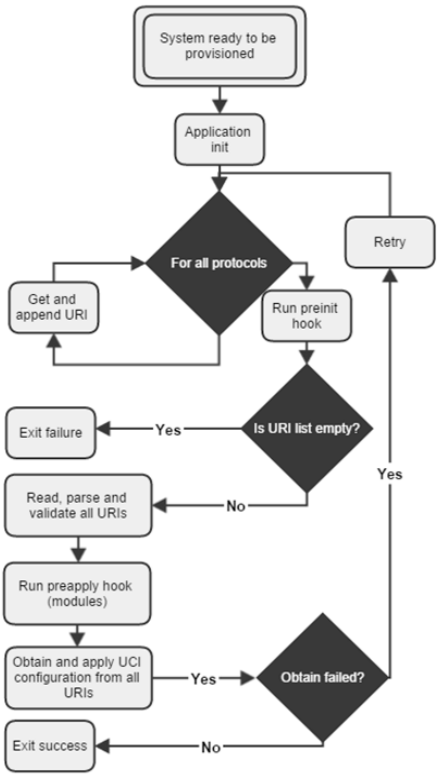

Fig. 2. Flow diagram of UCI provisioning client.

See flow diagram depicted in Fig. 2 to see how the UCI provisioning works and its description is following:

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).

5 PBX Use Case

The PBX module is a key part of the SeeSIP 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 SeeSIP is able to adapt to the users requirements [6]. 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.

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 [7], we keep information about academic network infrastructure (more than 50 VoIP Gateways and PBXs behind them with cca 500 thousand phones), the project SeeSIP 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 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 SeeSIP 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. The whole idea of PBX use case features can be found in the created datasheet [8].

Mutual integration of the Asterisk PBX and SIP proxy Kamailio on the SeeSIP platform with functionalities needed for operating as an SBC entity would allow the SeeSIP platform to be used as a telephony security node for the remote locations and locations without the high hardware requirements. On 2017 we are planning to create a cost effective and easy to deploy security measure for the telephony services in environments with low traffic, where commercially offered security solutions do not make sense from the economy point of view.

6 Security Use Case

The detection and classification of SIP attack data consist of two parts. First is the collection of attacks, second its classification. We propose the solution of both problems. The distributed network of nodes collects attack information and the centralized Beekeeper server serve as a data store and classifier. Because the classifier uses a modular design, it detects variable scale of attacks, without a need to focus on a specific kind of attack. This way we can achieve a reliable source of information and verbatim attack classifier. The architecture was closely described in a various papers [9, 10,11], however a quick review of the design follows.

Each node runs on prepared hardware or virtual image using the SeeSIP preconfigured software use case. The core of node is a Dionaea honeypot for emulation of SIP PBX. All traffic is filtered through the firewall, and only specific SIP traffic plus ICMP messages are allowed. Each node uses public IP address and is accessible to anyone. The source of attack data is Dionaea honeypot and dump of SIP traffic from Tcpdump packet sniffer. All attack data from nodes is periodically sent directly to the server's REST API via secure HTTPS channel. The detector node is ready for deploy with almost no configuration needed. The network administrator could deploy node in a network and use web user interface from the centralized server to access information about attacks from his concrete node. All attacks are classified automatically, without the need for human interaction and deep understanding of SIP related attacks.

The Beekeeper server is responsible for several tasks. The main task of the server is a collection of data received from nodes and its classification. Beekeeper stores all attack information in the raw format (as uploaded from the node) and in aggregated form. The Neural network algorithm then classifies aggregated attacks to one of 8 predefined attack classes. The server also automatically obtains geolocation information for each attack IP address, which brings additional information value to collected data. The Beekeeper also monitors and authenticates each node. Only authorized nodes can upload data to Beekeeper database.

Server in a practical deployment includes two neural network whose input consists of following input vector parameters: The number of connections in the attack, the number of all SIP messages, the number of messages from the subscriber, the ratio of the number of SIP messages on the connection, the duration of the attack. As noted above, subsequent classification divides attacks into 8 most frequent types of threat, namely:

- call_test - The attacker attempts to make calls over a victim's SIP server, detection with a SIP INVITE messages.
- opt_scan - Scanning of active SIP accounts with a large number of OPTIONS messages.
- opt_test - Scanning of an active SIP server using a small amount of OPTIONS messages.
- reg&call - The attacker tried to register and establish a call using the SIP REGISTER and INVITE messages.
- reg_attempt - Attempt to a legitimate registration or registration simple test.
- reg_test -The attacker was trying to obtain registration on a given server by using a greater number of SIP REGISTER messages.
- reg_test_high - Spamming attack or DoS attack using the SIP REGISTER messages.
- ukwSIP/noSIP - This is an attack by unknown SIP messages (other than RFC 3261) or the attack that was directed to the SIP port, but does not contain SIP messages.

The whole system is accessible through a web interface for intuitive and apt presentation of gathered data. Most parts of the system interact with the WUI – like node monitoring, node authorization or analytics module. The system scheme is shown in the Fig. 3.

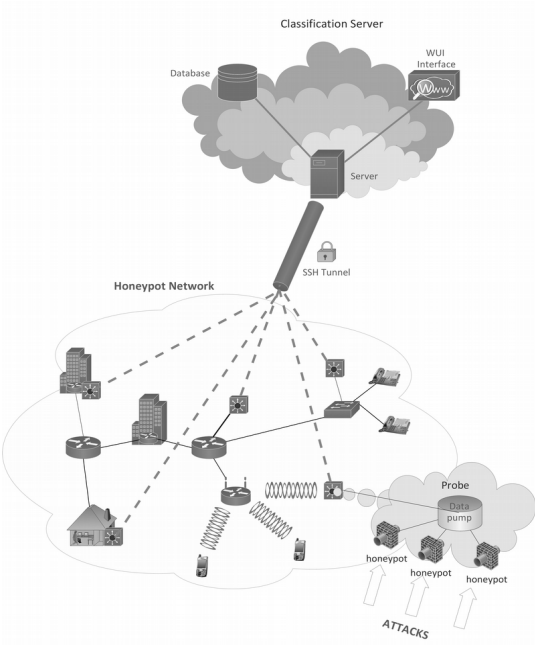

Fig. 3. The concept of the designed detection system.

7 Call Monitoring Use Case

Due to the nature of the SeeSIP architecture, which is focused mainly on the embedded and low-profile devices, SeeSIP can be deployed as the network monitoring probe as well. For these purposes, libraries allowing continuous monitoring of speech quality have been developed and incorporated into the SeeSIP software, thus allowing to create a powerful and reliable solution for the distributed speech quality monitoring in vast networks and inter-organization communication links. We called it Callmon. Although the speech quality monitoring is not the only possible implementation of the SeeSIP and the more general monitoring systems can be developed and implemented easily, the speech quality measurement and monitoring is of high importance due to the PBX module and SeeSIP's intended use. The monitoring module takes the advantage of the Asterisk PBX, which is the part of the SeeSIP's PBX use case, and is designed to work as the network probe. For this functionality only low-complexity operations are necessary and therefore it can be used even with the low end hardware.

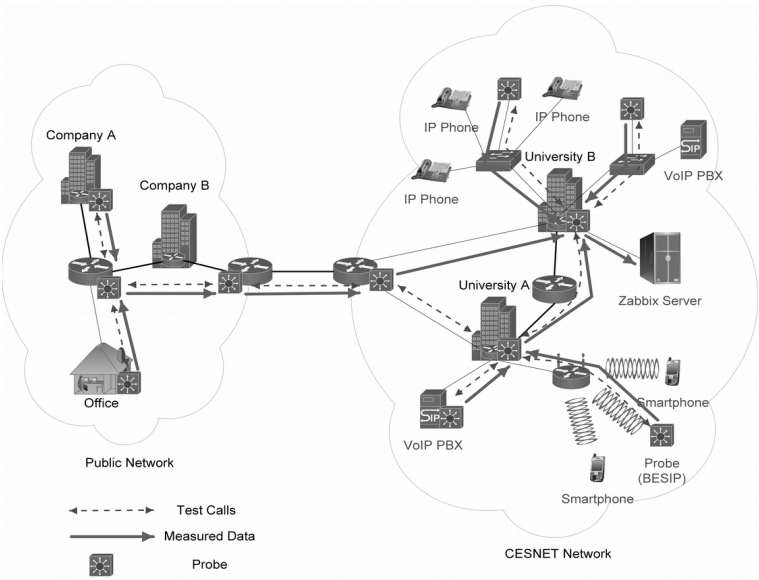

Fig. 4. The monitoring system use case architecture.

Before starting the system design, the research was conducted on the topic of what objective speech quality measurement model is ideal for our purpose and what management model should be used in the implementation. Based on the results that have been published [12, 13], we have made a decision that a centralized model is the most appropriate. It works on the principle of autonomous probes which are placed at key nodes in the network and periodically carry out test calls to adjacent probes. Once the test call is made, degraded samples are sent to the server, where are compared with the original signal using PESQ model.

Server side is based on the Zabbix monitoring tool, which is enhanced by our algorithms for data collection, processing and evaluation. Server also allows to monitor individual probes and respond to potential failure. An example of the system design is shown in Fig. 4. Once the data is processed on the server, the visualization is presented in the form of a map where peaks represent the probes and the measurement paths are shown in the form of evaluated edges. The user can easily monitor the quality of speech on different call routes and on the basis of this information he/she can optimize network flow, or edit voice codecs for VoIP traffic.

From the perspective of the technologies that were used in the implementation, it is again necessary to divide a whole system into two parts. Server uses a virtualized platform running on Zabbix monitoring system and it was necessary to implement a communication interface using the Python programming language. Detailed description of the interface itself and ongoing communication is then given in the following chapter. Probe side is based as the SeeSIP use case. However, the most important parts of the probe are classes for communication with the server, applications for control periodic processes and a test call generator itself. In addition, the probe includes classes for evaluation of speech quality and communication interface with other probes. As was already mentioned, the probes may be deployed in a network in many ways, allowing rapid expansion of the system and also gain a more accurate measurement.

After the first start of the probe and setting up the periodic cycles via Cron, the Listener class is executed. Then it waits for commands from the server so the list of the probes can be updated in case of changes. Once the probe receives the list of the neighboring probes from the server through JSON interface, the AsteriskCallList class creates and operates a call list, which is then used for generating test calls. These are actually controlled by AstConnection class that communicates with AMI (Asterisk Manager Interface) interface of SIP PBX Asterisk which is the part of each probe. The server side also uses scripts for database management and classes designed to run PESQ algorithm that compares the degraded sample with a reference one to determine the speech quality. Last mentioned class, which makes the interface between the running systém and classes containing particular functions is called Controller. It initializes various objects and allows the interaction and interconnection.

The initiation of calls is made by AstConnection class, which first started the test calls through the AMI interface of Asterisk using the reference sample and then it stores a degraded samples in a .wav format. Degraded files are stored in the form: *MAC_caller_probe-MAC_called_probe-timestamp.wav* (e.g.,000c29b28f9a-000c298f419d-20140116092039.wav), so they can be easily founded. These samples are then backed up and transferred using Rsync to a directory on Zabbix server. The list of files on the probe that has been transmitted is gradually crawled and is added to queue on the server using SSH for further analysis. This queue is stored in a MySQL database and script comparing the reference voice sample using PESQ algorithm is periodically triggered. Returned MOS values are then inserted into Zabbix application and are further visualized. The values are stored for each probe using the following string: *MOS-MAC_address_of_the_target_probe* (eg. 38-000c298f419d). To avoid the re-analysis at the next script startup, the file is then removed from the directory. Sequence diagram of communication between the probes and the server is shown in Fig. 5.

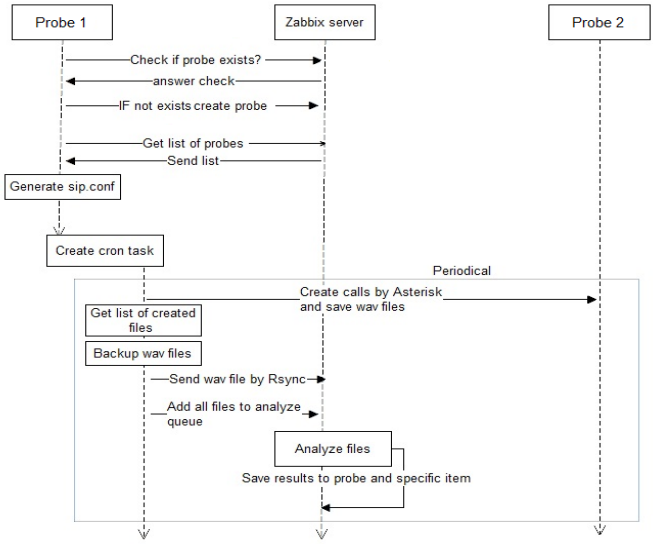

Fig. 5. Communication between Probes and Zabbix Server..

Based on the algorithm above, test calls between all neighboring probes (also called full-mesh) should be realized. The question is, which probe starts with testing first and how to perform all the test calls without duplications. For this purpose, the method has been developed and is described below.

First, the average number of connections per node is determined. In addition, the method creates an average number of connections for each probe. The first member of the list is then moved to the end. The method is shown below, where \textit{n} is the number of probes:

```
n = len(self.inp)
no_of_conns = n * (n - 1) / 2
conns_per_node = no_of_conns // n
rem_conns = no_of_conns % n
```

The visualization script was also implemented and it allows to generate a graphical map of probes from the database of results. The map is implemented using a graph where the nodes represent individual probes and the edges are connections between them. Edge weight shows the current MOS value measured between the respective probes in both directions (if both numbers are available). The graph is generated using the Zabbix server interface and is, therefore, available directly through the web application.

With the end of 2016, we have launched a website [14] which aims to promote Callmon use case for members of the CESNET association. Web contains detailed documentation of the probe and server, the option to download the latest version of the probe and live demo which shows the MOS values measured between multiple probes.

8 Conclusion

All the To ensure the continuity of the projects being developed and maintained and to further increase the capabilities of the software tools and methodologies incorporated to the SeeSIP deployment platform, the scheduled work will be focused on the deployment of the UCIPROV-enabled network probes for both honeypot and quality measurement use cases. Moreover, the SBC SeeSIP use case will further be developed in order to enable more efficient security measures in VoIP infrastructure of the remote locations. These new functions will then be merged with the trunk code branch and then the use case will be deployed in pilot environment.

Further increase in capabilities will also be visible in HoneyNode use case, where the majority of the work will attempt to integrate HEP-enabled plugin to the software to allow more precise and more standard (in sense of data structure and content) data to be collected and analyzed. The analysis that

takes place in the Beekeeper use case will be augmented so it can use a neural network based approach algorithms for both the phase of learning and the classification.

A new HOMER5 use case will be developed with intent to unify and standardize the communication of the network probes, honeypots and other tools deployed in the CESNET2 network. This will allow all the devices to send the collected data to the Beekeeper directly, where the evaluation will be performed in an automatic fashion.

9 Acknowledgment

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

10 References

[1] M. Voznak, J. Slachta, L. Macura, *Development of Advanced Concept of Voice Communication Server on Embedded Platform*, International Journal of Mathematical Models and Methods in Applied Sciences,Volume 7, Issue 2, 2013, ISSN 1998-0140, pp. 103-110.

[2] M. Voznak, J. Slachta, L. Macura L and K. Tomala, *Advanced solution of SIP communication server with a new approach to management*, Telecommunication Systems, 9 p., 2014.

[3] *Homeproj – SeeSIP project*, URL: https://homeproj.cesnet.cz/projects/besip/wiki .

[4] M. Voznak, J. Rozhon, F. Rezac, J. Slachta, *Real-Time Speech Quality Monitoring Using Non-Intrusive Method*, RECENT RESEARCHES in CIRCUITS, COMMUNICATIONS and SIGNAL PROCESSING, Milan, January 9-11, 2013, pp. 43-48.

[5] F. Rezac, J. Rozhon, J Slachta, M. Voznak, *Speech Quality Measurement in IP Telephony Networks by Using the Modular Probes*, CN2015 , In SPRINGER Communications in Computer and Information Science , Volume 522, 2015, pp 172-181 June 2015, Best paper award.

[6] J. Slachta, J. Rozhon, F. Rezac, M. Voznak, *Automation Techniques of Building Custom Firmwares for Managed and Monitored Multimedia Embedded Systems*, In Advances in Information Science and Applications - Volume II, 18th International Conference on Circuits, Systems, Communications and Computers (CSCC '14), Santorini, July 2014, ISBN: 978-1-61804-237-8, pp. 546-550.

[7] M. Voznak, Z. Chmelíkova, *Voice over IP in the network TEN-155 CZ*, 4th International Student Conference on Electrical Engineering POSTER 2000, Czech Technical University in Prague, 2000.

[8] F. Rezac, *Personal Private Branch Exchange*. URL: https://homeproj.cesnet.cz/attachments/download/2102/Datasheet_PBX.pdf .

[9] J. Safarik, J. Slachta, *Comparison of artificial intelligence classifiers for SIP attack data*, Proceedings of SPIE – The International Society for Optical Engineering, 9850, art. no. 985006.

[10] F. Rezac, J. Rozhon, J. Safarik, M. Voznak, Z. Bajakova, *Analysis of the IP Telephony Security Issues Using Automatic Neural Network Classifier*, INternational Conference on Software, Telecommunications and Computer Networks, Split, Croatia, 2016.

[11] J. Safarik, M. Voznak, F. Rezac, *Security Evaluation of Multimedia Systems*, TERENA Networking Conference TNC 2012, Reykjavik, Iceland, May 2012 In Proc. Networking to services, TNC2012, The 28th Trans European Research and Education Networking Conference, 21 - 24 May, 2012.

[12] M. Voznak, J. Rozhon, F. Rezac, E. Gresak, *VoIP Call Quality Assessment based on RPROP Neural Networks*, International Symposium on Telecommunications – BIHTEL 2016, Sarajevo, Bosna and Herzegovina, 2016.

[13] M. Voznak, J. Slachta, F. Rezac, J. Rozhon, *Modelling speech and video quality in networks based on internet protocol*, International Journal of Information and Communication Technology, Volume 8, Issue 1, 2016, pp. 95-111, DOI: 10.1504/IJICT.2016.073636.

[14] F. Rezac, *BeeSIP – Callmon*, URL: https://callmon.cesnet.cz/ .