Team project
“Software for self-testing of the Telecommunication network of University of Freiburg”

Arda Akcay
Tri Atmoko
Refik Hadžialić

October 6, 2011

Albert-Ludwigs-Universität Freiburg
Lehrstuhl für Komunikationsysteme
Prof. Dr. Gerhard Schneider

Supervisors:
Konrad Meier
Denis Wehrle

Sommersemester 2011

Contents

1 Introduction and Motivation

In the following report, the authors will try to give you a brief insight into our team project. The goal of our project was to develop a mechanism for automatic testing of our University Telecommunication network. The Telecommunication network of University of Freiburg consists of: our own internal GSM and telephone network systems; GSM redirecting device (if one initiates a call to one of the four external GSM networks, it redirects the calls to: T-mobile, 02, Vodaphone or E-Plus); a SIP gateway for landline calls inside of Germany (sipgate.de) and international calls. Since we did not have access to internal servers, our strategy was to exploit the existing systems and infer the results out of our findings. Before we had started working on our project, we had to analyze the overall network to come up with test cases that contain the highest information content. The next step in our procedure was to implement our ideas into a working piece of software. Gradually we implemented a bit-by-bit of the final software. Every single step was accompanied by testing and validation procedures. At the end we connected all the "black-boxes" into one big piece of software. We have fulfilled our requests and goals and made a fully working and operable test software. Despite developing a working software, all the way along we thought about the simplicity of the usage of the software. In the following chapters we will describe in more detail our approach and how each subsystem works.

2 Software concept

3 Introduction

3.1 Usage

4 Design

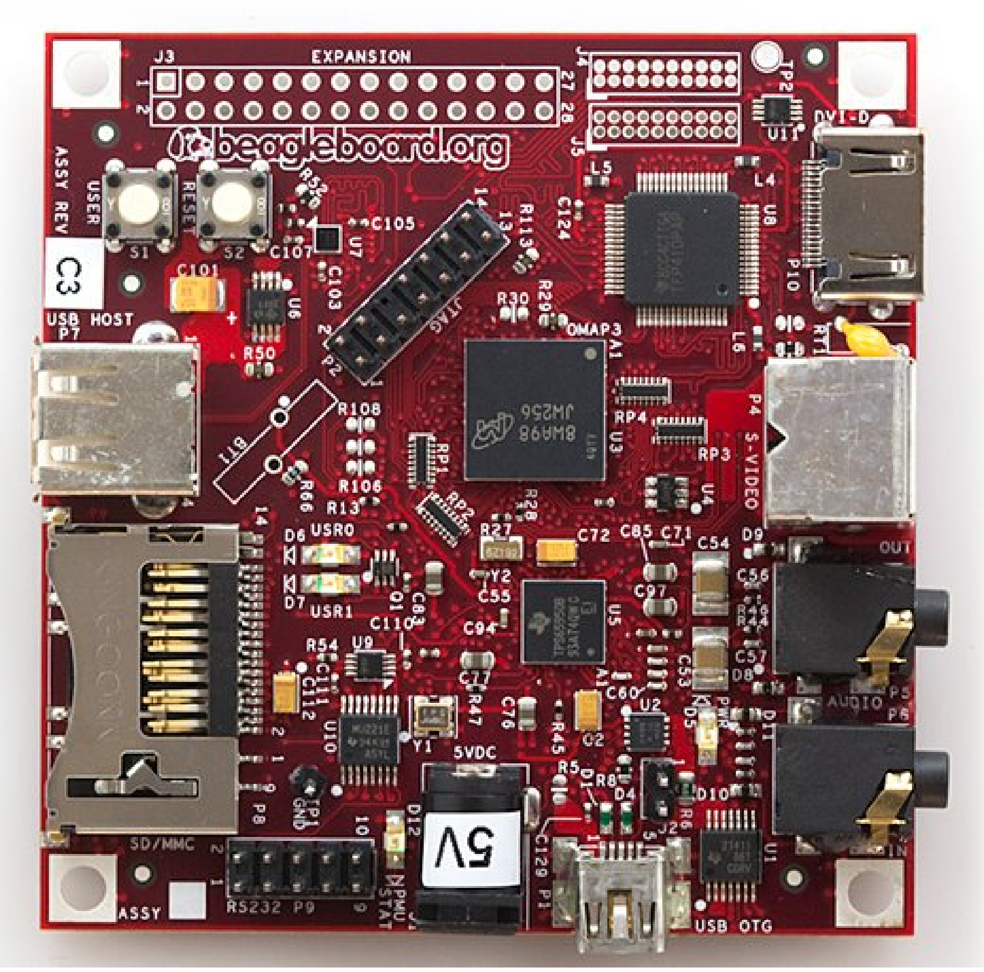

Figure 1: BeagleBoard, a linux-on-chip board where our controller software runs the GSM device

5 Protocol

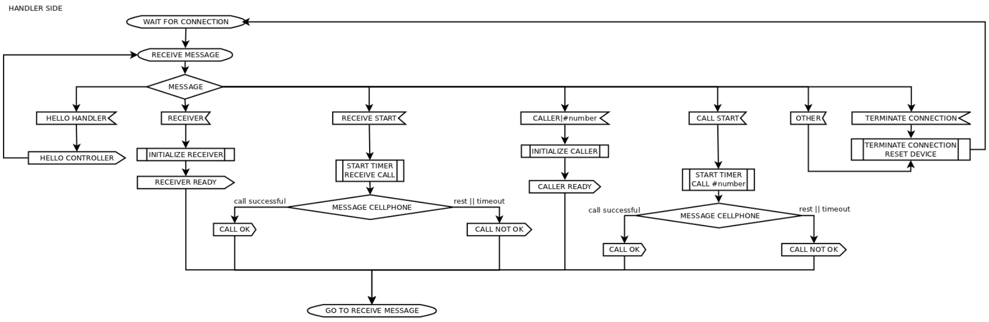

Figure 2: Flowchart of the protocol, on the handler side

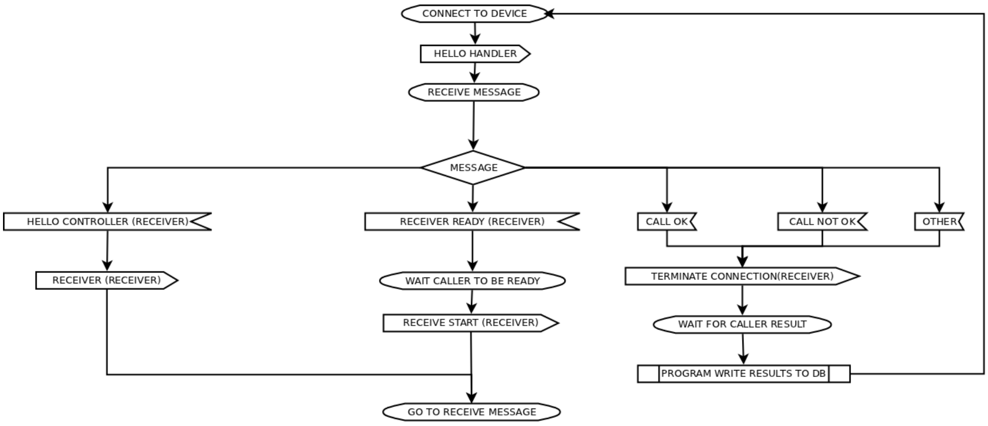

Figure 3: Flowchart of the protocol, on the controller side for the caller

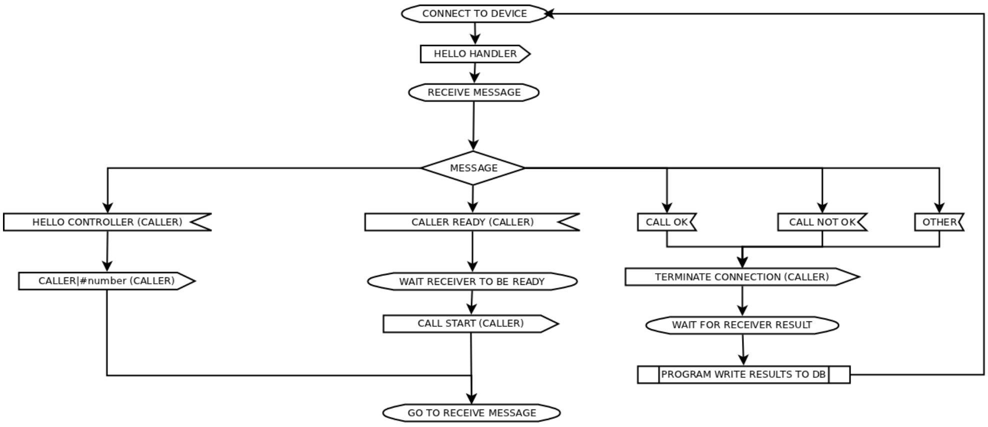

Figure 4: Flowchart of the protocol, on the controller side for the receiver

6 Security and safety of the test system

Safety and security of the software plays a major role in our project. It is of vital importance that only as few as possible people have access to our test system since the resulting data could be exploited to plan an attack (e.g. assume the University alarm system uses the SIP gateway to connect to the outside world and to alarm the police, if one knows that the SIP gateway is not working properly, a burglar could plan to rob the University building just at that moment.) Therefore the choice to go Open Source is justified due to the fact that one should know how every single detail of the system works. All the time, while we were working on the project, we were made aware of this issue by Denis and Konrad. We decided to use asymmetric key cryptography, where each side has two keys (private and public.) In the next sections we will explain in more details how we applied the methods.

6.1 Encryption of the communication channels

At first we thoought to encrypt the data before sending them but since none of us was an expert on encryption standards the idea was rejected. Alongside the fact that none of us had been an expert in the field of cryptography, we were neither experts in the field of internet programming. One could find maybe a way to disable our server software with various hacking methods (e.g. trying to open the port until the system runs out of memory and in our case the system which we used on the handler side was a BeagleBoard with ARM architecture running on a single chip TI OMAP processor, refer to the picture in figure 1.) We had to eliminate even the slightest possible threat in return for spending more time for debugging the test software system. Despite we were aware of all these facts, we had to choose one of the plenty implemented encryption standards on Linux. Denis and Konrad suggested using the SSH Tunneling method.

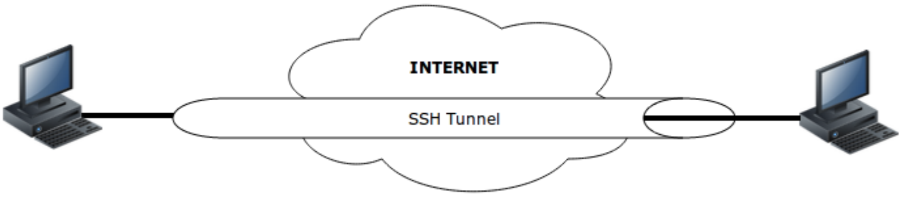

Figure 5: SSH Tunnel, all the communication inside the tunnel is encrypted

Using the SSH Tunnel port forwading method we could hide the real port we had used for our socket connection. On the other hand we could force the socket to accept only local connections (i.e. from the machine where the handler software was running.) The SSH Tunnel port forwarind method creates an encrypted tunnel between the two computers and then it creates two ports, one on the local and remote computer. All the data sent through the port on the local machine appear on the port at the remote machine.

The first problem we faced was that SSH required the username and password everytime we tried to make an SSH connection. We could avoid this problem by copying the public key from our server (where our test software runs) to the BeagleBoard [2]. This can be performed by executing the following commands in the terminal shell. One has to create first the private and public keys on the local machine(i.e. server computer, where the test software runs):

```
jsmith@local−host$ [Note: You are on local−host here]

jsmith@local−host$ ssh−keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jsmith/.ssh/id_rsa):[Enter key]
Enter passphrase (empty for no passphrase): [Press enter key]
Enter same passphrase again: [Pess enter key]
Your identification has been saved in /home/jsmith/.ssh/id_rsa.
Your public key has been saved in /home/jsmith/.ssh/id_rsa.pub.
The key fingerprint is:
33:b3:fe:af:95:95:18:11:31:d5:de:96:2f:f2:35:f9 jsmith@local−host
```

Then one needs to copy the public key to the remote machine (BeagleBoard) using ssh-copy-id:

```
jsmith@local−host$ ssh−copy−id −i ~/.ssh/id_rsa.pub remote−host
jsmith@remote−host's password:
Now try logging into the machine, with "ssh 'remote−host'", and check in:

.ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.
```

After we have created the public and private keys, and coppied the public key on the machine to which we want to connect, we can test if we can make an SSH connection to the remote machine:

```
jsmith@local−host$ ssh remote−host
Last login: Sun Nov 16 17:22:33 2008 from 192.168.1.2
[Note: SSH did not ask for password.]

jsmith@remote−host$ [Note: You are on remote−host here]
```

The test was successful. We tested it with our SSH Tunnel port forwarding class and it worked perfectly.

6.2 Security on the web site

Securing the communication channels without making certain the web site is safe would be worthless. We decided to use the *https* protocol instead of the *http* since a person in the middle could sniff our data (e.g. a person is connected with his/her smart-phone over an unprotected wireless network) [3]. At the same time the web site should be accessible only by the authorized personel. Our first approach to this problem was to build an PHP page with *MD5* hashed passwords, however we got a suggestion by Konrad and Denis to use a safer encryption method implemented in the Apache web server software, *.htaccess*. By using these two techniques we protected the web site of some vulnerabilities known to us. If the web site will be only accessed from our local university network, we can additionally add an IP filter mask as well.

7 Web page

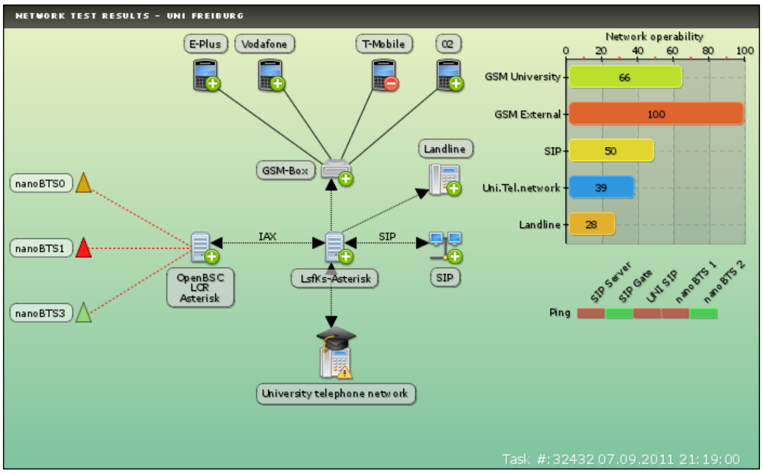

Figure 6: Result image showing working, defected and not tested subsystems

8 Conclusion

References

[1] H. Simpson, *Proof of the Riemann Hypothesis*, preprint (2003), available at `http://www.math.drofnats.edu/riemann.ps`.

[2] R. Natarajan, *3 Steps to perform SSH login without password using ssh-keygen & ssh-copy-id*, accessed on 18.08.2011, available at `http://goo.gl/fX68N`.

[3] P. Bramscher, *Creating Certificate Authorities and self-signed SSL certificates*, accessed on 05.09.2011, available at `http://www.tc.umn.edu/~brams006/selfsign.html`.