TOWSON UNIVERSITY
COLLEGE OF GRADUATE STUDIES AND RESEARCH

A SIP SERVER AND USER AGENT WITH SRTP FOR VoIP ON A BARE PC

By

Andre Alexander

A Dissertation

Presented to the faculty of

Towson University

in partial fulfillment

of the requirements for the degree

Doctor of Science

August 2010

Towson University
Towson, Maryland 21252

TOWSON UNIVERSITY
COLLEGE OF GRADUATE STUDIES AND RESEARCH

DISSERTATION APPROVAL PAGE

This is to certify that the dissertation prepared by **ANDRE ALEXANDER** entitled **A SIP SERVER AND USER AGENT WITH SRTP FOR VoIP ON A BARE PC** has been approved by his or her committee as satisfactory completions of the dissertation requirement for the degree **DOCTOR OF SCIENCE.**

Dr. Alexander Wijesinha Chair, Thesis Committee	Signature	8/24/10 Date
Dr. Ramesh Karne Committee Member	Signature	8-24-10 Date
Dr. Yanggon Kim Committee Member	Signature	Aug. 24, 2010 Date
Dr. Yeong-Tae Song Committee Member	Signature	8/30/2010 Date
Dr. Marius Zimand Committee Member	Signature	8/30/2010 Date
Dr. Chao Lu Dean, College of Graduate Studies and Research	Signature	9/21/10 Date

ACKNOWLEDGMENTS

Thank you to my advisors Dr. Wijesinha and Dr. Karne for their guidance and support during this journey; they are truly responsible for the successful completion of my dissertation. I am also deeply indebted to Dr. Zimand, Dr. Song and Dr. Kim for supporting this research. I also thank my family and friends for their support and words of wisdom. Finally, to Anna Alexander (my wife), Anaya Alexander (my daughter), Benzette Alexander-Fields (my mother) and Juanita Alexander (my grandmother), I dedicate this degree to you; your constant support, loving words, thoughtful gestures and sacrifices have kept me focused. Thank You! This moment marks the beginning of a wonderful future for the Alexander family and shows that with hard work and faith anything is possible!

"Ask and it will be given to you; seek and you will find; knock and the door will be opened to you." Matthew. 7:7

Andre Alexander

ABSTRACT

A SIP SERVER AND USER AGENT WITH SRTP FOR VoIP ON A BARE PC

Andre L. Alexander

Bare PC applications run on ordinary desktops and laptops without the support of an operating system (OS) or kernel. They provide immunity against attacks targeting an underlying OS, and have been shown to perform better than applications running on conventional systems due to their reduced overhead. In this dissertation, we describe a SIP server and user agent (UA) with SRTP that are designed for VoIP on a bare PC. We give details of their implementation and present experimental results evaluating their performance.

The server and UA include streamlined SIP functions and message handling, efficient CPU tasking, protocol and application intertwining, and direct Ethernet-level data manipulation. In particular, the server provides registration, proxy, and redirection services, and the UA is integrated with lean implementations of the necessary protocols within the bare PC softphone.

We evaluate the performance of the bare PC SIP server by determining its throughput and latency in a dedicated test network with and without authentication. We also report internal timings for the server. The server's performance is compared with that of the OpenSER and Brekeke SIP servers running on Linux and Windows respectively. Our results show that the bare PC SIP server has low cost for internal SIP-related operations, and higher throughput and lower latency than the OS-based servers except in a few cases that need further optimization.

We also implement SRTP to secure VoIP conversations on a bare PC softphone. Experiments to evaluate UA performance with SRTP are conducted using the bare PC softphone, and Twinkle and snom softphones running on Linux and Windows respectively. Pre-defined SRTP transforms based on AES counter mode encryption with HMAC-SHA-1 authentication are tested. Measured internal timings for SRTP operations indicate that authentication is more expensive than encryption regardless of key or tag size. Measured values of jitter, delta (packet interarrival time) and throughput show that the addition of SRTP protection to VoIP traffic over RTP has a negligible effect on voice quality.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I. INTRODUCTION

VoIP (Voice over Internet Protocol) is one of today's most researched telephony topics. The evolution of high-speed networks and data compression techniques has enabled VoIP to make influential changes to the technology landscape. Various studies have focused on the security and performance of VoIP in both real world and test environments.

SIP (Session Initiation Protocol) [1] and SRTP (Secure Real-time Protocol) [2] are two support protocols frequently used by VoIP systems. SIP is implemented in servers (known as SIP servers) and softphones (in the form of user agents or UAs), and is used for call setup and other call management functions. SRTP is implemented in softphones, and is used for encryption and authentication/integrity of voice data carried over RTP (Real-time Transport Protocol). An overview of SIP and SRTP is given at the end of this chapter.

Conventional VoIP systems require the support of an operating system (OS) or kernel. Bare Machine Computing (BMC) is an alternative approach to computing that enables applications to run on a bare machine with no OS or kernel support i.e., a bare PC. Bare PC applications are characterized by simplicity and efficiency, and have been shown to perform better than similar applications running on an OS-based system [3, 4, 5]. They also have inherent security advantages due to being immune to OS-based attacks. An overview of BMC systems is given in the next chapter.

The preceding considerations motivated us to build VoIP systems that support SIP and SRTP. Specifically, we design and implement a bare PC SIP server and SRTP with SIP UA support on a bare PC softphone. We also evaluate the performance of these implementations by measuring the throughput and latency of the SIP server, and the delay, jitter, and throughput (that serve as call quality metrics) for the voice data generated by the softphone. We compare performance for the bare PC systems with similar OS-based systems, and also measure the internal timings for key SIP and SRTP functions on the bare PC.

All components of SIP on the bare PC SIP server are implemented as part of this doctoral work. Also, we add new SIP UA and SRTP implementations to an existing bare PC softphone having only RTP/UDP/IP/Ethernet functionality. Furthermore, we add dynamic IP address configuration via DHCP (to the SIP server and softphone), NAT traversal via STUN, and domain name resolution via DNS to the softphone to facilitate plug-and-play capability. In keeping with the BMC approach, all protocols are implemented in a lean manner i.e., only essential functionality is implemented. Tests are conducted to verify correct operation of SIP, SRTP and the auxiliary protocols, as well as the interoperability of the SIP server and SRTP/SIP UA softphone with compatible OS-based systems.

A. *SIP Overview*

SIP 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 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. An optimized SIP server can help improve the overall performance of audio or video applications even though it is typically not directly involved in the actual transmission of audio or video. The throughput and latency of the SIP server when responding to requests from SIP user agent clients and other SIP servers are used as measures in evaluating its performance.

B. SRTP Overview

SRTP is an Internet standards-track profile of RTP that provides a framework for securing VoIP communications. The primary security considerations for VoIP are voice encryption, voice data authentication and integrity, and replay protection.

SRTP addresses these security aspects by providing security for RTP and its control protocol RTCP with low overhead. It can be used for encryption, message authentication/integrity and replay protection of RTP and RTCP traffic. While SRTP mandates message authentication for RTCP and adds new fields to an RTCP packet, we do not consider SRTP performance with respect to RTCP in our study since the overhead due to securing the periodic but infrequent RTCP messages is negligible.

The remainder of the dissertation is structured as follows. Chapter 2 contains a survey of related work on SIP and SRTP performance and implementation, and an overview of Bare Machine Computing (BMC). Chapter 3 describes the design and implementation of the SIP Server and SIP User Agent with SRTP, and the supporting protocols DHCP, STUN and DNS. This chapter also describes how the VoIP systems were tested. Chapter 4 reports the results of performance studies evaluating the bare PC SIP Server and SRTP implementations. Chapter 5 presents the conclusion and suggests possible future work. This dissertation includes material from our publications [6, 7, 8].

CHAPTER II. RELATED WORK

In this chapter, we present an overview of previous work on SIP, SRTP, and bare PC systems (also called bare machine computing or BMC systems). We discuss how they relate to this research and how they differ. The related work is divided into three sections dealing with SIP implementation and performance, SRTP implementation and performance, and BMC systems respectively.

A. SIP Implementation and Performance

There are numerous implementations of conventional SIP servers and softphones with SIP UAs that run on various OS platforms. In [9], a SIP server is implemented on top of an existing SIP stack. In [10], 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 [11]. In [12], the features of a new language called StratoSIP for programming UAs that can act as a UA server to one endpoint and as a UA client to another are presented. In [13], the UA is a SIP-based collaborative tool implemented by using existing SIP and SDP stacks. In [14], a Java-based SIP UA is proposed for monitoring manufacturing systems over the Internet. The focus of [15] is a SIP adaptor for both traditional SIP telephony and user lookup on a P2P network that does not have a SIP server.

While SIP servers usually run over UDP and in some cases over TCP, the use of SCTP as a transport protocol for SIP has also been studied [16]. An early study on SIP server performance [17] found that the overhead on a Java SIP server due to security mechanisms such as authentication and TLS was negligible. However, the study in [18], which measured throughput and latency in a dedicated gigabit Ethernet for stateless and stateful proxies over UDP and TCP, showed that authentication, TCP, or the operation/server configuration can significantly impact SIP server performance. Their experiments were conducted using a 3.06 GHz server class machine, and only the performance of a single SIP server (OpenSER on Linux) was evaluated. In

[19], SIP server performance for several stateful SIP proxies over UDP was evaluated. The authors concluded that the overhead due to string processing operations and memory management could consume significant processing time and that performance varied considerably depending on the proxy. Recent work on SIP servers has dealt with performance under overload conditions [20], scalability issues [21, 22], load balancing [23], and the impact of transport protocols on performance [24].

The main difference between previous studies on SIP and the present research is that we focus on a SIP server and SIP UA that run on a bare PC. Moreover, studies on SIP server performance typically use server machines, whereas the bare PC SIP server used for our experiments runs on an ordinary desktop (see Chapter IV). Another difference is that we evaluate SIP server performance not only for the usual register, invite, and redirect operations, but also for the register update, register logout, and invite-not-found operations that could be encountered in practice. We limit our studies to SIP over UDP with stateless proxying, which is a commonly used.

The goal of conventional SIP servers and SIP UAs is to offer enhanced services to clients by using existing low-level SIP stacks that rely on an OS. 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 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 immune to such attacks since they have no OS. Also, since bare PC applications have reduced code complexity and code size, it is easier to analyze their code for security flaws. Moreover, due to their simplicity and the limited services they offer, they have fewer avenues open for attack.

In addition to its security and low-cost benefits, a SIP server or SIP user agent running on a bare PC can be expected to operate efficiently. For example since there is no OS and the SIP applications have direct interfaces to the hardware, there is minimal system overhead. Also, lean versions of the necessary protocols and application-protocol intertwining enable the bare PC SIP server or SIP softphone application to reduce the overhead of inter-layer communication and improve performance. Consequently, the bare PC SIP server and UA have less overhead than an OS-based server or UA, and are more suited for secure low-cost environments.

B. SRTP Implementation and Performance

Previous work on SRTP primarily focuses on key exchange methods and ways to address drawbacks of the protocol. In [25], the requirements for a protocol that manages keys and parameters for SRTP and interoperates with SIP are described. The study also compares several existing approaches including SDP security descriptions, MIKEY, ZRTP and DTLS-SRTP, an extension of DTLS to manage keys in SRTP.

In [26, 27], the vulnerability of SRTP to denial-of-service flooding due to the high overhead of HMAC-SHA-1 authentication is addressed and an alternate lightweight authentication scheme SRTP+ is proposed. In [28], security protocols for VoIP and their impact on call quality are examined by measuring the mean opinion score (MOS).

This research differs from previous studies in that we implement SRTP on a bare PC softphone. Moreover, we 1) compare jitter, delta and throughput values with and without SRTP using a Windows softphone (snom), a Linux softphone (Twinkle) and a bare PC softphone; and 2) determine the time for the various internal operations in SRTP using a bare PC softphone. SRTP and the SIP UA also communicate directly and efficiently with each other and with the existing lower-layer protocols and cryptographic modules in the bare PC softphone. This enables the bare PC SRTP implementation and SIP UA to provide better call quality than a SIP UA with SRTP in an OS-based system.

C. Bare Machine Computing

Bare Machine Computing (BMC) is a novel approach to computing that enables application programs to control and manage hardware resources in a bare machine without an OS or kernel i.e., a bare PC. It is based on the application-centric dispersed operating system (DOSC) paradigm [29]. In this approach, the OS or kernel is eliminated. Instead, a single self-supporting application object (AO) encapsulates all of the necessary functionality for a few (typically one or two) applications to directly execute on the hardware. BMC 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 one or more applications and the boot code.

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 [30] 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 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.

BMC applications are intertwined with lean implementations of the necessary network protocols. For example, in bare PC Web servers and email servers, the application protocol (i.e., HTTP or SMTP) is intertwined with the TCP protocol [3, 31]. Protocol intertwining and other bare PC optimizations contribute to the improved performance of these servers over compatible OS-based servers [3, 4].

The design, implementation, and performance of a bare PC softphone are discussed in [5, 32]. A bare PC softphone with encryption and authentication capabilities is described in [33]. However as noted earlier, this softphone does not include a SIP UA and does not support SRTP.

CHAPTER III. SIP AND SRTP DESIGN AND IMPLEMENTATION

In this chapter, the design and implementation of a bare PC SIP Server for VoIP and a SIP UA with SRTP for a bare PC softphone are described. The bare PC SIP implementations are based on [1]. The SIP UA is integrated with SRTP and other protocols needed by the bare PC softphone.

A. *Bare PC SIP Server Overview*

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.

B. *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.

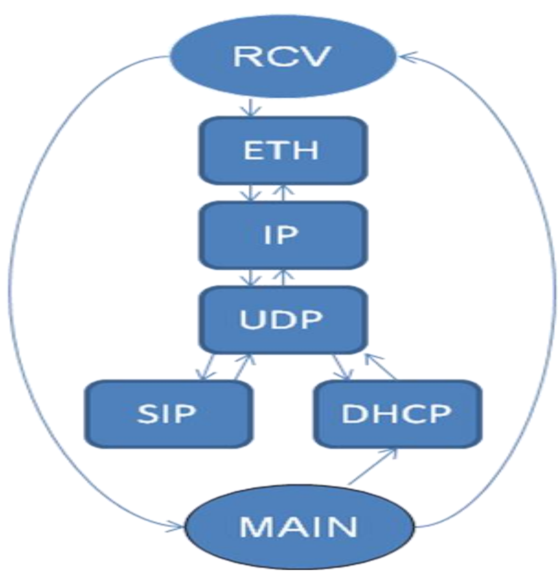


Figure 1. **SIP server protocol/task relationships**

C. SIP Server Internals

The bare PC SIP server uses only two CPU tasks, Main and Receive (Rcv). This 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.

D. 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 following 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.

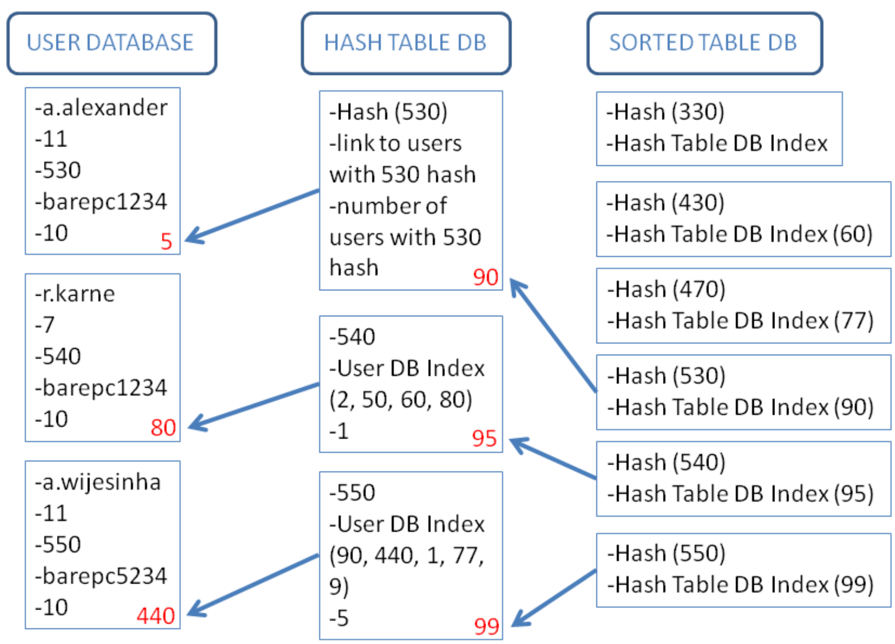


Figure 2. **Database, Hash, and Sorted Tables**

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.

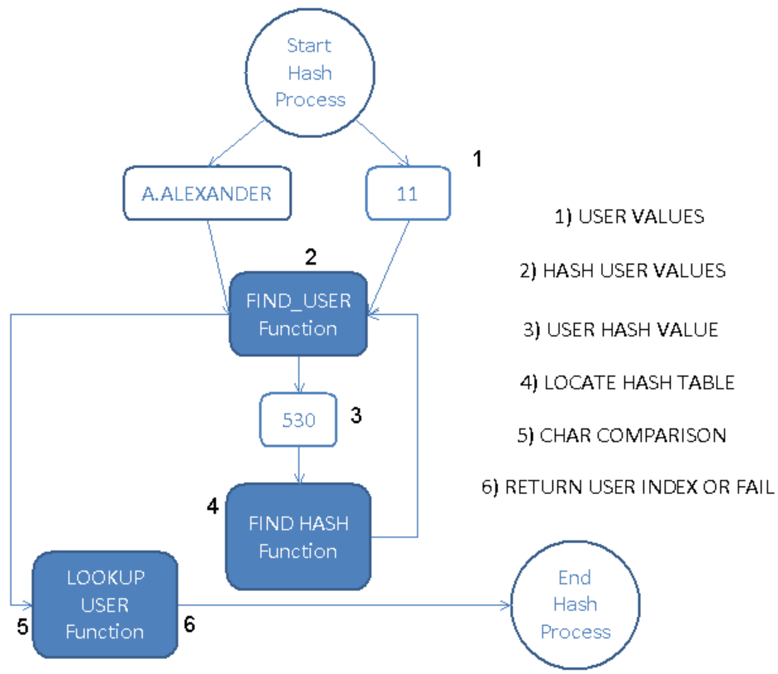


Figure 3. **User lookup process**

E. *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 is forwarded 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. A SIP message exchange including Invite for call setup and Bye for call termination is shown in Fig. 4.

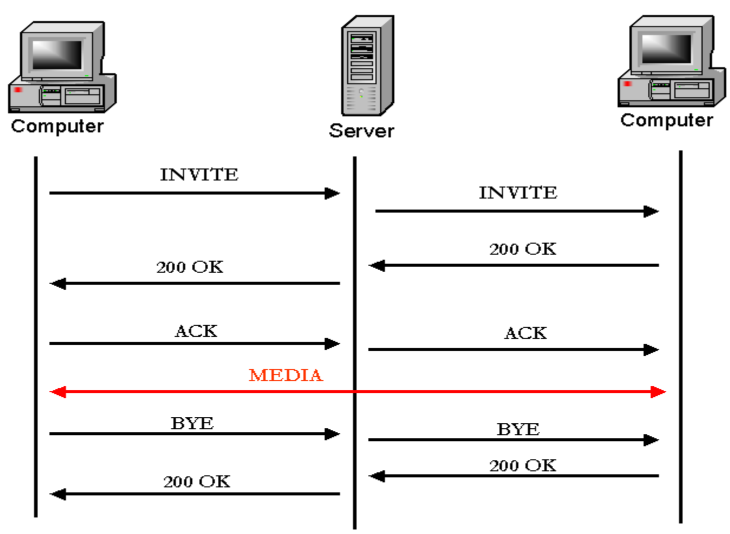


Figure 4. **SIP message exchange**

SIP Authentication: The Message format for an Invite request with authentication is shown in Fig. 5. 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.

```
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="6e91de67ad976997ff"
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 5. **SIP Invite with auth**

F. 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.

G. SIP UA

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.

H. 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 and obtains an IP address using DHCP. If this is a private address, the UA is behind a NAT and uses STUN [34] 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.

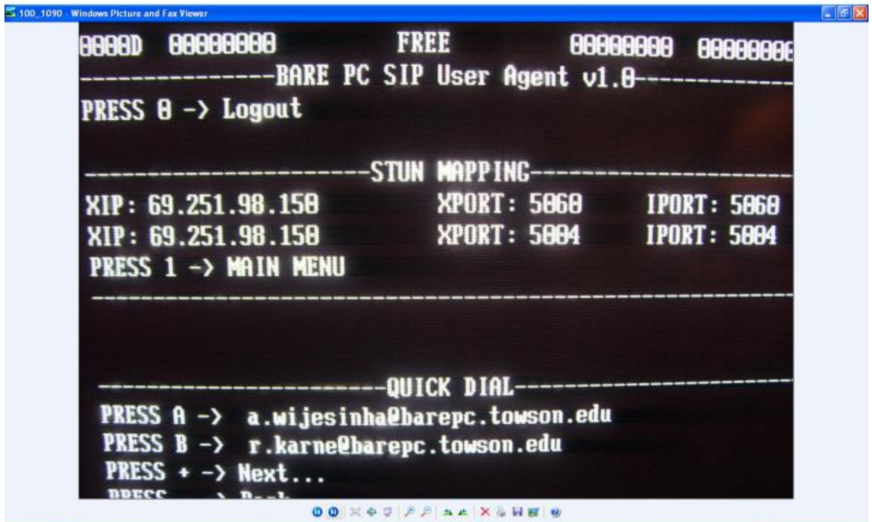


Figure 6. **UA main menu screen**

After the 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. 6. 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 status and connectivity information for the current call including 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 (such as HMAC, SHA-1, MD5, AES, and Base64) needed for SIP authentication, and key establishment and SRTP as described below; 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 object described below that provides VoIP security.

I. 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.

J. STUN/DHCP/DNS

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 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.

K. SRTP Implementation

As noted above, the SIP UA on the bare PC softphone is also integrated with SRTP. SRTP allows the UA to communicate securely with conventional SIP UAs that are SRTP capable. The bare PC SRTP implementation is based on the specification in [2].

The pre-defined cryptographic transforms for SRTP are AES in counter mode or f8 mode for encryption, and HMAC-SHA-1 for message authentication. The f8 mode is not supported by the bare PC softphone. When using AES in counter mode, SRTP encryption (which precedes authentication) consists of generating a pseudo-random keystream for each RTP packet and XORing the RTP data (excluding the RTP header) with the keystream.

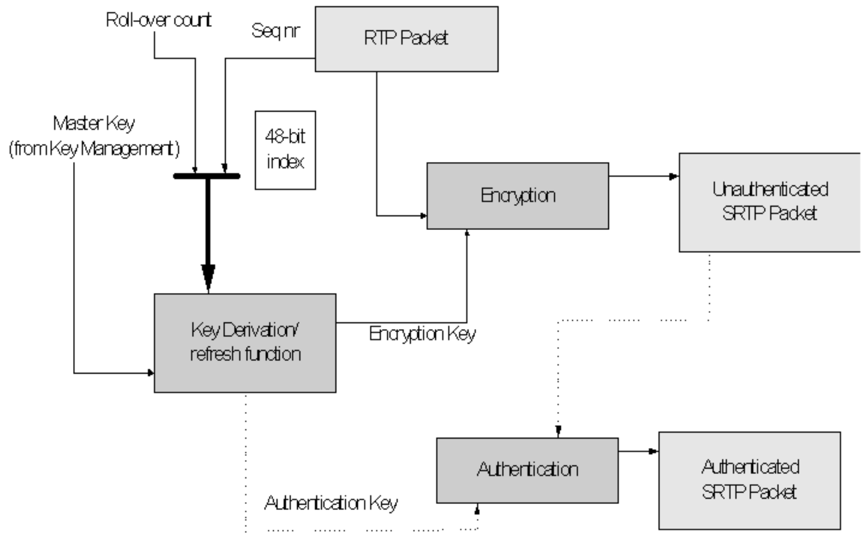


Figure 7. **SRTP processing**

Fig. 7 shows the main steps in SRTP processing on the bare PC SIP softphone. Key derivation produces the session encryption, authentication, and salting keys, while encryption and decryption use AES in counter mode as described earlier. To prevent replay attacks, the receiver checks the index of each packet using a replay list of processed RTP packets within a window of size 64. Packets are authenticated by using HMAC-SHA-1 with a 160-bit key and the result is truncated to obtain an 80-bit or 32-bit authentication tag that is appended to the end of the RTP packet.

L. Key Exchange

Secure VoIP calls require the exchange and management of keys for protection of the media sessions. The SRTP specification provides guidelines for selection of a key management system and mentions several standards but does not mandate a particular system. A variety of key exchange protocols are currently used by applications/providers with SRTP including ZRTP [35] SDES [36], MIKEY [37] and TLS [38]. In our experiments (described in Chapter IV), the snom and bare PC softphones use SDES/SIP, and the Twinkle softphone uses ZRTP for key exchange.

The SDES/SIP message exchange to set up a secure VoIP call is the same as shown in Fig. 4 for a normal SIP INVITE exchange. However, it also includes exchange of the master and master salt keys, and cryptographic transforms via SDES utilizing the SDP Offer/Answer model. Since SDES uses the inline tag within SDP, the latter does not require any protocol modifications. The bare PC UA and some conventional SIP softphones with SRTP currently implement this Offer/Answer model via SDES for key exchange. The keys used to generate the session keys are Base64 encoded by the bare PC softphone SRTP implementation prior to transmission. The SDES key exchange in this form is insecure since the SIP packets are sent in the clear. This problem can be addressed by using a TLS handshake over TCP (or DTLS over UDP) to protect the SDES key exchange over SIP/SDP.

However, other key exchange methods may have more overhead compared to SDES. For example, Fig. 8 shows the ZRTP message exchange used by the Twinkle softphone. ZRTP provides a tag within the SDP protocol for notification to the client that it is able to support ZRTP. It then utilizes the media channel of the VoIP call for key establishment. Compared to SDES/SIP, ZRTP requires 5 extra packets, which are sent over RTP, with an average size of 201 bytes. The experimental results for SRTP in Chapter IV show the impact of ZRTP overhead on Twinkle softphone performance.

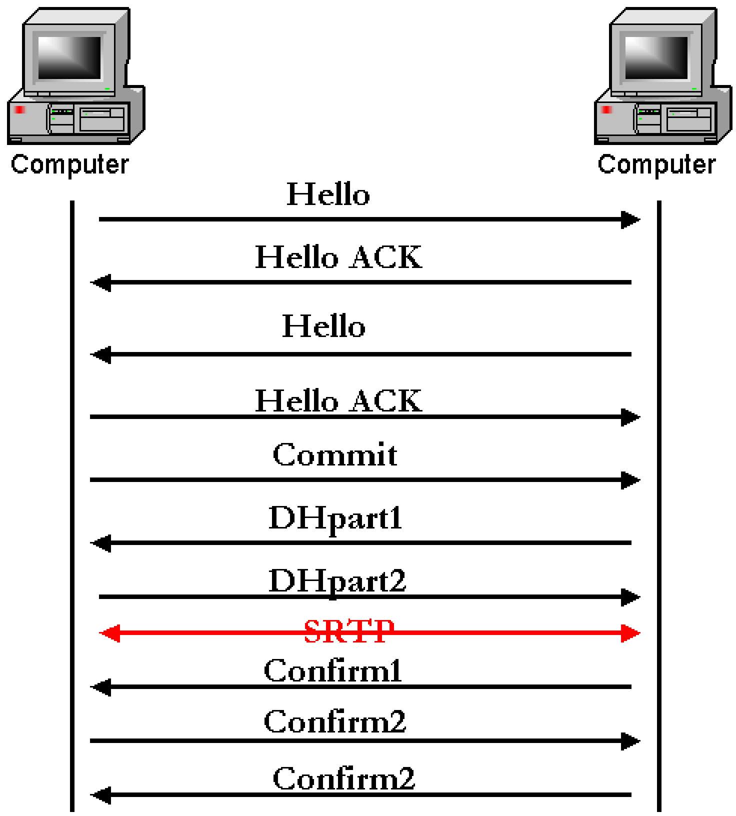


Figure 8. **ZRTP message exchange**

M. Testing

Operational tests (with and without SIP authentication) of the bare PC SIP server and SIP UA implementations with and without SRTP security were conducted. 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. 9.

The bare PC SIP server and user agents were first tested within the dedicated LAN. Testing was performed to verify correct operation between the bare PC SIP server and bare PC SIP softphones; interoperability of bare PC SIP softphones with the OpenSER server [39]; interoperability of the bare PC SIP server with snom360 softphones [40]; and interoperability of bare PC SIP softphones with the snom360 softphones. Specifics of these systems are given in the next chapter.

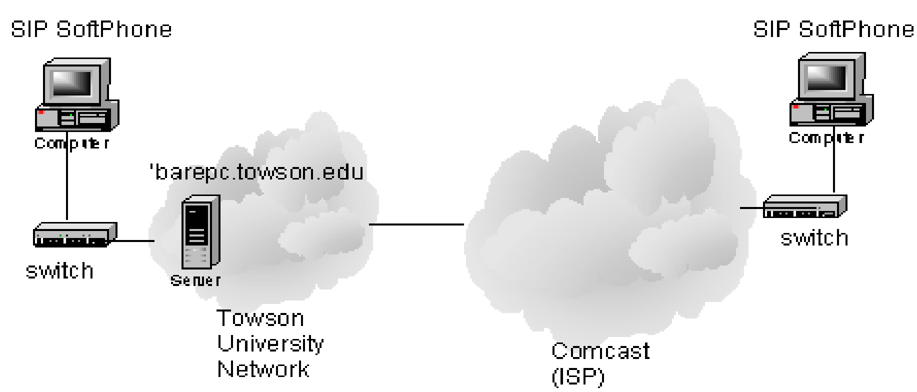


Figure 9. **Network for operational testing**

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.

CHAPTER IV. SIP AND SRTP PERFORMANCE

In this chapter, we describe the experimental setup and the experiments used to evaluate SIP and SRTP performance, and present the results. We also provide details of the systems and software used.

A. *Experimental Setup*

The dedicated test LAN consists of a 100 Mbps Ethernet to which the PCs (ordinary desktops) used for the various experiments are connected. To evaluate SIP server performance, the popular open source SIP workload generator SIPp [41] was used to generate call connection requests to the server for the SIP call flows of interest. The details of the SIP servers, hardware, and OSs used are as follows:

SIP servers: bare PC SIP server (no OS), OpenSer SIP Server [39] ver 1.3.2 –notls (Linux) OpenSer (Kamailio/OpenSIPS), and Brekeke SIP Server [42] ver 2.1.6.6 (Windows) utilizing the Jakarta Web Server and Java platform; PC hardware: Dell Optiplex GX-260 PCs with an Intel Pentium 4 (2.4 GHz) processor, 1.0 GB of RAM and 3COM

Ethernet 10/100 PCI network card; OSs: Microsoft Windows XP Professional ver. 2002 Service Pack 2 (XP SP2), and Linux Ubuntu 8.04 Kernel 2.6.24-16.

The Test LAN used to evaluate SRTP performance is shown in Figure 10. In addition, a Wireshark 1.0.3 packet sniffer [43] is used to capture packets, display message exchanges and report performance data. The PC hardware is the same as detailed above. Calls were made using the following softphones/UAs with SRTP: a snom softphone [38] v5.3 running on Windows XP SP2, a Twinkle softphone [44] version 1.4.2 running on Linux Ubuntu 8.04 kernel 2.6.24-16, and a bare PC softphone with no OS. The OpenSER SIP server (see above) is used to register user agents and set up (proxy) VoIP calls between the softphones.

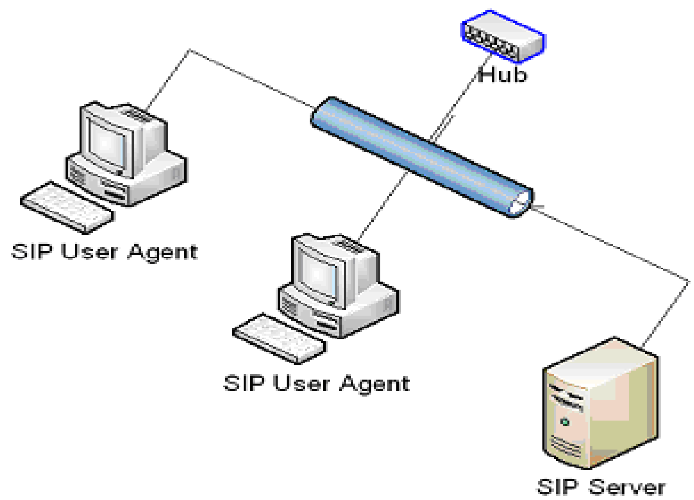


Figure 10. **Test LAN for evaluating SRTP performance**

B. *SIP Server Experiments*

In this section, we describe the experiments conducted to evaluate SIP server performance. We first obtain the values of throughput and latency (defined below) reported by the SIPp tool for the bare PC and OS-based SIP servers considering the register, register update, register logout, invite, invite-not-found, and redirect SIP operations (these operations are described below). We then measure internal timings on the bare PC server for the register operation.

For register updates, the SIP Server searches its user database for a match and then updates the corresponding user’s location data and registration expiration time; for the register logout operation, it removes the user from the database. The invite operation requires the server to lookup the callee's contact details in its database, forward the request to the callee, and send the response back to the caller. The invite-not-found operation is similar to invite except that the callee is not found in the database. For redirect, the server receives an invite message, but instead of forwarding the response to the callee, it forwards a temporarily moved message back to the caller.

For the register, register update, and register logout operations, latency measures the delay at the user agent between sending the register message and receiving the “200 OK” message. Latency for the invite operation measures the sum of two delays: the time between the invite message and “200 OK” messages; and the time between the “bye” and “200 OK” messages. Each of these operations was also tested with authentication enabled, which adds processing overhead due to verifying the MD5 hash, and extra message overhead due to the "unauthorized" message for registration and "407 proxy authentication" message for invite (and their responses).

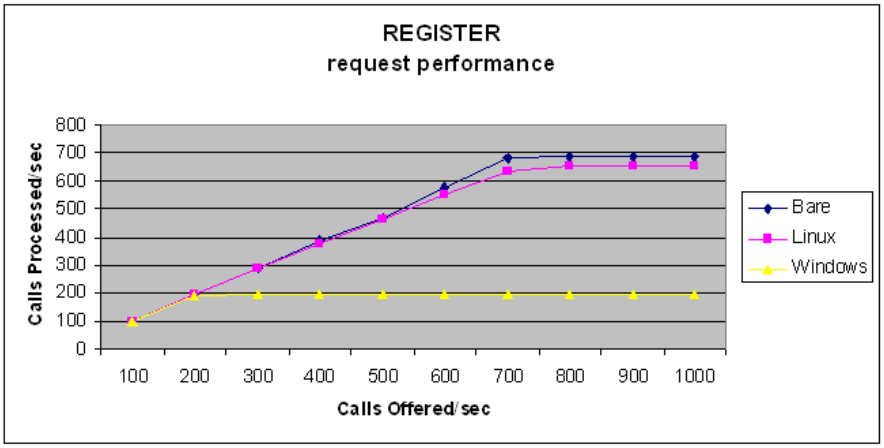
Latency for registration with authentication measures the sum of two delays: the time between the register request and the "unauthorized message"; and the time between the new register message with authentication credentials and the "200 OK" message. Latency for invite with authentication measures the sum of three delays: the time between the invite and "407 proxy authentication" messages; the time between the "invite with authentication" message and the "200 OK" messages; and the time between the "bye" and "200 OK" messages. For invite-not-found and redirect operations, the latency is similarly measured using the "404 not found" and "302 moved temporarily" messages.

The server throughput measures the number of calls per second successfully handled with respect to the offered load, which is the number of calls per second that are generated and sent to the server. The peak throughput is the highest throughput achieved under overload while the server remains stable (and produces consistent results).

To conduct the experiments, the servers were configured to operate in three configuration modes with and without authentication: registrar, proxy, and redirector. In addition, internal timings were measured by inserting timing points within the bare SIP server. Each SIP server was pre-loaded with 10,000 unique SIP username and password pairs. Call flow performance for register, invite-not-found, and redirect was measured for a maximum of 10000 unique users with rates varying from 10 to 1000 calls/s. Call flow performance for invite was similarly measured for a maximum of 5000 users, with rates varying from 50 to 100 calls/s. Each experiment was repeated a minimum of three times to ensure that the results were consistent.

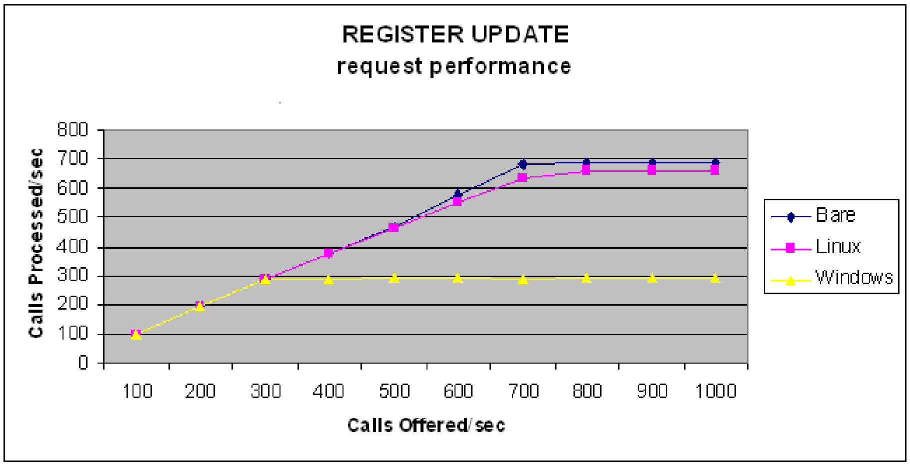
C. SIP Server Throughput

The throughput for the register and invite operations respectively without authentication is shown in Figs. 11-16. It can be seen that the peak throughput of the bare PC SIP server is always higher than that of the OS-based servers except in the case of invite redirect. The peak throughput of the bare PC server typically exceeds that of the Linux server by 50-125 calls/s depending on the operation (although peak is only 10 calls/s larger for invite, and peak is 150 calls/s smaller for invite redirect). For example, the bare PC SIP server has a peak throughput of 700 calls/s for register operations (without authentication), which is better than the peak throughput of Linux (650 calls/s); the Windows server has a much lower peak throughput (around 200 calls/s).



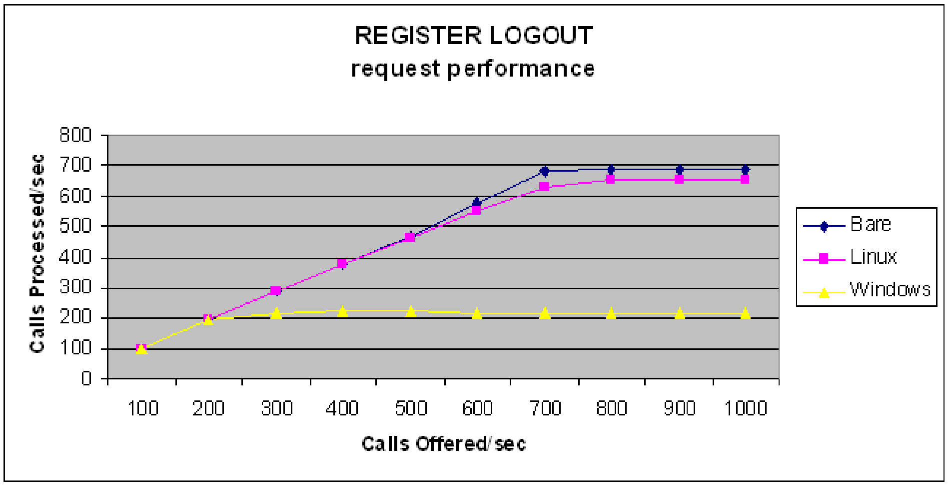
	Bare	Linux	Windows
100	98.492	98.508	98.492
200	194.058	194.118	190.647
300	286.87	286.87	197.106
400	386.341	376.918	197.227
500	470.504	464.447	196.259
600	580.496	549.813	194.943
700	683.022	632.391	194.647
800	735.072	654.407	196.502
900	810.072	654.407	196.502
1000	892.072	654.407	196.502

Figure 11. **SIP Throughput: Register without auth**



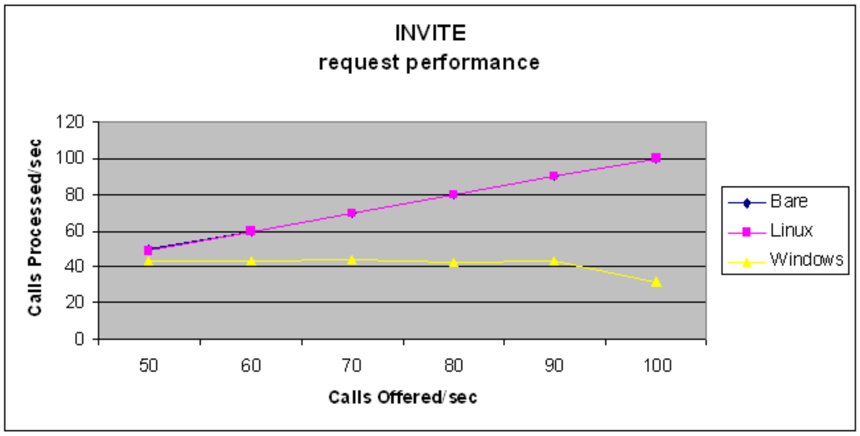
	Bare	Linux	Windows
100	98.492	98.492	98.507
200	194.058	194.114	194.058
300	286.87	287.002	286.738
400	376.918	376.918	291.307
500	470.504	464.447	295.744
600	580.496	549.843	293.574
700	683.022	632.391	288.809
800	735.072	659.805	292.107
900	810.072	659.805	292.107
1000	892.072	659.805	292.107

Figure 12. **SIP Throughput: Register Update without auth**



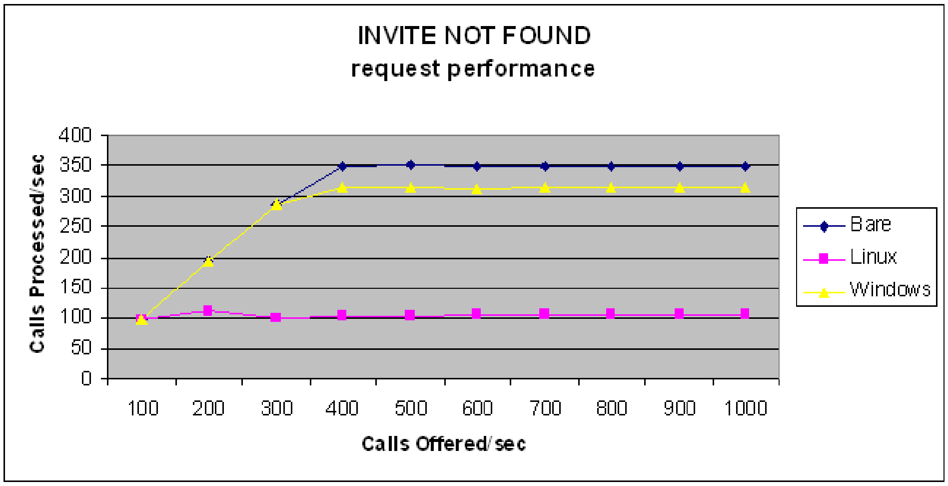
	Bare	Linux	Windows
100	98.507	98.492	98.508
200	194.118	194.058	194.058
300	286.738	286.738	216.361
400	376.918	376.918	220.916
500	470.504	464.447	220.687
600	580.496	549.813	217.836
700	683.022	631.792	217.912
800	735.072	657.117	216.071
900	810.072	657.117	216.071
1000	892.072	657.117	216.071

Figure 13. **SIP Throughput: Register Logout without auth**



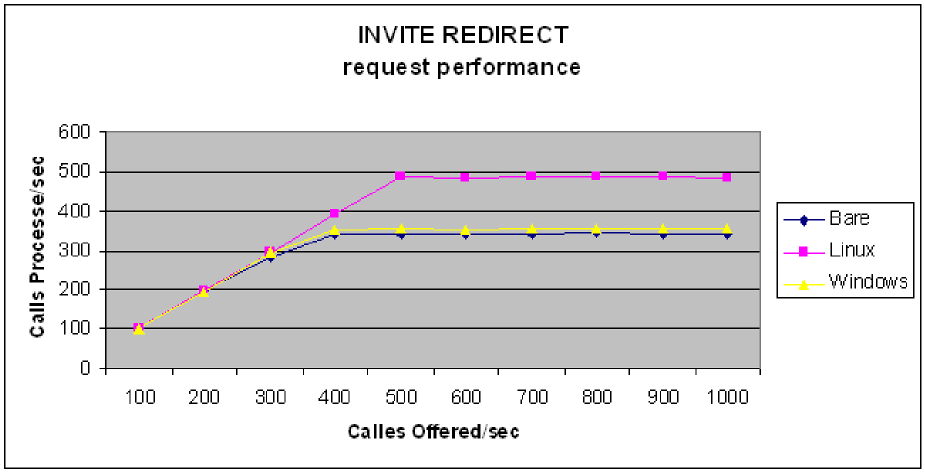
	Bare	Linux	Windows
50	49.867	48.537	43.71
60	59.802	59.801	43.728
70	69.732	69.641	43.914
80	79.662	79.681	42.678
90	89.565	89.611	43.08
100	99.799	99.502	42.689
110	108.507	109.471	42.689
120	119.825	109.894	42.689
130	127.316	109.973	42.689
140	127.141	109.973	42.689
150	127.87	109.973	42.689
160	127.505	109.973	42.689
170	127.257	109.973	42.689
180	127.654	109.973	42.689
190	127.654	109.973	42.689
200	127.654	109.973	42.689

Figure 14. **SIP Throughput: Invite without auth**



	Bare	Linux	Windows
100	98.492	98.431	98.477
200	194.058	110.25	194.058
300	286.738	100.203	286.738
400	348.396	103.728	315.736
500	410.92	104.472	314.961
600	468.202	105.281	313.421
700	510.15	105.68	315.428
800	510.15	105.315	315.577
900	510.15	105.315	315.577
1000	510.15	105.315	315.577

Figure 15. **SIP Throughput: Invite Not Found without auth**

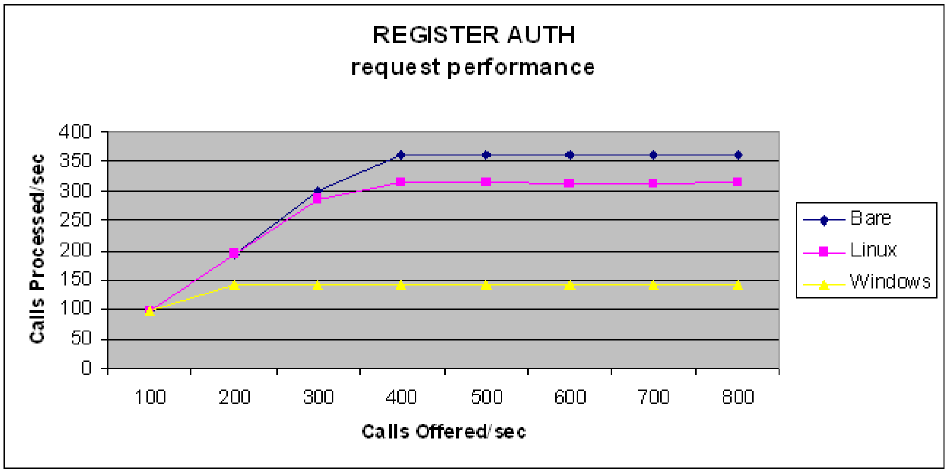


	Bare	Linux	Windows
100	99.502	99.534	99.467
200	198.02	198.145	195.031
300	284.414	295.753	295.281
400	396.341	391.665	353.057
500	490.504	487.805	355.568
600	590.496	484.872	353.257
700	693.022	489.333	355.316
800	765.072	486.334	355.29
900	842.231	486.334	355.872
1000	843.368	484.872	356.151

Figure 16. **SIP Throughput: Invite Redirect without auth**

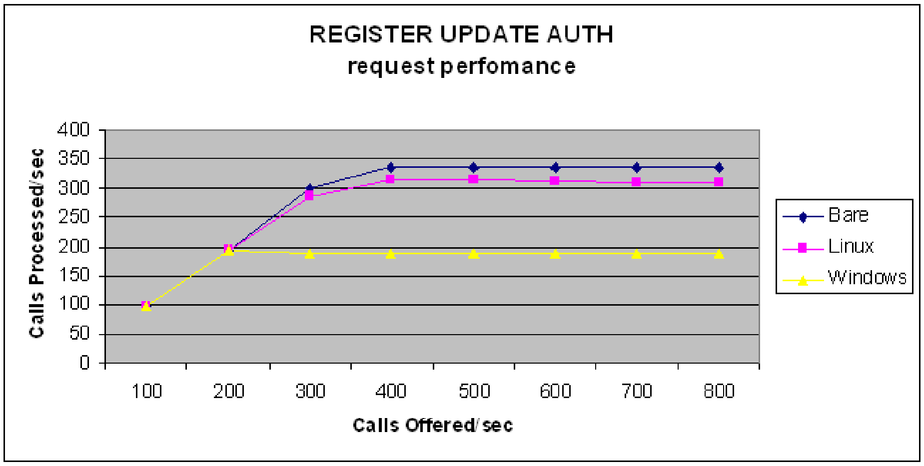
The peak throughput performance of the bare PC SIP server should be better than that of the OS-based servers, due to its simple design and the elimination of OS overhead. However, this performance advantage may be reduced or lost in certain cases due to inefficient algorithms or the lack of concurrency. The latter situation arises with the invite operation. The peak throughput of the bare PC server is only marginally higher than Linux in this case, but introducing a separate SIP task to handle an invite operation will improve performance. The apparent drop in performance of the bare PC server for invite redirect is due to a significant improvement in the performance of the Linux server in this case.

Implementing Linux's search algorithm on the bare PC SIP server should improve its performance. A more efficient search algorithm should also improve the performance for the invite-not-found operation. The peak throughput of a given server does not vary much across the three register operations since the work performed in each case is essentially the same. The increase in the peak throughput of the Windows server for register update compared to that for the other two register operations is possibly due to caching.



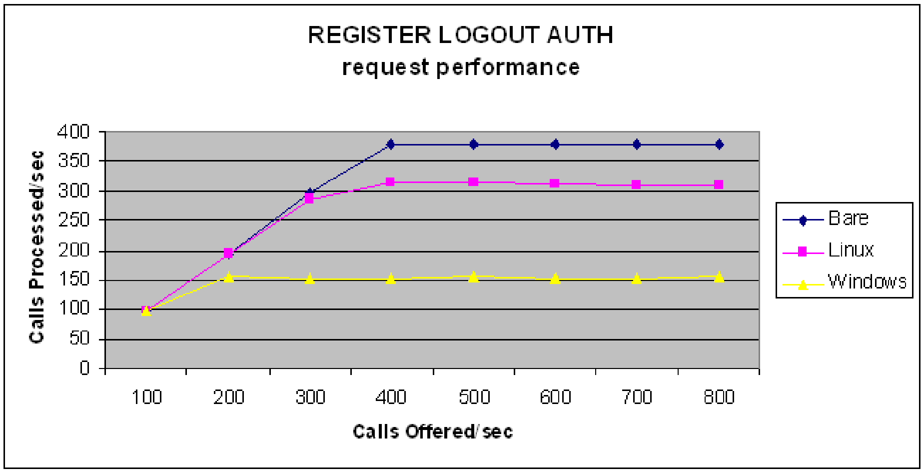
	Bare	Linux	Windows
100	98.507	98.465	98.492
200	193.825	193.915	142.602
300	298.316	286.508	142.286
400	361.141	315.04	140.752
500	361.141	315.567	141.561
600	361.141	312.637	141.405
700	361.141	312.539	141.655
800	361.141	314.13	141

Figure 17. **SIP Throughput: Register with auth**



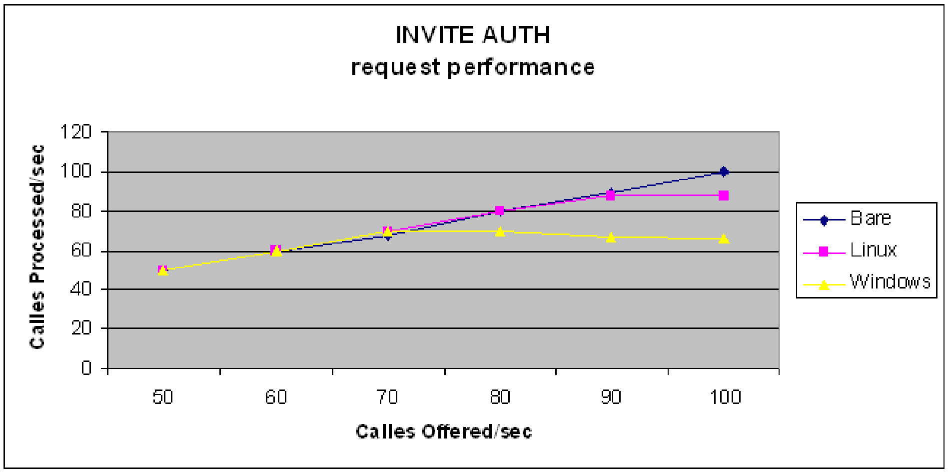
	Bare	Linux	Windows
100	98.492	98.459	98.492
200	194.058	193.979	193.998
300	298.44	286.451	188.512
400	334.441	315.398	188.349
500	334.441	314.06	188.455
600	334.441	312.725	187.684
700	334.441	310.472	187.959
800	334.441	310.771	188.402

Figure 18. **SIP Throughput: Register Update with auth**



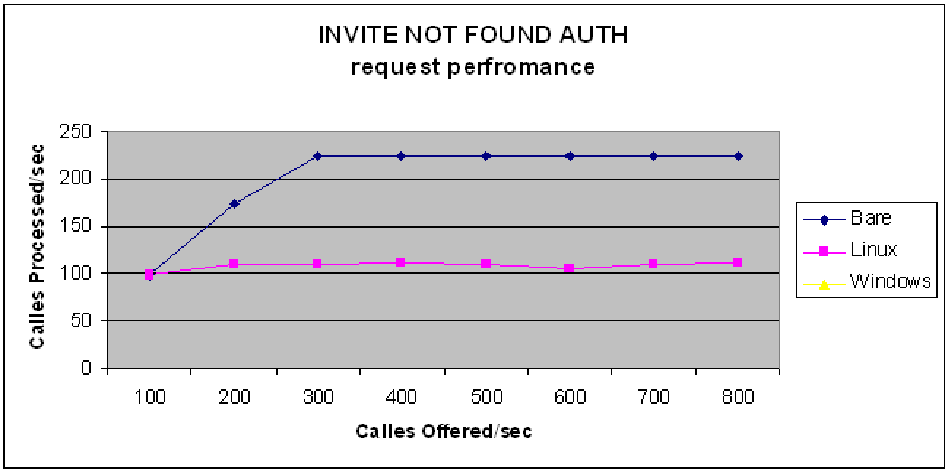
	Bare	Linux	Windows
100	98.492	98.459	98.477
200	194.058	193.979	155.642
300	297.807	286.451	153.516
400	380.178	315.398	153.402
500	380.178	314.06	153.884
600	380.178	312.725	153.219
700	380.178	310.472	153.402
800	380.178	310.771	153.624

Figure 19. **SIP Throughput: Register Logout with auth**



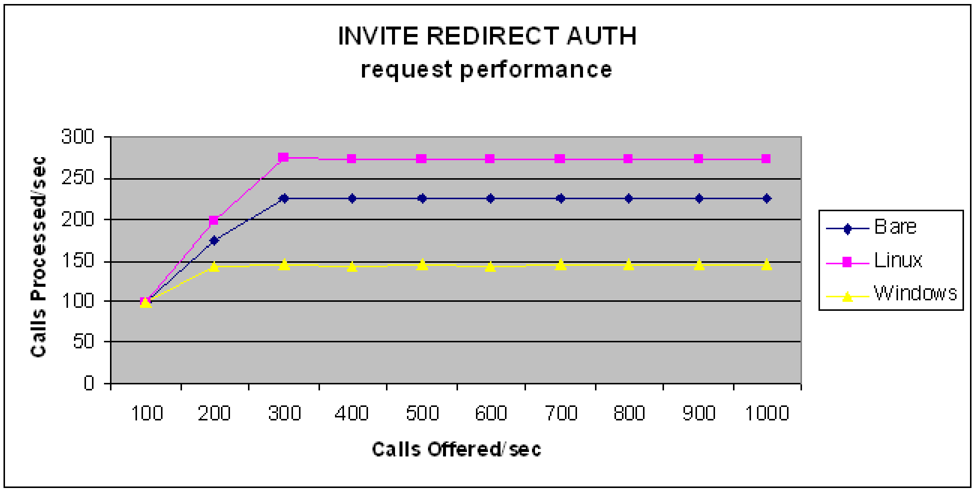
	Bare	Linux	Windows
50	49.859	49.86	49.875
60	59.802	59.802	59.802
70	67.548	69.718	69.748
80	79.802	79.662	69.399
90	88.802	87.561	66.89
100	99.802	87.473	65.601

Figure 20. **SIP Throughput: Invite with auth**



	Bare	Linux	Windows
100	98.492	98.461	
200	173.629	109.179	
300	224.771	109.179	
400	224.771	111.305	
500	224.771	109.853	
600	224.771	104.781	
700	224.771	110.232	
800	224.771	110.497	

Figure 21. **SIP Throughput: Invite Not Found with auth**



	Bare	Linux	Windows
100	99.471	99.471	99.502
200	173.629	197.894	143.951
300	224.726	273.973	144.601
400	224.726	272.346	143.951
500	224.726	273.03	144.663
600	224.726	272.807	143.625
700	224.726	272.109	144.601
800	224.726	272.109	144.471
900	224.726	273.508	144.928
1000	224.726	272.346	144.538

Figure 22. **SIP Throughput: Invite Redirect with auth**

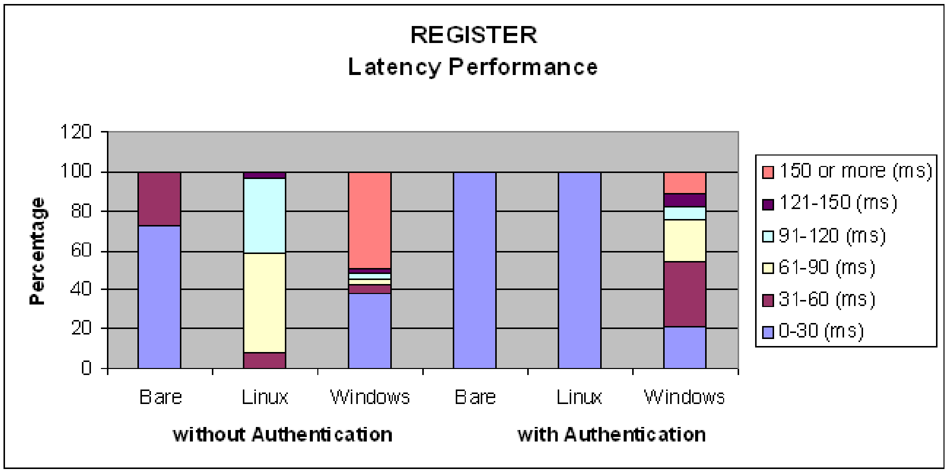
The results in Figs. 17-22 show that peak throughput of all servers is reduced as expected for both register and invite operations when authentication is added. This reduction in performance is due to the extra message overhead noted previously, and the overhead of computing and verifying the additional information needed for authentication with a message digest [17]. The negative impact of authentication on performance was also noted in [18].

There are no throughput values for the Windows server for invite-not-found with authentication since its message flow in this case could not be compared with that of the other two servers. It is evident that the peak throughput of the bare PC server with authentication shows a greater reduction versus its peak throughput without authentication compared to the OS-based servers. Adapting the approach used for authentication by Linux for the bare PC server could improve its performance.

D. SIP Server Latency

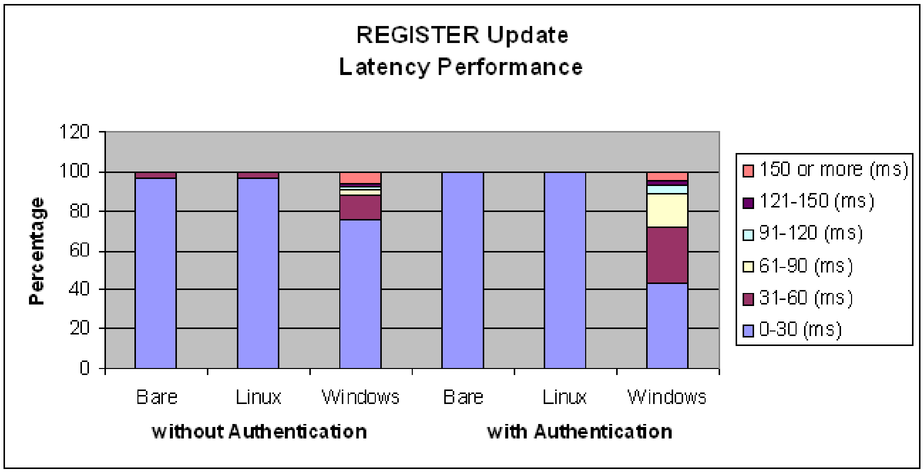
Figs. 23-28 compare the latencies for bare PC and OS-based SIP servers for the register and invite operations respectively, with and without authentication. In most cases, the bare PC server performs better than the OS-based servers.

As seen in the figures, the highest percentage of latencies for the bare PC server are usually in the 0-30 ms range, and it rarely has latencies that exceed 150 ms. The invite operation is an exception and latency performance in this case could be improved by enabling concurrency in the server as noted earlier. For all register operations and invite redirect with authentication, the latency performance of the bare PC and Linux servers is the same. Further studies are needed to determine if the approach used to implement authentication in the Linux server will improve the latency performance of the bare PC server in these cases.



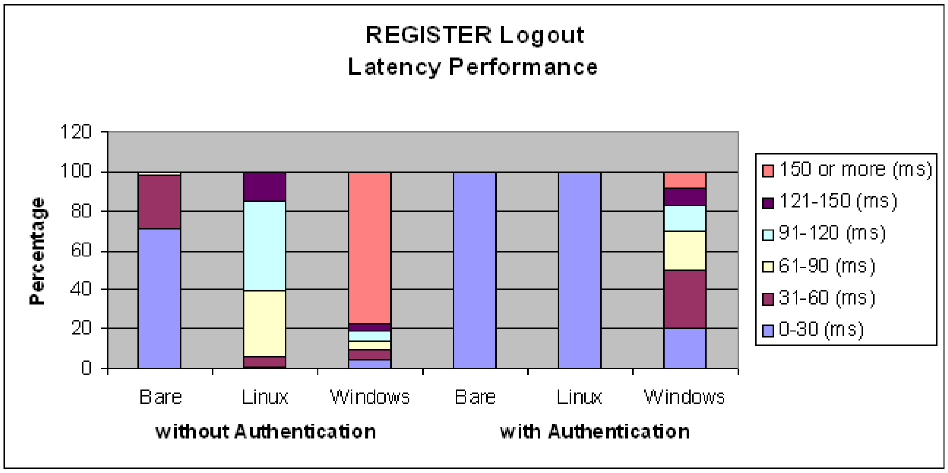
	Bare	Linux	Windows	Bare	Linux	Windows
0-30 (ms)	72.85499	0.010035	37.85248	100	100	20.72072
31-60 (ms)	26.90416	7.947817	4.887105	0	0	33.13313
61-90 (ms)	0.240843	50.87807	2.870045	0	0	21.32132
91-120 (ms	0	37.68189	2.528851	0	0	7.107107
121-150 (m	0	3.482188	2.257903	0	0	5.705706
150 or mor	0	0	49.60361	0	0	12.01201

Figure 23. **SIP Latency: Register**



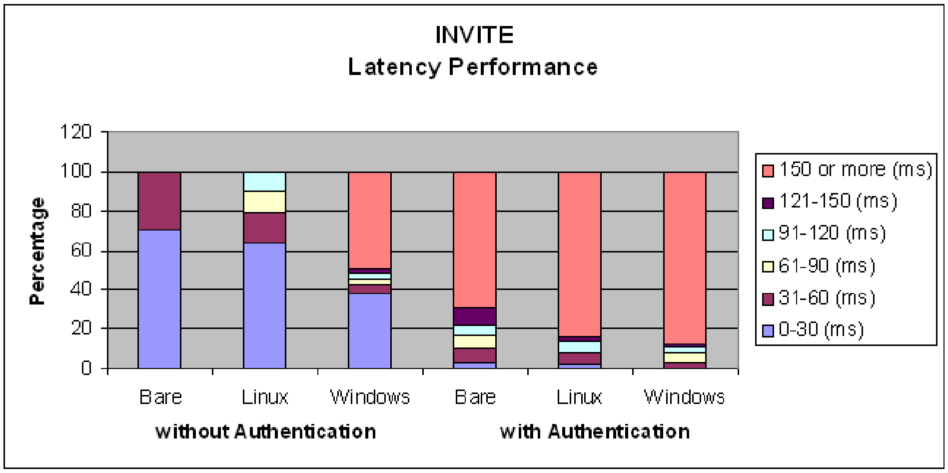
	Bare	Linux	Windows	Bare	Linux	Windows
0-30 (ms)	96.93929	96.86904	74.82188	100	100	43.44344
31-60 (ms)	3.060712	3.130958	12.70447	0	0	28.82883
61-90 (ms)	0	0	2.799799	0	0	16.21622
91-120 (ms	0	0	2.167587	0	0	4.404404
121-150 (m	0	0	1.274461	0	0	2.602603
150 or mor	0	0	6.231811	0	0	4.504505

Figure 24. **SIP Latency: Register Update**



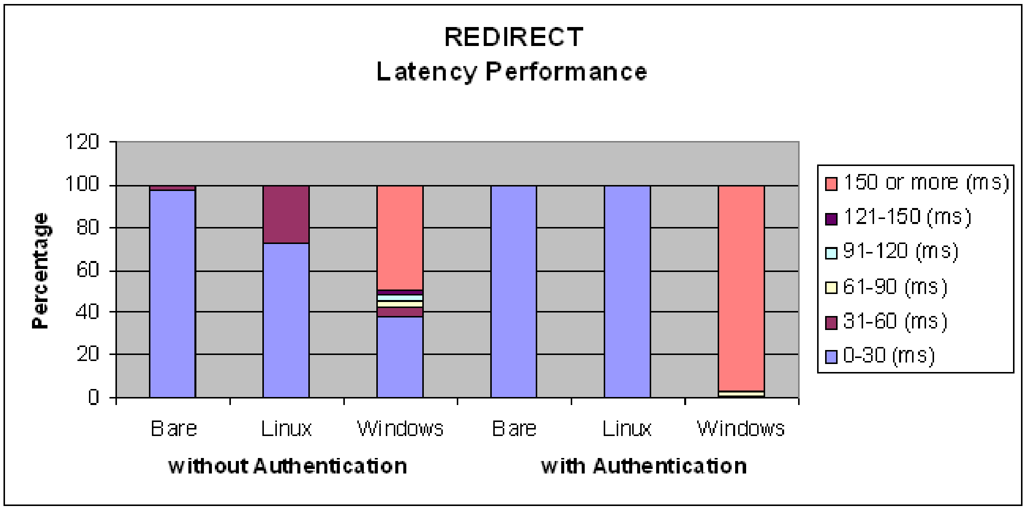
	Bare	Linux	Windows	Bare	Linux	Windows
0-30 (ms)	71.05871	0.983442	4.345208	100	100	20.42042
31-60 (ms)	27.44606	5.32865	4.887105	0	0	29.02903
61-90 (ms)	1.465128	33.35675	4.716508	0	0	20.02002
91-120 (ms	0.030105	45.20823	5.148018	0	0	13.71371
121-150 (m	0	15.12293	3.602609	0	0	8.408408
150 or mor	0	0	77.30055	0	0	8.408408

Figure 25. **SIP Latency: Register Logout**



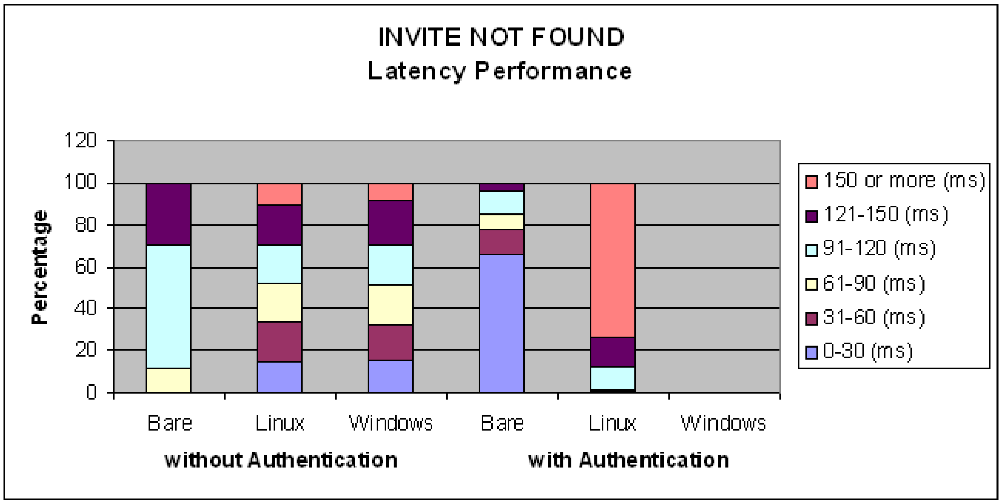
	Bare	Linux	Windows	Bare	Linux	Windows
0-30 (ms)	70.59359	63.76572	37.85248	3.009009	2.046046	0.1001
31-60 (ms)	29.00551	15.24098	4.887105	6.906907	5.614615	2.802803
61-90 (ms)	0.29537	10.59359	2.870045	7.104104	0.311311	4.504505
91-120 (ms	0	10.59359	2.528851	4.704705	6.106206	3.303303
121-150 (m	0	0	2.257903	9.106106	2.105405	1.301301
150 or mor	0	0	49.60361	69.16917	84.00064	87.98799

Figure 26. **SIP Latency: Invite**



	Bare	Linux	Windows	Bare	Linux	Windows
0-30 (ms)	97.2	72.85499	37.85248	99.89107	99.7998	0
31-60 (ms)	2.8	26.90416	4.887105	0.108932	0.2002	1.001001
61-90 (ms)	0	0.240843	2.870045	0	0	2.102102
91-120 (ms	0	0	2.528851	0	0	0.1001
121-150 (m	0	0	2.257903	0	0	0
150 or mor	0	0	49.60361	0	0	96.7968

Figure 27. **SIP Latency: Invite Redirect**

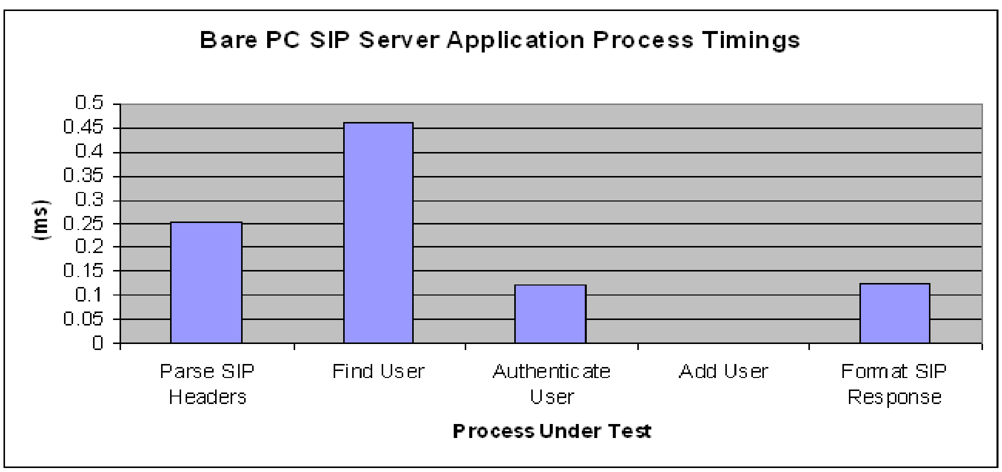


	Bare	Linux	Windows	Bare	Linux	Windows
0-30 (ms)	0	14.79678	15.3352	65.66567	1.001001	0
31-60 (ms)	0.220773	18.63645	17.31105	11.81181	0.600601	0
61-90 (ms)	11.50025	18.63645	18.46973	7.407407	0	0
91-120 (ms	58.45459	18.63645	19.49018	11.21121	10.6006	0
121-150 (m	29.68389	18.63645	20.40086	3.903904	14.2042	0
150 or mor	0.140492	10.65743	8.992967	0	73.59359	0

Figure 28. **SIP Latency: Invite Not Found**

E. SIP Server Internal Timings

Fig. 29 compares average values of internal timings for the bare PC SIP server collected during the register operation under maximum load conditions. It is seen that FindUser, which searches for a given user, and ParseSIPHeaders, which processes the SIP header are the most expensive operations, although the former is twice as expensive as the latter. The least expensive operation is AddUser, which simply adds the information for a new user, and thus takes an insignificant amount of time as would be expected. The AuthenticateUser and FormatSIPResponse operations have approximately the same cost, which is about half that of ParseSIPHeaders. We conducted tests on the OpenSER server using OProfile 0.9.5 [45], which showed that the timings for the AddUser and ParseSIPHeaders operations exceed the corresponding timings on the bare PC by factors of 4 and 7 respectively.



Parse SIP Headers	Find User	Authenticate User	Add User	Format SIP Response
0.2514	0.4632	0.1222	0.0004	0.126

Figure 29. **SIP server internal timings**

F. Analysis of Server Results

Further insight into the results on throughput may be obtained by considering sustainable throughput, which is defined as the maximum rate of calls for which the processed call rate matches the offered call rate. Sustainable throughput reflects the extent to which a server can cope with the offered load, and it can be determined from the preceding Figs. 11-22. For example, the bare PC server's sustainable throughput values for the register, register update, and register logout operations without authentication are respectively 400, 600, and 700 calls/s (for all three register operations without authentication, the peak throughput is the same as the latter value).

It can be seen that the sustainable throughput of the bare PC server exceeds that of the Linux server for all operations without authentication except for invite-not-found when it is the same. In contrast, the sustainable throughput for the two servers for all operations with authentication is the same (or differs by a small amount).

As noted earlier, in the case of peak throughput with and without authentication, the bare PC server's values are higher than those for the Linux server except for invite redirect. Thus, both sustainable and peak throughput values should be used to estimate server capacity with and without authentication.

The latency performance shown in the preceding Figs. 5 and 6 may be better understood by computing a latency coefficient $p_1*w_1+p_2*w_2+p_3*w_3+p_4*w_4+p_5*w_5-p_6$, where $p_1, \dots, p_6$ are the latency percentages of the groups 0-30 ms, … , 121-150 ms, and > 150 ms respectively; and $w_1, \dots, w_5$ are the weights of the first 5 groups with $0<=w_i<=1$ and $w_1+\dots+w_5=1$. The last term with a negative sign reflects the undesirability of latencies > 150 ms. The weights $w_1, \dots, w_5$ can be assigned based on the relative importance of the lower latency groups.

For example, suppose we assign $w_1=0.55$, $w_2=0.445$, $w_3=0.004$, $w_4=0.0007$, and $w_5=0.0003$. Then the latency coefficients for register logout without authentication for the bare PC, Linux, and Windows servers are 0.496, 0.185, and -0.7. These values show that the latency performance of the bare PC server in this case is much better than that of the Linux server, whereas the performance of the Windows server is far worse than both of them. It can also be verified that the latency coefficient of the bare PC server is greater than or equal to that of the Linux server except in the case of invite with authentication and invite-not-found without authentication. As noted above, concurrency and use of a more efficient search algorithm may help to improve bare PC server performance in these cases.

G. SRTP Experiments

In this section, we describe the experiments conducted to evaluate SRTP performance on the bare PC softphone. First, timing points as shown in Fig. 30 are inserted into the SRTP code on the bare PC softphone to get the processing times of major functions in SRTP including key derivation, encryption, decryption, replay protection and authentication, and also the time to process network headers in incoming and outgoing SRTP packets.

Key derivation produces the session encryption, authentication, and salting keys, while encryption and decryption use AES in counter mode as described earlier. Replay protection involves checking the index of each packet using a replay list of processed RTP packets within a window of size 64. Packets are authenticated by using HMAC-SHA-1 with a 160-bit key and the result is truncated to obtain an 80-bit or 32-bit authentication tag that is appended to the packet. The time to process network headers in incoming and outgoing SRTP packets is the time to transfer packets between the Ethernet and SRTP processing levels.

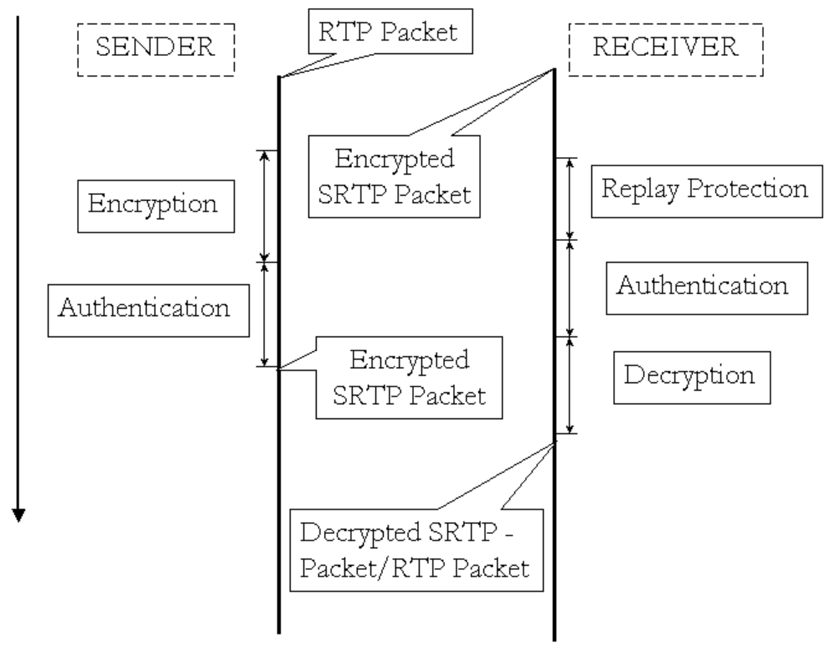


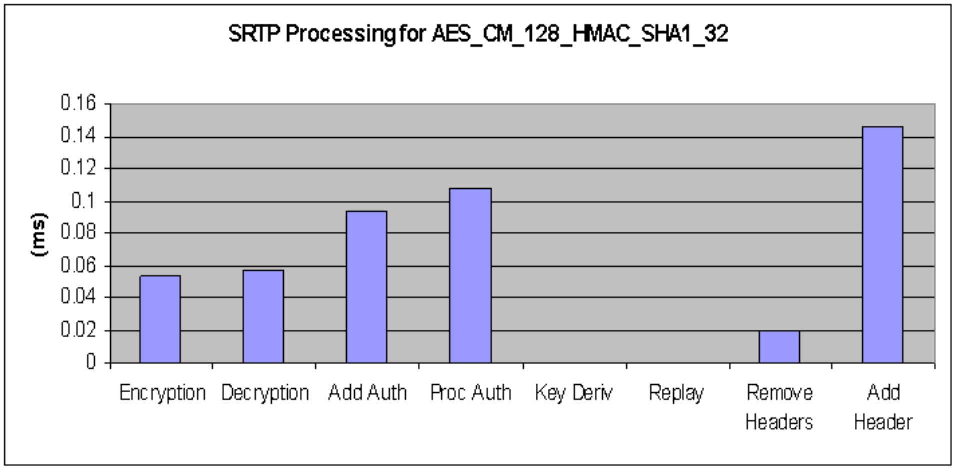
Figure 30. **SRTP timing points**

Next, VoIP call quality with and without SRTP is evaluated by comparing maximum and mean delta (packet interarrival time), maximum and mean jitter, and throughput (bits/s) reported by Wireshark for calls using the snom, Twinkle, and bare PC softphones. These values were computed based on 10,000 VoIP packets transferred in each direction between the softphones (i.e., about 3.5 minutes of voice traffic). The softphones used SRTP with a 128-bit AES encryption key and a 32-bit HMAC-SHA-1 message authentication tag. The bare PC softphone implementation of SRTP also allowed 192-bit and 256-bit encryption keys and an 80-bit authentication tag. The softphones were configured to use the G.711 codec and 20 ms voice packets consisting of 160 bytes. Since AES processes 16-byte blocks at a time, there are 10 AES invocations per packet.

H. SRTP Internal Timings

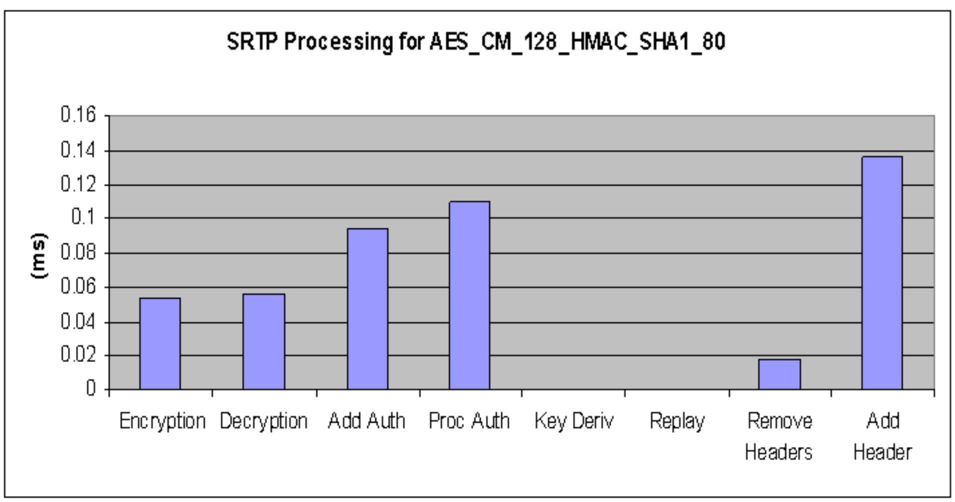
The internal timings (processing times) for various SRTP functions on the bare PC softphone with 128, 192, or 256-bit AES keys and a 32 or 80-bit HMAC/SHA-1 authentication tag are shown in Figs. 31-36. The most expensive internal step in the SRTP protocol is authentication processing. In contrast, the encryption and decryption processes consume much less time. It can also be seen that the times for the key derivation and replay processing steps are negligible. However, processing network headers on outgoing packets has higher cost than any of the other steps. Processing time increases by 10% when using a 192-bit AES key versus a 128-bit key, and by 20% when using 256-bit AES key versus a 128-bit key.

However, since the actual amount of processing time for all AES key sizes is very small, key size has no observable effect on call quality or VoIP throughput as is confirmed by the results in the next section. It can also be seen that processing times are about the same regardless of authentication tag size. This is because 160 bits are produced by HMAC/SHA-1 prior to truncating to a 32-bit or 80-bit authentication tag and the increase in processing time to compare the larger tag is insignificant compared to the nearly constant processing time of HMAC-SHA-1. Overall, the results clearly indicate that SRTP processing adds negligible overhead (less than 1 ms) to RTP processing.



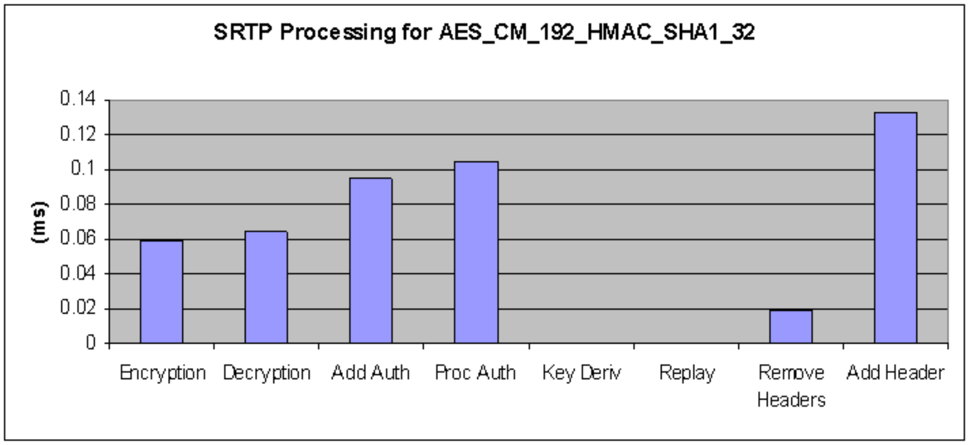
	Encryption	Decryption	Add Auth	Proc Auth	Key Deriv	Replay	move Head	Add Heade
128_32	0.0538	0.0565	0.0937	0.108	0	0.0004	0.0195	0.1449

Figure 31. **SRTP Timing: 128-bit encryption, 32-bit auth**



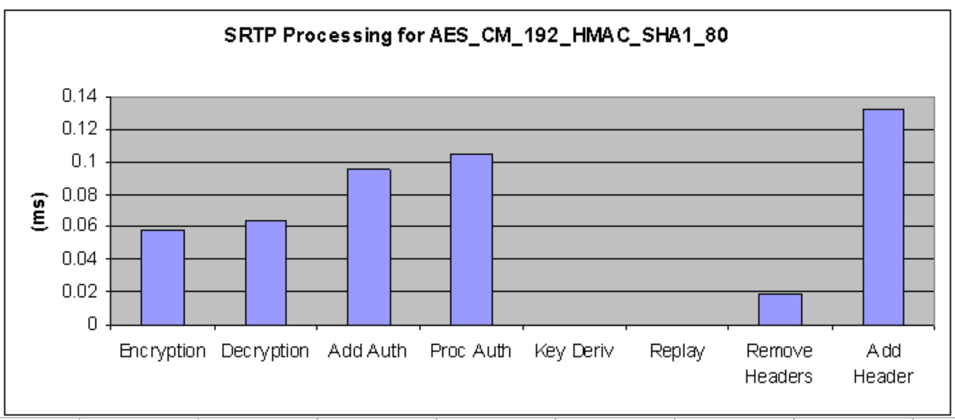
	Encryption	Decryption	Add Auth	Proc Auth	Key Deriv	Replay	move Head	Add Heade
128_80	0.0538	0.0558	0.0945	0.1094	0	0.0004	0.0177	0.1365

Figure 32. **SRTP Timing: 128-bit encryption, 80-bit auth**



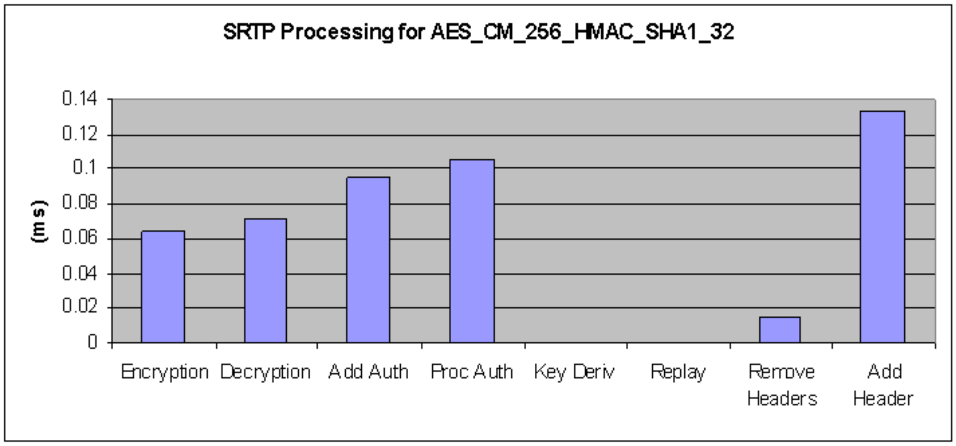
	Encryption	Decryption	Add Auth	Proc Auth	Key Deriv	Replay	move Head	Add Header
192_32	0.0587	0.0639	0.0951	0.1045	0	0.0004	0.0193	0.1322

Figure 33. **SRTP Timing: 192-bit encryption, 32-bit auth**



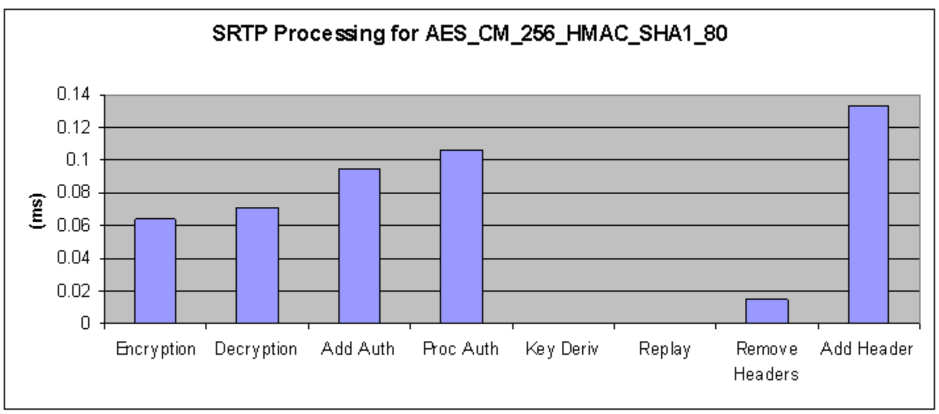
	Encryption	Decryption	Add Auth	Proc Auth	Key Deriv	Replay	move Head	Add Header
192_80	0.0587	0.0639	0.0951	0.1045	0	0.0004	0.0193	0.1322

Figure 34. **SRTP Timing:192-bit encryption, 80-bit auth**



	Encryption	Decryption	Add Auth	Proc Auth	Key Deriv	Replay	move Head	Add Heade
256_32	0.0644	0.0715	0.0954	0.1058	0	0.0004	0.0146	0.1332

Figure 35. **SRTP Timing: 256-bit encryption, 32-bit auth**

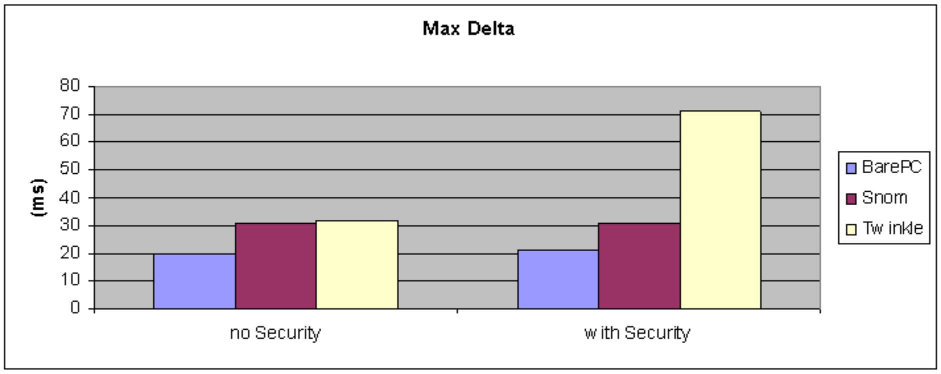


	Encryption	Decryption	Add Auth	Proc Auth	Key Deriv	Replay	move Head	Add Heade
256_80	0.0644	0.0715	0.0954	0.1058	0	0.0004	0.0146	0.1332

Figure 36. **SRTP Timing: 256-bit encryption, 80-bit auth**

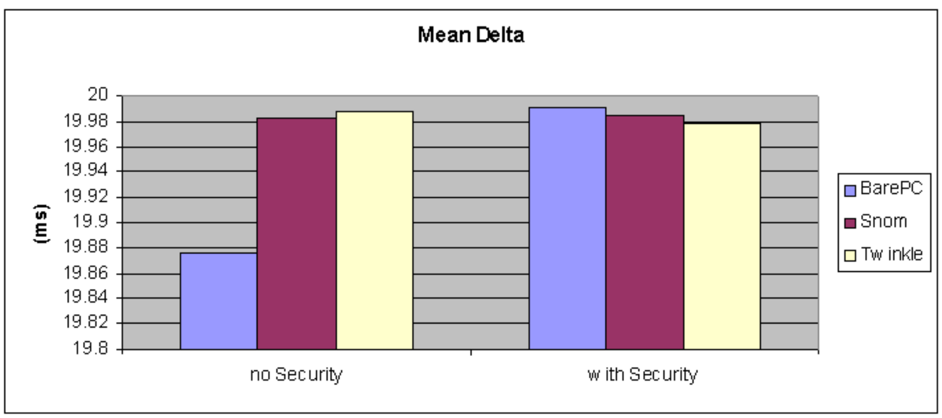
I. *SRTP Maximum and Mean Delta*

Maximum and mean delta values are shown in Figs. 37 and 38 respectively. Maximum delta without security is close to the ideal 20 ms value for the bare PC softphone, and 30 ms for the snom and Twinkle softphones. However, while the increase in maximum delta due to SRTP is less than 1 ms for the snom and bare PC softphones, it is over 40 ms for the Twinkle softphone. This increase in maximum delta for the Twinkle softphone is likely due to ZRTP exchanging its keys in the media channel as discussed in Chapter III. Mean delta values for all three softphones with SRTP are close to 20 ms.



	no Securit	with Secur
BarePC	20.11	20.84
Snom	30.66	30.97
Twinkle	31.77	71.07

Figure 37. **SRTP Maximum delta with and without SRTP**

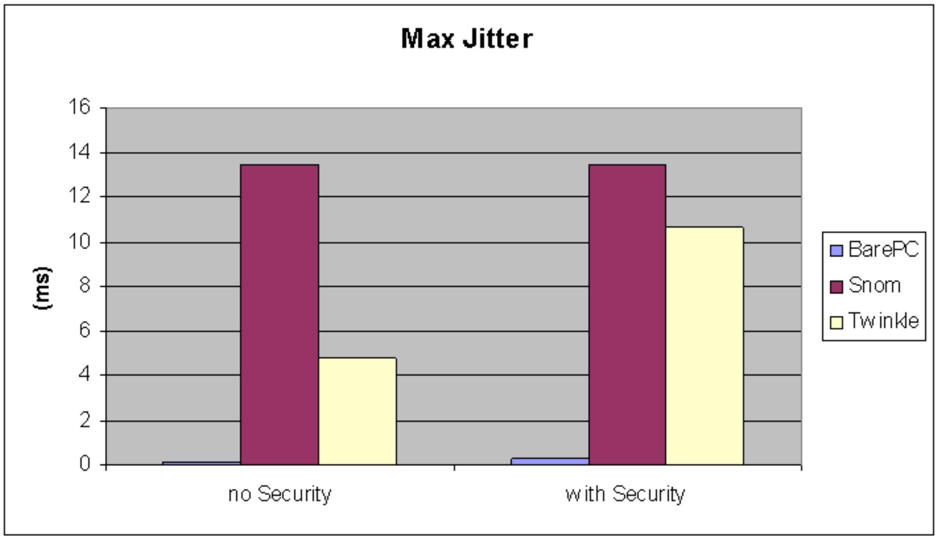


	no Securit	with Secur
BarePC	19.87561	19.99078
Snom	19.98277	19.98565
Twinkle	19.98792	19.97802

Figure 38. **Mean delta with and without SRTP**

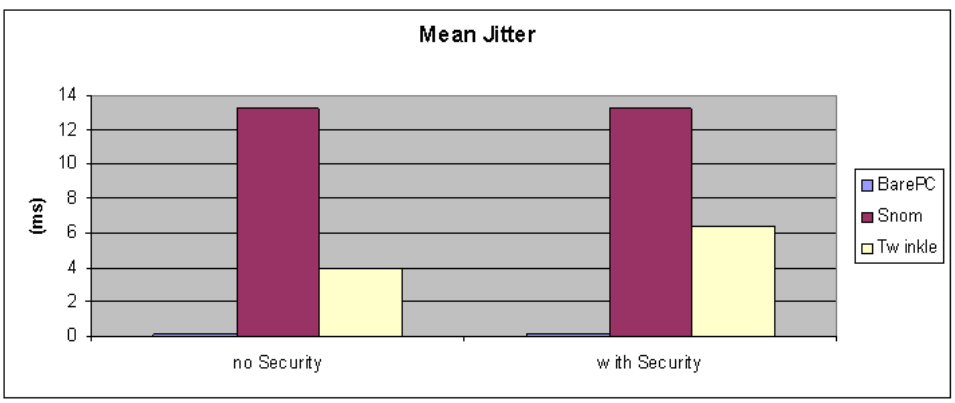
J. SRTP Maximum and Mean Jitter

Maximum and mean jitter values are shown in Figs. 39 and 40 respectively. For the snom softphone, maximum or mean jitter with or without SRTP is the same (13 ms). For the Twinkle softphone, maximum and mean jitter is 5 ms and 4 ms without security, and increases by 6 ms and 2 ms respectively with SRTP. Again, this performance drop in the Twinkle softphone is possibly due to the effects of ZRTP using the media channel. In contrast, maximum and mean jitter for the bare PC softphone with or without SRTP is close to zero.



	no Securit	with Secur
BarePC	0.12	0.22
Snom	13.44	13.46
Twinkle	4.73	10.6

Figure 39. **Maximum jitter with and without SRTP**



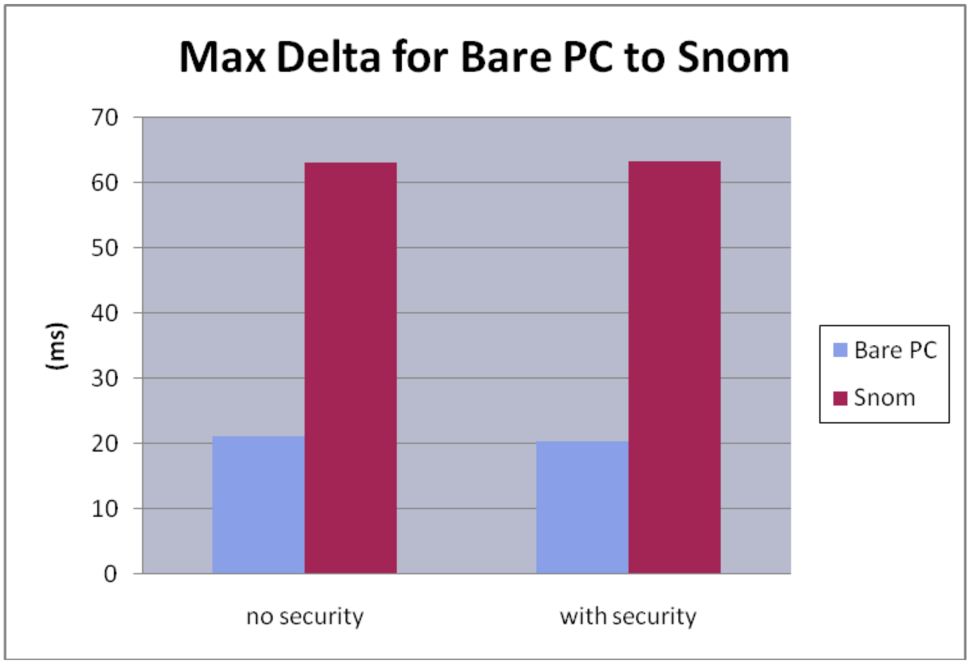
	no Securit	with Secur
BarePC	0.1	0.1
Snom	13.2	13.2
Twinkle	3.99	6.33

Figure 40. **Mean jitter with and without SRTP**

The above results for the bare PC softphone indicate that its streamlined processing of voice packets is able to reduce intrinsic delay and jitter with or without SRTP. Yet it is also evident that since delta and jitter values for all three softphones are within generally accepted limits, SRTP overhead has little or no effect on VoIP performance.

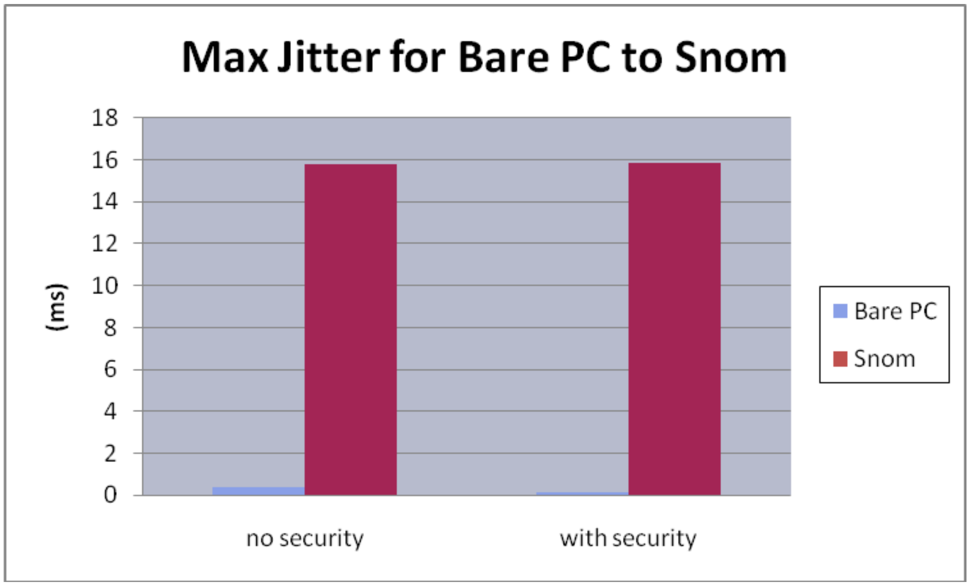
K. SRTP Delta and Jitter for snom-to-bare Calls

We also tested SRTP interoperability and VoIP performance when communicating between different softphones. This was done by measuring maximum delta, and maximum and mean jitter values on the respective softphones for calls between a snom softphone and a bare PC softphone using a 128-bit AES key and a 32-bit authentication tag. Maximum delta and maximum and mean jitter values with or without SRTP for bare PC to snom calls are shown in Figs. 41-43. These values can be compared with the corresponding values in Figs. 37, 39, and 40 respectively.



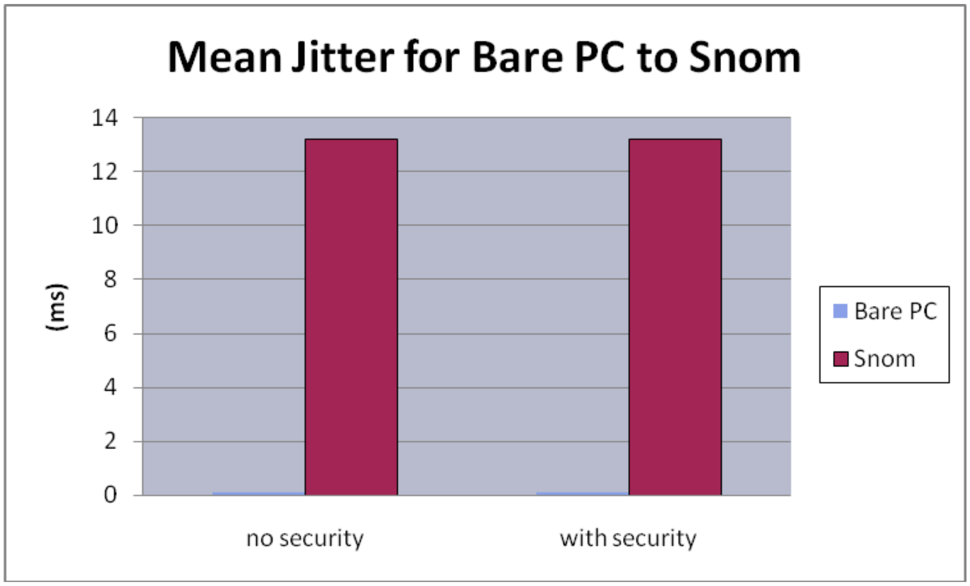
Max Delta	no security	with security
Bare PC	21.22	20.33
Snom	63.12	63.22

Figure 41. **SRTP Maximum delta for bare PC to snom**



Max Jitter	no security	with security
Bare PC	0.41	0.17
Snom	15.81	15.83

Figure 42. **SRTP Maximum jitter for bare PC to snom**

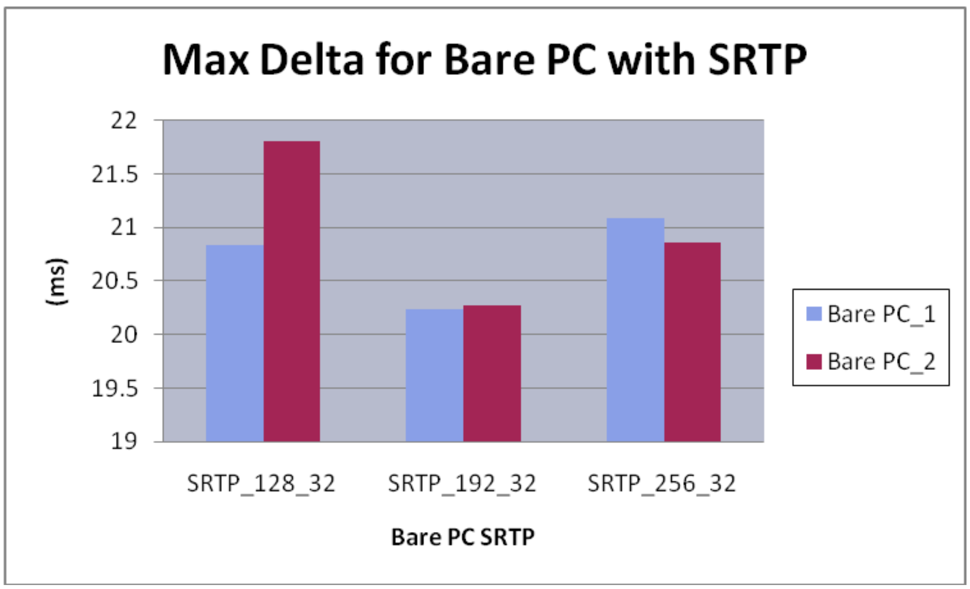


Mean Jitter	no security	with security
Bare PC	0.1	0.1
Snom	13.2	13.21

Figure 43. **SRTP Mean jitter for bare PC to snom**

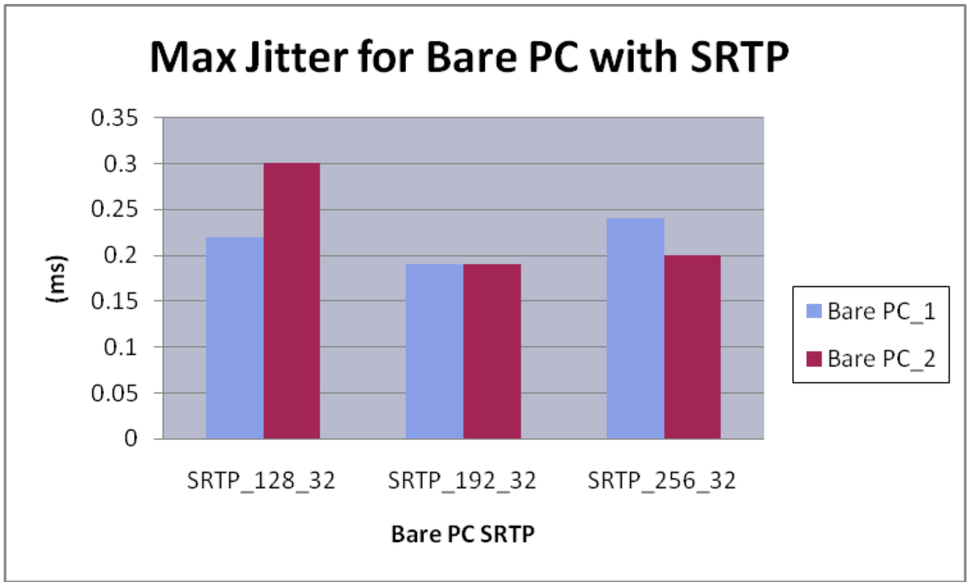
Maximum delta for the voice packet stream from the snom softphone is the same with or without SRTP but double that for snom to snom calls. However, maximum delta values for the stream from the bare PC softphone with or without SRTP are not significantly different compared to bare PC to bare PC calls. Maximum jitter values with or without SRTP are also the same but slightly higher for the stream from the snom softphone compared to snom to snom calls, but again, differences in maximum jitter values for the stream from the bare PC softphone are very small. Mean jitter values with or without SRTP for the stream from each softphone are unchanged for bare PC to snom calls. The increased values of maximum delta and maximum jitter for the stream from the snom softphone are possibly due to the difference in timing between the softphones when processing voice packets. More studies are needed to investigate these timing differences.

To evaluate the impact on VoIP performance with SRTP due to changing the AES key size, we measured maximum delta, and maximum and mean jitter values on a bare PC softphone with 192-bit or 256-bit AES keys and a 32-bit authentication tag (we were unable to test the snom softphone as it did not appear to support alternate AES key sizes). The results are compared with those for 128-bit AES keys (and a 32-bit authentication tag) in Figs. 44-45. The values of maximum delta and maximum jitter show little variation, and do not seem to have a simple relation to key size (the 192-bit key size has the best values and the least variation but the differences are very small). Also, the results for the two softphones are not identical. However, mean jitter is nearly constant for both bare PC softphones regardless of key size. Since the processing overhead for all authentication tag sizes is the same as explained above, the results using an 80-bit authentication tag would not be significantly different.



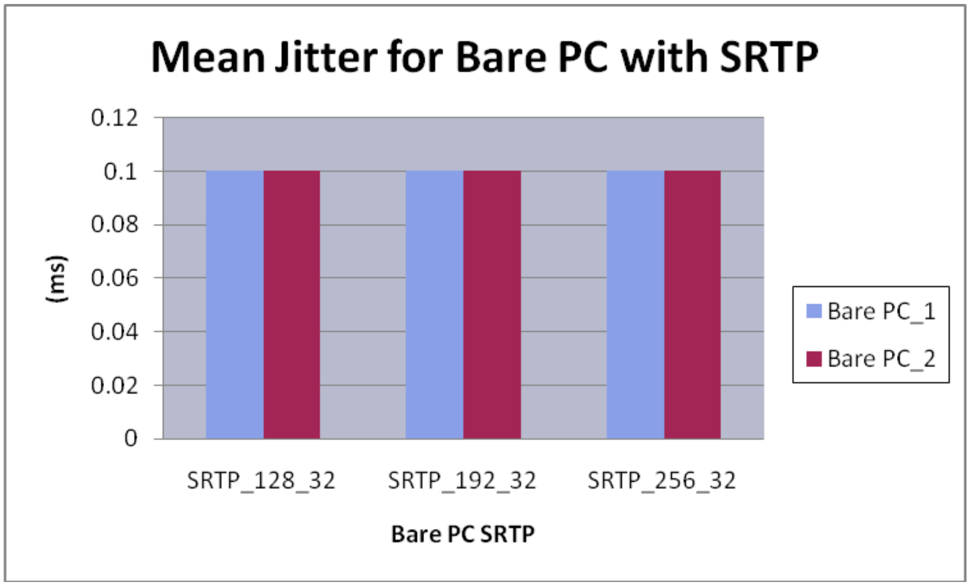
Max Delta	SRTP_128_32	SRTP_192_32	SRTP_256_32
Bare PC_1	20.84	20.24	21.09
Bare PC_2	21.81	20.27	20.86

Figure 44. **SRTP Max delta: varying AES key size**



Max Jitter	SRTP_128_32	SRTP_192_32	SRTP_256_32
Bare PC_1	0.22	0.19	0.24
Bare PC_2	0.3	0.19	0.2

Figure 45. **SRTP Max jitter: varying AES key size**



Mean Jitter	SRTP_128_32	SRTP_192_32	SRTP_256_32
Bare PC_1	0.1	0.1	0.1
Bare PC_2	0.1	0.1	0.1

Figure 46. **SRTP Mean jitter: varying AES key size**

L. *SRTP VoIP Throughput*

VoIP throughput for all three softphones without SRTP is 81.6 kbps without SRTP, and 83.23 kbps with SRTP when using a 128-bit AES key and a 32-bit authentication tag. Since SRTP encryption does not increase the size of the voice packet, the only increase in size is due to the 32-bit (or 80-bit) authentication tag. In an Ethernet, the total packet size including all network headers but excluding the CRC is 214 bytes without SRTP, and 218 bytes (or 224 bytes) with SRTP. Thus, the 2% increase in throughput with SRTP in our case simply reflects the 4-byte increase in packet size due to the authentication tag i.e., the increase in processing time due to SRTP is negligible and does not alter the throughput. Furthermore, all three softphones have the same throughput since their mean delta values are the same.

CHAPTER V. CONCLUSION

This dissertation presents new research on VoIP systems in a bare machine computing (BMC)/bare PC environment. The focus of this work is the implementation of SIP, SRTP, and other support protocols for VoIP systems on a bare PC, and the evaluation of these systems by conducting experiments to measure their performance. Specifically, this research has demonstrated that the development of interoperable, dynamically configurable and secure VoIP systems that run on a bare PC with no OS or kernel is a viable option to its OS-based counterparts. The bare PC VoIP SIP server and SIP user agent/softphone with SRTP, which were the focus of this research, are characterized by simple tasking, lean protocol implementations, and immunity against OS-based attacks.

We first described the design, implementation, and operations of a bare PC SIP server and SIP user agent with SRTP. These VoIP systems provide essential SIP and SRTP functionality with less overhead and better system security due to the absence of an OS. The tests conducted show that the bare PC SIP server can interoperate with bare PC and OS-based SIP softphones, and the bare PC SIP softphone can interoperate with OS-based softphones and SIP servers.

We then evaluated the performance of a bare PC SIP server by measuring its throughput and latency for registration, proxying, and redirection, with and without authentication. We compared its performance with that of an OpenSER server running on Linux and a Brekeke server running on Windows. We also determined timings for internal operations on the bare PC SIP server. The results show that the bare PC server performs better than the OS-based servers in most cases.

The exceptions are throughput performance for the invite redirect operation, and latency performance for the invite operation with authentication and the invite-not-found operation without authentication, for which the Linux server is better. It is expected that the performance of the bare PC server can be improved in these cases by optimized processing techniques and the use of more efficient search algorithms. The bare PC SIP server implementation can also be modified based on internal timings to reduce the cost of the most expensive operations. Our results serve as a baseline to assess the minimal overhead associated with basic SIP server operations for both OS-based and bare PC servers, and to help improve the performance of bare PC SIP servers. They also indicate the feasibility of deploying bare PC SIP servers in secure environments where OS-based vulnerabilities are a concern.

Finally, we compared VoIP performance with SRTP on a bare PC SIP softphone with snom and Twinkle softphones running on Windows and Linux respectively. In particular, we determined packet interarrival times (delta) and jitter, with and without SRTP, for these softphones. Maximum delta and maximum and mean jitter for the bare PC softphone, which has no operating system, are smaller than for the snom and Twinkle softphones. Mean delta values for all three softphones are close to the ideal value. We also verified that VoIP throughput on the bare PC softphone with

SRTP is close to the expected value. Measurement of internal processing times for SRTP operations on the bare PC softphone revealed that SRTP authentication is expensive than AES encryption. However, no SRTP operation degrades VoIP performance. Overall, the results indicate that SRTP adds negligible overhead to VoIP processing and has no observable effect on VoIP call quality.

Future research can investigate the use of TLS (Transport Layer Security) by the bare PC SIP server to secure the signaling channel, and for key exchange. An implementation of the bare PC SIP server that runs on TCP will provide flexibility, and further extend its capability to interoperate with OS-based servers. A beta version of such a SIP server exists and is being tested and improved. In summary, we have shown that the performance of VoIP SIP servers and softphones with SRTP may be improved with lean protocol implementations, simple tasking, and other bare PC-like softphone optimizations. We have also shown that bare PC VoIP systems can co-exist with OS-based systems.

CHAPTER VI. 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] M. Baugher, D. McGrew, M. Naslund, E. Carrara and K.Norrman,"The secure real-time transport protocol (SRTP)," RFC 3711, March 2004.

[3] 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.

[4] 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.

[5] 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.

[6] A. Alexander, A. L. Wijesinha, and R. Karne, "Implementing a VoIP SIP Server and a User Agent on a Bare PC," 2nd International Conference on Future Computational Technologies and Applications (Future Computing 2010), In Press.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] P. Zave, E. Cheung, G. W. Bond, and T. M. Smith, "Abstractions for Programming SIP Back-to-Back User Agents," IPTComm'09, 2009.

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

[14] 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.

[15] 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.

[16] K. Ono and H. Schulzrinne, The Impact of SCTP on SIP Server Scalability and Performance, GLOBECOM, pp. 1421-1425, 2008.

[17] S. Salsano, L. Veltri, and D. Papalilo, SIP security issues: The SIP authentication procedure and its processing load, *IEEE Network*, pp. 38-44, 2002.

[18] E. M. Nahum, J. M. Tracey, and C. P. Wright, Evaluating SIP server performance, in: 17th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Urbana-Champaign, Illinois, June 2007.

[19] M. Cortes, J. R. Ensor, and J. O. Esteban, On SIP Performance. *Bell Labs Technical Journal*, 9(3), pp. 155-173, 2004.

[20] C. Shen, H. Schulzrinne, and E. M. Nahum, Session Initiation Protocol (SIP) Server Overload Control: Design and Evaluation, IPTComm, pp. 149-173, 2008.

[21] V. A. Balasubramaniyan, A. Acharya, M. Ahamad, M. Srivatsa, I. Dacosta, and C. P. Wright, SERvartuka: Dynamic Distribution of State to Improve SIP Server Scalability, ICDCS, pp. 562-572, IEEE Computer Society, 2008.

[22] K. Ono and H. Schulzrinne, One Server Per City: Using TCP for Very Large SIP Servers, IPTComm, pp. 133-148, 2008.

[23] H. Jiang, A. Iyengar, E. M. Nahum, W. Segmuller, A. Tantawi, and C. P. Wright, Load Balancing for SIP Server Clusters, INFOCOM 2009.

[24] K. K. Ram, I. C. Fedeli, A. L. Cox, and S. Rixner, Explaining the Impact of Network Transport Protocols on SIP Proxy Performance, ISPASS, pp. 75-84, 2008.

[25] D.Wing, S. Fries, H. Tschofenig, and F. Audet, "Requirements and analysis of media security management protocols," RFC 5479, April 2009.

[26] S. Garg, N. Singh, and T. Tsai, "SRTP+, An efficient scheme for RTP packet authentication," Retrieved Nov. 2, 2008 from pubs.research.avayalabs.com/pdfs/ALR-2004-001-paper.pdf.

[27] S. Garg, N. Singh, and T. Tsai., "Schemes for enhancing the denial-of-service tolerance of SRTP," pp. 409-411, 1st Int. Conf. onSecurity and Privacy for Emerging Areas in Communications Networks (SECURECOMM 05), 2005.

[28] S. Spinsante, E. Gambi, E. Bottegoni, "Security solutions in VoIP applications," IEEE International Symposium on Consumer Electronics (ISCE 2008), pp. 1-4, 2008.

[29] 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.

[30] 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.

[31] G. Ford, R. Karne, A. L. Wijesinha, and P. Appiah-Kubi, The Design and Implementation of a Bare PC Email Server, with G. Ford et. al, 33rd Annual IEEE International

Computer Software and Applications Conference (COMPSAC), 2009.

[32] 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.

[33] G. H. Khaksari, A. L. Wijesinha, and R. Karne, “Secure VoIP using a Bare PC”, 3rd International Conference on New Technologies, Mobility and Security (NTMS), 2009.

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

[35] P. Zimmermann, A. Johnston, and J. Callas, “ZRTP: Media Path Key Agreement for Secure RTP,” Internet-Draft, March 2009.

[36] F. Andreason, M. Baugher, and D. Wing, “Session description protocol (SDP) security descriptions for media streams,” RFC4568, July 2006.

[37] D. Ignjatic, L. Dondeti, F. Audet, P. Lin, “MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia InternetKEYing (MIKEY),” RFC 4738, November 2006.

[38] T. Dierks and C. Allen, "The TLS protocol version 1.0," RFC 2246, January 1999.

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

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

[41] SIPp, http://sipp.sourceforge.net/doc/reference.html

[42] Brekeke SIP Server, http://www.brekeke.com/sip/

[43] Wireshark, http://www.wireshark.org.

[44] Twinkle, http://www.xs4all.nl/~mfnboer/twinkle/index.html.

[45] Oprofile-A System Profiler for Linux, July 31, 2009. [Online]. Available: http://oprofile.sourceforge.net/news/. Accessed: May 28, 2010.

CHAPTER VII. CURRICULUM VITA

NAME: **Andre Alexander**

PERMANENT ADDRESS: 1612 Chapel Ridge Ct. Hanover, MD 21076

PROGRAM OF STUDY: Applied Information Technology

DEGREE AND DATE TO BE CONFERRED: Doctor of Science, August 2010

Secondary education:

College attended	Dates	Degree	Graduation
Coppin State College	**09/1996 – 12/2001**	**B.S.**	**Dec 2001**

Major: **Computer Science**

Minor(s), if applicable: N/A

College attended	Dates	Degree	Graduation
Towson University	**01/2002 – 12/2003**	**M.S.**	**Dec 2003**

Major: **Applied Information Technology**

Minor(s), if applicable: N/A

Professional publications:

[1] Alexander, A. L. Wijesinha, and R. Karne, "Implementing a VoIP SIP Server and a User Agent on a Bare PC," 2nd International Conference on Future Computational Technologies and Applications (Future Computing 2010), In Press.

[2] 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.

[3] 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.

Professional positions held:

1) Department of Defense, Network Engineer Feb/2002 until Now
2) Social Security Administration, Network Engineer (Intern Program)