Implementing a VoIP SIP Server and User Agent on a Bare PC

Andre L. Alexander, Alexander L. Wijesinha, and Ramesh K. Karne
Department of Computer & Information Sciences
Towson University
Towson, MD 21252
USA

***Abstract*—Bare PC applications run on ordinary desktops and laptops without the support of an operating system or kernel. They provide immunity against attacks targeting an underlying operating system, and have been shown to perform better than applications running on conventional systems due to their reduced overhead. We describe a bare PC SIP server and a SIP user agent designed for VoIP and give details of their internal implementation. The server and user agent include streamlined SIP functions and message handling, efficient CPU tasking, protocol and application intertwining, and direct Ethernet-level data manipulation. The SIP server provides registration, proxy, and redirection services, and the user agent is integrated with lean implementations of the necessary protocols within the bare PC softphone. Bare PC SIP servers and SIP softphones can be used for building secure and efficient low-cost VoIP systems, or deployed in existing VoIP networks with conventional SIP servers and user agents.**

Keywords-bare machine computing; bare PC; SIP implementation; SIP server; SIP user agent; VoIP.

I. INTRODUCTION

Telephony systems over the Internet continue to evolve with the development of new or enhanced VoIP technologies. SIP [1] is an important protocol that provides support for VoIP by handling functions such as call set up, user authentication, user registration and location, and billing support. Although SIP is a general-purpose protocol that can also be used for video conferencing, instant messaging and gaming, it is predominantly used today in VoIP systems. Conventional SIP implementations in servers and softphones require the support of a traditional operating system (OS) such as Windows or Linux, or an OS kernel. SIP phones are also frequently implemented in hardware/firmware typically with an embedded OS. The SIP implementations in OS-based systems take advantage of their rich supporting environment and capabilities and are convenient to use.

However, an OS-based full SIP implementation is not always needed. If a higher level of security or performance is desired at low cost, a customized SIP server or a SIP softphone running on a bare PC (ordinary desktop or laptop without a conventional OS or kernel) would be more easily secured or designed for high performance. For example, an OS-based system may be difficult to secure against attacks that target vulnerabilities of the underlying OS. Bare PC systems are inherently immune against such attacks since they have no OS. In addition, compared to their OS-based counterparts, they also have reduced code complexity and code size, making it easier to analyze their code for security flaws. Moreover, due to their simplicity and the limited services they offer, they also have fewer avenues open for attackers to exploit. Also, studies of bare PC Web servers [2] and email servers [3] have shown that they perform better than their OS counterparts.

Thus, a SIP server or SIP user agent running on a bare PC can be expected to provide secure and efficient low-cost operation. Moreover, since there is no OS, lean versions of the necessary protocols can be intertwined with the bare PC SIP server or SIP softphone application to reduce the overhead of inter-layer communication and improve performance. A preliminary performance study of the bare PC SIP server (see the related work section below) confirms that it performs better than Linux or Windows-based SIP servers with very few exceptions.

In this paper, we describe the design and implementation of a bare PC SIP server and a bare PC SIP UA. In particular, we discuss the details of SIP operations, message handling, and task structure on the bare PC SIP systems. We also examine possible causes for the few performance bottlenecks identified in the bare PC SIP server performance study and note possible future design improvements.

As with other bare PC applications, the SIP server or user agent implementation and interfaces to the hardware constitute a single self-contained executable. The SIP UA is also integrated with the bare PC softphone. The bare PC SIP server and SIP UA implement only the key elements of SIP and have minimal functionality compared to conventional OS-based SIP servers and SIP UAs. Also, the SIP implementations are UDP-based, and the server is stateless. A SIP server implementation over TCP is under development. The bare PC SIP server and bare PC SIP softphone currently run on an IA32 (Intel Architecture 32-bit) or Intel 64-bit architecture in 32-bit mode. They can be used for building secure or high-performance bare PC-only VoIP networks, or interoperate with conventional OS-based SIP servers and SIP softphones as discussed below in the section on testing.

The rest of this paper is organized as follows. In Section II, we briefly survey related work. In Section III, we give an overview of bare machine computing. In Sections IV and V respectively, we describe the design and implementation of the bare PC SIP server and UA. In Section VI, we discuss testing of the SIP server and UAs. In Section VII, we present the conclusion.

II. RELATED WORK

There are numerous implementations of conventional SIP servers and SIP softphones on various OS platforms. These SIP servers and UAs run on conventional OSs. In [4], a SIP

Copyright (c) IARIA, 2010 ISBN: 978-1-61208-107-6

server is implemented on top of an existing SIP stack. In [5], SIP servers and SIP UAs are implemented on the Solaris 8 OS. A client-side SIP service offered to all applications based on a low-level SIP API is described in [6]. In [7], the features of a new language StratoSIP for programming UAs that can act respectively as a UA server to one endpoint and as a UA client to another are presented. In [8], the UA is a SIP-based collaborative tool implemented by using existing SIP and SDP stacks. In [9], a Java-based SIP UA is proposed for monitoring manufacturing systems over the Internet. The focus of [10] is a SIP adaptor for both traditional SIP telephony and user lookup on a P2P network that does not have a SIP server. The goal of such SIP servers and SIP UAs is to offer enhanced services to clients by using existing low-level SIP stacks that rely on an OS. In contrast, bare PC SIP servers and UAs that are implemented directly on the hardware will have less overhead and are more suited for secure low-cost environments.

Intertwining bare PC Web server or email server application and application protocols (i.e., HTTP or SMTP) with the TCP protocol contributes to its improved performance over OS-based servers [2, 3]. In [11], the performance of a bare PC SIP server is compared with that of OS-based servers, and it is shown that the bare PC server performs better except in a few cases. It is likely that the performance drops are due to using a simple (non-optimized) search algorithm for user lookup as discussed in the section on server implementation below. The performance study did not discuss the SIP server design details or its implementation. The design, implementation, and performance of a bare PC softphone are discussed in [12, 13]. However, the softphone does not include a SIP UA and hence lacks the ability to communicate with SIP servers and set up calls with other SIP softphones.

III. BARE MACHINE COMPUTING

Bare PC application development is based on the bare machine computing paradigm, also referred to as the dispersed operating system (DOSC) paradigm [14]. In this paradigm, a single self-supporting application object (AO) encapsulating all of the necessary functionality for a few (typically one or two) applications executes on the hardware without an OS. Bare machine applications only use real memory; a hard disk is not used. The AO, which is loaded from a USB flash drive or other portable storage medium, includes the application and boot code. The application code is intertwined with lean implementations of the necessary network and security protocols. If required by the application, the AO also includes cryptographic algorithms, as well as network interface and other device drivers, such as an audio driver in case of the bare PC softphone. The interfaces enabling the application to communicate with the hardware [15] are also included in the AO. The AO code is written in C++ with the exception of some low-level assembler code. The AO itself manages the resources in a bare machine including the CPU and memory. For example, every bare PC AO has a main task that runs whenever no other task is running, and network applications require a Receive (Rcv) task that handles incoming packets. Additional tasks may be used depending on the applications included in the AO, such as an audio task for the bare PC softphone.

IV. BARE PC SIP SERVER IMPLEMENTATION

The bare PC SIP server supports registrar, redirector, or proxy modes with or without authentication. The server is designed in a modular fashion to allow for easy updates and implementation of new features, and to facilitate analysis of the server code. Since the bare PC SIP server implementation is lean, only specific content from an incoming SIP packet is parsed. The bare PC SIP server AO contains about 2000 lines of code.

A. Boot Sequence

The bare PC SIP server is booted by directly loading its AO from a USB flash drive. The protocol/task relationships for the server are shown in Fig. 1. The bare PC SIP Server boot sequence begins when the Main task invokes the DHCP handler to send a DHCP request for an IP address (unless the server has been preconfigured to use a specific IP address). When a response arrives, the Rcv task is invoked to process it. Next, a file containing username and password combinations of authorized users is transferred from another host on the network using an adaptation of trivial FTP. As discussed later, multiple data structures to facilitate server operations such as user lookup, username and password lookup, and state lookup are then created in memory. The last step in the boot process is to display the user interface for administering the server.

B. SIP Server Internals

The bare PC SIP server uses only two CPU tasks, Main and Receive (Rcv), which simplifies task management and increases efficiency. The Main task runs continually and activates the Rcv task whenever packets arrive in the Ethernet buffer and need to be processed. After a response is sent, the Rcv task terminates and the Main task runs again. For example, when the SIP Server AO's Rcv task is activated by the Main task upon the arrival of a SIP request in the Ethernet buffer, a single thread of execution handles the request all the way from the Ethernet level to the SIP (application) level till a response is sent, which simplifies server design and reduces the processing overhead. Thus, if an arriving packet is designated for the default SIP UDP port 5060, the Rcv task causes the Ethernet, IP, and UDP handlers to be invoked to process the respective protocol headers using a single copy of the message. As shown in Fig. 1, the Rcv task only terminates after the SIP request is processed and a SIP response is sent by the server (after invoking the respective protocol handlers to attach the headers).

The bare PC SIP server AO consists of several objects. In addition to the Ethernet, IP, UDP, and SIP objects, the server also requires the DHCP, FTP, and MD5 objects. The role of the DHCP and FTP objects were discussed earlier. The MD5 object is used to provide support for user authentication via standard SIP authentication (i.e., HTTP-Authentication) if it is needed.

Copyright (c) IARIA, 2010 ISBN: 978-1-61208-107-6

C. User Database Lookup

After the usernames and passwords from the file are read into memory, the bare PC SIP server runs the sipservergetdb() function to store them in the USER_DATABASE structure:

```
Struct USER_DATABASE {
char username [20];
int username_size;
int username_hash;
char Password [20];
int Password_size;
};
```

The data structures HASH_TABLE and SORTED_TABLE shown below are also used.

```
Struct HASH_TABLE {
int hash_hit;
int hash_reg_db_loc[HASH_REG_DB_SIZE];
int hash_hit_size
};
Struct SORTED_TABLE {
int hash;
int hash_link;
};
```

In essence, the hash of each username is then used as an index into HASH_TABLE, which is used together with SORTED_TABLE to facilitate looking up the user in the USER_DATABASE structure, and retrieving information when making or receiving calls, or registering a user. The HASH_TABLE structure links back to the SORTED_TABLE and USER_DATABASE structures. The details are as follows. First, the hash values are stored in a SORTED_TABLE array (which allows for efficient searching for a given hash value), and each position in the sorted array is linked to the specific HASH_TABLE array corresponding to that hash value. In turn, each position in the HASH_TABLE array corresponds to a user that hashed to that value and contains a link back to the USER_DATABASE entry for that user. The HASH_TABLE structure links the index in the USER_DATABASE structure to the hash value of the SORTED_TABLE as shown in Fig. 2.

The user lookup process in Fig. 3 is done by using two functions: the find_hash_hit() function, which is based on a particular hash value, and the find_user() function that is based on the username and size. In performance tests, this search operation was found to be a likely bottleneck because of the username comparisons triggered by collisions on a single hash value. The find_user() function takes a username and username size as input. It then hashes the username and passes the value to the find_hash_hit() function, which finds the corresponding hash table containing all the users with that same hash value. The hash table is passed back to the find_user() function, which calls the lookup_user() function. The latter goes through each user in that specific hash table and first compares the sizes of the usernames; if they match, it looks for a second match on the full username. If the user is found, the location containing the user's information in the database, including the IP Address and port, is returned. To improve performance, future bare PC SIP server implementations will use adaptations of data structures and search techniques used by popular Linux SIP servers.

D. SIP Message Processing

The siphandler() function manages the processing of received SIP messages. This function, which is called directly by the udp_handler() function after verifying the SIP port in the UDP header, is the key element in the bare PC SIP server. The siphandler() function calls the parse_headers() function which goes through the SIP packet and parses out specific identifiers to identify the type of message (for example, REGISTER, INVITE, ACK, BYE, 180 Ringing, 200 OK and 100 Trying). Within the parse_headers() function are specific functions built to handle the following SIP tags: Header, Via, From, To, Expires, Authorization, Proxy Authorization, CallId, CSeq, Contact, and Content Length. In keeping with the lean SIP implementation, only the indicated tags are parsed to expedite the processing of SIP packets (other tags are bypassed). Once the tags are parsed and the relevant data from the packet is stored, control returns to the siphandler() function. Further processing is determined according to the request_type returned. Only the following SIP messages are routable by the Bare PC SIP Server: Register Invite, 100 Trying, 180 Ringing, 200 OK, Ack, Bye, and Unsupported. When the system (the siphandler function) has decided what to do with the SIP request, processing is carried out to forward the SIP message, or a reply is sent to the SIP User Agent by utilizing the generate_sip_response() function. This function generates the SIP reply (or 100 Trying response) based on the values retrieved earlier by parsing the SIP request. It then calls the sipsenddata() function, which calls the relevant protocol handlers to format the headers in the SIP reply.

Register Message: To process a Register message, the bare PC SIP server parses the Via (IP address:port), From and To (usernames@domain/IP), and Contact tags. It then calls the function check_registered_users(). A process similar to that described earlier is used to determine if the user is already registered (i.e., is found in the Registered_Users_Database). If so, only the relevant information is updated; otherwise, the system stores all necessary information parsed from the SIP request including the username, IP address and port number. This information is used to generate replies back to the UA on future requests until the UA re-registers or one of the parameters is updated. After the information is stored or updated, the server generates a 200 OK message and sends the reply back to the SIP UA.

Invite Message: For an Invite message, the bare PC SIP server parses almost all of the same fields as for the Register message. The server then sends messages to the caller and callee. A 100 Trying message is sent back to the caller letting the UA know that the SIP Server is processing the request. To send this message, the server looks up the IP address of the caller using the process described earlier. It also looks up the registration information for the callee and forwards the Invite message to its UA.

Copyright (c) IARIA, 2010 ISBN: 978-1-61208-107-6

SIP Authentication: The Message format for an Invite request with authentication is shown in Fig. 4. SIP authentication is done by challenging the initial request (Invite or Register) sent by the SIP UA. SIP uses HTTP authentication techniques. The bare PC SIP Server is designed so that each request is not authorized unless it receives the proper response for a given challenge. The server can be configured at start-up to operate with or without authentication. An authorization flag indicates if a particular request is approved or denied based on authentication. The bare PC SIP server processes the initial request, and then sends a challenge response back to the requesting SIP UA. The SIP server generates a challenge response that depends on the values of realm and nonce. The realm is typically set to the domain of the SIP server (for example, barepc.towson.edu or the IP address). The nonce is a string that is randomly generated by the server. Once the server receives the reply to the challenge, the fields in the authorization request are parsed from the SIP packet. Then the response value is computed using the MD5 algorithm and matched against the response value sent by the SIP UA. The response value is a hash that depends on the concatenation of all values in the authorization request. If the computed response matches the response sent by the SIP UA, the request is approved (authorized) and normal SIP call flow processing is allowed.

E. User Interface

The bare PC SIP Server has a simple user interface that displays its basic configuration and state information when the interface function sipserverstate() is called. The displayed information includes the number of users added to the username and password database, and the server's configuration mode (proxy, redirector, authentication, stateless, or stateful). The server can also show the username, ip address, and port for each user logged into the system. An administrator can toggle through the list of users, or configure the server so that the display is triggered every time a user is added or removed from the Registered_User_Database by calling sipserverstate() from the Main task.

V. BARE PC SIP UA IMPLEMENTATION

The bare PC SIP user agent (UA) is integrated with the bare PC softphone enabling calls to be set up. Its operational characteristics are similar to those of a SIP UA in a conventional OS-based SIP softphone. However, the UA implementation is different due to the absence of an OS and a built-in protocol stack, and results in a UA with less overhead and better security. The UA can also directly communicate with a peer (without using a SIP server) provided the peer can be contacted via a known (public) destination IP address and port number.

A. UA Operation/User Interface

As in the case of the bare PC SIP server, only two tasks Main and Rcv are needed for the UA, and arriving SIP messages and responses are processed in a single thread of execution as described earlier. When the UA is booted, if an IP address for the UA has not been preconfigured, the UA sends out a request for an IP address and obtains one using DHCP. If this is a private address, the UA is behind a NAT and uses STUN [16] to learn its public IP address and port. In this case, the UA first sends a DNS request and obtains the IP address of a public STUN server. The bare PC STUN implementation is described in more detail below.

After UA completes the initialization process it displays the main login menu, which enables the user to login-in to a particular SIP server or to communicate directly with a peer as noted earlier. In case SIP server login is selected, the UA sends a SIP Register request to the server after performing a DNS resolution if needed. Once the 200 OK messages are received from the SIP server, the UA displays a "main menu" screen as in Fig. 5. The menu has several options, which enables the user to see the IP configuration information from DHCP, and NAT mappings from STUN that show the external IP address and internal/external SIP and RTP ports for the softphone. Such information is useful to troubleshoot connectivity problems. In addition, a separate option shows call status and connectivity information, and indicates whether security is on. A "quick dial" option for selecting specific users is also available.

The software design of the bare PC SIP UA is simple and modular. The essential UA functionality contained in the SIPUA object consists of 3000 lines of C++ code. This object is supplemented by 1) objects for cryptographic and other algorithms needed for key establishment (HMAC, SHA-1, MD5, AES, and Base64); 2) objects implementing the essential elements of the necessary auxiliary protocols (STUN, DHCP, and DNS); and 3) objects needed by the bare PC softphone including the Ethernet, IP, and UDP objects, the RTP, audio, and G.711 objects that handle voice data processing, recording, and playback on the bare PC softphone, and the SRTP [17] object that provides VoIP security.

B. User Agent Client and User Agent Server

The bare UA consists of two independent components: the SIP user agent server (UAS) and SIP user agent client (UAC). The UAS is operationally similar to the bare PC SIP server with respect to its handling of SIP packets. For example, it listens for call requests and its actions are activated by the Rcv task when a packet arrives as discussed earlier for the case of the SIP server. The UAC can be activated by keyboard input. The UA functionality is contained in a SIPUA object that is responsible for processing SIP messages and SDP tags, displaying the SIP UA interface, and interacting with the user. The SIPUA object is integrated in a single AO with several other objects needed to implement the UA.

C. STUN/DHCP/DNS/SRTP

The public IP address and port learned from the public STUN server is used in SIP Invite requests to enable the peer to communicate with the UA behind the NAT. The bare PC SIP UA sends out multiple STUN messages to find the external port for its voice channel over RTP. Since the signaling channel is proxied through the SIP server, STUN

Copyright (c) IARIA, 2010 ISBN: 978-1-61208-107-6

is not needed to discover the external SIP signaling port. After the bare PC client is booted, STUN messages for the media channel are sent every 30 seconds until the SIP UA establishes the call. The Invite message contains the last known media channel external port number. Since the NAT binding may change, the UA sends voice packets to the destination host using a sequence of consecutive ports. The UA stops sending on the other ports once voice packets are received on a particular port.

Since there is no OS and no built-in protocol stack on the bare PC softphone, the bare PC SIP UA also needs to send DHCP messages to automatically obtain an IP address and other essential configuration information at start-up. The DHCP messages follow the typical DHCP call flow (Discover, Offer, Request, and Ack). The softphone can also send DNS requests to resolve the domain name of the SIP or STUN server. As noted earlier, the implementation of the DHCP and DNS protocols have only the minimal features needed by the bare PC SIP softphone.

The bare PC SIP UA is also integrated with SRTP. The implementation and performance of SRTP on a bare PC softphone are presented in [18]. SRTP allows the UA to communicate securely with conventional SIP UAs that are SRTP capable. The bare PC softphone AO includes implementations of SHA-1, MD5, HMAC, and AES in counter mode, which are used by SRTP. The bare PC SRTP implementation also supports addition of a recommended authentication tag to the end of the RTP packet. The UA currently implements the SDP Offer/Answer model via SDES for key exchange. This method is used by several conventional SRTP clients. The keys used to generate the session keys are Base64 encoded by the bare PC softphone SRTP implementation prior to transmission. Since this approach for transmitting keys is not secure, TLS is used by some conventional softphones for SIP signaling.

VI. TESTING

Operational tests of the bare PC SIP server and SIP softphone implementations with and without authentication and SRTP security were conducted using Dell GX-260 desktops with Intel Pentium 4 2.4 GHz processors, 1.0 GB RAM, and a 3COM Ethernet 10/100 PCI network card. The test network consists of a dedicated LAN within the Towson University network, and an external network connected through an ISP as shown in Fig. 6. The bare PC SIP server and user agents were first tested within the dedicated LAN. Testing was performed to verify 1) correct operation between the bare PC SIP server and bare PC SIP softphones; 2) interoperability of bare PC SIP softphones with the OpenSer v3.0.0 server [19]; 3) interoperability of the bare PC SIP server with Snom360-5.3 softphones [20]; and 4) interoperability of bare PC SIP softphones with the Snom softphones.

Similar tests were conducted over the Internet by establishing calls between a softphone on the external network and another on the dedicated LAN when the SIP servers are connected to the LAN. These tests also served to verify that the UA and the lean DHCP, STUN, and DNS implementations on the bare PC SIP softphone work correctly when it is connected to the Internet. In particular, the bare PC STUN implementation was found to be adequate for connecting between clients behind NATs on the dedicated test LAN and on an ISP network.

VII. CONCLUSION

We described the design, implementation, and operations of a bare PC SIP server and SIP UA, which provide essential SIP functionality with less overhead and better security at lower cost due to the absence of an OS. The underlying bare PC system enables the SIP server and UA to benefit from simple tasking, lean protocols, and efficient data handling. The tests conducted show that the bare PC SIP server can interoperate with both bare PC and OS-based UAs, and also that the bare PC SIP UA can interoperate with both an OS-based UA and an OS-based SIP server.

REFERENCES

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

[2] L. He, R. Karne, and A. Wijesinha, “The Design and Performance of a Bare PC Web Server”, International Journal of Computers and Their Applications, vol. 15, pp. 100 - 112, June 2008.

[3] G. Ford, R. Karne, A. Wijesinha, and P. Appiah-Kubi, “The Peformance of a Bare email server”, 21st International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2009, October 28-31, Sao Paulo, Brazil, pp.143-150, 2009.

[4] L. Chen, and C. Li, “Design and Implementation of the Network Server Based on SIP Communication Protocol,” World Academy of Science, Engineering and Technology 31, pp. 138-141, 2007.

[5] S. Zeadally and F. Siddiqui, “Design and Implementation of a SIP-based VoIP Architecture,” AINA 2004.

[6] A. Singh, A. Acharya, P. Mahadeva, and Z-Y, Shae, **“**SPLAT: a unified SIP services platform for VoIP applications,” International Journal of Communication Systems, Volume 19, Issue 4, pp. 425-444, 2006.

[7] P. Zave, E. Cheung, G. W. Bond, and T. M. Smith, “Abstractions for Programming SIP Back-to-Back User Agents,” IPTComm’09, 2009.

[8] S Siddique, RK Ege, SM Sadjadi, “X-Communicator: Implementing an advanced adaptive SIP-based User Agent for Multimedia Communication,” SoutheastCon, pp. 271-276, 2005.

[9] K. J. Kim, Y, Jang, J. W. Chung, and J. H. Seo, “Design and implementation of SIP UA for a manufacturing network,” International Journal of Advanced Manufacturing Techniques, Volume 28, Number 7-8, pp. 822-826, 2006.

[10] K. Singh and H. Schulzrinne, “Peer-to-Peer Internet Telephony using SIP,” International Workshop on Network and Operating System Support for Digital Audio and Video, pp. 63-68, 2005.

[11] A. Alexander, A. L. Wijesinha, and R. Karne, “A Study of Bare PC SIP Server Performance,” 5th International Conference on Systems and Network Communications, ICSNC 2010, In Press.

[12] G. H. Khaksari, A. L. Wijesinha, R. K. Karne, L. He, and S. Girumala, “A Peer-to-Peer bare PC VoIP Application,” IEEE Consumer and Communications and Networking Conference (CCNC 2007), pp. 803-807, 2007.

[13] G. H. Khaksari, A. L. Wijesinha, R. Karne, Q. Yao, and K. Parikh, “A VoIP Softphone on a Bare PC”, Embedded Systems and Applications Conference (ESA), 2007.

[14] R. K. Karne, K. V. Jaganathan, T. Ahmed, and N. Rosa, “DOSC: Dispersed Operating System Computing,” 20th Annual ACM Conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA ’05), Onward Track, pp. 55-61, 2005.

Copyright (c) IARIA, 2010 ISBN: 978-1-61208-107-6

[15] R. K. Karne, K. V. Jaganathan, and T. Ahmed, "How to Run C++ Applications on a Bare PC?" 6th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2005), pp. 50-55, 2005.

[16] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," RFC 5389, 2008.

[17] M. Baugher, D. McGrew, M. Naslund, E. Carrara and K.Norrman, "The Secure Real-time Transport Protocol (SRTP)," RFC 3711, 2004.

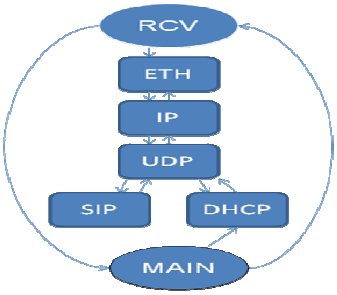

Figure 1. SIP Server Protocol/Task Relationships.

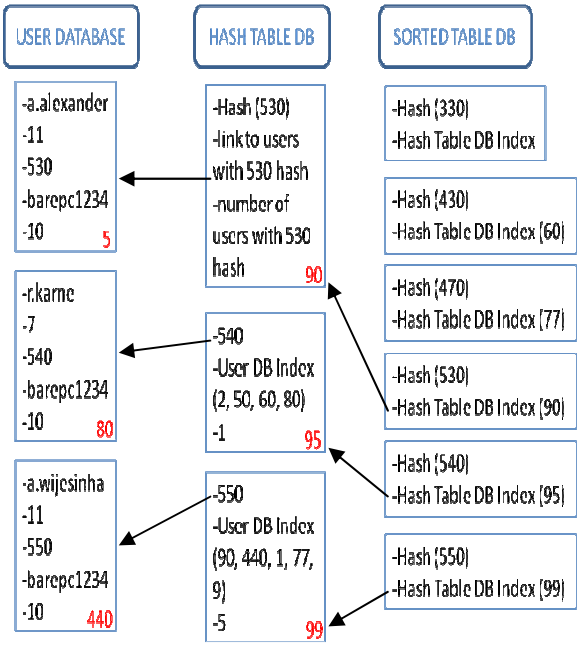

Figure 2. Database and Hash Table Relationships.

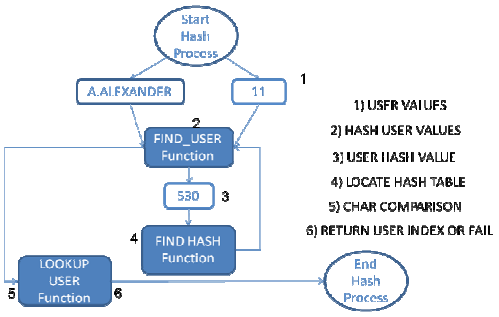

Figure 3. User Lookup Process.

[18] A. Alexander, A. L. Wijesinha, and R. Karne, "An Evaluation of Secure Real-Time Protocol (SRTP) Performance for VoIP," 3rd International Conference on Network and System Security (NSS), pp. 95-101, 2009.

[19] Kamailio (OpenSER) SIP server, http://sourceforge.net/projects/openser

[20] Snom VoIP phones, http://www.snom.com/download/snom360-5.3.exe

```
INVITE sip:67890111@barepc.towson.edu:5060 SIP/2.0
Via:SIP/2.0/UDP192.168.1.56:5060;brach=0320
From:<sip:0123456@ barepc.towson.edu>;tag=0
To: <sip: 67890111@ barepc.towson.edu>
Max-Forwards: 70
Call-ID: 0010-0003-DA76506F-0@AAE2A42DF82D1D0AA
 CSeq: 297386 INVITE
Contact: <sip:123456@192.168.1.56:5060>
Content-Type: application/sdp
Proxy-Authorization:Digest
username="8000",realm="BAREPC",nonce="3bd76584",
uri="sip:123456@192.168.2.81",response="6e91de67ad976997
ff"
User-Agent: BarePC SIP UA v1.0
Content-Length: 276
v=0
 o=Vega400 4 1 IN IP4 192.168.1.56
 s=Bare PC Sip Call
 t=0 0
 m=audio 10006 RTP/AVP 4 18 8 0 96
 c=IN IP4 192.168.1.56
 a=rtpmap:8 PCMA/8000
 a=rtpmap:0 PCMU/8000
 a=rtpmap:96 telephone-event/8000
 a=fmtp:96 0-15,16
 a=sendrecv
```

Figure 4. SIP Invite with Authentication.

Figure 5. UA Main Menu Screen.

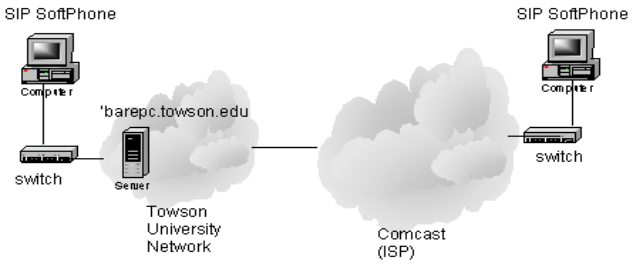

Figure 6. Test Network.