BOUN-SIM API Reference

Çağatay Yıldız[†], Barış Kurt[†], Taha Ceritli[†],
A. Taylan Cemgil[†], Bülent Sankur[⋆]

Boğaziçi University, Dept. of Computer Eng.[†]
Boğaziçi University, Dept. of Electrical Eng.

FIRST DRAFT - December 2016

This document details the API created at Boğaziçi University for Netaş Turkey.

Contents

1 Introduction

Session Initiation Protocol (SIP) [1] is one of the most popular open standard signaling protocols designed for Voice Over Internet Protocol (VoIP). Yet, difficulties of access to a real SIP data set prevent researchers from studying on SIP-related tasks. The motivation of this work is to present a tool that eliminates the real data set obstacle. We developed BOUN-SIM to generate real-time SIP traffic by simulating behaviors of a number of users. We present a detailed analysis of our software in this report.

To use our system, one needs to setup a framework consisting of 3 components: *Trixbox* (a SIP proxy server), *Monitor* (network monitoring and recording system), and *Simulator*. As a SIP server, we used Trixbox[2]. Monitor is the unit that captures the network packets and extracts several sets of features.[3]. Finally, Simulator is responsible for mimicking SIP phones and generating SIP network packets. Each component can be installed via the instructions presented in later sections.

The rest of the document is organized as follows: Section 2 introduces the framework used for simulation. Section 3 details the model behind the Simulator. Then, we illustrate the software design to display the architecture. Next, an installation manual for the framework is given in Section 5. Lastly, we present instructions to run the software in Section 6.

2 Framework

2.1 The SIP Server: Trixbox

Since users in a SIP network communicate through a SIP server, the simulation setup requires a server. Our network traffic generator is designed agnostic to any SIP Proxy server. However, the system is tested only with the Trixbox, a publicly available VOIP phone system based on the Asterisk.

Prior to running Simulator, simulated users must be subscribed to the server. Recommended way of using Simulator is to prepare a Trixbox server with subscribed users dedicated for the simulation. We explain the subscription process in Section 5.

2.2 Monitor

Simulator creates network traffic passing through the SIP Server. For online or one-time tasks, a module that listens the network interface and parses network packets would be sufficient. On the other hand, tasks such as model selection and parameter optimization enforce researchers to use the exactly same network traces several times. For this, we implemented a monitoring unit [3]. In addition to the histogram of 28 different SIP packets, Monitor tracks server statistics, such as CPU-Memory usage percentages and operating system statistics.

A more detailed explanation of Monitor can be found in [3]. We leave installation procedure and usage of monitoring system to Section 5 and 6, respectively.

2.3 Simulator

Simulator is a realization of an event simulation. In the simulation, a number of SIP clients calls one another in real time. The fundamental elements of Simulator are users and their characteristics. Each user may perform a set of actions. Their characteristics of performing these actions and relationships with each other depend on a social network model. The details of Simulator are given in the next section.

3 Simulator Details

This section presents Simulator details. The goal is to allow readers to see the underlying structure behind Simulator. More specifically, we explain how to adapt a social network model, which determines the characteristics of users, to SIP network, and how those characteristics affect the generated traffic.

3.1 Generative Model

The simulation is driven by a probabilistic generative model to mimic the user behavior. One can easily modify the parameters to simulate different kinds of users. Those parameters control the structure of the social network and the characteristics of each individual in this network. The list of parameters is given in Table 1.

3.1.1 The Social Network

The relationship between users in the simulation is structured by a stochastic graph block model[4]. N users in the simulation are first divided into K different groups. The probability of a user to be in group k, denoted by a K dimensional vector π_k, is sampled from a Dirichlet distribution with parameter α (The distribution is by default flat):

$$\pi \sim \mathcal{D}ir(\pi; \alpha) \tag{1}$$

$$p(\pi|\alpha) = \frac{\Gamma(\sum_k \alpha_k)}{\prod_k \Gamma(\alpha_k)} \prod_k \pi_k^{\alpha_k - 1} \tag{2}$$

We use a $N \times K$ dimensional binary-valued matrix $G = (g_1^T, g_2^T \ldots, g_N^T)^T$ to denote group identities. Each row g_n consists of $n-1$ zeros and a single one ($g_{n,k} = 1$ if and only if n'th user belongs to k'th group). Rows are sampled from a Categorical distribution with parameter π:

$$g_n \sim \mathcal{C}at(g_n; \pi) \tag{3}$$

$$p(g_n|\pi) = \prod_k \pi_k^{[g_{n,k}=1]} \tag{4}$$

where $[x = y]$ is the *indicator* function, which returns 1 if $x = y$ and returns 0 otherwise.

3.1.2 Phone Books

First, a phone book b_i is generated for each user i to determine persons that user will most likely to call. b_i is defined as a vector of length N, such that $b_{i,j}$ is the probability of user i calling user j whenever it decides to make a call. This vector is drawn from a Dirichlet distribution with a parameter α_i specific to each user.

$$b_i|\alpha_i \propto \mathcal{D}ir(b_i; \alpha_i) \tag{5}$$

The hyper parameter α is chosen as $\alpha_{i,i} = 0$ and $\alpha_{i,j} = 1$ for $i \neq j$. This means that the phone book b_i of user i is drawn from an (almost) flat Dirichlet distribution, but b_i itself is not flat. Each user i have different call probabilities of calling other users.

Whenever user i decides to make a call, it selects a user c_i from its phone book to call from a Categorical distribution

$$c_i|b_i \propto \mathcal{C}at(c_i; b_i) \tag{6}$$

Parameter	Detail	Default Value
N	Number of users	100
K	Number of social groups	4
α	Group distribution parameter	$[1, 1, 1, 1]$
a, b	Phone book shape parameters	2, 0.4
μ, ψ	Initial unregistered time shape parameters	2, 10
ρ, ϕ	Re-registration period shape parameters	30, 10
k, θ	Call rate shape parameters	20, 10
$\delta_{min}, \delta_{max}$	Upper and lower limits of average conversation time	20,200
a_{min}, a_{max}	Upper and lower limits of probability of answering a call	0.8, 1

Table 1: System Parameters

3.1.3 Registration

When the simulation starts, all users are offline. The amount of time user i spends before registration is denoted by r_i and generated as follows:

$$r_i \sim \mathcal{G}(r_i; \mu, \psi) \tag{7}$$

$$p(r_i|\mu, \psi) = r_i^{\mu-1} \frac{e^{-r_i/\psi}}{\psi^{\mu}\Gamma(\mu)} \tag{8}$$

Furthermore, each user i sends a re-register packet with a fixed period t_i that is again sampled from a Gamma distribution with parameters ρ and ϕ:

$$t_i \sim \mathcal{G}(t_i; \rho, \phi) \tag{9}$$

$$p(t_i|\rho, \phi) = t_i^{\rho-1} \frac{e^{-t_i/\phi}}{\phi^{\rho}\Gamma(\rho)} \tag{10}$$

3.1.4 Call Rates

The rate of phone calls for each user i is determined by the scale parameter β_i. We generate β_i from a Gamma distribution with parameters k and θ:

$$\beta_i \sim \mathcal{G}(\beta_i; k, \theta) \tag{11}$$

$$p(\beta_i|k, \theta) = \beta_i^{k-1} \frac{e^{-\beta_i/\theta}}{\theta^{k}\Gamma(k)} \tag{12}$$

During the simulation, whenever user i becomes idle, it waits for a random t_i seconds before calling a person from his phone book, which is drawn from an Exponential distribution with rate β_i.

$$t_i|\beta_i \propto \mathcal{E}xp(t_i; \beta_i)$$
$$p(t_i|\beta_i) = 1/\beta_i \exp(-t_i/\beta_i)$$

3.1.5 Call Durations

Each user i has its own average call duration δ_i, drawn from a Uniform distribution with parameters $\delta_{min}, \delta_{max}$.

$$\delta_i|\delta_{min}, \delta_{max} \propto \mathcal{U}(\delta_i; \delta_{min}, \delta_{max}) \tag{13}$$

Some users tend to make short calls, while others prefer to talk longer. Whenever a call is set up between users i and j, they both sample the duration of the call independently and the one who wants to talk shorter hangs up. Therefore, we generate a call duration $d_{i,j}$ as follows

$$d_i|\delta_i \propto \mathcal{E}xp(d_i; \delta_i) \tag{14}$$
$$d_j|\delta_j \propto \mathcal{E}xp(d_j; \delta_j) \tag{15}$$
$$d_{i,j} = \min(d_i, d_j) \tag{16}$$

3.1.6 Call Responses

The actions that users take for incoming calls is given in Algorithm 1:

Algorithm 1 Algorithm to Generate Responses to Calls

User A calls user B
if B.status == "Available" **then**
  generate $x \in U[0, 1)$
  if x $\leq$ B.answering_probability **then**
    return "ACCEPT"
  else
    return "REJECT"
  end if
else
  return "BUSY"
end if

Every user i is given a call answering probability a_i by the model. We draw this probability from the following uniform prior:

$$a_i \propto \mathcal{U}(a_i; 0.8, 1) \tag{17}$$

Whenever a user that is already talking on the phone receives a call, s/he returns busy signal and the call is rejected. In case that the user is in available state, we draw a Bernoulli variable $r_i \in \{0, 1\}$ for the decision.

$$r_i \propto \mathcal{B}(r_i; a_i) \tag{18}$$

The call is simply rejected if $r_i = 0$. Otherwise users start talking.

4 Software Design

Simulator is written in *python* by using the *PJSIP*[5] library for generating SIP messages. PJSIP is a free and open source multimedia communication library written in C, implementing SIP and some other protocols.

4.1 Issues with PJSIP's Limitations

We successfully run a simple network simulation where 8 users can register themselves and call each other at random times. However, in order to simulate a larger and more complex VOIP network traffic, we needed to increase the number of users in the simulation. By doing so, we have reached the PJSIP library's limitation on the number of user accounts, calls and transports. One way to combat with this issue is to modify and re-compile the library itself, but we decided to switch to multi-process software architecture.

In our implementation, a master process forks a number of slave processes (we also refer those as *batch*es). User behaviors are simulated in slave processes, where each slave process can host up to 8 user accounts. All kinds of events (such as initiation of a call, hangup, etc.) take place in slave process. This new architecture can also support distributed simulation by running the slave processes in different machines connected to the Internet. Current implementation is able to simulate not more than 8 users. When the goal is to simulate a more complex real-world scenario, multiple processes, or *batch*es, ought to be initiated.

4.2 Assumptions and Constraints

Below are some of the details indicating the capabilities and limitations of our program. The program is open to improvements as needs arise.

- Currently, there is only one SIP server in the network. We may need to add a few more servers to generate more realistic traffic.
- We assume that all the users in the simulation are indeed subscribed to the SIP server being used. Also note that we have not tested our software with more than 1000 users, which we also set as the maximum number of users supported in the simulation. One may update `__MAX__NUM_USERS__` field in `simulator.py` but no guarantees.
- Number of users in each process is limited to 8, which is set by PJSIP. This is also the main reason why we go with multi-process architecture.
- Despite all our effort to workaround PJSIP's limitations, the number of simultaneous calls a single user can make is limited to 32. This is actually fine for a lot of applications but we need to take this into account in case we want to simulate network attacks.
- We have run this program in Unix-like operating systems. If there is no platform dependency issue in PJSIP, the program should run in other operating systems as well.
- Each user in the simulation uses a different network port to send and receive data. The number of port opened by a user is the same as its user id. By default, we specified port, or id, range $10000 - 10999$.

4.3 Class Schema

In this work, we implemented a number of classes that extend PJSIP and Python modules and we also implemented a few from scratch. In the following subsections, we presented more details of each module.

4.3.1 `simulator.py`

Simulation starts with the execution of `main.py`. This module is responsible only for forking new processes (i.e., `TrafficGeneratingBatch` instances), which in turn run the simulation. Currently, a single process is forked as the number of users is set to 8.

4.3.2 `model.py`

User parameters and decision-making functions are stored in this module. Currently, there is only one class, `Model`. Depending on the needs, one may add more parametrization to functions in this class or may create new classes, which basically corresponds to nothing but a different generative model. This class is currently responsible for

- Setting user parameters
- Generating response to a call
- Generating waiting time between two consecutive outgoing calls
- Picking a callee
- Setting the simulation intensity

4.3.3 `batch.py`

This module contains a class named `Batch`, which extends `Process` class in Python library. Another class in this module, `TrafficGeneratingBatch`, extends `Batch` and is responsible for

- initializing PJSIP libraries and `User` instances.
- controlling the flow of simulation, thanks to each `TrafficGeneratingBatch` instances having its own heap that stores call and hangup events.

If one decides to implement a different type of simulation, s/he should implement his/her own batch that extends `Batch`.

4.3.4 `user.py`

This module governs the behavior of an SIP agent and also provides an interface to PJSIP library. We have 3 different classes:

- `UserAccountCallback`: This class extends PJSIP's `AccountCallback` implementation and handles incoming call interrupts. We overwrote `on_incoming_call` function, which sends SIP packets depending on the response generated by `Model`.
- `UserCallCallback`: This class extends PJSIP's `CallCallback` implementation and handles call state changes. We overwrote `on_state` function, which is signaled when the state of a call changes. This function helps us to update users' statuses and add new call/hangup events to their heaps.

- `User`: This class is at the heart of our software. It stores user parameters, phone book, status, as well as pointers to many other objects. It also realizes user actions such as generating/answering/terminating calls.

5 Installation Manual

This section aims to refer the dependencies of the software and lead users through the installation steps. The network traffic generator works with a SIP server with subscribed users and requires Python 2.7 with some additional packages installed. Hence, we explain setup procedure of Trixbox alongside Simulator. We also cover installation of monitoring system in this manual.

The system is easily installed and works without a problem on the Linux environment. It has been mainly tested on Ubuntu Linux 14.04.

5.1 Trixbox

We prefer running Trixbox on a Linux virtual machine due to its usefulness. Therefore, following instructions implement installation of Trixbox on a virtual machine. However, it may be used on a real computer as well since installing directly on a computer differs slightly.

To install trixbox, we followed the tutorial on [6].

1. Make new VM boot from the trixbox ISO which can be found in [7]:
   (a) Select trixbox and open settings by clicking Settings.
   (b) Select Storage and choose Empty under IDE Controller. Then click the folder icon.
   (c) Click Add on the VirtualBox and browse for trixbox ISO file.
   (d) Click select after choosing the trixbox ISO.
2. Configure network settings.
   (a) Click Network and choose Attached to: Bridged Adapter and select your active network interface card under Name.
3. Install trixbox.
   (a) Click Start.
   (b) Hit Enter in the boot screen.
   (c) Select your keyboard layout and hit Enter after getting to OK.
   (d) Select your time zone and hit Enter after getting to OK.
   (e) Enter your root account password and hit Enter after getting to OK.
   (f) Select CD/DVD Devices in the devices menu and click Unmount CD/DVD Device.
   (g) Select Reset on the Machine menu.

The recommended way of using Simulator is to prepare a Trixbox server with subscribed users dedicated for the simulation. Simulator supports a maximum of 1000 users, (SIP extensions), such that the User Id's are strictly between 10000-10999 and the all passwords are *tamtam*. Those values are not configurable in this current version.

The subscription of users can be done via the *PBX→Bulk Extensions* module of the Trixbox server. The necessary comma separated file for subscribing the necessary users can be found at *trixbox_extensions.txt*.

User Id	User Name	Password
10000	user10000	tamtam
10001	user10001	tamtam
⋮	⋮	⋮
10999	user10999	tamtam

Table 2: SIP extenstions for traffic generation.

5.2 Monitor

This subsection states the required modules and describes installer commands for them. We present modules required for the system with their roles as follows:

1. *scipy* : required for random process generation
2. *numpy* : required for random process generation
3. *dpkt*: required for IP packet parsing
4. *twisted*: required for SIP message parsing
5. *psutil*: required for tracking OS resource usage
6. *pylibpcap* : required for capturing the network traffic

We used *pip* as a package installer. The *pip* installer commands for the modules except `pylibpcap` are given below:

```
sudo pip install scipy
sudo pip install numpy
sudo pip install dpkt
sudo pip install twisted
sudo pip install psutil
sudo pip install matplotlib
```

To install pcap library, first install `libpcap-dev` with

```
sudo apt-get install libpcap-dev
```

Then, install `pylibpcap` from https://sourceforge.net/projects/pylibpcap/.

5.3 Simulator

The dependencies of Simulator can be categorized into 2 categories: PJSUA Python library and Python modules.

The network generator depends on the *PJSUA* Python library, which is the Python version of the *PJSIP* library written in C, that implements several communication protocols including the SIP. Additional python modules are required for data generation and processing. The details are given in the following 2 subsections, sequentially.

5.3.1 PJSIP

The installation of the PJSIP library requires building it from the source code. Then the Python module must be built afterwards. For a detailed explanation of the installation process, please refer to the original PSJIP manual. Also, following documentation presents the process more clearly: PJPROJECT manual.

We summarize the instructions to install the library successfully:

1. Download the source code from PJSIP Source.
2. Install g++:
   (a) sudo apt-get update
   (b) sudo apt-get install g++
3. Install PJSIP:
   (a) cd to pjproject-2.x
   (b) Create shell script named `configure-linux.sh` containing following two commands:
      - export CFLAGS=”$CFLAGS -fPIC”
      - ./configure
   (c) ./configure-linux.sh
   (d) make dep
   (e) make
4. Install PJSIP Python
   (a) sudo apt-get install python-dev
   (b) cd pjsip-apps/src/python
   (c) sudo python ./setup.py install

5.3.2 Python Modules

We present modules required for the system with their roles as follows:

1. *scipy* : required for random process generation
2. *numpy* : required for random process generation
3. *scikit-learn* : required for the Simulator

We used *pip* as a package installer. The *pip* installer commands for the modules are given below:

```
sudo pip install scipy
sudo pip install numpy
sudo pip install -U scikit-learn
```

6 How to Run

In this section, we explain command line usage of the software. Simulator is a standalone application and can be run by a single command. However, initiating Monitor takes two commands. The first one is for starting a *server* application in the SIP server, which listens the network, tracks the operating system and extracts features. Other one triggers a *client* application to which the server program delivers data. Those two applications must be running either at same LAN or in computers connected to the Internet. Since data transmission through the Internet takes some time, the latter setup results in overhead and may not be feasible in applications where the frequency of data arrival is high, for example 10 arrivals in a second.

6.1 Simulator

Upon installing dependencies, Simulator can be started via command line. The parameters are as follows:

```
sudo python simulator.py -n [num_users] -k [num_groups]
-a [server_ip] -i [traffic_intensity]
-f [simulation_config_file] -id [base_id]
-t [simulation_duration] -v [verbose]
```

where

- `num_users` and `num_groups` are integers,
- `traffic_intensity` is either *low* or *high* in default settings
- `simulation_config_file` is the absolute path of the file
- `base_id` is the lowest user id (which also corresponds to the lowest port number used by the program)
- Also, add the `-v` option to print out the logs of the interaction between users.

For example, command above simulates a traffic with 6 users. SIP server has a IP address of 79.123.176.157. The simulation takes 5000 seconds. Lastly, the traffic intensity is high.

```
sudo python simulator.py -n 6
-a 79.123.176.157 -i high -t 5000
```

Note that a simple help menu is displayed with the commands

```
python simulator.py -h
python simulator.py --help
```

6.2 Monitor

In this section, we show how to use our server and client applications. Note that those are our default applications and one may create its own applications depending on the requirements of the task.

6.2.1 Server

Since our goal is to listen the network traffic passing through the SIP server, Monitor should run on the machine where Trixbox is installed. Therefore, Monitor's dependencies must be installed to SIP server. Similar to Simulator, Monitor can be called from command line. After getting to root directory of the software, following command runs Monitor:

```
cd monitor
python boun_server.py -p [port]
```

where `port` is an integer representing the port that server uses to communicate with client applications. By default, we set the port number to 5010.

6.2.2 Client

In addition to server, we illustrate a client application. A nice property of our framework is that multiple `MessageHandler`s can be registered to a single client application, each of which being responsible for performing a distinct task. All registered `MessageHandler`s wake up when data arrives to the client. Currently, we provide five `MessageHandler`s. Four of them is used for visualization (of packet histograms, Asterisk logs, CPU-memory percentages and operating system statistics), and the other simply logs the data.

Below command runs a simple client:

```
cd monitor
python boun_client.py -i [interface] -o [log_file] -a
[server_ip] -p [port] -v [verbose]
```

where

- `interface` is interface parameter that monitor listens.
- `log_file` is name of file to which data is recorded.
- `server_ip` is the IP address of the server application to be connected.
- `port` is an integer representing the port that client application uses to communicate with server.
- `verbose` is a boolean used to print simulation logs to screen.

When no log file name is set, program does not create logs.

6.3 Bayesian Change Point Model (BCPM)

In addition to Monitor and Simulator, we provide an implementation of Bayesian multiple change point model. We run the model on the data generated via Simulator and a vulnerability scanning tool named Nova V-Spy [8] that is able to perform DDoS attacks. You can see `bcpm/demo.py` and `bcpm/demo.py2` for more detail.

References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and Schooler E. RFC 3261: SIP: Session Initiation Protocol. Technical report, IETF, 2002.

[2] Trixbox. http://www.fonality.com/trixbox, 2016. [Online; accessed 29-April-2016].

[3] Ç. Yıldız, B. Kurt, Y. T. Ceritli, A. T. Cemgil, and B. Sankur. A sip network simulation and monitoring system. 2016.

[4] Anna Goldenberg, Alice X. Zheng, Stephen E. Fienberg, and Edoardo M. Airoldi. A survey of statistical network models. *Foundations and Trends in Machine Learning*, 2(2):129–233, 2010.

[5] pjsip. http://www.pjsip.org/, 2016. [Online; accessed 29-April-2016].

[6] Trixbox Tutorial. http://call-cheap.blogspot.com.tr/2010/12/installing-trixbox-in-virtualbox.html, 2016. [Online; accessed 29-April-2016].

[7] Trixbox Download URL. https://sourceforge.net/projects/asteriskathome/, 2016. [Online; accessed 29-April-2016].

[8] Nova V-SPY. http://www.netas.com.tr/en/innovation-productization/nova-cyber-security-products/, 2016. [Online; accessed 29-April-2016].