

# Large Scale SIP-aware Application Layer Firewall

Eilon Yardeni and Henning Schulzrinne, Department of Computer Science, Columbia University  
Gaston Ormazabal, Verizon Labs

**Abstract**—Placing voice traffic on the data network exposes it to the same attacks that plague the existing Internet infrastructure. Traditional perimeter security solutions cannot cope with the complexity of VoIP protocols at carrier-class performance. We have implemented a large-scale, SIP-aware application layer firewall capable of filtering VoIP traffic at carrier-class rates. We have also built an integrated testing and analysis tool and a testbed that validates its functionality and performance. Our testing tool is unique in its use of finer granularity measurements of pinhole opening and closing delays of the system under test (SUT) than previously available. The tool is further enhanced with an array of ten standard Pentium 3GHz processors to distribute VoIP traffic load generation, reaching the desired target performance rates. This paper describes our novel and original application layer firewall solution, the testing methodology and the testbed used to benchmark our solution.

## I. INTRODUCTION

PERIMETER security devices, such as firewalls, are meant to prevent hosts on the Internet from accessing hosts on the protected network. Firewalls primarily enforce access control policies that define which hosts and services are allowed to traverse the perimeter. The simplest type of packet filtering firewalls, work at the network layer and compare each packet to a set of security policy criteria to determine whether it should be forwarded or dropped. Rules may include source and destination IP address, source and destination port number and protocol used. Blocking through packet filtering, any random or malicious unauthorized network traffic from entering the protected network is the first line of defense against Denial of Service (DoS) attacks, which target the availability of services running in the protected network to legitimate users. Packet filtering firewalls need no further understanding of the traffic being limited. Some protocols, however, use additional randomly chosen port numbers for a protocol specific channel (e.g., FTP). In order to support these types of protocols, the firewall needs to examine packets at the application layer and to dynamically change packet filtering criteria according to application specific needs. Specifically, protecting VoIP networks utilizing the Session Initiation Protocol (SIP) [1], requires SIP application awareness in the perimeter firewall along with dynamic packet filtering capabilities.

SIP is an application layer signaling protocol, carried on the well-known port 5060, for creating, modifying, and terminating media sessions, such as Internet telephony calls, between one or more participants. SIP does not transport media content itself, but allows communicating parties to agree on how and what media to exchange. This is accomplished by using an offer/answer model that includes the Session Description Protocol (SDP) [2] as an important component of the SIP message scheme. Typically, SIP User Agents (UA) negotiate SIP sessions through a series of network elements called proxy servers. SIP proxy servers route SIP requests and responses to and from communicating SIP UAs. A SIP UA chooses a port number, associated with its IP address, for receiving the media stream, and sends it to the opposite SIP UA in the SDP body. Once the respective session parameters are negotiated and signaled by the UAs, unidirectional media streams begin to flow directly between the UAs through the dynamically allocated ports.

A SIP aware application layer firewall needs to intercept and parse the SIP session creation messages exchanged by the UAs, extract the negotiated port numbers and IP addresses from the SDP, and dynamically open the pinholes for the media streams. When a UA terminates the session, the firewall intercepts the appropriate message and closes the ports. For this purpose, the firewall needs to maintain two state tables. The first is a **session state** table, which contains active session entries. The second is a **connection state** table, which contains 5-tuple entries (source and destination IP address, source and destination port number and protocol used) for each active media stream. Each session entry from the session state table references one or more connection table entries, representing the session related media streams. The session state table is updated when a session is created or terminated and the connection table is then updated accordingly. The connection table, which represents the security policy enforced by the firewall, is traversed for every arriving packet to determine whether it should be forwarded or dropped. This stateful mode carries the burden of being extremely consumptive of CPU cycles, due to the full connection state table traversal for every arriving media packet and the session signaling processing. CPU exhaustion limits the currently available firewalls ability to handle high volumes of concurrent calls, usually to no more than a few hundreds.

Carrier-to-carrier VoIP peering, hosted IP Centrex and other multimedia packet-based services present new challenges for IP networks and edge networking technologies. Carriers are

confronted with deployment barriers such as service level assurance, NAT traversal and most importantly, security. Security is ideally provided with cryptographic techniques applied on the traffic content and supplemented with perimeter protection as a first line of defense. Securing the data channels with protocols such as IPsec and Transport Layer Security (TLS), however, involves the use of digital certificate-based key systems, which are today still difficult to manage at the carrier-class scale. The resulting alternative is protecting crucial network assets such as media gateways, signaling gateways, and application servers, through the use of network perimeter protection devices that can block potentially nefarious unwanted traffic from ever reaching those assets. The correct implementation of a SIP-aware firewall performing dynamic pinhole filtering at the network edge provides a good level of protection at a level of granularity not otherwise achievable with other current security technologies.

We have implemented a SIP-aware application layer firewall that filters SIP based VoIP traffic. Our firewall is based on a call control architecture, where the call control application logic is outside the firewall and handled by a SIP proxy. This fully SIP conformant solution alleviates firewall traversal that is fairly complex for a protocol such as SIP. Our solution can support up to 30,000 concurrent VoIP calls of signaling and media, thus satisfying carrier-class rates requirements. We have also designed a benchmarking methodology, built a generic testing tool for measuring the performance of SIP-aware dynamic packet filter devices at carrier-class rates, and applied the testing methodology to our firewall.

The remainder of this paper is organized as follows: we first describe our SIP-aware firewall implementation, second, we describe the benchmarking methodology and testing architecture, and finally we describe the benchmarking results.

## II. APPLICATION LAYER FIREWALL ARCHITECTURE

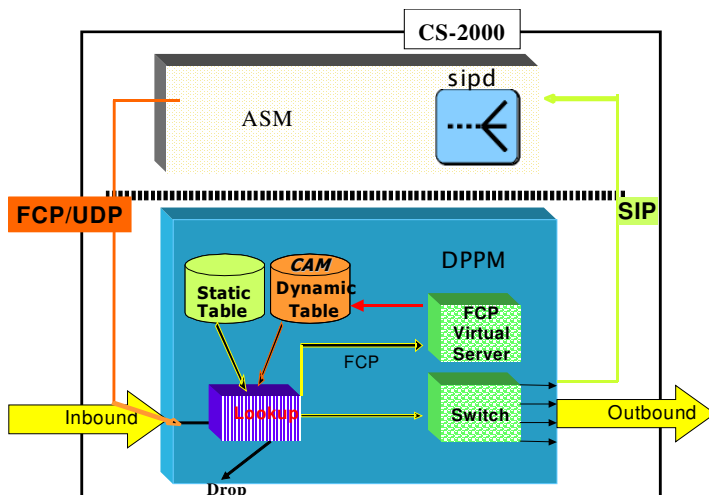


Figure 1: The sipd proxy resident in the CS-2000 Application Server Module (ASM), intercepts SIP messages and passes instructions to the Deep Packet Processing Module (DPPM) using a Firewall Control Protocol (FCP).

### A. Overview

Our solution combines two components: the Columbia developed SIP-Proxy *sipd*, which is part of the Columbia InterNet Multimedia Architecture (CINEMA) [3], and the commercial hardware CloudShield CS-2000 [4] fast packet processing application server.

The *sipd* proxy is a SIP redirect, forking proxy and registration server that provides name mapping, user location and scripting services. As such, *sipd* implements a full SIP stack and supports multiple transport protocols (e.g., TLS, TCP and UDP).

The CloudShield CS-2000 is designed to run deep packet inspection applications on high-speed networks. The CS-2000 consists of a Deep Packet Processing Module (DPPM) based on the Intel IXP 2800 network processor, consisting of 16 programmable data plane computers, a silicon database using Content Addressable Memory (CAM) technology, and a separate onboard Pentium-based Linux Application Server Module (ASM) for application development, management and control. Applications are written in a high-level proprietary language, called Rapid Application and Visualization Environment (RAVE), in the ASM, and subsequently “pushed” to the DPPM and converted into DPPM application logic for execution in real time. The CS-2000 has five GigE input ports and a total of 5Gb/s packet processing throughput.

In the proposed architecture (Figure 1), *sipd* is the SIP-aware application that controls the CS-2000 server, which performs dynamic packet filtering. The *sipd* proxy is executed in the CS-2000 ASM and instructs the DPPM which ports should be opened and closed for the Real Time Protocol (RTP) [5] media streams negotiated by SIP. This is done using a Firewall Control Protocol (FCP) that we have designed and implemented. This architecture allows segregation of the signaling processing and the media packet filtering which have been traditionally done on a shared CPU. The high concurrent call volume is achieved by keeping the connection state table in hardware using the CAM database.

### B. Call Flow Example

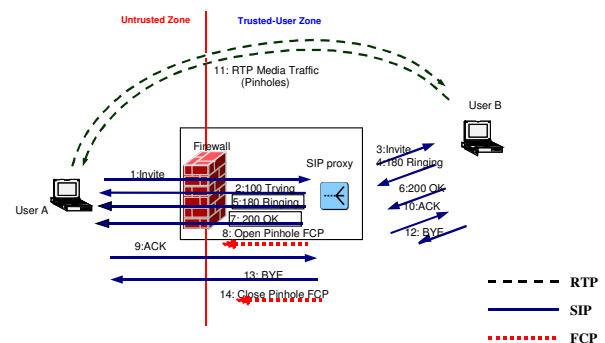


Figure 2: SIP and Firewall Control Protocol (FCP) call flow

```

1 INVITE User A -> sipd

INVITE sip:UserB@there.com SIP/2.0
Via: SIP/2.0/UDP here.com:5060
From: BigGuy <sip:UserA@here.com>
To: LittleGuy <sip:UserB@there.com>
Call-ID: 12345600@here.com
CSeq: 1 INVITE
Contact: BigGuy <sip:UserA@here.com>
Content-Type: application/sdp
Content-Length: 147

v=0
o=UserA 2890844526 2890844526 IN IP4 here.com
s=Session SDP
c=IN IP4 100.101.102.103
t=0 0
m=audio 49172 RTP/AVP 0
a=rtptime:0 PCMU/8000

6 200 OK B -> sipd

SIP/2.0 200 OK
Via: SIP/2.0/UDP here.com:5060
Record-Route: <sip:UserB@there.com;maddr=ss2.wcom.com>,
               <sip:UserB@there.com;maddr=ssl.wcom.com>
From: BigGuy <sip:UserA@here.com>
To: LittleGuy <sip:UserB@there.com>;tag=314159
Call-ID: 12345601@here.com
CSeq: 1 INVITE
Contact: LittleGuy <sip:UserB@there.com>
Content-Type: application/sdp
Content-Length: 147

v=0
o=UserB 2890844527 2890844527 IN IP4 there.com
s=Session SDP
c=IN IP4 110.111.112.113
t=0 0
m=audio 34560 RTP/AVP 0
a=rtptime:0 PCMU/8000

```

Figure 3: User A's INVITE Request (message 1) and User B's 200 OK Response (message 6)

A simple call flow example is shown in Figure 2. In this scenario User A sends an **INVITE** (message 1) request (Figure 3) to User B through the *sipd* SIP proxy. The *sipd* proxy intercepts the **INVITE** request, fetches the media IP address and port number from the SDP body (100.101.102.103 and 49172, respectively) and forwards the request to User B (message 3). User B responds with a **200 OK** (message 6) response (Figure 3) that contains the media IP address and port number in the SDP body. The *sipd* proxy fetches User B's media IP address and port number (110.111.112.113 and 34560, respectively), updates its state table and sends an "open" FCP command (message 8) to the DPPM. The DPPM updates the CAM database with the open pinholes and the RTP media streams can now flow through the firewall (message 11). Later on, User B terminates the session by sending a **BYE** request (12). The *sipd* proxy forwards the **BYE** request (message 13) to User A, removes the session from its state table and sends a "close" FCP command (message 14) to the DPPM. The DPPM removes the connection from the CAM database, which closes the pinholes.

### C. Firewall Control Module

We have enhanced the *sipd* proxy resident in the ASM, with a Firewall Control Module. This module parses the media streams' IP addresses and port numbers from the SDP, manages a session state table, and sends FCP commands to the DPPM. The *sipd* proxy intercepts **INVITE/200 OK** messages, and extracts the IP addresses and port numbers that were

negotiated for the RTP media streams. Subsequently, *sipd* constructs 5-tuple connection entries for an "open" pinhole FCP message, instructing the DPPM to open four pinholes; two pinholes for the RTP streams in each direction, and two additional pinholes for the accompanying Real Time Control Protocol (RTCP) channels. The *sipd* proxy maintains a session state table in memory, which contains active sessions, indexed by the SIP Call-ID header field— a globally unique-per-call identifier. The *sipd* proxy associates the 5-tuple entries to a session state table entry, so that when the corresponding **BYE** message arrives, the matching pinholes can be closed. Either side of the VoIP call can choose termination by sending a **BYE** message. The *sipd* proxy intercepts the **BYE** message, deletes the session from its session state table, and instructs the DPPM to close the pinholes using a "close" pinhole FCP message. The Firewall Control Message set consists of either "open" or "close" pinhole commands, encapsulated in a UDP packet, and containing four 5-tuples for the four pinholes that are opened or closed for each VoIP call. The need for the FCP surfaced when the CS-2000 built-in API used for pushing data updates from the ASM to the DPPM, in real time, failed to perform at high speeds. A delay of the order of two seconds to open a port was experienced, causing an unacceptable degradation in service, as a user answering a ringing phone, hears no audio for the first two seconds.

The DPPM is programmed to act as a dynamic packet filter peeking at layers 3 and 4 of the packet headers. It uses two connection state tables, **static** and **dynamic**, that are stored in the CAM database resident in the DPPM. The static table contains statically configured pinholes such as port 5060 for SIP traffic. The dynamic table contains 5-tuple connection entries that are updated by *sipd*. When a packet arrives at one of the DPPM input ports, it first constructs a 5-tuple connection identifier from the packet headers. The DPPM then tries to look for a match in the static connection table in the CAM. If there is a match, the packet is forwarded to an output port; otherwise a second attempt is made to lookup for a match in the dynamic connection table. If there is no match, the packet is dropped. .

The FCP message is sent by *sipd* from the ASM network interface, to a Virtual Server (VS), written in RAVE, which resides in the DPPM (Figure 1), and intercepts FCP UDP packets. The FCP packet enters the DPPM and is processed like any other incoming network packet. The VS identifies FCP packets by the special IP address and port (1.1.1.1 and 9999, respectively) they are sent to. The VS extracts the 5-tuple connection entries from the FCP packet and writes them to, or deletes them from the dynamic table in the CAM database. The FCP packet is not forwarded to an output port but processed and terminated at the VS. The FCP message also sets a timer for the updated four CAM entries correlating to a single VoIP call. The timer removes the CAM entries if the call is idle, meaning no RTP or RTCP packets are sent for the duration of the timer. Conversely, any CAM match will reset the timer. This functionality guarantees that there are no open pinholes left in case of a signaling fault, a scenario that could leave a security backdoor. Since the DPPM is designed to peek

into packets at wire-speed, the FCP allows *sipd* to send real-time data updates to the DPPM at GigE rates.

#### D. Signaling Performance

Signaling processing is somewhat limited in our implementation as *sipd* runs on a 1GHz CPU within the ASM. Singh *et al* [6] showed that *sipd* can handle up to 900 calls/second when running on a Pentium IV 3GHz CPU with 1GB memory. Our proposed architecture is highly scalable in signaling processing power, as *sipd* could be placed on a separate, speedier, host located in the trusted network. For even higher signaling processing power, an array of *sipd* servers, running in load sharing mode [6] and controlling a single CS-2000 server, could also be used. An array of six *sipd* servers was proved to handle up to 2800 calls per second. This roughly translates to a carrier-class grade of 10 million BHCA (Busy Hour Call Attempts).

#### E. Security Considerations

As the FCP packets are conveyed to the DPPM through a general traffic port, a security issue is raised because in principle a malicious user could write directly into the CAM database. In our implementation, one of the GigE input ports is reserved for the FCP channel, so that FCP packets are segregated from the general traffic. The ASM running the *sipd* is connected to the DPPM through this reserved input port. If more than one SIP-proxy is used to control the firewall, they should be placed in a private subnet, isolated from the external network, and connected to the reserved input port. Additional mechanisms to protect this channel can be applied, for example, an Access Control List (ACL) specifying the FCP hosts. Cryptography based authentication mechanisms could not be currently used as the DPPM lacks cryptographic capabilities, but is recommended when the DPPM is enhanced with this technology.

### III. BENCHMARKING METHODOLOGY

#### A. Overview

The benchmarking methodology’s primary aim is the verification of the proper functioning of a dynamically allocating pinhole implementation, and its scalability and performance at carrier-scale traffic rates. The stateful packet filtering we describe is very consumptive of both memory and CPU. The memory utilization is expected to be linear according to the size of the 5-tuple connection state table. The CPU usage has a direct impact on the speed of pinhole opening and closing since a full connection state table traversal is needed for every arriving RTP packet. As the number of concurrent calls or incoming call rate rises, the CPU may get overloaded. An overloaded CPU may delay signaling processing and miss SIP **BYE** messages, causing pinhole closing delays that would start getting longer and longer until at some point the CPU can no longer handle any new calls. Determination of opening and closing delays is a significant

measure of firewall operation efficiency. These delays should be measured as a function of the call rate and the number of concurrent calls.

#### B. Testing Apparatus and Instrumentation

The measurements are performed in a controlled IP telephony test bed consisting of several SIP UAs, generating VoIP calls that traverse the SIP firewall. The firewall’s onboard SIP-proxy routes the signaling traffic between the UAs. The methodology proposed here includes the use of “originator” and “target” Integrated End-Points (IEPs) and “loader” and “handler” traffic generators.

#### Integrated End Points

The IEPs incorporate traffic generation and analysis tools. Specifically, the IEPs have VoIP traffic generation for both SIP signaling and RTP media, scanning probes and a protocol analyzer. The IEPs’ traffic generation component is based on a Columbia developed SIP UA called *sipua*, capable of generating both signaling and correlating media traffic for simulating VoIP calls. The originator IEP is placed at the untrusted side of the firewall, and the target IEP is placed at the trusted side. The originator IEP also includes the **nmap** [7] port scanner, used for scanning probes, whereas the target IEP includes the **snort** [8] protocol analyzer. The IEPs are used as traffic injection tools at the originator end, and as a traffic analyzer at the target end. The IEPs run on two Centrino 1.8GHz CPU, with 512 MB of memory, running Redhat Linux (Mandrake), IBM T42 ThinkPad hosts.

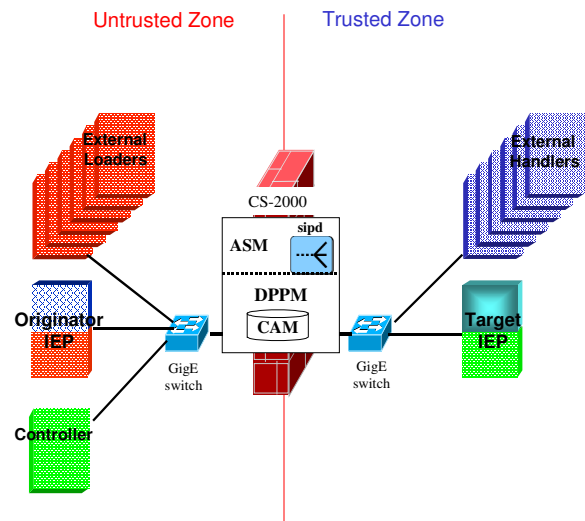


Figure 4: Load Generation and Pinhole Measurement Testbed

#### Large Scale Testing Environment

The IEPs, were supplemented with five pairs of additional hosts running *sipua*, each consisting of a “loader” placed at the untrusted side, and a “handler” placed at the trusted side. These hosts generate VoIP calls that traverse the SIP firewall for load generation purposes. The external loaders and handlers run on ten Pentium IV 3GHz CPUs, with 1GB of memory each, running Redhat Linux (Fedora). A central benchmark manager, called the controller, placed at either side

of the firewall and using SSH as a control channel, coordinates the execution of the benchmark. The controller runs on a Pentium III 1GHz CPU, with 512MB memory. The *sipd* proxy runs on a Pentium III 1GHz CPU, with 1GB memory Linux server, which resides in the ASM. Three of the five input ports of the CS-2000 are used. Two ports are used for connecting GigE switches, one from the untrusted side and one from the trusted side. The third port is connected to the ASM interface that is used by *sipd*.

### C. Measurements Methodology

#### Verification that only “signaled” pinholes are open

The objective of the testing is to launch traffic at the originating end and verify what traffic traversed the firewall and can be detected at the target end. To verify that all the ports that are not defined within the firewall rule-set, and hence not currently dynamically allocated are closed, traffic must be generated across the entire UDP and TCP port ranges from a legitimate IP address, while monitoring this traffic on the target end. The IEP is used to launch calls associated with a pair of legitimately opened pinholes. The IEP scanning probe component, launched from the same legitimate IP address, probes the entire TCP and UDP port ranges for that particular originating IP address. The destination IEP analyzes the arriving traffic and discriminates between the allowed traffic per the firewalls rules and traffic addressed to ports other than the legitimate dynamic ports. The expected result is that no traffic other than that addressed to ports dynamically allowed should appear at the target end. The presence of any ports other than those dynamically allocated would indicate a failure in the implementation.

#### Measurement of pinhole open and close delays

This measurement verifies two areas:

- 1) The speed with which a firewall correlates the information from the **INVITE/200OK** messages and the opening of the pinhole, the pinhole opening delay. This measures the implementation’s ability to prevent blocking the beginning of the audio conversations.
- 2) The length of time a pinhole remains open after a call has effectively terminated, the closing delay. The closing delay is defined by the time the last RTP packet sent from the originating IEP is detected by the target IEP. This is a very important measurement as it helps to characterize the firewall in terms of its commitment to provide absolute security.

The opening and closing delays are measured using a manipulation of the RTP sequence number and marker bit header fields, along with monitoring of the packets being received by both IEPs. For the open delay, the IEP starts the RTP media stream with a zero sequence number. RTP packets are then sent with sequentially increasing sequence numbers every 20 ms. At the analysis stage, the IEP uses the first recorded RTP sequence number as the indicator of the number of packets that were dropped by the firewall, before the pinhole was opened. The pinhole open delay is thus computed by multiplying the number of dropped packets by the 20 ms

packetization interval. For the closing delay, the originator IEP does not stop the RTP stream after it sends a **BYE** message to the target IEP. The IEP sets the marker bit in RTP packets that are sent after the **BYE**. Some RTP packets may traverse the firewall while the **BYE** message is processed and until the pinhole is actually closed. The set marker bit distinguishes RTP packets, which traversed the firewall after the **BYE** message. At the analysis stage, the target IEP counts the number of packets having the marker bit set. The pinhole closing delay is computed using the same multiplication operation. For finer granularity measurement the originating IEP can be set to send these “post **BYE**” RTP packets at a shorter packetization interval. We used a 10 ms packetization interval for our closing delay measurements.

#### Measurement under load

For measuring pinhole opening and closing delays while the firewall is loaded, we use our distributed VoIP call generation architecture. The cost of a commercial VoIP traffic generator is on the order of a few hundred thousand dollars. The proposed architecture is based on “loaders” and “handlers” standard Pentium CPU machine pairs running the *sipua* tool in traffic generation mode (Figure 4). The array of machines is used to generate external load on the firewall before internal load is generated for pinhole opening and closing delay measurements. The controller reads an input benchmark configuration file that specifies:

- The loader and handler user names
- The *sipd* proxy IP address
- The IP addresses of the loaders, handlers and IEPs
- The calls per second rate
- The total number of calls to generate

The controller then establishes the configurable number of concurrent calls that are handled by the firewall. Once the load is established, the controller invokes the IEPs for measuring the pinhole opening and closing delays. The IEPs create and destroy calls at the configured call rate. Once pinhole measurements are done, the controller tears down all the established calls and analyzes the IEPs’ output.

## IV. BENCHMARKING RESULTS

### A. Traffic Volume

We generated 6,000 concurrent calls from each pair of loader and handler hosts. Each call consists of two RTP streams with 160 bytes RTP packet payload. We could not generate more than 6,000 calls from a single host due to CPU exhaustion. As the CPU is loaded, signaling processing is delayed and RTP packets are sent further and further apart. As a result, at some point no more new calls can be established and total generated bandwidth is limited to about 40 Mb/s. From the five pairs of loaders and handlers we were able to generate up to 30,000 concurrent calls, namely 30,000 RTP streams in each direction. The IEP machines could not generate more than 300 calls per second since higher call rates introduced an increasing delay in the 20ms RTP packetization

intervals, which is essential for our pinhole opening and closing delay measurements.

### B. Results

Table 1 describes our testing results. Note that we do not show the results of measurements taken with lower calls per second rates, as they all show zero opening and closing delays.

concurrent calls	calls per second	open delay (ms)	close delay (ms)
10,000	300	15	0
15,000	300	14.8	0
20,000	300	15	0
25,000	300	15	0
30,000	300	15.4	3.4

Table 1: Pinhole open and close delay test results

The results show flawless behavior of our SIP firewall implementation. The opening delays are negligible as on average less than one RTP packet was dropped before the pinhole is opened. Some minor closing delays were detected when 30,000 concurrent calls were established. The signaling processing and session state table lookup that occurs in the *sipd* proxy is assumed to cause this delay. Upon termination of existing calls, this table is traversed before a “close” pinhole FCP message is sent to the DPPM. It should be noted, however, that the state table look up in *sipd* is executed only for signaling packets, representing approximately 3% of the total payload handled by the system. The far more intensive state table look times required by the arriving RTP packets, is handled by the CAM table in the DPPM, and processed in one CPU cycle, thus affording the exceptional processing speeds of this system. Improvements to *sipd* table lookup can be implemented by increasing the host’s CPU power or by load balancing a networked array of *sipds* to match the processing power provided by the DPPM.

## V. CONCLUSION

As perimeter security has become of prime importance to VoIP services carrier providers, our work suggests a highly scalable solution. We have also developed the methodology for verifying this SIP based dynamic pinhole filtering functionality. This implementation leverages a fast parallel processing packet-processing server - that uses a CAM database for storing the huge connection state table associated with high volumes of concurrent calls - with the full SIP conformance afforded by the Columbia SIP Proxy - supplying the signaling processing out of band. By following the call control architecture, and a novel and original method of data exchange, we gained fast and simple development of our SIP-aware firewall. The resulting prototype described in this paper has the very unique property of being simultaneously fully SIP conformant while also performing at carrier-class rates. The compactness of the hardware platform also makes this solution the most cost-effective for a device performing at these rates. Furthermore, the testing and benchmarking methodology afforded us the ability to apply it to our solution, demonstrating that it performed flawlessly. Using our current test environment we showed that the firewall can handle 30,000 concurrent calls at 300 calls per second rates. The fully

scalable architecture for the signaling processing module indicates that the signaling processing can be improved linearly with the addition of other SIP proxies in a load-sharing configuration. Testing and evaluating such a device will be the subject of follow-up research.

## ACKNOWLEDGMENT

We would like to thank Jim Sylvester, Vice President of Systems Integration and Testing, and Stu Elby, Vice President of Network Architecture, at Verizon Laboratories, for their sponsorship of this work, and their continued interest and support throughout the year.

We would also like to thank Jonathan Lennox, the primary architect of our SIP-proxy *sipd*, and Kundan Singh the primary architect of our SIP UA *sipua* for their contributions throughout this work.

## REFERENCES

- [1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. R. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, “SIP: Session Initiation Protocol”, RFC 3261, June 2002.
- [2] J. Rosenberg, H. Schulzrinne, “An offer/answer model with the session description protocol (SDP)”, RFC 3264, June 2002.
- [3] K. Singh, W. Jiang, J. Lennox, S. Narayanan, and H. Schulzrinne, “CINEMA: Columbia InterNet Extensible Multimedia Architecture”, Tech. Rep. CUCS-011-02, Department of Computer Science, Columbia University, New York, New York, USA, May 2002. B. Smith, “An approach to graphs of linear forms (Unpublished work style),” unpublished.
- [4] CloudShield Technologies, [http://www.cloudshield.com/what\\_we\\_do/cs2000.html](http://www.cloudshield.com/what_we_do/cs2000.html)
- [5] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, RFC 3550, July 2003.
- [6] K. Singh and H. Schulzrinne, “Failover and Load Sharing in SIP Telephony”, International Symposium on Performance Evaluation of Computer and Telecommunication systems (SPECTS), Philadelphia, PA, July 2005.
- [7] nmap, “Network Mapper”, <http://www.insecure.org/nmap/>
- [8] snort, <http://www.snort.org/>