Installation and Performance Analysis of Asterisk on a wireless router

Dorin HINTEA, Tudor Mihai BLAGA, Virgil DOBROTA
Technical University of Cluj-Napoca
Faculty of Electronics, Telecommunications and Information Technology
G. Baritiu Str., 26 – 28 No., 400027, Cluj-Napoca, ROMANIA
E-mail: dorin.hintea@student.utcluj.ro {tudor.blaga;virgil.dobrota}@com.utcluj.ro

Abstract - This paper presents the installation of OpenWRT Linux distribution and Asterisk IP PBX on a Linksys WRT54GL wireless router. The testbed scenario includes two Windows XP computers running a traffic generation software called SIPp. This software is used for measuring the maximum call rate that can be supported by Asterisk running on a wireless router. The final goal of these tests is to evaluate whether an embedded device could support Asterisk VoIP PBX in a production environment and to determine the maximum number of calls per second supported.

I. INTRODUCTION

While many people think of VoIP (Voice over IP) as just a little more than a method of obtaining free long-distance calls, the real advantage of VoIP is that it allows voice to become just another application that uses the network to transmit data [1].

As many of the networks used inside residential or business buildings are under-utilized, a lot of cost savings are done using a single network which carries voice and data. With the continuous development of VoIP telephony, Asterisk software PBX (Private Branch eXchange) has grown in popularity among many residential and business users. Because of the open-source characteristic and the portability of the software, Asterisk as a VoIP is preferred to proprietary and expensive PBXs. Asterisk can be configured as the core of an IP or hybrid PBX, switching calls, managing routes, enabling features, and connecting callers with the outside world over IP, analog (POTS – Plain Old Telephone Service) and digital (ISDN/GSM/T1/E1) connections [2].

II. TESTBED CONFIGURATION

Figure 1 below presents the testbed architecture. It contains a Linksys WRT54GL wireless router running OpenWRT on which Asterisk is installed.

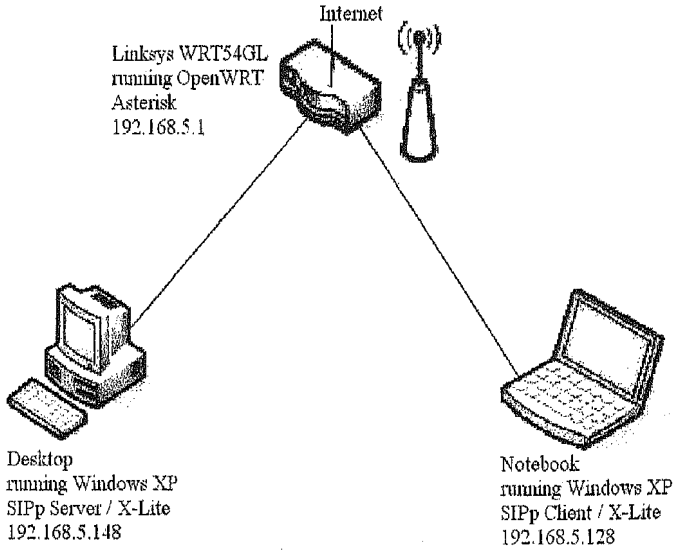

Figure 1: Testbed Architecture

The IP Address of the router is 192.168.5.1. It also includes two Windows XP computers, a notebook and a desktop, each running SIPp, one being the client and the other one the server.

A. Hardware and Software used

1) Linksys

The main hardware used is a Linksys WRT54GL v1.1 wireless router. The platform of this embedded device is Broadcom 5352 and it runs at 200 MHz. The Wireless NIC is Broadcom (integrated) [3]. The switch is also integrated in the CPU. It has 16 MB of RAM and 4MB of Flash Memory, a Serial Port, a JTAG and no USB ports [4].
Standard endowment for the exterior of the router is: 1 WAN port, 4 LAN ports and 2 external antennas. As for the interior, it includes standard firmware, SPI firewall, DDNS, DHCP, MAC filtering, 802.11 b/g capabilities, WEP, WPA, WPA2 (TKIP/AES/TKIP+AES) encryptions.

2) OpenWRT

OpenWRT is an extensible Linux distribution that runs on Linksys WRT54G/GS series, as well as on some other chipsets and manufacturers (Netgear, D-Link, Asus and others). OpenWRT is built from the ground to be a full-featured, easy modifiable operating system for these routers.
Because of the open architecture, the user is enabled to use stateful packet inspection, intrusion detection and many more matters that other systems offer at a much larger cost [5]. An important feature of OpenWRT is a fully writable JFFS2 file system, which allows package management via the `ipkg` package system.
The first version of OpenWRT was the Whiterussian RC1, which was intended only for Broadcom hardware. After these versions got up to RC6, the next version of OpenWRT that was released was Whiterussian 0.9, which is the latest and final release of the Whiterussian series. Support is still available, but the development has been stopped in early 2007. The new series, Kamikaze, supports multiple hardware architectures and uses the 2.6 kernel for all targets, except a legacy Broadcom target. It uses configuration files, instead of nvram.

3) Asterisk

Asterisk is an open-source, converged telephony platform, which is designed primarily to run on Linux. Among the advantages lies in it's fully customizable nature [1]. Asterisk's architecture is designed for maximum flexibility and supports VoIP in many protocols and can interoperate with almost all standard-based telephony equipments using low cost hardware [2].

4) SIPp

The software chosen for the load generation is SIPp. It is a free Open Source test tool / traffic generator for the SIP protocol. It includes a few basic SipStone user agent scenarios, user agent Server (UAS) and user agent client (UAC). It works by establishing and releasing multiple calls with the INVITE and BYE commands of the SIP protocol. It is also possible to write custom xml scenario files which can be loaded by SIPp. It features dynamic display of statistics about running tests (call rate, round trip delay and message statistics), CSV statistics dumps and dynamically adjustable call rates [6].

B. Installation

Using the Linksys web GUI, a version of OpenWRT Linux distribution was installed on the router. It was the Whiterussian 0.9, chosen because it's the latest stable version and it was developed especially for Broadcom hardware. Because the release included many packages, including the webif package (a web interface similar to the Linksys proprietary one), an error occurred when trying to install the Asterisk package. The size of the Flash was too small (4 MB), so the Asterisk package would not fit. Because of that, it was needed to erase the current firmware and install a size-reduced one.
Size of the Flash, obtained with "#df" command:

```
root@OpenWrt:/# df
Filesystem 1k-blocks Used Available Use% Mounted
on
/dev/root         832    832      0  100%   /rom
None             7152     28   7124    0%   /tmp
/dev/mtdblock/4 2432   1468    964   60%   /jffs
/jffs
```

The release chosen was Whiterussian 0.9 micro release. This release contains only basic packages for the router to function. A lite version of Asterisk was chosen too, to assure that it would fit on the Flash. The package asterisk-mini_1.0.10 was successfully installed using the "`ipkg`" command of Linux.
Before the installation of the Asterisk package, an updated package list must be obtained from the OpenWRT download site. The list was updated using the command "`#ipkg update`". Next, the Asterisk package was downloaded and installed. The following output presents all the steps:

```
root@OpenWrt:/# ipkg update
root@OpenWrt:/# ipkg list
root@OpenWrt:/# ipkg install asterisk-mini
```

On the two computers which run Windows XP, the X-Lite soft-phone was installed, to test Asterisk. Also, SIPp was installed on both computers, and configured using xml files. One of the machines was configured as a user agent server and the other one as a user agent client.

C. Configuration

Asterisk has been configured by modifying the *sip.conf* and *extensions.conf* files from the "/etc/asterisk" folder of the OpernWRT distribution installed on the router. Two users have been added in the *sip.conf* configuration files, in order to simulate calls between them. Because the configuration of the calls is being done mainly in SIPp, the *extensions.conf* file contains only the minimum requirements for a call to be initiated [7]. The configuration files can be seen below:

```
Sip.conf
[general]
context=default
port=5060
bindaddr=0.0.0.0

[client]
username=client
callerid=client
type=friend
context=default
host=dynamic
disallow=all
allow=all
canreinvite=no

[server]
username=server
callerid=server
type=friend
context=default
host=192.168.5.148
disallow=all
allow=all
canreinvite=no

extensions.conf
[general]
static = yes
writeprotect = no

[default]
exten => client,1,Dial(SIP/client,20)
exten => server,1,Dial(SIP/server,20)
exten => t,1,Hangup()
```

The next step is to setup the traffic generation tool, SIPp, for our custom scenario. It works by using two user agents: client and server. The client initializes calls towards the server. SIPp features the possibility to import configuration files, having the format of "xml" files. When SIPp is launched, by using the option "-sf" followed by the name of the xml file in the command line, it is possible to load specific configuration into the program. Below is a part of an xml configuration file used to load configuration in the user agents.

```
<send retrans="500">
    <![CDATA[
      INVITE
sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via:                    SIP/2.0/[transport]
[local_ip]:[local_port];branch=[branch]
      From:                                    sipp
<sip:sipp@[local_ip]:[local_port]>;tag=[call_num
ber]

      To:                                      sut
<sip:[service]@[remote_ip]:[remote_port]>
      Call-ID: [call_id]
      CSeq: 1 INVITE
      Contact: sip:sipp@[local_ip]:[local_port]
      Content-Length: [len]
    ]]>
  </send>
```

This part is the "send" function, implemented in the configuration file of the user agent client, in which the "INVITE" message is transmitted to Asterisk [8]. Along with this SIP message, SDP was used to describe the SIP sessions [9].

Although the Asterisk configuration may look like a very simple one, it was difficult to make the SIPp user agent client to find the user agent server. This is because in the first stages of configuration the "host" parameter of the client peer in *sip.conf* was set as "dynamic". This is why the client could not find the server. After the parameter was changed to the actual IP address of the server, the calls created by the client were arriving successfully at the server side.

III. Performance Analysis

The performance analysis was done using two categories of tests. The first set of tests points out how many calls could Asterisk support when installed on a wireless router. For this set, there was not any actual voice data exchanged between the two parties involved in the call. The calls followed the simple steps of initiating and terminating a telephone call. The parameters tested in this situation were the routers' CPU Load and Memory Load. Figure 2 shows the call flow described above:

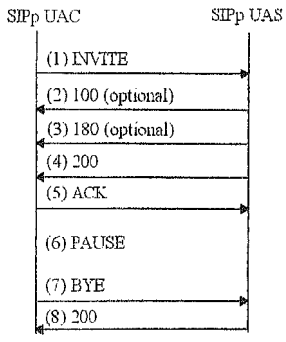

Figure 2: Call Flow - No RTP

The second set of tests included RTP traffic between the two peers. The parameters of interest were the ones from the first set of tests, and also the maximum number of simultaneous calls that Asterisk could support. For this situation, the call flow can be seen in Figure 3:

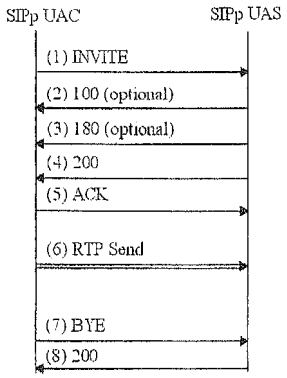

Figure 3: Call Flow - With RTP

This was made much more interesting as the peers involved in communication were using different type of codecs. There are a couple of situations that can apply here. The two peers could be using the same codec or they could be using a different codec. The situation where they use the same codec is covered by the first case, in which there is no voice transmitted between the two peers. It must be taken into account that in a SIP – SIP phone call, only the signaling passes through Asterisk, while the RTP data is transmitted directly between peers on top of either TCP or UDP. This is why from the point of view of testing the Asterisk PBX sending no traffic or sending traffic that does not need transcoding is the same thing. Figure 4 explains everything:

The maximum number of calls that can be initiated in one

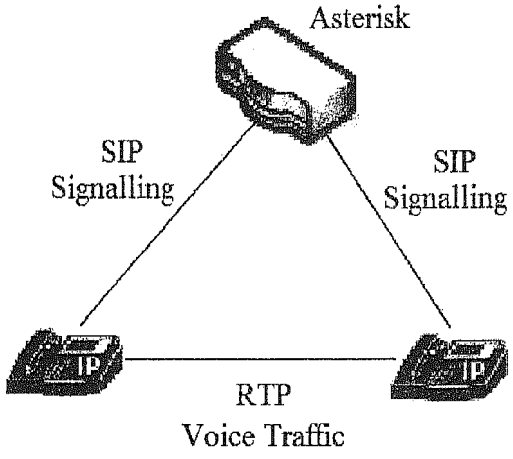

Figure 4: SIP

IV. Experimental Results

The results for the first set of tests can be observed in Figure 5 below:

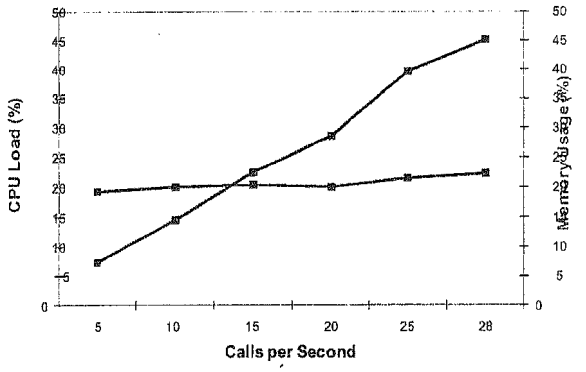

Figure 5: First set of tests results

second is 28. At 29 calls per second, Asterisk begins to drop calls. It can be seen that with the increase of the number of calls per second, the CPU Load is also increasing, while the Memory Usage is the same for all values of calls per second.

For the second set of tests, there were a number of different scenarios that were analyzed. These scenarios can be seen in Figure 6, Figure 7 and Figure 8 along with the results:

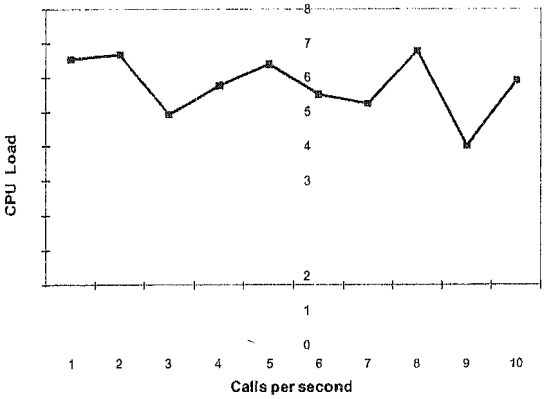

Figure 6: A-Law to μ-Law transcoding

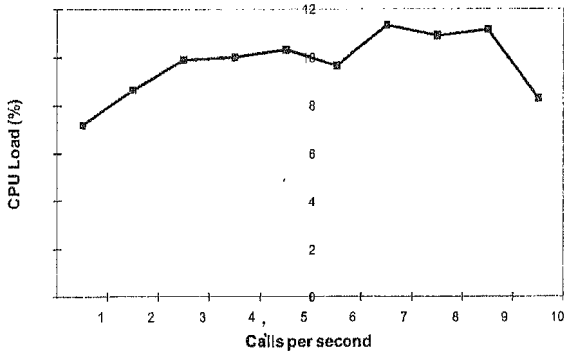

Figure 7: A-Law to GSM transcoding

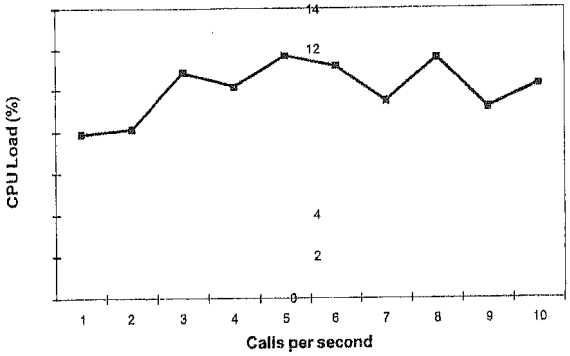

Figure 8: μ-Law to GSM transcoding

In these tests, the parameter varied was the calls per second and the one of interest was the CPU Load. It can be seen that the CPU Load is increasing for a higher number of calls per second. Also, the amount of CPU Load varies according to the codec used.

The next scenarios the parameter varied was the maximum number of simultaneous calls that can be supported and the observed parameter was the Memory Usage. The results can be seen in Figure 9, Figure 10 and Figure 11 below:

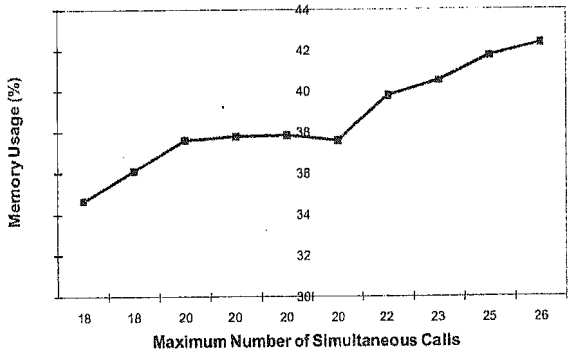

Figure 9: A-Law to μ-Law transcoding

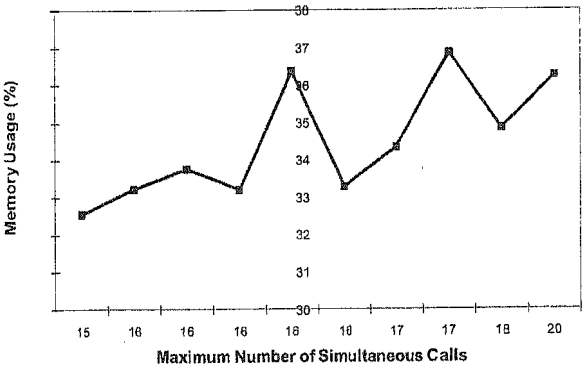

Figure 10: A-Law to GSM transcoding

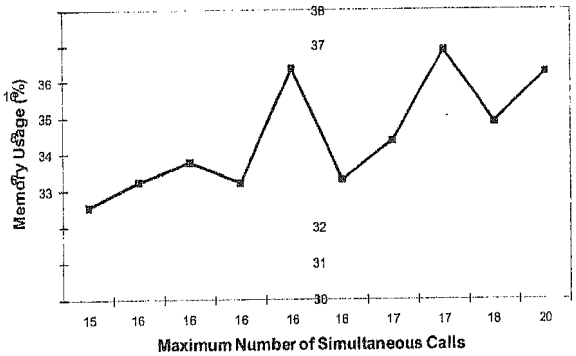

Figure 11: μ-Law to GSM transcoding

It can be seen that the more simultaneous calls are taking place, the more memory Asterisk uses.

V. CONCLUSIONS

As it was suspected, Asterisk can be a viable option when run on a wireless router. From the cost point of view it is a good solution for residential users and SOHO companies, as they can use their own wireless router to host Asterisk, thus no need for an extra computer in the house. The price of a wireless router is considerably lower than the price of a full-sized computer. The power consumption is another point where the router is more efficient than a computer. Although the proposed solution might seem perfect, the installation and configuration is not a trivial task. It is not a proprietary solution, so no ready-to-use version is available. The size of the flash memory available on the embedded device must be taken into account. Based on this, a compatible firmware solution can be chosen. Also, because of the limited available flash memory we had to choose the mini version of Asterisk 1.0.10-1, which does not include all the existing voice codecs. Only G.711 A-Law, G.711 μ-Law and GSM were available. Depending on the remaining free space other packages can be installed afterwards.

Once Asterisk was working properly on the wireless router its performance was of interest. The evaluation included the number of SIP calls per second (with or without RTP data) and the resources (CPU and memory) used by Asterisk.

The first set of test took into account the number of calls per second that can be initiated without voice traffic. Results show that a maximum rate of 28 calls per second is supported. At higher rates Asterisk begins to drop calls although the CPU load is only 45% and the memory usage is 22%.

For the second set of tests calls RTP data was used. It must be noted that the two instances of SIPp, each running a different user agents used different codecs, so transcoding was needed. As the tests confirmed it, depending on

the type of transcoding, a mean value of 15 simultaneous calls carrying voice could be supported. In this scenario the CPU load was 10% and the memory usage is about 45%.

Further evaluation could be done using other VoIP protocols (IAX, MGCP, H.323). Results can be improved by using a different hardware with more Flash memory available.

REFERENCES

[1] Jim van Meggelen, Leif Madsen, Jared Smith, "Asterisk: The Future of Telephony", O'Reilly, 2nd edition, 2007

[2] http://asterisk.org/support/about

[3]]http://www.broadcom.com/products/Wireless-LAN/802.11-Wireless-LAN-Solutions/BCM5352EL

[4] http://oldwiki.openwrt.org/OpenWrtDocs(2f)Hardware(2f)Linksys(2f)WRT54GL.html

[5] http://oldwiki.openwrt.org/OpenWRTDocs(2f)About.html

[6] http://sipp.sourceforge.net

[7] Spencer, Mark, Mack Alison, Christopher Rhodes, The Asterisk Documentation Team, "Asterisk Handbook", Digium, version 2, 2003

[8] Russel, Travis, "Session Initiation Protocol (SIP) Controlling Convergent Networks", McGraw-Hill Comunications, 2008

[9] http://www.ietf.org/rfc/rfc4566.txt