International Journal of Multimedia and Ubiquitous Engineering
Vol. 2, No. 3, July, 2007

Towards a Distributed Wireless VoIP Infrastructure

[1]Yuebin Bai, [1]Syed Aminullah, [1]Chao Li, [2]Qingmian Han, and [3]Huabin Lu

[1]School of Computer Science and Engineering, Beihang University, Beijing 100083, China
[2]School of Telecommunications Engineering, Xidian University, Xi'an, 710071 China; Communication Telemetry & Telecontrol Research Institute, CETC, Shijiazhuang, 050002, China
[3]Communication Telemetry & Telecontrol Research Institute, CETC, Shijiazhuang, 050002, China
[1] yuebinb@163.com

Abstract

It is the requirement of the next generation communication systems that devices should provide all existing and future communication services in integral way. A novel distributed wireless VoIP infrastructure based on SIP protocol, WiFi, and smart devices is proposed in this paper. The proposed architecture of the infrastructure is simple, manageable, stable, and extendable. The prototype test bed is an Infrastructure of distributed wireless VoIP Server based SIP, which provides text, voice and video communication to both wired and wireless devices. One of the main characteristic of this model is roaming and mobility. User authentication is centralized, and SIP server uses remote centralized AAA server for user authentication . Thus mobile users don't need to worry about roaming and changing the Server. The whole architecture is modular, and the reason for isolating the core modules from the extended modules is flexibility and extendibility. At the end of the paper, detailed test approaches ,test results and result analysis of the prototype system are presented. Test results show that the prototype system reach the design goal of this research work..

***Keyword**: VOIP, SIP, wireless network, distributed system, NGN*

1. Introduction

According to the experts in Communication industry, IP based communication would replace the traditional telephony network in near future. It is because the IP bases communication is not only efficient but offers coast effective advanced communication services with ease. Most of such services are not even possible with traditional telephony.

The IP network is moving fast towards wireless networks, and there is a huge demand and market for wireless VoIP services. With increasing number of wireless devices like Laptop, PDA and wireless terminals, users would desire for wireless communication. They would need wireless IP hot spots for cheep and exciting wireless communication as they use http for e-mail and web surfing.

Session Initiation Protocol (SIP) [1] for IP based communication is as important as http for World Wide Web. Industry experts predict that SIP would be the standard protocols for VoIP multimedia communication in near future. SIP not only offers basic telephony services; which are easy to manage and implement, but also provides advanced and complex value added services. Hence it is the most favorable candidate for 3G networks [2].

The proposed and developed architecture in this research work is the best combination of hardware and software for low cost distributed wireless SIP applications [3]. The maximum user load per second, the registration time, and call establishment time were tested using advances testing equipments and the results were more than satisfactory. It is worth mention that all advanced 3rd generation communication services can be provided with very low capital investment and using this proposed architecture. Possible application areas are university campuses, air ports and railway stations. More importantly, this architecture gives a solid research platform to SIP based application developers and researchers. Developers can not only use it as a SIP server for SIP client's development, yet can further extend its core capabilities. They can also use it for VoIP related QoS and Security research work as well

The remainder of the paper is organized as follows – the next section gives a description of the distributed server architecture, section 3, 4 and 5 present the design and implementation of prototype embedded system, SIP Server and Network Interface respectively. Section 6 shows the system working diagram and Section 7 gives the details of the server behavior. Section 8 and Section 9 describes initial experiments, test results and analysis; section 10 concludes the paper and presents the future work.

2. The Distributed Server Architecture

Figure 1 shows two wireless SIP servers in distributed environment [4]. A centralized AAA Server is used for all Wireless SIP Servers, so that each SIP Server authenticates its own users via this centralized AAA server. As the user data is in the centralized location and each SIP server register its clients using this centralized authentication server, the SIP servers don't care about the client's location. This means that mobile client's authentication process would not be affected when they change the server.

Each Wireless SIP Server is designed and implemented in such a way that it can deal with the incoming Invite messages. Depending on the destination SIP URI, the server intelligently forwards the invite message to appropriate destination client of server. Once the destination client replies with an OK message a direct RTP channel will be established between the two clients. It is then the responsibility of the clients to communicate and use appropriate ports and coding for media transfer [10]. The SIP server deals with signaling only while the clients deal with media specific issues. It is important to notice here that, in this design, the server creates its own clients table on fly, which is used for routing the messages.

Figure 1 also represents a sample call establishment procedure, which is briefly described here:

① Client C1 send Registration message to Wireless SIP Server WSS1.

② The Server WSS1 send authentication request to AAA server

③ The AAA server authenticates the Client C1

④ C1 sends invite message to C2

⑤ As C2 is not local client, the WSS1 server forwards the Invite to WSS2

⑥ WSS2 forward the invite message to C2 which is its local client

⑦ C2 Return ok messages to C1

⑧ C1 acknowledge the ok and call is established over RTP or RTPS

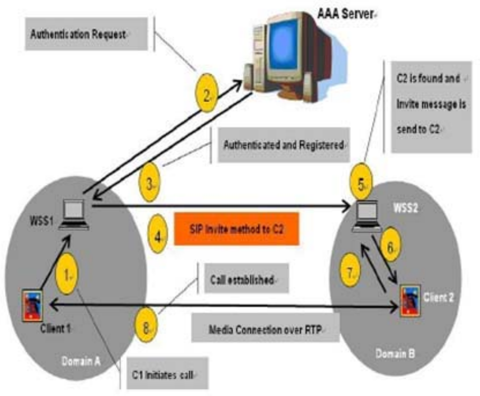


Figure 1. Distributed Server Architecture

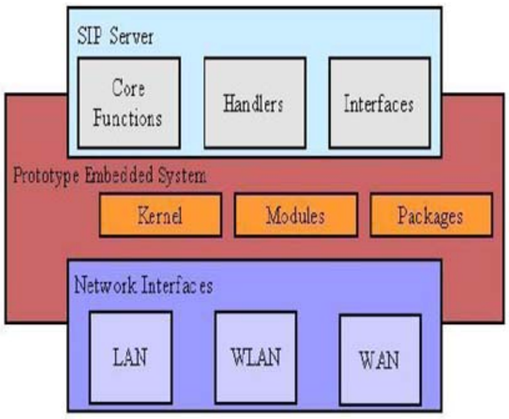


Figure 2. Major Design Components

It is important to notice here that, in this design, the server creates its own client table on fly, which is used for routing the messages. The authentication is centralized while the routing is automated. Clients are not bound to specific servers; either server can register the clients C1 and C2. This intelligent design behavior is the main factor in simple server implementation.

3. Prototype Embedded System design

The following figure 2 shows the block diagram and the major components of the whole design architecture.

As shown in the figure, the main components are:

● Prototype embedded system design and implementation

● SIP Server Design and implementation

● Network Interfaces

We will discuss these components in detail separately.

The prototype embedded system was proposed to fulfill the needs of the customized SIP Server and to test the SIP Server behavior while changing different core level parameter. As part of this research work, this prototype embedded system has been designed and developed, upon which the SIP server we will run.

The system consists of the following major components:

● Wireless hardware device

● Embedded Linux

● Drivers for interfaces

● Supporting tools and utilities

● System configuration and implementation

For the moments, it consists of an embedded wireless

Device [8], with 200MHz CPU, 4MB Flash and 16 MB RAM running embedded Linux [7]. On one hand the memory and processing power is very low, and on the other hand, a full fledged Linux environment is need for our SIP Sever. Considering all these factors in mind, a design that is the most efficient, low cost and practical in implementation was suggested. The system was designed with only required modules so that it provides best efficiency and resource management. Linux Kernel 2.4.x has been used as the base system. Many open source utilities were also implemented inside the prototype system to provide other required functionalities for the overall architecture. A customized version of SIP Express Router [6] which runs on this MIPS based architecture device provides SIP signaling.

3.1. Development Environment and Development tools

As Before starting the custom prototype system building procedure, a focus on some critical issues involved in the development is necessary. First a platform upon which we could run our prototype system is selected. The choice depends on market and the availability of the hardware in the local market. A low cost MIPS architecture based wireless device has been selected as the targeted platform. The Prototype System then, would run on this embedded device.

The prototype system deployment uses the Broadcom Chipset as a deployment platform. For the development patform, a x86 architecture based personal computer is used. This x86 development architecture has a compilation toolchain, which is used to compile application that runs on x86 architecture. As the embedded system has a different processor, a cross-compilation toolchain is needed.

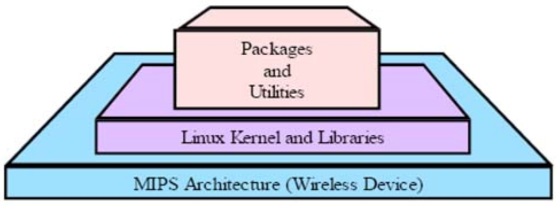

Packages
and
Utilities
Linux Kernel and Libraries
MIPS Architecture (Wireless Device)

Figure 3. Prototype System Model

Cross cross-compilation toolchain is a compilation toolchain that runs on one type of architecture and generates code for the other type of architecture. Here the development system has x86 architecture and the target system uses MIPS Architecture. Thus a cross-compilation toolchain would be used, that runs on x86 and generates code for MIPS architecture.

The exact specifications of both developing and deployment architectures are important. For the development environment, a personal computer (PC) with the following specification was used

CPU: 2.0GHz Celeron, x86 Architecture

Target System: 200MHz MIPS Architecture from Broadcom Corporation.

RAM: 256 M

OS: Fedora Core 4

Tools and Libraries: gcc, binutils, patch, bzip2, flex, bison, make, gettext, pkg-config, unzip, libz-dev and libc headers.

Cross Compilation Tool Chain: OpenWrt [3] cross-compilation tool chain that runs on x86 and generates code for MIPS, our wireless deployment architecture.

OpenWrt System Development Kit on Fedora box to generate the cross architecture packages for the MIPS architecture.

3.2. Prototype System Model

As The prototype embedded system consists of various building blocks. Figure 3 is the block diagram of the prototype System Model, where each block is represented as a stack. Actually each block acts as a foundation for the other. The main components of our Prototype System Model are:

1. MIPS based wireless device
2. Embedded Linux and Libraries
3. Usefule Utilities and Packages

As As shown in the figure 3, the base foundation is the wireless device. This device has it own unique specifications which are used as a reference during Linux Kernel development. It is a MIPS base small device; with it own CPU, RAM, Flash memory, Wireless card, Ethernet cart, and other supporting devices. However the device doesn't have LCD, Serial port and VGA Card. Thus, telnet, remote login, http, https and SSH etc should be use to access its command line interface.

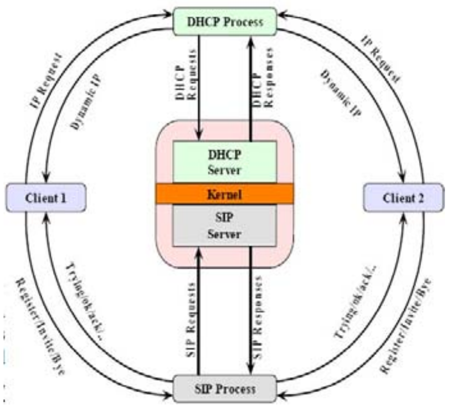


Figure 4. Client-Server Interaction

Linux is versatile in nature and runs on any device including this wireless device. Linux Kernel runs on top of the hardware and functions as a bridge between hardware and application software. Linux provide provides countless services with the help of other utilities. Although any standard MIPS based device can be used as the base for embedded Linux, yet the choice still depends on the device availability in the local market. Once such device is selected as the base, its exact technical specifications are important to know. The device used for our Prototype system has the following technical specifications.

CPU Type: Broadcom 4712 Wi-Fi access point processor chip

CPU Speed: 200MHz

Flash Memory: 4MB

RAM: 16MB

Wireless NIC: BCM4320 IEEE 802.11g MAC/baseband Wireless Controller

Antenna: Integrated

The next important component of the system is the Linux kernel and libraries. Linux kernel runs at the heart of the design and functions as a bridge between network interfaces and packages. Linux was a natural choice for this prototype system, because it runs on almost any device with maximum efficiency. The result was a robust, fast and efficient System for deploying SIP Server.

The Prototype System would be incomplete without many useful and important packages and utilities. These packages provide extended utilities which extend the functionalities of the embedded system. The OpenWRT system development kit (SDK) was used for the compilation and building of these packages. These useful packages running side by side with the SIP Server and provide handy commands for device manipulation and debugging.

In the Distributed Wireless SIP Serve design architecture, a device must first become member of LAN or WLAN network, and then it should register with the SIP Server. Such services and many others are part of Prototype System Design. These services include; dynamic IP allocation, firewall functionless, NAT traversal, routing functionalities and DNS service etc. For all these services, third party packages as discussed earlier are needed. The Prototype Systems runs service daemons and offer services to the clients in a manner shown in figure 4.

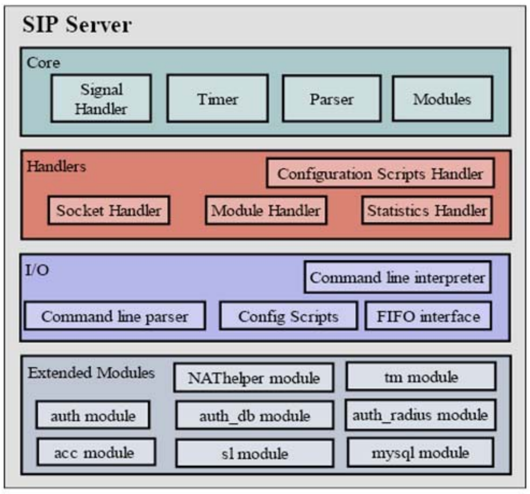


Figure 5. SIP Server Components

4. SIP Server Design

Once the prototype system is ready, the SIP server would be designed and built. The server is modular, and only required modules have been selected. The OpenWrt [3] system development kit (SDK) has been used to build our server and modules. The ipkg package installation utility of prototype system would be used to implement the SIP server on wireless device.

We have selected many third party modules to provide accounting, authentication, database access, radius authentication, distributed NAT traversal and NAT client support functionalities. These modules enable us to use centralized authentication, database storage and most importantly provide NAT Traversal for client behind NAT devices.

4.1. SIP Server Design Objective

Speed: Speed is one of the main objectives of SIP server design. The server must provide hundred of calls per second. As described in our testing chapter, the server is capable of hundreds or even thousands of calls per second even on low processing wireless device. This competitive capacity allows setting up networks which are inexpensive and easy to manage due to low number of devices required.

Flexibility: The Server allows its users to define its behavior, and server functions are defined in textual scripts, written by qualified administrator ranging from SIP routing decision to the main functions of SIP Server it self.

Extensibility: As said earlier, the server is modular in nature, and new modules can be developed to extend its functionality. This enables the developers to extend its basic function and add new extension when needed.

Portability: The server has been written in ANSI C language. It can be ported to PC/Linux, BSD and Sun Solaris platforms. This is also the reason why we have deployed it on our embedded wireless device. The ANSI C is also the key factor in server performance and speed.

Interoperability: When we talk about distributed systems, we mean the collaboration between participating parties. This is only possible, when all parties follow same standards. The server is based on the open SIP standard and is interoperable with standard SIP servers and clients. The Server is also compatible with SIP Interoperability Tests (SIPIT). Thus it is best suited for distributed environment.

Small Size: The SIP server is small in size and can best fit in our prototype system discussed in previous chapter. The Footprint of the core is 300k, and with extra add-on modules it can still fit in our 4Mb flash device aside with our Embedded Linux and other useful utilities. Even the size is small, the server should serve thousands of users per second, and our server is capable of doing so.

4.2. SIP Server Design Model

The sip server design model shown in figure 5 consists of various block elements. These blocks are core, handlers, input output and extended module. The main startup functions are defined in the man.c file which is part of core block and which takes care of the program startup. The signal handler block take care of the signals related to graceful shutdown, print server statistics etc. Some of the signals are SIGINT, SIGPIPE, SIGUSR1, SIGTERM, SIGHUP, and SIGCHLD. An internal timer takes care of the timing subsystems of the server. The command line parser parses the commands coming from the command line interpreter. The command line interpreter gets raw commands from the FIFO via standard input. The FIFO interface on the front hand takes commands and sends them to the command line interpreter or generates special signals to the Signal handler. As shown, The Server is modular in nature. There are two types of modules, core and external. The module handler gracefully handles all such modules manipulations. The server uses the statistic handler to generate and manipulate the important and useful statistics. It is the function of socket handler to deal all socket related activities. The configuration Scripts handler is vital in a sense that it does not only compile the configuration scripts on the fly, but also provide the socket, module, and statistics related information to the corresponding handlers. Based on the configuration setting defined in configuration files, the server can function in various manners. These configurations would be detailed later and particular configuration would be suggested to best suit our Wireless SIP Server infrastructure.

The design model provides a flexible plug-in model for new applications. Third parties can easily link their plug-ins with the server code and thereby provide advanced and customized services. There are many such plug-ins which were used with this model. few these are authentication, accounting, RADIUS, and mysql plug-ins. The Server is extremely configurable to allow the creation of various routing and admission policies as well as setting up new and customized services.

Based on the configuration and deployment scenario, the server includes support for registrar, proxy and redirect mode. Further it acts as an application server with support for instant messaging and presence including a 2G/SMS and Jabber gateway, a call control policy language, call number translation, private dial plans and accounting, ENUM, authorization and authentication (AAA) services. The server runs on, UNIX, Linux and Solaris and supports both IPv4 and IPv6. Hosting multiple domains and database redundancy is also supported.

4.3. Network Interfaces

The selected embedded device for the prototype system test bed support multiple network interfaces. This includes LAN, WAN and WLAN interfaces. The device has 4 LAN ports, one WAN and one WLAN interface. To better utilities all these ports and minimize the configuration headache, all these port were bridged together. The main objective of such design is that, SIP Server provides transparent service to SIP clients at all interfaces. The communication between these interfaces is straight forward. The LAN and WLAN ports are used for local SIP clients. To access the remote AAA server for authentication the WAN port is used, which can be configures for PPP, or PPPoE what ever is the case. This WAN port is also used for remote SIP communication and to access remote wireless SIP server.

5. Complete System Working Diagram

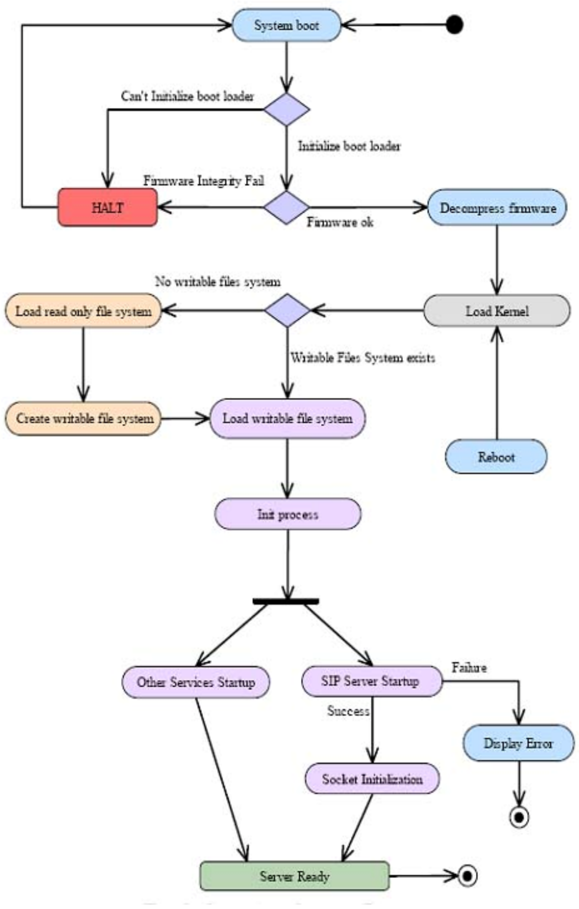


Figure 6. Complete System Diagram

Up to this point every thing for the whole architecture has been completed and system is ready for testing various performance factors and deployment cases. To complete the Distributed Wireless SIP Server design phase, the complete system working and flow diagram as shown in the figure 6 is represent.

As shown in the following figure 6, the system start with system boot up. This is for the first boot after the image has been transferred to the device. After system boot it check for the boot loader. This boot loader performs the power on self test (POST) and validated the firmware integrity. If the firmware is corrupt, the system will not proceed further and end up with halt, and a hard reboot would be required. If we continuously face this situation, we would debug our firmware and flash our device with new bug free image. Some time the image is ok but not transferred properly. As we suggested earlier, the image transfer takes enough time and the time is not constant. During our research work, we found that the device behavior is very unpredictable at this stage. So enough time should be give after firmware transfer and before hard boot. There is no way to destroy the hardware except during this firmware transfer process. Thus extra care is strongly recommended.

After firm ware integrity test pass, the image is decompressed in the memory, and the kernel is loaded. At this stage the kernel takes over the device and locates the writable file system. The kernel also loads the device drivers and performs many other tasks behind the scene. It is important to mention that the device has both writable and non writable file system.

The kernel checks the writable file system, and if this writeable file system exists, the kernel will load the file system and the configurations files from this file system. If the writable file system does not exist, the kernel loads the read only file system. The scripts would then create and load the writeable file system. After that, the init process "parent of all processes" is initiated and it starts all the other processes defined in system configuration files. As shown the init process starts SIP Server as well as other process.

The process of SIP server startup it self is more complicated than shown in the figure. The SIP server by itself has its own startup routine and performs many tasks. Upon the startup, the server initiates signal handler, parser timer and core module. The server then parses the command line parameters, following the parsing of configuration scripts. Once the configurations are know, the server starts corresponding handlers. Based on the configuration file, all relevant extended modules are loaded and configured. The server finally checks the port setting, initiate the sockets and listen on the defined port and IP address. The server may not respond or due to wrong configuration may not startup. If this happen, a debug message is printed on the screen and program quits. However the rest of the prototype system runs smoothly and only SIP server need to be fixed, repaired and restarted.

The system in normal operation may reboot due to any reason. This reason might be power failure, an interrupt signal or a reboot command. If this happen, the whole process we described is not followed, but a different path as shown in the figure completed the process. When the system is booted for the second time; instead of going through all procedure till firmware decompress, the device directly load the kernel and the rest of the procedures took place. After everything starts normally the server provides SIP calling facility to both local LAN and WLAN clients as well as to the remote WAN clients.

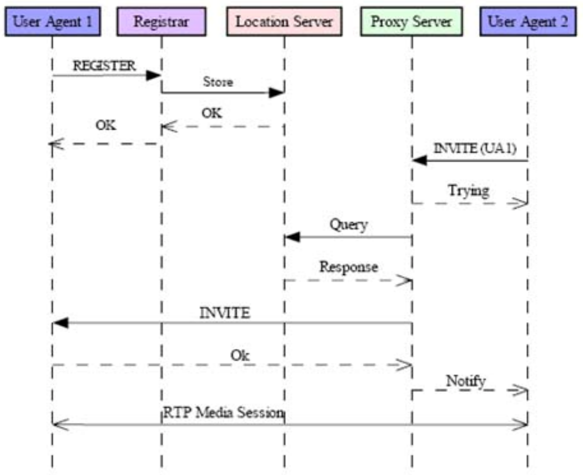


Figure 7. SIP Server Behavior

6. Server Behavior

SIP protocol specification suggests various server behaviors and a SIP server can behave in following different ways and figure7 shows the interactions between such servers, when implemented separately. It is the important design requirement of our Distribute Wireless SIP Server to provide such service in an integral way. Thus our SIP server is Registrar, Proxy and Redirect Server.

Stateless User Agent Server (UAS) Behavior: A stateless SIP Sever is a Server that does not maintain transaction state. It replies to requests normally, but discards any state that would ordinarily be retained by a server after a response has been sent. If a stateless Server receives a retransmission of a request, it regenerates the response and resends it, just as if it were replying to the first instance of the request. A Server cannot be stateless unless the request processing for that method would always result in the same response. This rules out stateless registrars, for example. Stateless Server does not use a transaction layer; they receive requests directly from the transport layer and send responses directly to the transport layer. The stateless server role is needed primarily to handle unauthenticated requests for which a challenge response is issued. If unauthenticated requests were handled state-fully, then malicious floods of unauthenticated requests could create massive amounts of transaction state that might slow or completely halt call processing in a UAS, effectively creating a denial of service condition;

Redirect Server: In some architecture it may be desirable to reduce the processing load on proxy servers that are responsible for routing requests, and improve signaling path robustness, by relying on redirection. Redirection allows servers to push routing information for a request back in a response to the client, thereby taking themselves out of the loop of further messaging for this transaction while still aiding in locating the target of the request. When the originator of the request receives the redirection, it will send a new request based on the URI(s) it has received. By propagating URIs from the core of the network to its edges, redirection allows for considerable network scalability.

A redirect server is logically constituted of a server transaction layer and a transaction user that has access to a location service of some kind. This location service is effectively a database containing mappings between a single URI and a set of one or more alternative locations at which the target of that URI can be found. A redirect server does not issue any SIP requests of its own. After receiving a request other than CANCEL, the server either refuses the request or gathers the list of alternative locations from the location service and returns a final response of class 3xx. For well-formed CANCEL requests, it SHOULD return a 2xx response. This response ends the SIP transaction. The redirect server maintains transaction state for an entire SIP transaction. It is the responsibility of clients to detect forwarding loops between redirect servers. When a redirect server returns a 3xx response to a request, it populates the list of (one or more) alternative locations into the Contact header field. An "expires" parameter to the Contact header field values may also be supplied to indicate the lifetime of the Contact data. The Contact header field contains URIs giving the new locations or user names to try, or may simply specify additional transport parameters. A 301 (Moved Permanently) or 302 (Moved Temporarily) response may also give the same location and username that was targeted by the initial request but specify additional transport parameters such as a different server or multicast address to try, or a change of SIP transport from UDP to TCP or vice versa.

However, redirect servers MUST NOT redirect a request to a URI equal to the one in the Request-URI; instead, provided that the URI does not point to itself, the server MAY proxy the request to the destination URI, or MAY reject it with a 404.

Registration: A registrar is a server that accepts REGISTER requests and places the information it receives in those requests into the location service for the domain it handles. SIP offers a discovery capability. If a user wants to initiate a session with another user, SIP must discover the current host(s) at which the destination user is reachable. This discovery process is frequently accomplished by SIP network elements such as proxy servers and redirect servers which are responsible for receiving a request, determining where to send it based on knowledge of the location of the user, and then sending it there. To do this, SIP network elements consult an abstract service known as a location service, which provides address bindings for a particular domain.

Registration creates bindings in a location service for a particular domain that associates an address-of-record URI with one or more contact addresses. Thus, when a proxy for that domain receives a request whose Request-URI matches the address-of-record, the proxy will forward the request to the contact addresses registered to that address-of-record. Registration entails sending a REGISTER request to a special type of UAS known as a registrar. A registrar acts as the front end to the location service for a domain, reading and writing mappings based on the contents of REGISTER requests. This location service is then typically consulted by a proxy server that is responsible for routing requests for that domain.

Proxy Server: A proxy server is same as the registrar. In addition Proxy server is an intermediary server that acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, possibly after translation, to other servers. A proxy interprets, and, if necessary, rewrites a request message before forwarding it

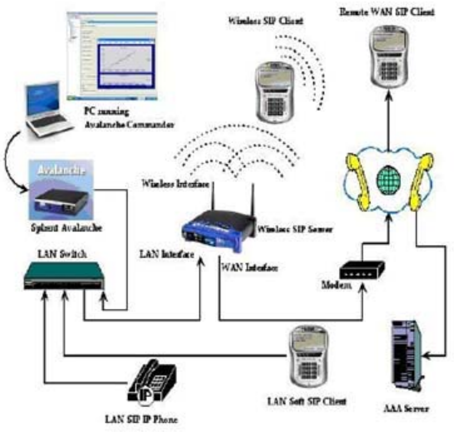
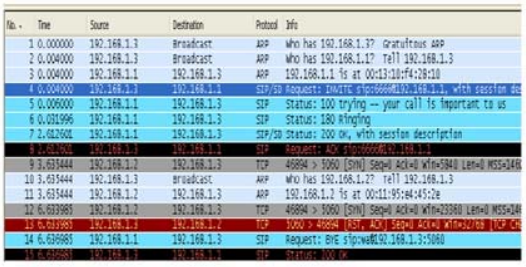


Figure 8. Test-bed Architecture



```
No. -  Time       Source        Destination   Protocol  Info
    1  0.000000   192.168.1.3   Broadcast     ARP       Who has 192.168.1.3?  Gratuitous ARP
    2  0.004000   192.168.1.3   Broadcast     ARP       Who has 192.168.1.1?  Tell 192.168.1.3
    3  0.004000   192.168.1.1   192.168.1.3   ARP       192.168.1.1 is at 00:13:10:f4:28:10
    4  0.004000   192.168.1.3   192.168.1.1   SIP/SD    Request: INVITE sip:666@192.168.1.1, with session de
    5  0.006000   192.168.1.1   192.168.1.3   SIP       Status: 100 trying -- your call is important to us
    6  0.031996   192.168.1.1   192.168.1.3   SIP       Status: 180 Ringing
    7  2.612601   192.168.1.1   192.168.1.3   SIP/SD    Status: 200 OK, with session description
    8  2.612601   192.168.1.3   192.168.1.1   SIP       Request: ACK sip:666@192.168.1.1
    9  3.635444   192.168.1.2   192.168.1.3   TCP       46894 > 5060 [SYN] Seq=0 Ack=0 Win=5840 Len=0 MSS=146
   10  3.635444   192.168.1.3   Broadcast     ARP       Who has 192.168.1.2?  Tell 192.168.1.3
   11  3.635444   192.168.1.2   192.168.1.3   ARP       192.168.1.2 is at 00:11:95:e4:45:2e
   12  6.633985   192.168.1.2   192.168.1.3   TCP       46894 > 5060 [SYN] Seq=0 Ack=0 Win=23360 Len=0 MSS=14
   13  6.633985   192.168.1.3   192.168.1.2   TCP       5060 > 46894 [RST, ACK] Seq=0 Ack=0 Win=32768 [TCP CH
   14  6.636985   192.168.1.1   192.168.1.3   SIP       Request: BYE sip:wa@192.168.1.3:5060
   15  6.636985   192.168.1.3   192.168.1.1   SIP       Status: 200 OK
```

Figure 9. SIP Packets Captured

7. Test Objectives and Methods

The system would be tested against its design targets, which was a robust, stable, secure, reliable, extendable and simple system.

A complex test bed network as shown in the above figure was created, to test different network segments and analyze the resulting data. The network is very dynamic and different possibilities exist for SIP calls. All possible connections that may establish between two SIP clients running on any network segment were tested. In addition to call establishment tests, maximum users per second were also simulated. This was done with the help of Spirent Avalanche hardware [12] and Avalanche Commander Software tool. The Ethereal Packet Analyzer was also used to calculate call establishment time and for SIP Packet analysis. This system brings patients, doctors and pharmacists together online and allows them to interact in the process of diagnosis, prescription and dispensing to the effect of increasing efficiency of medical services and rendering the medical services more user-friendly.

8. Test Results and Analysis

Following figure 9 shows the output of the Ethereal Packet Analyzer tool. Ethereal is a nice free packet analyzer tool, that can be use to capture real time network packets. The screen shot shows SIP packets captured during a LAN to LAN SIP session. For each type of network segment discussed in previous section, we captured SIP packets and analyzed accordingly. We only present on such image, while other same looking images for other network segments are not included due to massive space consumption.

The next important test was the SimUsers Per Second as shown in the following figure 10. This test was performed with the help of Avalanche hardware and Avalanche Commander Software from Spirent Communication. Starting we 1000 maximum users per second, and a step increase of 50 users and constant load, the results are show in the following figure. The light color in the graph is for desired users, and the dark color in the graph is for current users. As shown in the figure, the two graphs over lap, and are same, it is thus concluded that the server can easily sustain for 1000 users per second. Although 1000 users are much more than a WLAN can have, we still expect that there might be more SIP Requests than this number.

To find more about the SIP server behavior, we increased the users per second and performed the next test.

The following figure 11 shows the tests results of simulating maximum users per second. The user load was step increased to a maximum of 10000 users per second. After the user load of 4000, the server started malfunctioning. The server however, sustains minimum 4000 users per second without any crash or reboot, which shows its stable behavior.

The unstable behavior after 4000 users per second is mainly due to resource limitation. On average, this delicate low processing and low memory device can not handle users beyond 4000. While the SIP Server it self can handle much more users per second, if enough processing power is available. Actually 4000 users per second limitation is not supposed to be a design drawback. In fact this is a reasonable number and when many such small devices make a distributed network, the accumulative users per second would be multiple of this number.

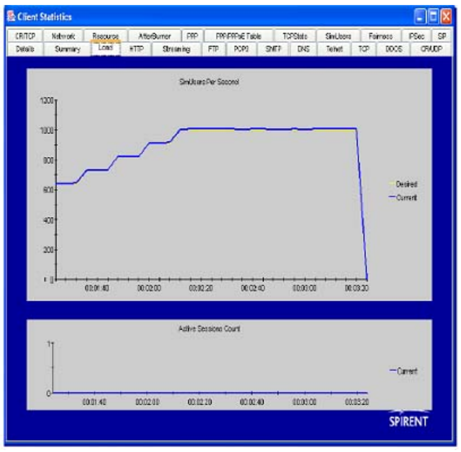


Figure 10. Output Similating Users Per Second

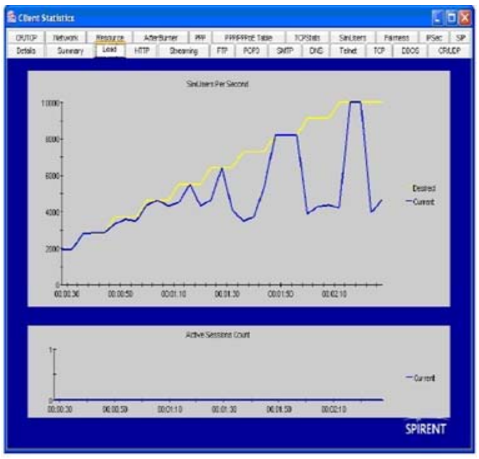


Figure 11. Step Load Test Result

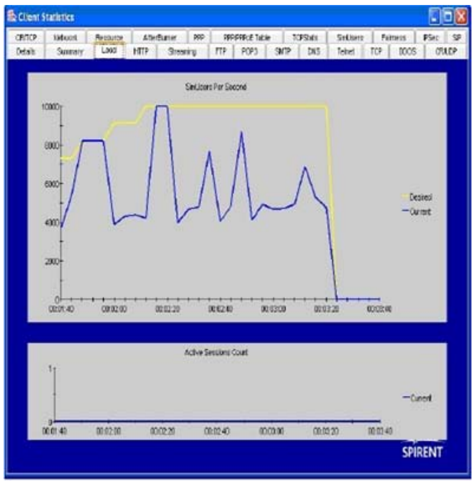
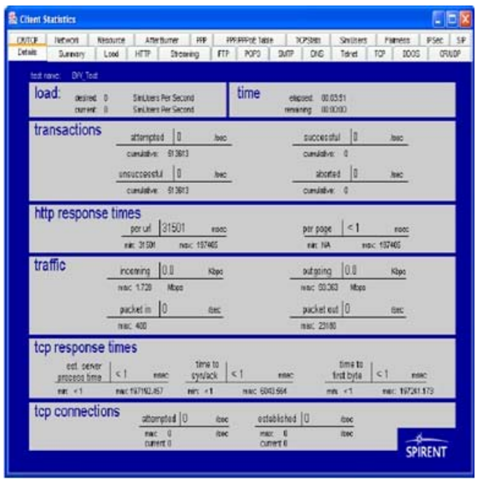


Figure 12. Constant Load Test Result



```
test name:  DIY_Test
load:   desired 0   SimUsers Per Second        time   elapsed    00:03:51
        current 0   SimUsers Per Second               remaining  00:00:00
transactions
        attempted     | 0      /sec          successful | 0      /sec
        cumulative:   513813                 cumulative:  0
        unsuccessful  | 0      /sec          aborted    | 0      /sec
        cumulative:   513813                 cumulative:  0
http response times
        per url  | 31501    msec             per page | < 1    msec
        min: 31501   max: 197405             min: NA    max: 197405
traffic
        incoming  | 0.0     Kbps             outgoing  | 0.0    Kbps
        max: 1.720   Mbps                    max: 50.363  Mbps
        packet in | 0       /sec             packet out | 0     /sec
        max: 400                             max: 23180
tcp response times
        est. server          time to            time to
        process time | < 1 msec   syn/ack | < 1 msec   first byte | < 1 msec
        min: < 1  max: 197192.457   min: < 1  max: 6043.564   min: < 1  max: 197241.573
tcp connections
        attempted | 0    /sec          established | 0    /sec
        max:  0          /sec          max:  6            /sec
        current 0                      current 6
```

Figure 13. Numerical Test Results

The following figure 12 shows the simulation users per second test result for step increase and a steady constant load of 10000. Despite the many ups and downs in the graph, the average still remains around 4000 users per second. Same resource limitation is the major cause for unstable behavior of the server.

The Avalanche Commander comes with nice statistical results interface as well. Results in statistical form let us know the exact numerical values. Following figure shows the numerical values for the above bandwidth test graph. 1.728 Mbps incoming and 93.363 Mbps outgoing traffic can be seen in the following figure. There are tags for each protocol, which were used during analysis. We can also see the load, time, transactions, http response times, traffic, TCP response etc.

In previous section, the server was tested for maximum user load using advanced testing tools. The same kind of tests should be performed for Registration Response Time under various load conditions. We also suggest that tests for the Call Setup time under various load should be performed as well using advance devices. However, these tests were not possible to be formed with the help of Avalanche testing device, so the Ethereal Packet Analyzer tool was used to calculate these values. As Ethereal not draw the required time graphs for various load condition, we manually performed the same tests again and again to get useful data. The real time and data would be different than that we presented in the following table 6, if the test are performed under various loads. It is thus requested to use other means or tool to calculate the exact values.

Manual tests using Ethereal to calculate the Registration Response Time and Call Establishment Time were also performed. The Time value has been taken from Ethereal and we have used the following formula for Registration Time, and Call Establishment time:

$$\mathbf{T_{reg} = (T_{rep} - T_{req})}$$

And $\mathbf{T_{cet} = (T_{rep} - T_{req})}$

Where $\mathbf{T_{reg}}$ **= Registration Time**

$\mathbf{T_{cet}}$ **= Call Establishment Time**

$\mathbf{T_{req}}$ **= Request Time**

$\mathbf{T_{rep}}$ **= Response Time**

Table 1. Registration and Call Establishment Time

S.No	**Registration Time (sec)**	**Call Establishment Time (sec)**
1	0.003520	0.62989
2	0.003015	0.72891
3	0.004636	1.03357
4	0.004069	0.89253
5	0.003027	1.40811
Average	0.003653	0.93860

Or in other words:

Registration Time = REGISTER Method Response Time – REGISTER Request Time

Call Establishment Time = Media Channel Establishment Time – INVITE Request Time

It is clear from the above table that registration time and call establishment time are both below a second. These values are much less than corresponding H.323 and PSTN values. We found that these values are not always same for LAN, WLAN and WAN clients. The main reason for this difference in values for different network segment is due to difference in bandwidth on different network segment. The architecture we proposed and developed is the best combination of hardware and software for low cost distributed wireless SIP application [11].

9. Conclusions and Future Work

In this research paper, a Distributed Wireless SIP Server was proposed, designed and implemented to evaluate its performance and applicability factors. It was found that the server is extremely feasible for practical low cost multimedia communication use and the proposed architecture can be deployed in various scenarios.

The architecture has three main blocks and each block is modular in nature. Keeping simple deployment factor in design model, the servers do not need any special configuration on installation site. The centralized AAA server makes the architecture easy to manage and trouble shoot.

The capacity of an IEEE 802.11b[5] network carrying voice calls depends on many factures, including varying delay constraints, channel conditions and voice call quality requirements. All these factors fall into the client specific SIP research area, the server itself has nothing to do with these choices and performance constrains.

The prototype was tested for maximum user load per second, the Registration Time, and for the Call Establishment time, using advanced network testing tools and software. These are all server dependant factors, and part of our design objectives.

Even very low processing power and cheap hardware was used as the deployment platform, yet the architecture support thousands of users per second. The Registration and the Call Establishment time both were bellow a second, which is ideal for real time media communication. However it was found that the server can handle limited 4000 users per second. Since a single server is part of a large Distributed Wireless SIP Servers architecture, where many such SIP servers are deployed, the overall user base would be multiple of this number. Hence a huge number of users per second can be entertained.

Security is one of the main concerns of any computer system including VoIP. During this research work, many security threads were observed. Due the limited time frame, we could not implemented security in our architecture. However, a secure trusted model for future research work was worked out.

It is very important for all Distributed Wireless SIP Servers to exchange critical information in a secure way. This information exchanged between participating parties includes authentication, pricing, capabilities, usage reports, etc.Open Settlement Protocols (OSP) provides specification for secure exchange of critical information. The OPS can be implemented in our architecture as show in the figure. OSP is an operational support system (OSS) protocol well suited for managing inter-domain routing, access control and accounting of SIP transactions. The content of an OSP transaction is an http message formatted according to the standards for MIME or SMIME. Individual components in the message are XML documents and the message may be signed with an S/MIME digital signature [13]. These messages are transported via http using TCP port 80 or using SSL and port 443 for secure communication.

In the future, we will try to design and implement the secure trusted peering using OSP and investigate the secure call establishment model, call termination and usage reports model.

Acknowledgements

This research work is supported by the National Natural Science Foundation of China (granted Nos. 90612004, 90412011, 60673180 and 90104022), and the Co-Funding Project of Beijing Municipal Commission of Education under granted No.SYS100060412. The authors would like to thank great support.

References

[1] Rosenberg, H. Schulzrinne, G. Camarillo, “SIP: Session Initiation Protocol”, IETF RFC 3261, 2002.

[2] OpenWrt embedded Linux,http://www.openwrt.org.

[3] Open SIP Express Router, http://www.openser.org

[4] Linux Based Device Directory, http://www.linuxdevices.com

[5] Daniel Collins, *Carrier Grade Voice over IP*, Mc. Graw Hill press, 2003

[6] M. Handley and V. Jacobson, *SDP:Session Description Protocol*, IETF RFC 2327, April 1998

[7] David P. Hole and Fouad A. Tobagi. *Capacity of an IEEE 802.11b Wireless LAN Supporting VoIP* In Proc. IEEE Conference on Communications (ICC) 2004

[8] Theo Kanter, Christian Olrog, Gerald Q. Maguire Jr. *VoIP over Wireless for Mobile Multimedia Applications,* Proceedings of the Sixth IEEE International Workshop on Mobile Multimedia Communications IEEE, ISBN 0-7803-5904-6, San Diego November 1999, pp. 179-183.

[9] Wei Li, *A Service Oriented SIP Infrastructure for Adaptive and Context-Aware Wireless Services*, Department of Computer and Systems Sciences, Royal Institute of Technology, www.ep.liu.se/ecp/011/014/ecp011014.pdf

[10] 3GPP, http://www.3gpp.org

[11] Spirent Communications. www.spirent.com

[12] IETF RFC 2802: *Digital Signatures for the v1.0 Internet Open Trading Protocol (IOTP).*

Authors

Yuebin Bai

Received his Ph.D. degree in computer architecture from Xi’an Jiaotong University, China, 2001. From 2001 to 2003, he was engaged in postdoctoral research in College of Science and Technology, Nihon University, Tokyo, Japan. In 2003, he joined the faculty of Beihang

University, China, where he is currently an associate professor in School of Computer Science and Engineering. His research focuses on ad hoc and wireless sensor networks, pervasive computing and networked embedded systems.

Syed Aminullah

Received his B.Eng. degree in electrical engineering from Balochistan University of Engg. & Technology, Khuzar, Pakistan,2000, and M.S. degree in computer science from Beihang University, China, 2006. then he joined the Huawei-3COM Co Ltd (H3C), Hangzhou, China, where he is currently an oversea product manager. His research interests include VOIP, SIP and wireless networks.

Qingmian Han

Received her B.Eng. degree in radio-technology from HeFei University of Technology, China,1989, and M.Eng. degree in signal and information system from Communication Telemetry & Telecontrol Research Institute (CTI), China Electronic Technology Group Corporation (CETC),1992. She is currently a Ph.D. candidate majored in communication and information system, in Xidian University, China, and working as a senior engineer in CTI, CETC. Her research interests include Next Generation Network (NGN) and Integrated Service Digital Network (ISDN).

Huabin Lu

Graduated in computer communications from University of Electronic Science and Technology of China., 1988. and received his M.Eng. degree in computer application from Beijing Institute of Technology , China, 1996. He worked as a director of software sector in North China Fujitsu Communication Equipment Co. Ltd. from 1998 to 2003. He is working as a senior engineer in Communication Telemetry & Telecontrol Research Institute (CTI), China Electronic Technology Group Corporation (CETC). His research interests include Next Generation Network (NGN), Integrated Service Digital Network (ISDN) and software engineering.

Chao Li

Received his B.Eng degree in Computer Science and Technology from ZhengZhou University, China.2005. He is currently reading for his Masters Degree in Software Engineering in BeiHang University, China. He is working as a M.Eng degree candidate in Sino-German Joint Software Institute (JSI). Beihang University.China His research interests include computer networks and pervasive computing.